Thapa_iaf603_final_project.ipynb and Sanga_iaf603_final_project.ipynb

```
import os # operating system
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # standard graphics
import seaborn as sns # fancier graphics
from scipy import stats
from sklearn import preprocessing
from functools import reduce
import sqlite3


from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving AB_US_2020.csv to AB_US_2020.csv

```
#bnb = pd.read_csv("Downloads/AB_US_2020.csv", sep=",",low_memory=False)
#bnb
#bnb = pd.read_csv('AB_US_2020.csv',sep=",",low_memory=False)
#bnb
import io
bnb = pd.read_csv(io.BytesIO(uploaded['AB_US_2020.csv']))
bnb
```

```
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2718: Dty
  interactivity=interactivity, compiler=compiler, result=result)
```

	id	name	host_id	host_name	neighbourhood_group	ne
0	38585	Charming Victorian home - twin beds + breakfast	165529	Evelyne	NaN	
1	80905	French Chic Loft	427027	Celeste	NaN	
2	108061	Walk to stores/parks/downtown. Fenced yard/Pet...	320564	Lisa	NaN	
3	155305	Cottage! BonPaul + Sharky's Hostel	746673	BonPaul	NaN	
4	160594	Historic Grove Park	769252	Elizabeth	NaN	
...	...	...	...	...	...	
226025	45506143	DC Hidden In Plain "Site"	25973146	Marci	NaN	Ch Q

```
bnb.dtypes
```

```
id                                 int64
name                              object
host_id                            int64
host_name                         object
neighbourhood_group               object
neighbourhood                     object
latitude                         float64
longitude                        float64
room_type                         object
price                              int64
minimum_nights                     int64
number_of_reviews                  int64
last_review                       object
reviews_per_month                float64
calculated_host_listings_count     int64
availability_365                   int64
city                              object
dtype: object
```

```
bnb.head()
```

	id	name	host_id	host_name	neighbourhood_group	neighbourh
0	38585	Charming Victorian home - twin beds + breakfast	165529	Evelyne	NaN	28
1	80905	French Chic Loft	427027	Celeste	NaN	28
2	108061	Walk to stores/parks/downtown. Fenced yard/Pet...	320564	Lisa	NaN	28
3	155305	Cottage! BonPaul + Sharky's Hostel	746673	BonPaul	NaN	28

```
bnb.tail()
```

	id	name	host_id	host_name	neighbourhood_group	neighbourho
226025	45506143	DC Hidden In Plain "Site"	25973146	Marci	NaN	Downto Chinatown, P Quarters, Mo Ver
226026	45511428	DC 3 BR w/ screen porch 3 blck to metro w/ par...	231133074	Thomas	NaN	Brookla Brentwo Lang
226027	45514685	Charming Penthouse Apt w/ Rooftop Terrace in L...	33758935	Bassem	NaN	Shaw, Lo Ci
226028	45516412	Adams Morgan/Nat'l Zoo 1 BR Apt #32	23193071	Michael	NaN	Kalora Heights, Ada Morgan, La Heig
226029	45517735	Beautiful large one-bedroom w/ washer and dryer	17789858	Adam	NaN	Edgewo Bloomingd Truxton Cir Ecking

```
bnb.shape
```

```
(226030, 17)
```

```
bnb.describe()
```

	id	host_id	latitude	longitude	price	minimum_
count	2.260300e+05	2.260300e+05	226030.000000	226030.000000	226030.000000	2.2603
mean	2.547176e+07	9.352385e+07	35.662829	-103.220662	219.716529	4.5254
std	1.317814e+07	9.827422e+07	6.849855	26.222091	570.353609	2.1033
min	1.090000e+02	2.300000e+01	18.920990	-159.714900	0.000000	1.0000
25%	1.515890e+07	1.399275e+07	32.761783	-118.598115	75.000000	1.0000
50%	2.590916e+07	5.138266e+07	37.261125	-97.817200	121.000000	2.0000
75%	3.772624e+07	1.497179e+08	40.724038	-76.919322	201.000000	7.0000
max	4.556085e+07	3.679176e+08	47.734620	-70.995950	24999.000000	1.0000

```
bnb.isnull().sum()
```

```
id                                   0
name                                28
host_id                              0
host_name                           33
neighbourhood_group                115845
neighbourhood                        0
latitude                             0
longitude                            0
room_type                            0
price                                0
minimum_nights                       0
number_of_reviews                    0
last_review                      48602
reviews_per_month                48602
calculated_host_listings_count       0
availability_365                     0
city                                 0
dtype: int64
```

```
miss = bnb.isna().sum()
miss /= bnb.shape[0]
miss *=100
miss = miss.to_frame().rename(columns={0:'Precentage Of Missing Values'})
miss
```

	Precentage Of Missing Values
id	0.000000
name	0.012388
host_id	0.000000
host_name	0.014600
neighbourhood_group	51.252046
neighbourhood	0.000000
latitude	0.000000
longitude	0.000000
room_type	0.000000
price	0.000000
minimum_nights	0.000000
number_of_reviews	0.000000
last_review	21.502455

```
to_drop = ['id',
           'host_id',
           'neighbourhood_group',
           'host_name',
           'last_review']

bnb.drop(to_drop, inplace = True, axis = 1)
bnb.head()
```

	name	neighbourhood	latitude	longitude	room_type	price	minim
0	Charming Victorian home - twin beds + breakfast	28804	35.65146	-82.62792	Private room	60	
1	French Chic Loft	28801	35.59779	-82.55540	Entire home/apt	470	
2	Walk to stores/parks/downtown. Fenced yard/Pet...	28801	35.60670	-82.55563	Entire home/apt	75	
3	Cottage! BonPaul + Sharky's Hostel	28806	35.57864	-82.59578	Entire home/apt	90	
4	Historic Grove Park	28801	35.61442	-82.54127	Private room	125	

```
bnb.isnull().sum()
```

```
name                                28
neighbourhood                        0
latitude                             0
longitude                            0
room_type                            0
price                                0
minimum_nights                       0
number_of_reviews                    0
reviews_per_month                48602
calculated_host_listings_count       0
availability_365                     0
city                                 0
dtype: int64
```

```
bnb['reviews_per_month'].fillna(bnb['reviews_per_month'].mean(),inplace=True)
```

```
bnb.dropna(inplace=True)
```

```
bnb.shape
```

```
(226002, 12)
```

```
bnb.isnull().sum()
```

```
name                            0
neighbourhood                   0
latitude                        0
longitude                       0
room_type                       0
price                           0
minimum_nights                  0
number_of_reviews               0
reviews_per_month               0
calculated_host_listings_count  0
availability_365                0
city                            0
dtype: int64
```

```
bnb.dtypes
```

```
name                             object
neighbourhood                    object
latitude                        float64
longitude                       float64
room_type                        object
price                             int64
minimum_nights                    int64
number_of_reviews                 int64
reviews_per_month               float64
calculated_host_listings_count    int64
availability_365                  int64
```

```
city                                    object
dtype: object
```

```
bnb.price.value_counts()
```

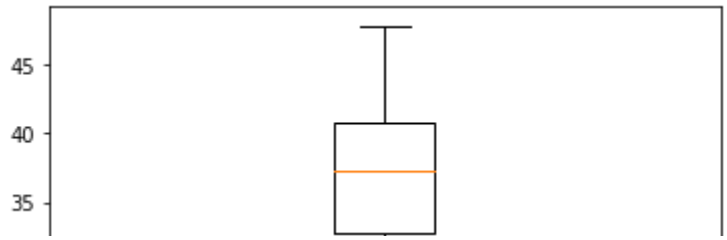
```
100     5825
150     5721
75      4710
50      4401
80      3906
        ...
1236       1
1364       1
1620       1
1876       1
2047       1
Name: price, Length: 1975, dtype: int64
```

