3-Sigma Technique (Standard Deviation)

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import statistics
import pandas as pd
```

In [ ]:
```
data = pd.read_csv("./data/raw_sales.csv")
```

In [ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29580 entries, 0 to 29579
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   datesold      29580 non-null  object
 1   postcode      29580 non-null  int64 
 2   price         29580 non-null  int64 
 3   propertyType  29580 non-null  object
 4   bedrooms      29580 non-null  int64 
dtypes: int64(3), object(2)
memory usage: 1.1+ MB
```

In [ ]:
```
data.head(5)
```

Out[ ]:

	datesold	postcode	price	propertyType	bedrooms
0	2007-02-07 00:00:00	2607	525000	house	4
1	2007-02-27 00:00:00	2906	290000	house	3
2	2007-03-07 00:00:00	2905	328000	house	3
3	2007-03-09 00:00:00	2905	380000	house	4
4	2007-03-21 00:00:00	2906	310000	house	3

In [ ]:
```
type(data)
```

Out[ ]: pandas.core.frame.DataFrame

In [ ]:
```
# Function to Detection Outlier on one-dimentional datasets.
def find_anomalies(data):
    #define a list to accumlate anomalies
    anomalies = []

    # Set upper and lower limit to 3 standard deviation
    random_data_std = statistics.stdev(data)
    random_data_mean = statistics.mean(data)
    # 3-standard deviation

    anomaly_cut_off = random_data_std * 3

    lower_limit  = random_data_mean - anomaly_cut_off
    upper_limit = random_data_mean + anomaly_cut_off

    # Generate outliers
    for outlier in data:
        if outlier > upper_limit or outlier < lower_limit:
            anomalies.append(outlier)
    return anomalies
```

In [ ]:
```
data.price
```

Out[ ]:
```
0        525000
1        290000
2        328000
3        380000
4        310000
          ...
29575    500000
29576    560000
29577    464950
29578    589000
29579    775000
Name: price, Length: 29580, dtype: int64
```

