Problem Statement

The SkyBee Real Estate aims determine the accurate value of a property, basing it on past property transactions, to establish the price for the property they intend to sell.

- Note:Price of the property will be the dependent variable in my analysis and other factors which impact the price which I identified base on primary and secondary research are the independent variables and also different parts of data came from different sources and I collated all the data into a single tabular format

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
```

In [3]:
```
df=pd.read_csv('House_Price.csv', header=0)
df.head()
```

Out[3]:

	price	crime_rate	resid_area	air_qual	room_num	age	dist1	dist2	dist3	dist4	teachers	p
0	24.0	0.00632	32.31	0.538	6.575	65.2	4.35	3.81	4.18	4.01	24.7	
1	21.6	0.02731	37.07	0.469	6.421	78.9	4.99	4.70	5.12	5.06	22.2	
2	34.7	0.02729	37.07	0.469	7.185	61.1	5.03	4.86	5.01	4.97	22.2	
3	33.4	0.03237	32.18	0.458	6.998	45.8	6.21	5.93	6.16	5.96	21.3	
4	36.2	0.06905	32.18	0.458	7.147	54.2	6.16	5.86	6.37	5.86	21.3	

In [9]:
```
df.shape
```

Out[9]: (506, 19)

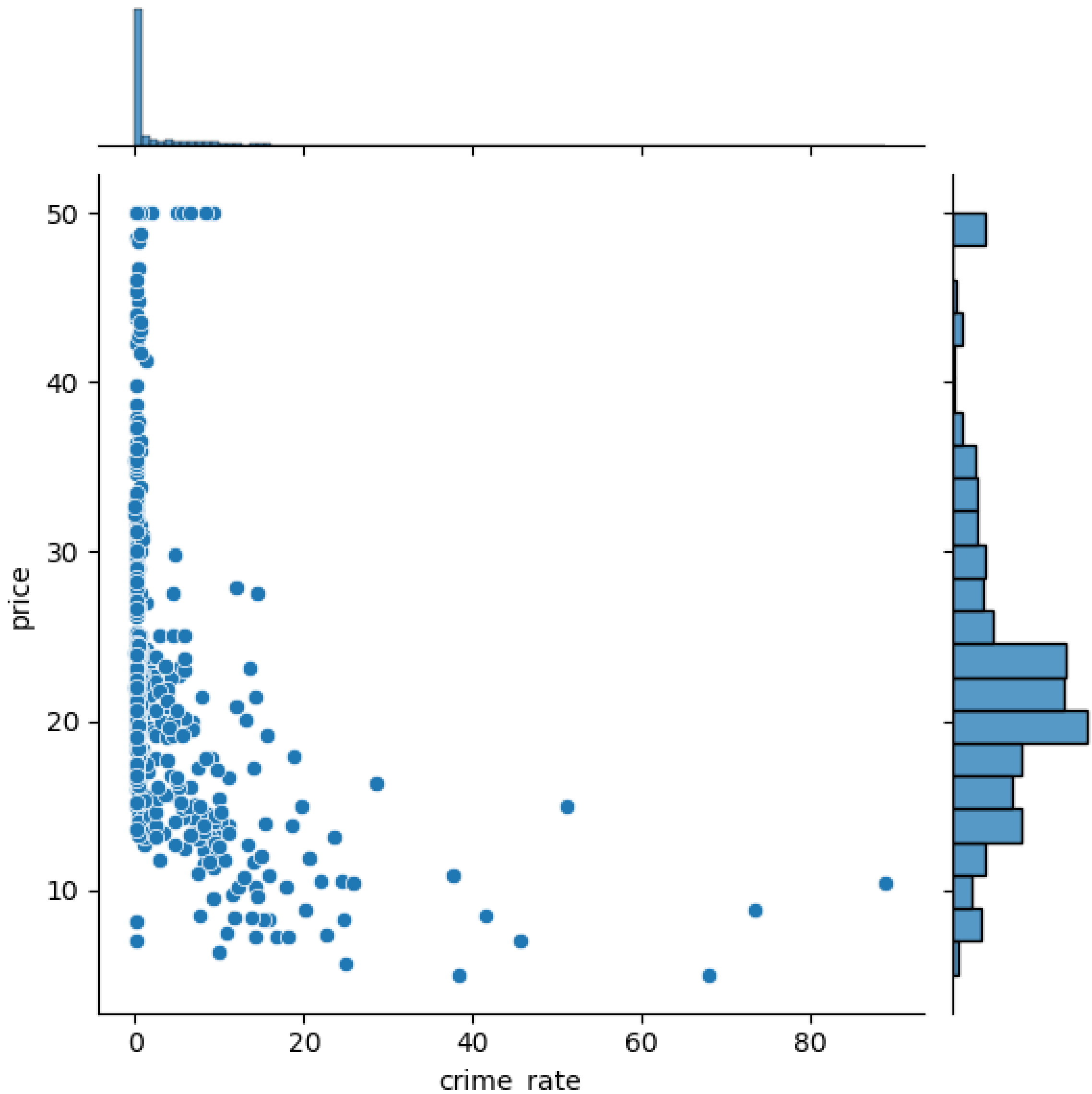
In [10]:
```
df.describe()
```

Out[10]:

	price	crime_rate	resid_area	air_qual	room_num	age	dist1	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.(
mean	22.528854	3.613524	41.136779	0.554695	6.284634	68.574901	3.971996	3.(
std	9.182176	8.601545	6.860353	0.115878	0.702617	28.148861	2.108532	2.1
min	5.000000	0.006320	30.460000	0.385000	3.561000	2.900000	1.130000	0.9
25%	17.025000	0.082045	35.190000	0.449000	5.885500	45.025000	2.270000	1.9
50%	21.200000	0.256510	39.690000	0.538000	6.208500	77.500000	3.385000	3.(
75%	25.000000	3.677083	48.100000	0.624000	6.623500	94.075000	5.367500	4.9