```
index = bnb[(bnb['price'] >= 10000)|(bnb['price'] <= 10)].index
bnb.drop(index, inplace=True)
bnb['price'].describe()
```

```
count    225645.000000
mean        208.506473
std         403.825272
min          11.000000
25%          75.000000
50%         121.000000
75%         200.000000
max        9999.000000
Name: price, dtype: float64
```

```
# checking the skewness
print(bnb['latitude'].skew())
bnb['latitude'].describe()
# Box plot
plt.boxplot(bnb['latitude'])
plt.show()
# Identifying outliers for price
print(bnb['latitude'].quantile(0.10))
print(bnb['latitude'].quantile(0.90))
```

```
-0.7446626923779746
```




```
print(bnb['latitude'].quantile(0.50))
print(bnb['latitude'].quantile(0.95))
```

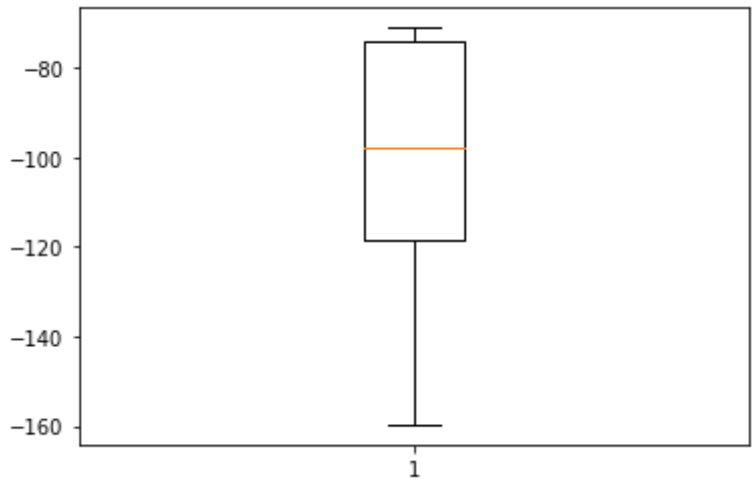
```
37.26332
45.140235999999994
```



```
bnb['latitude'] = np.where(bnb['latitude'] > 45, 37, bnb['latitude'])


# checking the skewness
print(bnb['longitude'].skew())
bnb['longitude'].describe()        # skewness for price is -0.4866482922764107
# Box plot
plt.boxplot(bnb['longitude'])
plt.show()
# Identifying outliers for price
print(bnb['longitude'].quantile(0.10))
print(bnb['longitude'].quantile(0.90))
```

```
-0.5062768759703664
```




```
-123.018948
-73.94145
```

```
print(bnb['longitude'].quantile(0.50))
print(bnb['longitude'].quantile(0.95))
```

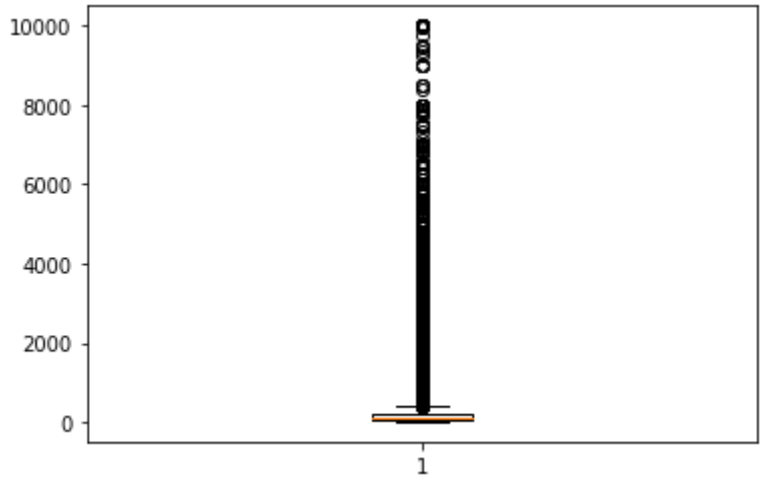
```
-97.81409000000001
-73.876804
```

```
bnb['longitude'] = np.where(bnb['longitude'] > -73.88, -97.81, bnb['longitude'])
```

```
bnb['longitude'] = np.where(bnb['longitude'] > -73.86, -97.81, bnb['longitude'])
```

```
# checking the skewness
print(bnb['price'].skew())
bnb['price'].describe()
# Box plot
plt.boxplot(bnb['price'])
plt.show()
# Identifying outliers for price
print(bnb['price'].quantile(0.10))
print(bnb['price'].quantile(0.90))
```

```
12.15337506589865
```




```
50.0
375.0
```

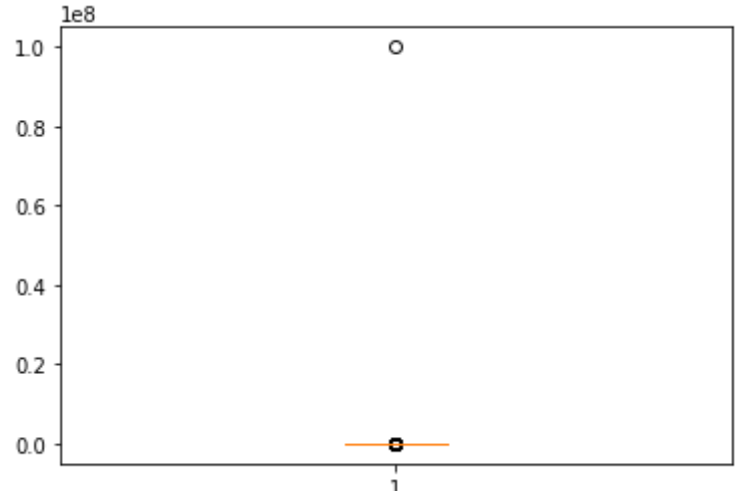
```
print(bnb['price'].quantile(0.50))
print(bnb['price'].quantile(0.95))
```

```
121.0
597.0
```

```
bnb['price'] = np.where(bnb['price'] > 597, 121, bnb['price'])
```

```
# checking the skewness
print(bnb['minimum_nights'].skew())
bnb['minimum_nights'].describe()        # skewness for price is 1.5996741660233658
# Box plot
plt.boxplot(bnb['minimum_nights'])
plt.show()
# Identifying outliers for price
print(bnb['minimum_nights'].quantile(0.10))
print(bnb['minimum_nights'].quantile(0.90))
```

```
475.0210421471036
```




```
print(bnb['minimum_nights'].quantile(0.50))
print(bnb['minimum_nights'].quantile(0.95))
```

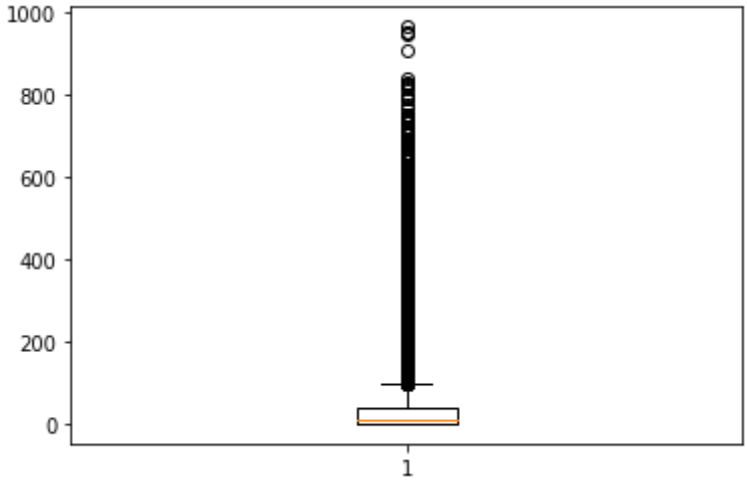
```
2.0
30.0
```