In [ ]:
```
list_1 = find_anomalies(data['price'])
```

In [ ]:
```
len(list_1)
```

Out[ ]:
```
461
```

In [ ]:
```
len(data)
```

Out[ ]:
```
29580
```

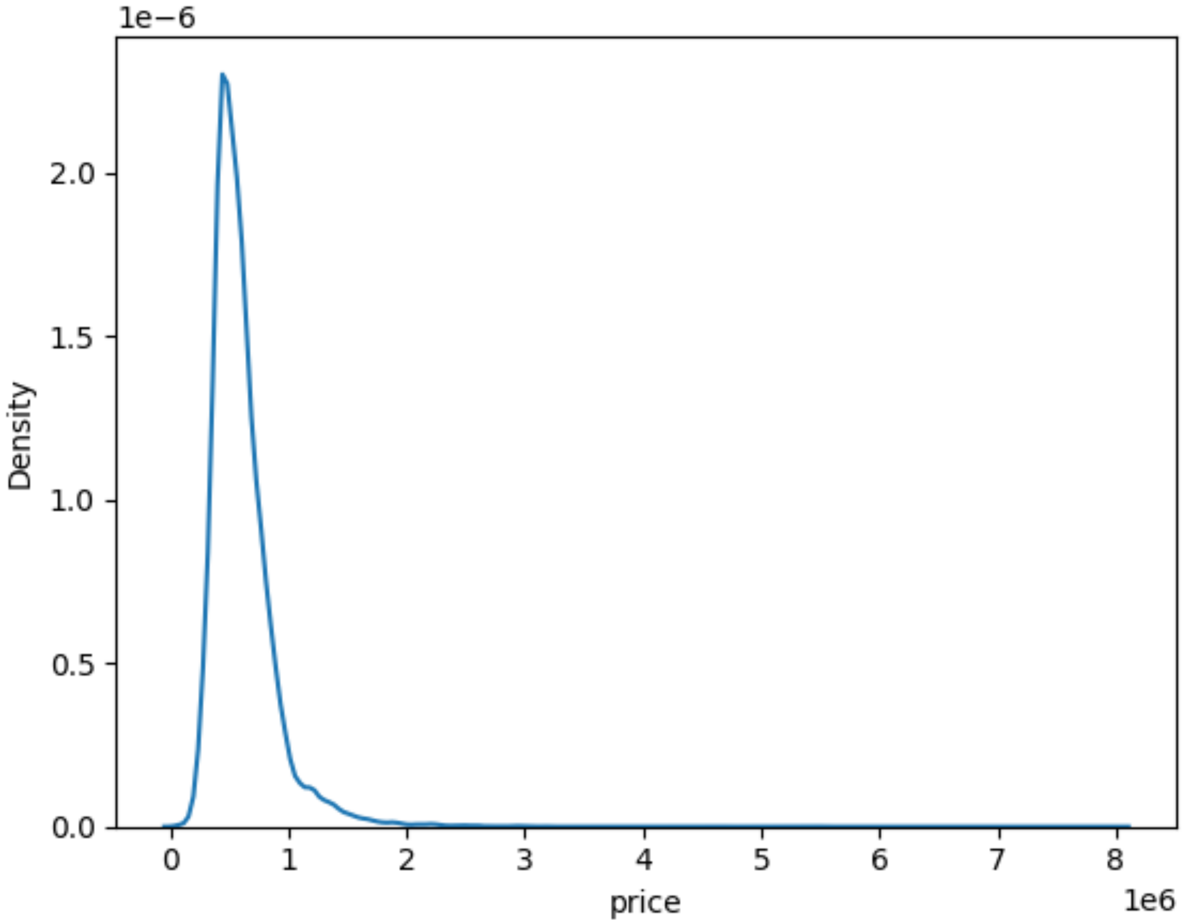
In [ ]:
```
data.price.skew()
```

Out[ ]:
```
4.312009366902366
```

In [ ]:
```
import seaborn as sns
```

In [ ]:
```
sns.kdeplot(data.price)
```

Out[ ]: <AxesSubplot:xlabel='price', ylabel='Density'>



In [ ]:
```
data['price_transformed'] = np.log(data.price)
```

In [ ]:
```
data.price_transformed.skew()
```

Out[ ]: 0.4731646269984763

In [ ]:
```
list_2 = find_anomalies(data.price_transformed)
```

In [ ]:
```
len(list_2)
```