max	50.000000	88.976200	57.740000	0.871000	8.780000	100.000000	12.320000	11.9

In [11]:
```
sns.jointplot(x="crime_rate", y ="price", data = df)
```

Out[11]: <seaborn.axisgrid.JointGrid at 0x1a9f3736648>

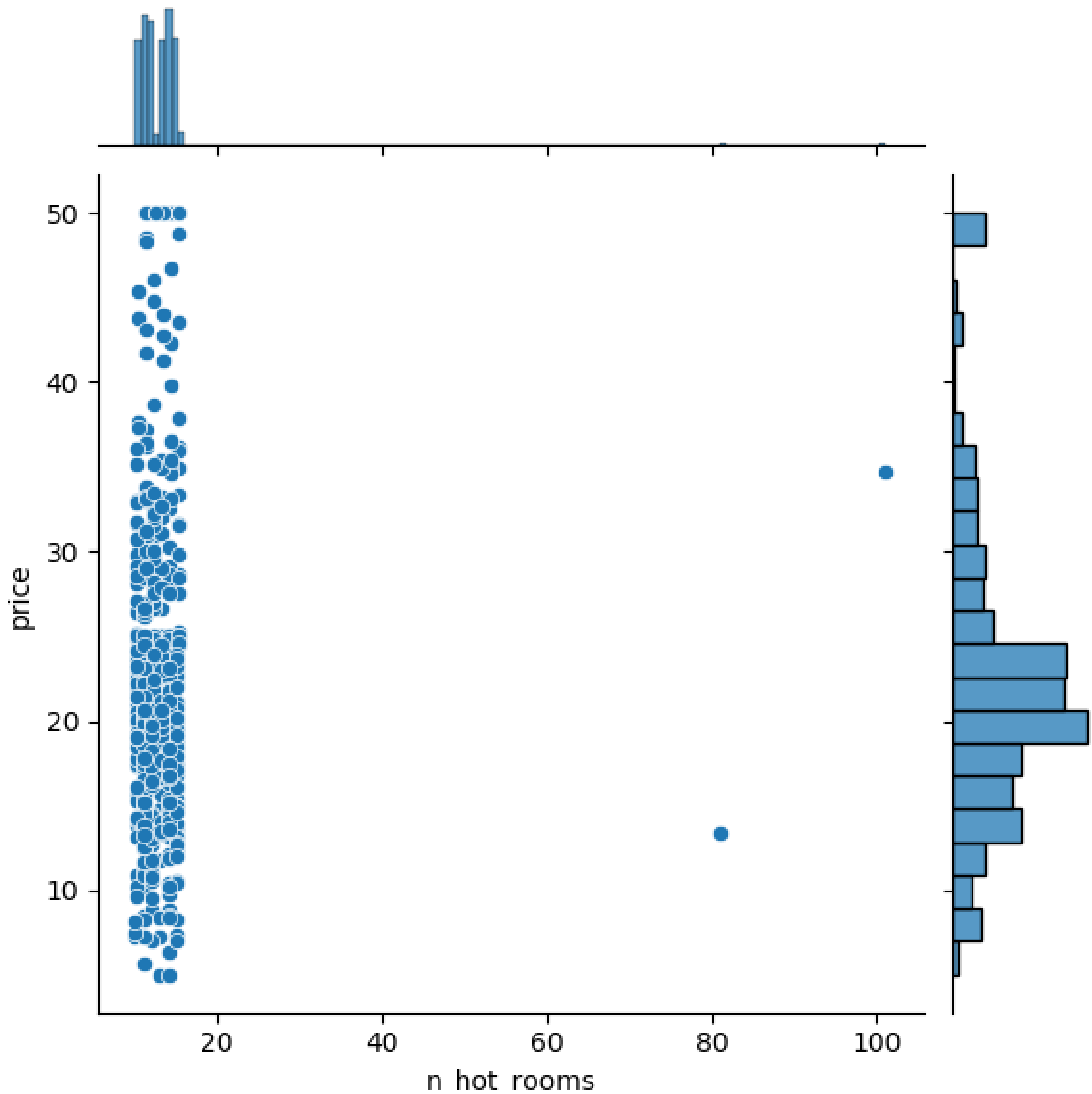


- n_hos_beds has 8 missing values
- Crime_rate- different, age, poor_prop,n_hot_rooms,rainfall, between min and max are huge need more investigate

- I could easily identified 2 outliers here, most of my data lies between 0-20

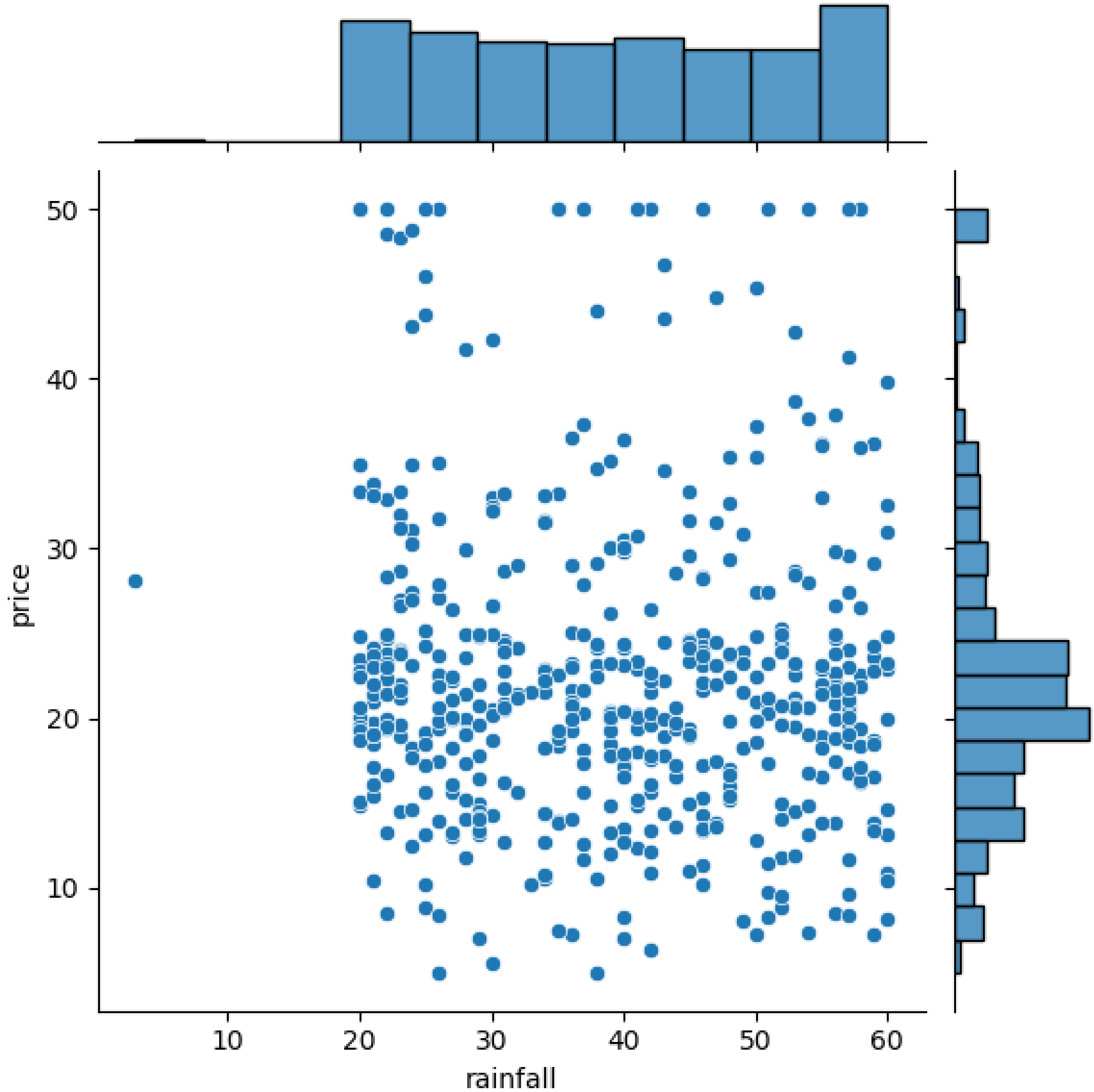
In [12]:
```
sns.jointplot(x='n_hot_rooms', y= 'price', data=df)
```

Out[12]: <seaborn.axisgrid.JointGrid at 0x1a9f3b12f88>



In [13]:
```
sns.jointplot(x='rainfall', y= 'price', data=df)
```

Out[13]: <seaborn.axisgrid.JointGrid at 0x1a9f4194c88>



- most of my rainfall values are lies between 20-60 and also I can see one outlier is a single point with the value is almost like 4 or 5