```
bnb['minimum_nights'] = np.where(bnb['minimum_nights'] > 30, 2, bnb['minimum_nights'])
bnb.describe()
```

	latitude	longitude	price	minimum_nights	number_of_reviews
count	225645.000000	225645.000000	225645.00000	225645.000000	225645.000000
mean	35.134465	-104.528139	146.87959	7.099763	34.537331
std	6.333456	25.258623	104.46235	10.262867	63.626603
min	18.920990	-159.714900	11.00000	1.000000	0.000000
25%	32.761780	-118.598890	75.00000	1.000000	1.000000
50%	37.000000	-97.814090	121.00000	2.000000	8.000000
75%	40.690100	-80.119560	184.00000	5.000000	39.000000
max	44.999970	-73.880010	597.00000	30.000000	966.000000

```
# checking the skewness
print(bnb['number_of_reviews'].skew())
bnb['number_of_reviews'].describe()        # skewness for price is 3.2033875173642374
# Box plot
plt.boxplot(bnb['number_of_reviews'])
plt.show()
# Identifying outliers for price
print(bnb['number_of_reviews'].quantile(0.10))
print(bnb['number_of_reviews'].quantile(0.90))   # outliers for price is 1.0, 122.0
```

```
3.562643224187558
```




```
0.0
103.0
```

```
print(bnb['number_of_reviews'].quantile(0.50))
print(bnb['number_of_reviews'].quantile(0.95))
```

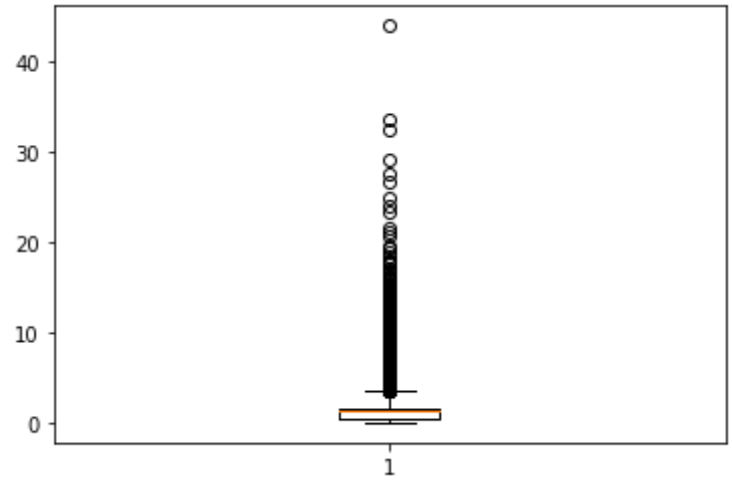
```
8.0
161.0
```

```
bnb['number_of_reviews'] = np.where(bnb['number_of_reviews'] > 161, 8, bnb['number_of_
bnb.describe()
```

	latitude	longitude	price	minimum_nights	number_of_reviews
count	225645.000000	225645.000000	225645.00000	225645.000000	225645.000000
mean	35.134465	-104.528139	146.87959	7.099763	22.423852
std	6.333456	25.258623	104.46235	10.262867	33.883824
min	18.920990	-159.714900	11.00000	1.000000	0.000000
25%	32.761780	-118.598890	75.00000	1.000000	1.000000
50%	37.000000	-97.814090	121.00000	2.000000	8.000000
75%	40.690100	-80.119560	184.00000	5.000000	29.000000
max	44.999970	-73.880010	597.00000	30.000000	161.000000

```
# checking the skewness
print(bnb['reviews_per_month'].skew())
bnb['reviews_per_month'].describe()        # skewness for price is 2.3179279392801484
# Box plot
plt.boxplot(bnb['reviews_per_month'])
plt.show()
# Identifying outliers for price
print(bnb['reviews_per_month'].quantile(0.10))
print(bnb['reviews_per_month'].quantile(0.90))
```

```
2.6157076592195514
```




```
0.10999999999999999
3.26
```

```
print(bnb['reviews_per_month'].quantile(0.50))
print(bnb['reviews_per_month'].quantile(0.95))
```

```
1.37
4.41
```

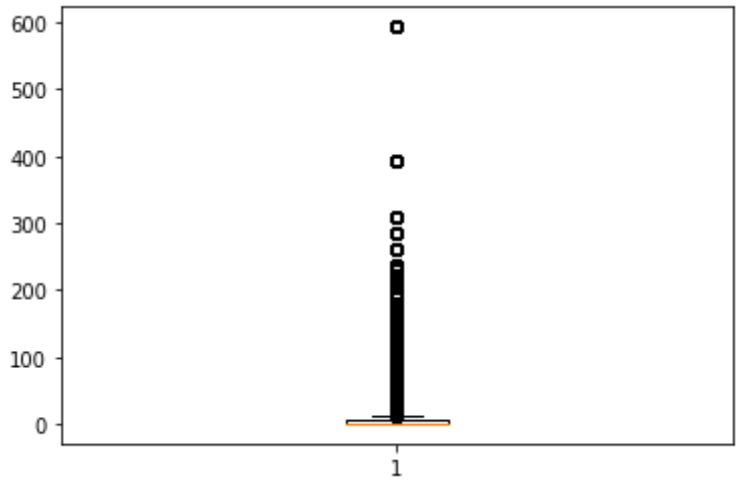
```
bnb['reviews_per_month'] = np.where(bnb['reviews_per_month'] > 4.41, 1.37, bnb['reviev
bnb.describe()
```

	latitude	longitude	price	minimum_nights	number_of_reviews
count	225645.000000	225645.000000	225645.00000	225645.000000	225645.000000
mean	35.134465	-104.528139	146.87959	7.099763	22.423852
std	6.333456	25.258623	104.46235	10.262867	33.883824
min	18.920990	-159.714900	11.00000	1.000000	0.000000
25%	32.761780	-118.598890	75.00000	1.000000	1.000000
50%	37.000000	-97.814090	121.00000	2.000000	8.000000
75%	40.690100	-80.119560	184.00000	5.000000	29.000000
max	44.999970	-73.880010	597.00000	30.000000	161.000000

```
# checking the skewness
print(bnb['calculated_host_listings_count'].skew())
bnb['calculated_host_listings_count'].describe()
# Box plot
plt.boxplot(bnb['calculated_host_listings_count'])
plt.show()
# Identifying outliers for price
print(bnb['calculated host listings count'].quantile(0.10))
```

```
print(bnb['calculated_host_listings_count'].quantile(0.90))
```

```
6.252764437340647
```




```
1.0
37.0
```

```
print(bnb['calculated_host_listings_count'].quantile(0.50))
print(bnb['calculated_host_listings_count'].quantile(0.95))
```