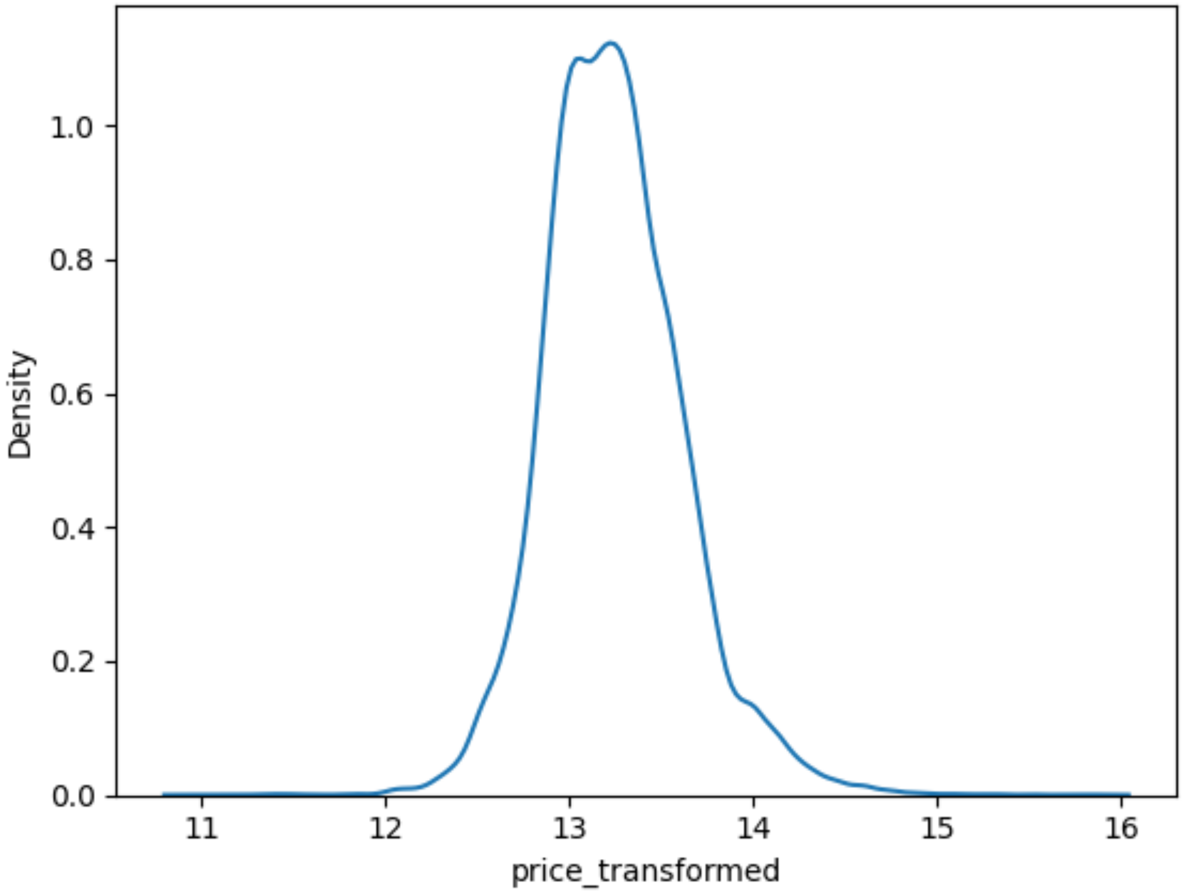
Out[ ]: 266

In [ ]:
```
len(data)
```

Out[ ]: 29580

In [ ]:
```
sns.kdeplot(data.price_transformed)
```

Out[ ]: <AxesSubplot:xlabel='price_transformed', ylabel='Density'>



In [ ]:
```
data['price_transformed_double'] = np.log(data.price_transformed)
```

In [ ]:
```
data['price_transformed_double'].skew()
```

Out[ ]: 0.33092530655758573

In [ ]:
```
list_3 = find_anomalies(data.price_transformed_double)
```

In [ ]:
```
len(list_3)
```

Out[ ]: 251

In [ ]:
```
len(data)
```

Out[ ]: 29580

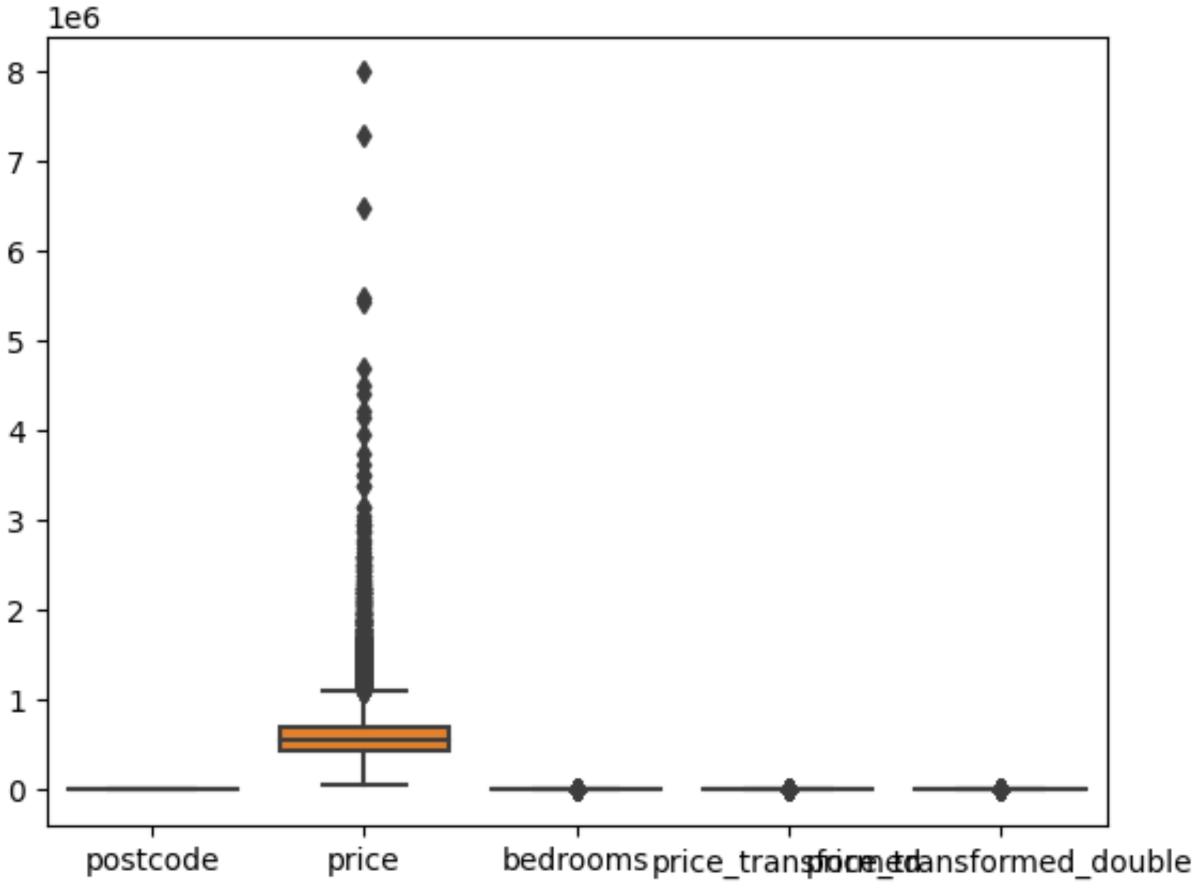
- Anything below 40, or above 80 are considered as outliers