```
2.0
90.0
```

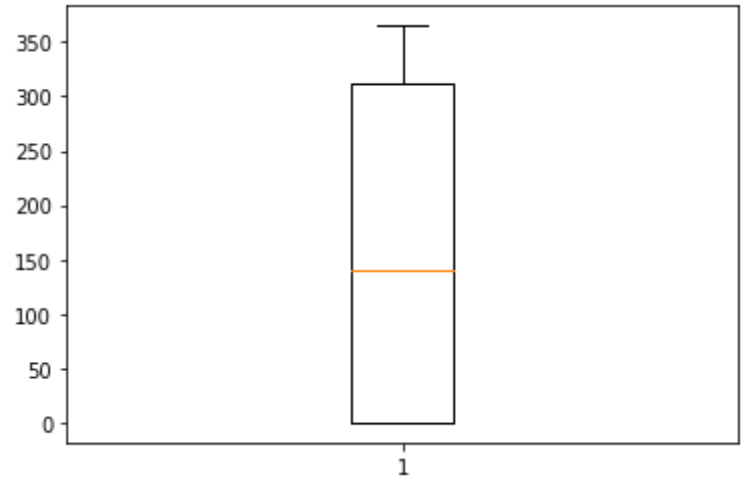
```
bnb['calculated_host_listings_count'] = np.where(bnb['calculated_host_listings_count'
                                                 bnb['calculated_host_listings_
bnb.describe()
```

	latitude	longitude	price	minimum_nights	number_of_reviews
count	225645.000000	225645.000000	225645.00000	225645.000000	225645.000000
mean	35.134465	-104.528139	146.87959	7.099763	22.423852
std	6.333456	25.258623	104.46235	10.262867	33.883824
min	18.920990	-159.714900	11.00000	1.000000	0.000000
25%	32.761780	-118.598890	75.00000	1.000000	1.000000
50%	37.000000	-97.814090	121.00000	2.000000	8.000000
75%	40.690100	-80.119560	184.00000	5.000000	29.000000
max	44.999970	-73.880010	597.00000	30.000000	161.000000

```
# checking the skewness
print(bnb['availability_365'].skew())
bnb['availability_365'].describe()        # skewness for price is 0.24268434096714664
# Box plot
plt.boxplot(bnb['availability_365'])
```

```
plt.show()
# Identifying outliers for price
print(bnb['availability_365'].quantile(0.10))
print(bnb['availability_365'].quantile(0.90))
```

```
0.23388915433100738
```




```
0.0
361.0
```

```
print(bnb['availability_365'].quantile(0.50))
print(bnb['availability_365'].quantile(0.95))
```

```
140.0
365.0
```

```
bnb['availability_365'] = np.where(bnb['availability_365'] > 365, 140,
                                                    bnb['availability_365'])
bnb.describe()
```

	latitude	longitude	price	minimum_nights	number_of_reviews
count	225645.000000	225645.000000	225645.00000	225645.000000	225645.000000
mean	35.134465	-104.528139	146.87959	7.099763	22.423852
std	6.333456	25.258623	104.46235	10.262867	33.883824
min	18.920990	-159.714900	11.00000	1.000000	0.000000
25%	32.761780	-118.598890	75.00000	1.000000	1.000000
50%	37.000000	-97.814090	121.00000	2.000000	8.000000
75%	40.690100	-80.119560	184.00000	5.000000	29.000000
max	44.999970	-73.880010	597.00000	30.000000	161.000000

```
bnb_db = "bnb.db"
conn = sqlite3.connect(bnb_db)
cursor = conn.cursor()
```

```
cursor.execute("DROP TABLE IF EXISTS BNB")

try:
    cursor.execute('''
      CREATE TABLE BNB

      (name   TEXT  DEFAULT NULL,

      neighbourhood   TEXT   DEFAULT NULL,

      latitude     FLOAT   DEFAULT 0,

      longitude     FLOAT   DEFAULT 0,

      room_type TEXT DEFAULT NULL,

      price  INTEGER DEFAULT 0,

      minimum_nights  INTEGER DEFAULT 0,

      number_of_reviews  INTEGER DEFAULT 0,

      reviews_per_month  FLOAT DEFAULT 0,

      calculated_host_listings_count  INTEGER DEFAULT 0,

      availability_365  INTEGER DEFAULT 0,

      city   TEXT   DEFAULT NULL

       );''')

    print("BNB Table created successfully")
except Exception as e:
    print(str(e))
    print('BNB Table creation failed!!!')
finally:
    conn.close()
```

```
BNB Table created successfully
```

```
conn = sqlite3.connect(bnb_db)
cursor = conn.cursor()
try:
  bnb.to_sql("BNB",conn,if_exists="append",index=False)
  print("Insertion Successful")
except Exception as e:
  print(e)
  print("Insertion Failed!!!")
finally:
  conn.close()
```

```
Insertion Successful
```

```
conn = sqlite3.connect(bnb_db)
cursor = conn.cursor()
cursor.execute("SELECT COUNT(*) FROM BNB;")
outs = cursor.fetchall()
for out in outs:
    print(out)
```

```
(225645,)
```

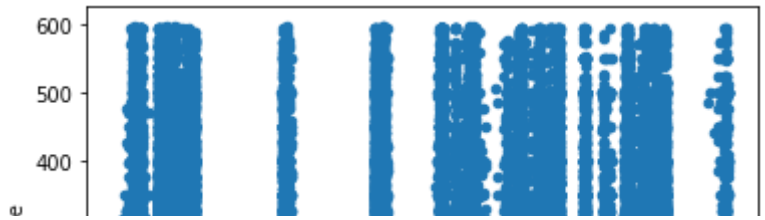
```
#Is there any relationship between latitude and price?
conn = sqlite3.connect(bnb_db)
df = pd.read_sql_query('''SELECT latitude, price FROM BNB;''', conn)
df
```

	latitude	price
0	35.651460	60
1	35.597790	470
2	35.606700	75
3	35.578640	90
4	35.614420	125
...	...	...
225640	38.903880	104
225641	38.920820	151
225642	38.911170	240
225643	38.926630	60
225644	38.911569	79

225645 rows × 2 columns

```
df.plot.scatter('latitude','price')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5c16166860>
```




```
#Is there any relationship between number of reviews and price?
conn = sqlite3.connect(bnb_db)
df = pd.read_sql_query('''SELECT number_of_reviews, price FROM BNB;''', conn)
df
```

	number_of_reviews	price
0	138	60
1	114	470
2	89	75
3	8	90
4	58	125
...	...	...
225640	0	104
225641	0	151
225642	0	240
225643	0	60
225644	0	79

225645 rows × 2 columns

```
df.plot.scatter('number_of_reviews','price')
```

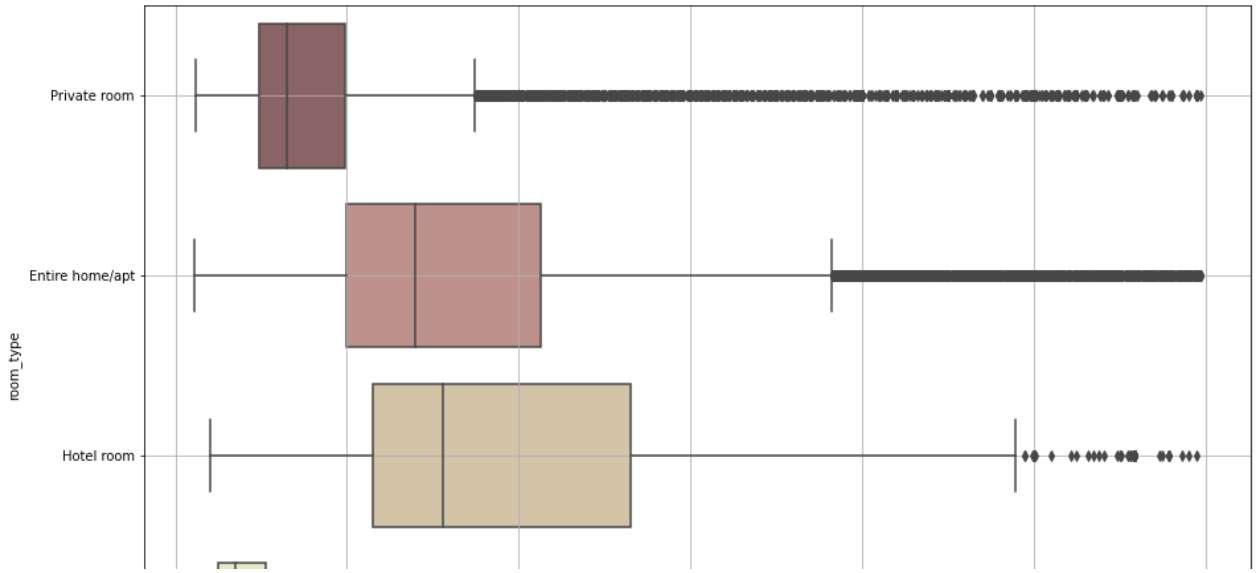
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5c17534860>
```

```
#How do prices per night differ by room type?'
conn = sqlite3.connect(bnb_db)
df = pd.read_sql_query('''SELECT room_type, price FROM BNB;''', conn)
df
```

	room_type	price
0	Private room	60
1	Entire home/apt	470
2	Entire home/apt	75
3	Entire home/apt	90
4	Private room	125
...	...	...
225640	Entire home/apt	104
225641	Entire home/apt	151
225642	Entire home/apt	240
225643	Entire home/apt	60
225644	Entire home/apt	79

225645 rows × 2 columns

```
plt.figure(figsize=(15,10))
sns.boxplot(y = "room_type", x = "price", data = df, palette = 'pink')
plt.grid()
plt.show()
```




```
conn = sqlite3.connect(bnb_db)
df = pd.read_sql_query('''SELECT * FROM BNB;''', conn)
df
```

	name	neighbourhood	latitude	longitude	room_type	price
0	Charming Victorian home - twin beds + breakfast	28804	35.651460	-82.627920	Private room	60
1	French Chic Loft	28801	35.597790	-82.555400	Entire home/apt	470
	Walk to				Entire	

```
#What is the maximum price per night?
conn = sqlite3.connect(bnb_db)
maxim = pd.read_sql_query('''SELECT MAX(price) FROM BNB;''', conn)
maxim
```

	MAX(price)
0	597

Downtown,

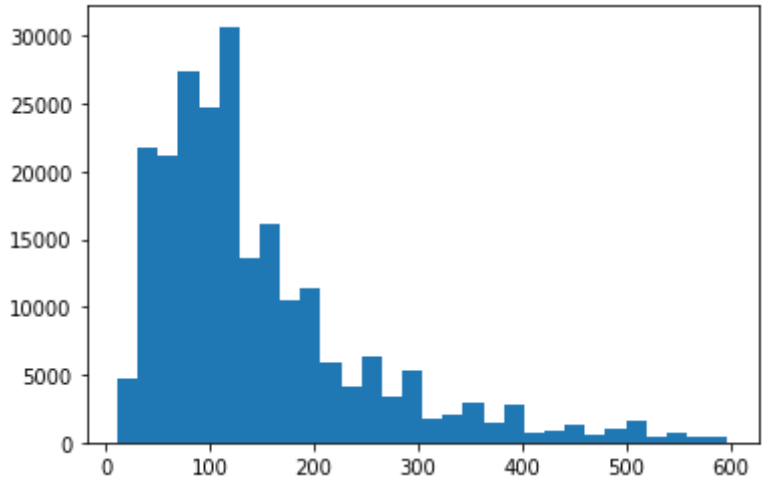
```
#What is the minimum price per night?
conn = sqlite3.connect(bnb_db)
minim = pd.read_sql_query('''SELECT MIN(price) FROM BNB;''', conn)
minim
```

	MIN(price)
0	11

```
df.describe()
```

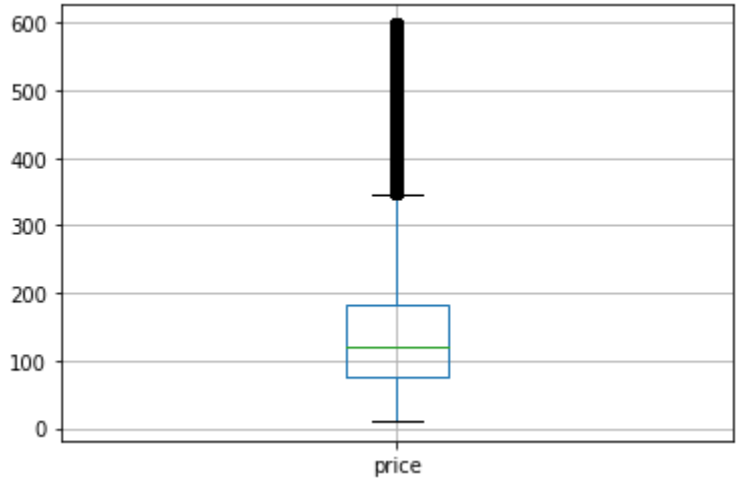
	latitude	longitude	price	minimum_nights	number_of_reviews
count	225645.000000	225645.000000	225645.00000	225645.000000	225645.000000
mean	35.134465	-104.528139	146.87959	7.099763	22.423852
std	6.333456	25.258623	104.46235	10.262867	33.883824
min	18.920990	-159.714900	11.00000	1.000000	0.000000
25%	32.761780	-118.598890	75.00000	1.000000	1.000000
50%	37.000000	-97.814090	121.00000	2.000000	8.000000
75%	40.690100	-80.119560	184.00000	5.000000	29.000000
max	44.999970	-73.880010	597.00000	30.000000	161.000000

```
plt.hist(df["price"], bins=30)
plt.show()
```




```
df.boxplot(column=["price"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5c162c5c18>
```




```
#Which regions offer listings with a minimum number of nights between 14 and 21 days?
conn = sqlite3.connect(bnb_db)
df = pd.read_sql_query('''SELECT DISTINCT(city) FROM BNB
WHERE minimum_nights BETWEEN 14 AND 21;''', conn)
df
```