Boxplots

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.boxplot(data=data)
```

Out[ ]: <AxesSubplot:>



The above code displays the plot below.

- As you can see, it considers everything above 75 or below ~ -35 to be an outlier. The results are very close to method 1 above.

In [ ]:
```
df = pd.DataFrame(data)
```

In [ ]:
```
len(df)
```

Out[ ]: 29580

In [ ]:
```
df
```

Out[ ]:

	datesold	postcode	price	propertyType	bedrooms	price_transformed
0	2007-02-07 00:00:00	2607	525000	house	4	13.171154
1	2007-02-27 00:00:00	2906	290000	house	3	12.577636
2	2007-03-07 00:00:00	2905	328000	house	3	12.700769
3	2007-03-09 00:00:00	2905	380000	house	4	12.847927
4	2007-03-21 00:00:00	2906	310000	house	3	12.644328
...	...	...	...	...	...	...
29575	2019-07-25 00:00:00	2900	500000	unit	3	13.122363
29576	2019-07-25 00:00:00	2612	560000	unit	2	13.235692
29577	2019-07-26 00:00:00	2912	464950	unit	2	13.049685
29578	2019-07-26 00:00:00	2601	589000	unit	2	13.286181
29579	2019-07-26 00:00:00	2612	775000	unit	2	13.560618

29580 rows × 7 columns

In [ ]:

```
# Statistical information of the dataframe columns

df.describe()
```

Out[ ]:

	postcode	price	bedrooms	price_transformed	price_trans
count	29580.000000	2.958000e+04	29580.000000	29580.000000	
mean	2730.249730	6.097363e+05	3.250169	13.244695	
std	146.717292	2.817079e+05	0.951275	0.375214	
min	2600.000000	5.650000e+04	0.000000	10.941996	
25%	2607.000000	4.400000e+05	3.000000	12.994530	
50%	2615.000000	5.500000e+05	3.000000	13.217674	
75%	2905.000000	7.050000e+05	4.000000	13.465953	
max	2914.000000	8.000000e+06	5.000000	15.894952	

Inter Quartile Range

IQR = Q3 - Q1

In [ ]:
```
list1 = [43, 54, 56, 61, 62, 66, 68, 69, 69, 70, 71, 72, 77, 78, 79, 85,
```

In [ ]:
```
len(list1)
```

Out[ ]: 25

In [ ]:
```
max(list1)
```

Out[ ]: 99

In [ ]:
```
min(list1)
```

Out[ ]: 43

In [ ]:
```
import statistics

statistics.mean(list1)
```

Out[ ]: 76.96

In [ ]:
```
sorted(list1)
```

Out[ ]:
```
[43,
 54,
 56,
 61,
 62,
 66,
 68,
 69,
 69,
 70,
 71,
 72,
 77,
 78,
 79,
 85,
 87,
 88,
 89,
 93,
 95,
 96,
 98,
 99,
 99]
```

- To find the 90th percentile for these (ordered) scores, start by multiplying 90 percent times the total number of scores, which gives 90% ∗ 25 = 0.90 ∗ 25 = 22.5 (the index). Rounding up to the nearest whole number, you get 23.

In [ ]:
```
list2 = sorted(list1)
```

In [ ]:
```
list2
```

Out[ ]:
```
[43,
 54,
 56,
 61,
 62,
 66,
 68,
 69,
 69,
 70,
 71,
 72,
 77,
 78,
 79,
 85,
 87,
 88,
 89,
 93,
 95,
 96,
 98,
 99,
 99]
```

Hence, 98 is the 90th percentile for this dataset

Now say you want to find the 20th percentile. Start by taking 0.20 x 25 = 5 (the index); this is a whole number, which tells you the 20th percentile is the average of the 5th and 6th values in the ordered data set (62 and 66).

`so, 20th percentile is 62+66/2 = 64`

The median (the 50th percentile) for the test scores is the 13th score: 77.