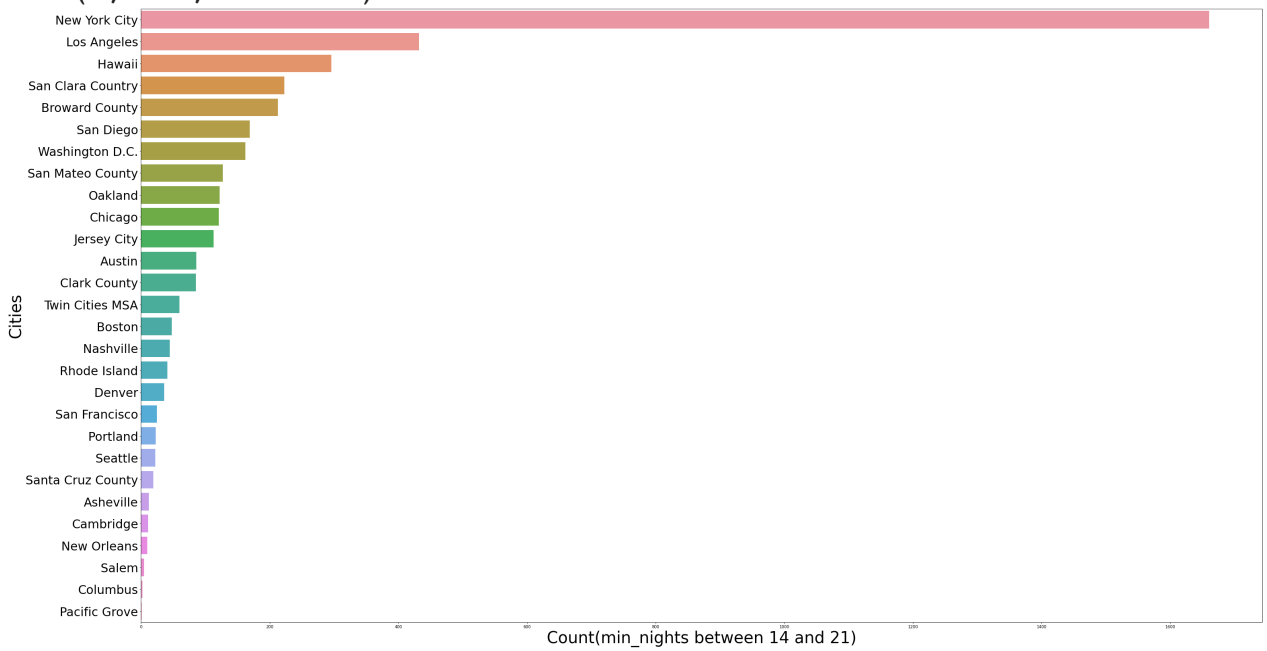
	city
0	Asheville
1	Austin
2	Boston
3	Broward County
4	Cambridge
5	Chicago
6	Clark County
7	Columbus
8	Denver
9	Hawaii
10	Jersey City
11	Los Angeles
12	Nashville
13	New Orleans
14	New York City
15	Oakland
16	Pacific Grove
17	Portland
18	Rhode Island
19	Salem

```
#How many total listings are there for these cities with minimum number of nights betv
conn = sqlite3.connect(bnb_db)
df = pd.read_sql_query('''SELECT DISTINCT(city), COUNT(minimum_nights) as count FROM I
                       WHERE minimum_nights BETWEEN 14 AND 21
                       GROUP BY city
                       ORDER BY count DESC ;''', conn)
df
```

	city	count
0	New York City	1660
1	Los Angeles	432
2	Hawaii	296
3	San Clara Country	223
4	Broward County	213
5	San Diego	169
6	Washington D.C.	162
7	San Mateo County	127
8	Oakland	122
9	Chicago	121
10	Jersey City	113
11	Austin	86
12	Clark County	85
13	Twin Cities MSA	60
14	Boston	48
15	Nashville	45
16	Rhode Island	41
17	Denver	36
18	San Francisco	25
19	Portland	23
20	Seattle	22
21	Santa Cruz County	19
22	Asheville	12

```
plt.figure(1, figsize=(50, 28))
ax = sns.barplot(x="count", y="city", data=df)
ax.set_yticklabels(ax.get_yticklabels(),fontsize=30)
plt.rc('xtick',labelsize=30)
ax.set_xlabel('Count(min_nights between 14 and 21)',fontsize=40)
ax.set_ylabel('Cities', fontsize=40)
```

```
Text(0, 0.5, 'Cities')
```



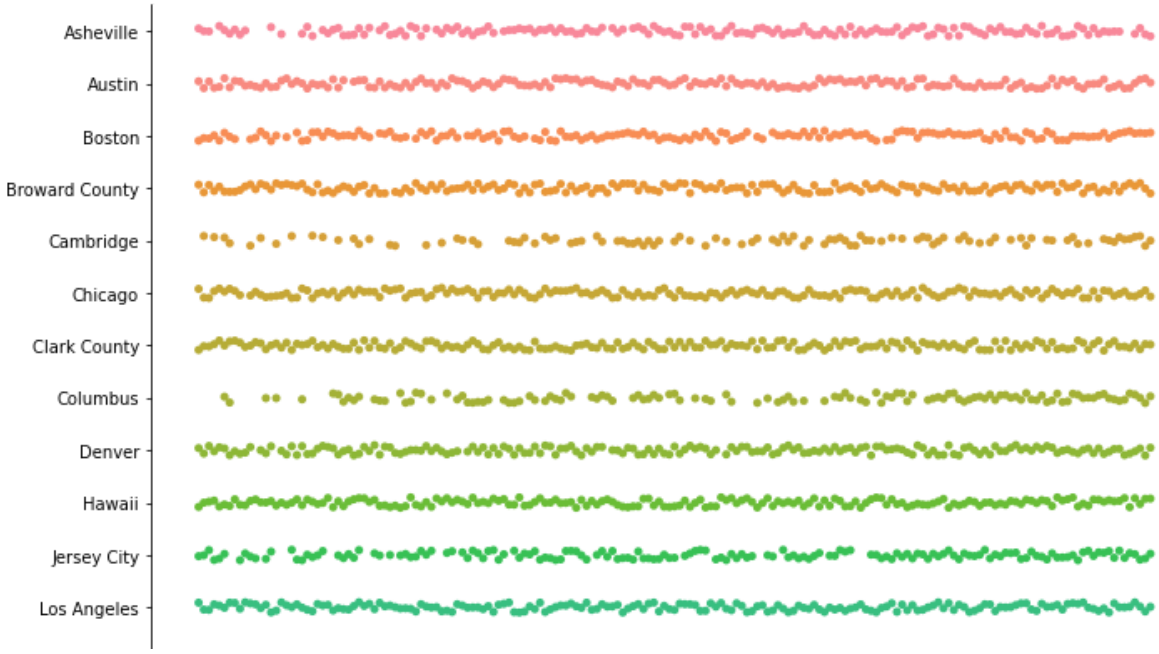

```
#How are the listings whose number of days in a year greater than 180 days that is ava
conn = sqlite3.connect(bnb_db)
df = pd.read_sql_query('''SELECT DISTINCT city, availability_365   FROM BNB
                          WHERE availability_365 > 180;''', conn)
df
```

	city	availability_365
0	Asheville	288
1	Asheville	298
2	Asheville	294
3	Asheville	207
4	Asheville	339

```
#What are the distinct cities whose number of days in a year greater than 180 days tha
conn = sqlite3.connect(bnb_db)
df2 = pd.read_sql_query('''SELECT DISTINCT city FROM BNB
                        WHERE availability_365 > 180;''', conn)
df2
```

	city
0	Asheville
1	Austin
2	Boston
3	Broward County
4	Cambridge
5	Chicago
6	Clark County

```
g = sns.catplot(x = 'availability_365', y = 'city', data = df,
                  height = 13,
                  aspect = 0.8)
```

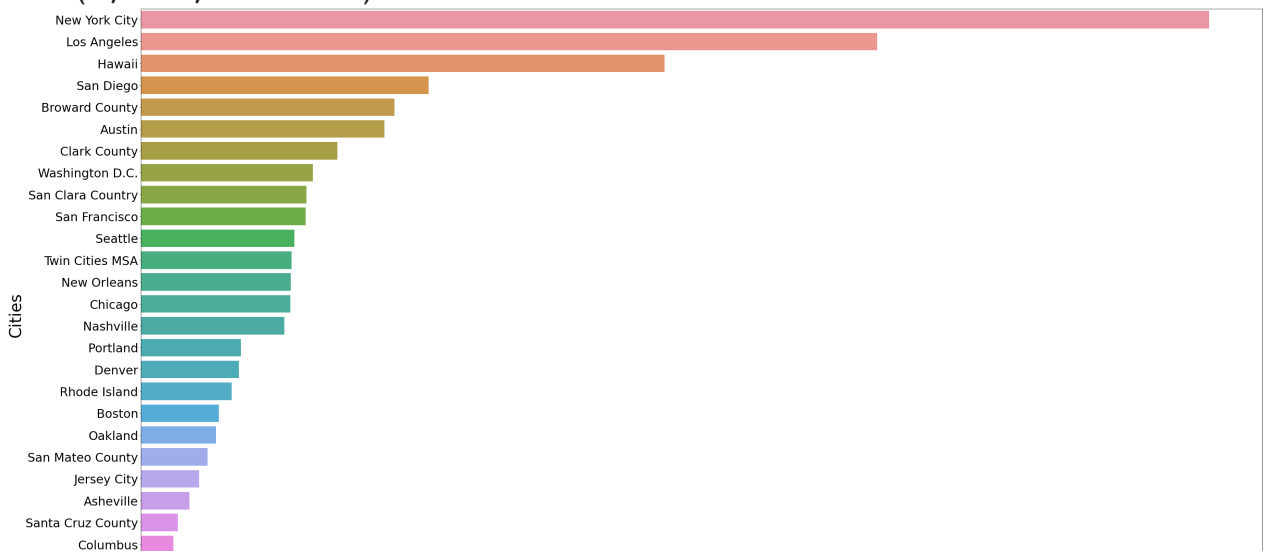



```
#How are the number of listings in different cities distributed?
conn = sqlite3.connect(bnb_db)
df = pd.read_sql_query('''SELECT DISTINCT(city), COUNT(city) FROM BNB
                       GROUP BY city
                       ORDER BY COUNT(city) DESC;''', conn)
df
```

	city	COUNT(city)
0	New York City	45701
1	Los Angeles	31495
2	Hawaii	22410
3	San Diego	12299
4	Broward County	10847
5	Austin	10419
6	Clark County	8408
7	Washington D.C.	7346
8	San Clara Country	7075
9	San Francisco	7048
10	Seattle	6564
11	Twin Cities MSA	6448

```
plt.figure(1, figsize=(50, 28))
ax = sns.barplot(x="COUNT(city)", y="city", data=df)
ax.set_yticklabels(ax.get_yticklabels(),fontsize=30)
plt.rc('xtick',labelsize=30)
ax.set_xlabel('Count',fontsize=40)
ax.set_ylabel('Cities', fontsize=40)
```

```
Text(0, 0.5, 'Cities')
```




```
!apt update
!apt-get clean
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://www-us.apache.org/dist/spark/spark-2.4.7/spark-2.4.7-bin-hadoop2.7.tg
!tar xf spark-2.4.7-bin-hadoop2.7.tgz
!pip install -q findspark
```

```
Get:1 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Ign:2 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64
Get:3 https://cloud.r-project.org/bin/linux/ubuntu bionic-cran40/ InRelease [3,6
Get:4 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic InRelease [15.
Ign:5 https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu
Hit:6 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64
Hit:7 http://archive.ubuntu.com/ubuntu bionic InRelease
Hit:8 https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu
Get:9 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:10 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Packages
Get:11 http://ppa.launchpad.net/graphics-drivers/ppa/ubuntu bionic InRelease [21
Get:12 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:15 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic/main Sources
Get:16 http://archive.ubuntu.com/ubuntu bionic-updates/universe amd64 Packages [
Get:17 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 Packages [2,24
Get:18 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic/main amd64 Pa
Get:19 http://ppa.launchpad.net/graphics-drivers/ppa/ubuntu bionic/main amd64 Pa
Fetched 8,650 kB in 3s (2,654 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
15 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-2.4.7-bin-hadoop2.7"
```

```
import findspark
findspark.init()
from pyspark.sql import SparkSession
```

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
spark
```

SparkSession - in-memory

SparkContext

Spark UI

Version
    v2.4.7
Master
    local[*]
AppName
    pyspark-shell

```
#Getting data ready for model
categorical_cols = ['room_type']
bnb2 = pd.get_dummies(bnb, columns = categorical_cols)
bnb2.head()
```

	name	neighbourhood	latitude	longitude	price	minimum_nights
0	Charming Victorian home - twin beds + breakfast	28804	35.65146	-82.62792	60	1
1	French Chic Loft	28801	35.59779	-82.55540	470	1
2	Walk to stores/parks/downtown. Fenced yard/Pet...	28801	35.60670	-82.55563	75	30
3	Cottage! BonPaul + Sharky's Hostel	28806	35.57864	-82.59578	90	1
4	Historic Grove Park	28801	35.61442	-82.54127	125	30

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
bnb2['city'] = label_encoder.fit_transform(bnb2['city'])
bnb2.head()
```

	name	neighbourhood	latitude	longitude	price	minimum_nights
0	Charming Victorian home - twin beds + breakfast	28804	35.65146	-82.62792	60	1
1	French Chic Loft	28801	35.59779	-82.55540	470	1
	Walk to					

```
to_drop = ['name', 'neighbourhood']
bnb2.drop(to_drop, inplace = True, axis = 1)
bnb2.head()
```

	latitude	longitude	price	minimum_nights	number_of_reviews	reviews_per_mon
0	35.65146	-82.62792	60	1	138	1.
1	35.59779	-82.55540	470	1	114	1.
2	35.60670	-82.55563	75	30	89	0.
3	35.57864	-82.59578	90	1	8	2.
4	35.61442	-82.54127	125	30	58	0.

```
bnb2.shape
```

```
(225645, 13)
```

```
airbnb = spark.createDataFrame(bnb2)
```

```
airbnb.printSchema()
```

```
root
 |-- latitude: double (nullable = true)
 |-- longitude: double (nullable = true)
 |-- price: long (nullable = true)
 |-- minimum_nights: long (nullable = true)
 |-- number_of_reviews: long (nullable = true)
 |-- reviews_per_month: double (nullable = true)
 |-- calculated_host_listings_count: long (nullable = true)
 |-- availability_365: long (nullable = true)
 |-- city: long (nullable = true)
 |-- room_type_Entire home/apt: long (nullable = true)
 |-- room_type_Hotel room: long (nullable = true)
 |-- room_type_Private room: long (nullable = true)
 |-- room_type_Shared room: long (nullable = true)
```

```
newdf = airbnb.drop("name","neighbourhood")
newdf.show()
```

```
+------------------+------------------+-----+--------------+------------------+--
|          latitude|         longitude|price|minimum_nights|number_of_reviews|re
+------------------+------------------+-----+--------------+------------------+--
|          35.65146|-82.62791999999999|   60|             1|               138|
|          35.59779|           -82.5554|  470|             1|               114|
|           35.6067|-82.55563000000001|   75|            30|                89|
|          35.57864|          -82.59578|   90|             1|                 8|
|          35.61442|          -82.54127|  125|            30|                58|
|35.618559999999995|-82.55275999999999|  134|             7|                54|
|          35.58345|          -82.59713|   48|             1|               137|
|          35.59635|          -82.50655|   65|             3|                57|
|          35.61929|          -82.48114|   71|            28|                 8|
|          35.55537|          -82.53539|   50|             2|                31|
|35.644529999999996|          -82.52586|  289|            30|                24|
|          35.58217|          -82.59997|   78|             4|                 8|
|          35.49111|          -82.48438|  125|             2|                40|
|35.601820000000004|          -82.56174|  126|             3|                 8|
|          35.56118|-82.57784000000001|  118|             2|                 8|
|          35.60371|          -82.55621|   85|             2|                 8|
|          35.61115|-82.54375999999999|   50|            30|               130|
|          35.60075|           -82.5539|   97|            30|                 8|
|          35.57318|          -82.59976|   74|             2|                 8|
|          35.60418|          -82.54964|  160|            30|                69|
+------------------+------------------+-----+--------------+------------------+--
only showing top 20 rows
```

```
#counting rows
newdf.count()
```

```
225645
```

```
#counting columns
len(newdf.columns)
```

```
13
```

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
```

```
train_data,test_data =newdf.randomSplit([0.8,0.2],seed=495)
```

```
def create_and_score_linear_model(train_data,test_data,featurelist,target,metric="mse'
  #setup for PySpark workflow
  assembler = VectorAssembler(inputCols=featurelist, outputCol = 'Attributes')
  output = assembler.transform(train_data)
  train = output.select("Attributes",target)
  output = assembler.transform(test_data)
```

```
    output = assembler.transform(test_data)
    test = output.select("Attributes",target)
    #initialize model
    regressor = LinearRegression(featuresCol = 'Attributes', labelCol = target)
    #fit the model
    regressor = regressor.fit(train)
    #make predictions
    pred = regressor.evaluate(test)
    #initialize evaluator
    eval = RegressionEvaluator(labelCol=target, predictionCol="prediction", metricName=m
    #score mode
    score = eval.evaluate(pred.predictions)
    return score


features=['latitude', 'longitude', 'minimum_nights', 'number_of_reviews', 'reviews_per
          'calculated_host_listings_count', 'availability_365','city','room_type_Entire
                                          'room_type_Hotel room', 'room_type_Private roon
create_and_score_linear_model(train_data=train_data,test_data=test_data, featurelist=f
```

```
    8793.544491971337
```

```
#This exists only to time the code
%%time

#The Backwards selection starts here:
features=['latitude', 'longitude', 'minimum_nights', 'number_of_reviews', 'reviews_per
          'calculated_host_listings_count', 'availability_365', 'city','room_type_Enti
          'room_type_Hotel room', 'room_type_Private room', 'room_type_Shared room']
new_best = features
new_score = create_and_score_linear_model(train_data=train_data,test_data=test_data,fe
current_best = []
#we will also keep track of how many models we've evaluated
count=1
while current_best!=new_best:
  current_best = new_best
  current_score = new_score
  for var in current_best:
    contender = [i for i in current_best if i != var]
    contender_score = create_and_score_linear_model(train_data=train_data,test_data=te
    if contender_score <= current_score:
      new_best = contender
      new_score = contender_score
    count+=1
print(current_best)
print(current_score)
print("Number of models considered: "+str(count))
```

```
    ['latitude', 'longitude', 'minimum_nights', 'number_of_reviews', 'reviews_per_mon
    8793.544491971337
    Number of models considered: 13
```

```
CPU times: user 536 ms, sys: 86.1 ms, total: 622 ms
Wall time: 1min 40s
```

```
print(newdf.columns)
```

```
['latitude', 'longitude', 'price', 'minimum_nights', 'number_of_reviews', 'revie
```

```
#Input all the features in one vector column
assembler = VectorAssembler(inputCols=['latitude', 'longitude',
                                       'minimum_nights', 'number_of_reviews', 'reviews
                                       'calculated_host_listings_count', 'availability
                                       'room_type_Hotel room', 'room_type_Private roor
```

```
output = assembler.transform(newdf)
```

```
finalized_data = output.select("Attributes","price")
```

```
finalized_data.show()
```

```
+--------------------+-----+
|          Attributes|price|
+--------------------+-----+
|[35.65146,-82.627...|   60|
|[35.59779,-82.555...|  470|
|[35.6067,-82.5556...|   75|
|[35.57864,-82.595...|   90|
|[35.61442,-82.541...|  125|
|[35.6185599999999...|  134|
|[35.58345,-82.597...|   48|
|[35.59635,-82.506...|   65|
|[35.61929,-82.481...|   71|
|[35.55537,-82.535...|   50|
|[35.6445299999999...|  289|
|[35.58217,-82.599...|   78|
|[35.49111,-82.484...|  125|
|[35.6018200000000...|  126|
|[35.56118,-82.577...|  118|
|[35.60371,-82.556...|   85|
|[35.61115,-82.543...|   50|
|[35.60075,-82.553...|   97|
|[35.57318,-82.599...|   74|
|[35.60418,-82.549...|  160|
+--------------------+-----+
only showing top 20 rows
```

```
#Split training and testing data
train_data,test_data = finalized_data.randomSplit([0.8,0.2],seed=495)
```

```
regressor = LinearRegression(featuresCol = 'Attributes', labelCol = 'price')
```

```
#Learn to fit the model from training set
```

```
regressor = regressor.fit(train_data)

#To predict the prices on testing set
pred = regressor.evaluate(test_data)

#Predict the model
pred.predictions.show()
```

```
+--------------------+-----+------------------+
|          Attributes|price|        prediction|
+--------------------+-----+------------------+
|(12,[0,1,2,4,5,8]...|   99| 163.2855487613581|
|(12,[0,1,2,4,5,8]...|  199|171.28023407473327|
|(12,[0,1,2,4,5,10...|  310| 78.58337582836268|
|[18.92099,-155.68...|   50|  94.1606881548987|
|[18.9295,-155.679...|   40| 94.91765639266205|
|[18.9836100000000...|   75|118.23531521825899|
|[19.03121,-155.63...|  205|219.26797188654768|
|[19.0406899999999...|  135|211.76363083686633|
|[19.0442,-155.660...|   50|  130.075640867225|
|[19.04692,-155.63...|  167| 201.3710693505442|
|[19.04845,-155.65...|   89| 200.8293780739411|
|[19.04956,-155.76...|   90| 92.78564060687071|
|[19.05732,-155.60...|   82|180.71137660150225|
|[19.0589899999999...|  155|240.60488144541122|
|[19.05954,-155.61...|  199|237.99826558998427|
|[19.0607600000000...|   97| 101.1611205403673|
|[19.06154,-155.76...|  137|169.94230293208943|
|[19.06716,-155.61...|   95|178.01855925521227|
|[19.07684,-155.79...|   95|199.09471407527616|
|[19.0835799999999...|   45| 199.1728660047325|
+--------------------+-----+------------------+
only showing top 20 rows
```

```
#coefficient of the regression model
coeff = regressor.coefficients

#X and Y intercept
intr = regressor.intercept

print ("The coefficient of the model is : %a" %coeff)
print ("The Intercept of the model is : %f" %intr)
```

```
The coefficient of the model is : DenseVector([-0.5297, -0.2783, -1.036, -0.2101
The Intercept of the model is : 127.580396
```

```
from pyspark.ml.evaluation import RegressionEvaluator
eval = RegressionEvaluator(labelCol="price", predictionCol="prediction", metricName="r

# Root Mean Square Error
rmse = eval.evaluate(pred.predictions)
print("RMSE: %.3f" % rmse)
```

```
# Mean Square Error
mse = eval.evaluate(pred.predictions, {eval.metricName: "mse"})
print("MSE: %.3f" % mse)

# Mean Absolute Error
mae = eval.evaluate(pred.predictions, {eval.metricName: "mae"})
print("MAE: %.3f" % mae)

# r2 - coefficient of determination
r2 = eval.evaluate(pred.predictions, {eval.metricName: "r2"})
print("r2: %.3f" %r2)
```

```
RMSE: 93.734
MSE: 8786.093
MAE: 66.739
r2: 0.194
```