Note:

- since rainfall variable is uniformly distributed regardless of price. it was saying that the variable may not be having a significant impact on the dependent variable. so, I have an option to delete the rainfall variable

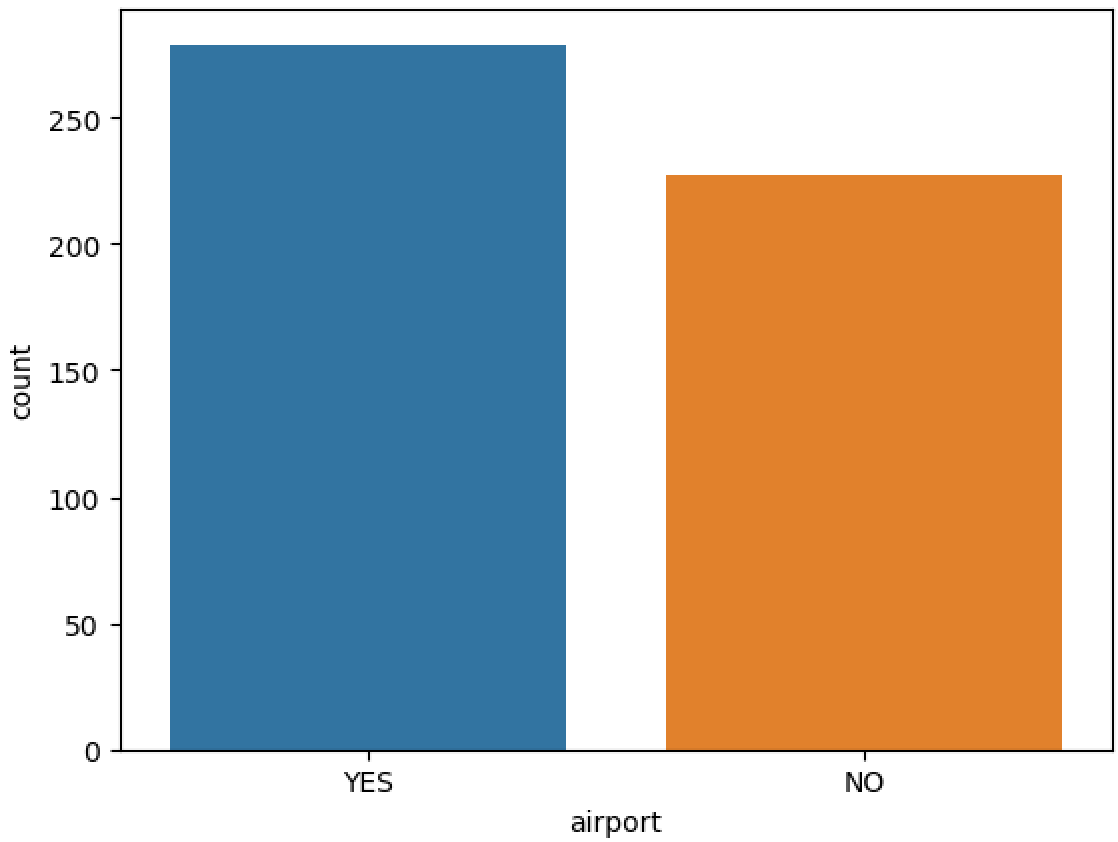
In [14]:
```
df.head()
```

Out[14]:

	price	crime_rate	resid_area	air_qual	room_num	age	dist1	dist2	dist3	dist4	teachers	p
0	24.0	0.00632	32.31	0.538	6.575	65.2	4.35	3.81	4.18	4.01	24.7	
1	21.6	0.02731	37.07	0.469	6.421	78.9	4.99	4.70	5.12	5.06	22.2	
2	34.7	0.02729	37.07	0.469	7.185	61.1	5.03	4.86	5.01	4.97	22.2	
3	33.4	0.03237	32.18	0.458	6.998	45.8	6.21	5.93	6.16	5.96	21.3	
4	36.2	0.06905	32.18	0.458	7.147	54.2	6.16	5.86	6.37	5.86	21.3	

In [15]:
```
# To identify my categorical variables
sns.countplot(x='airport', data=df)
```

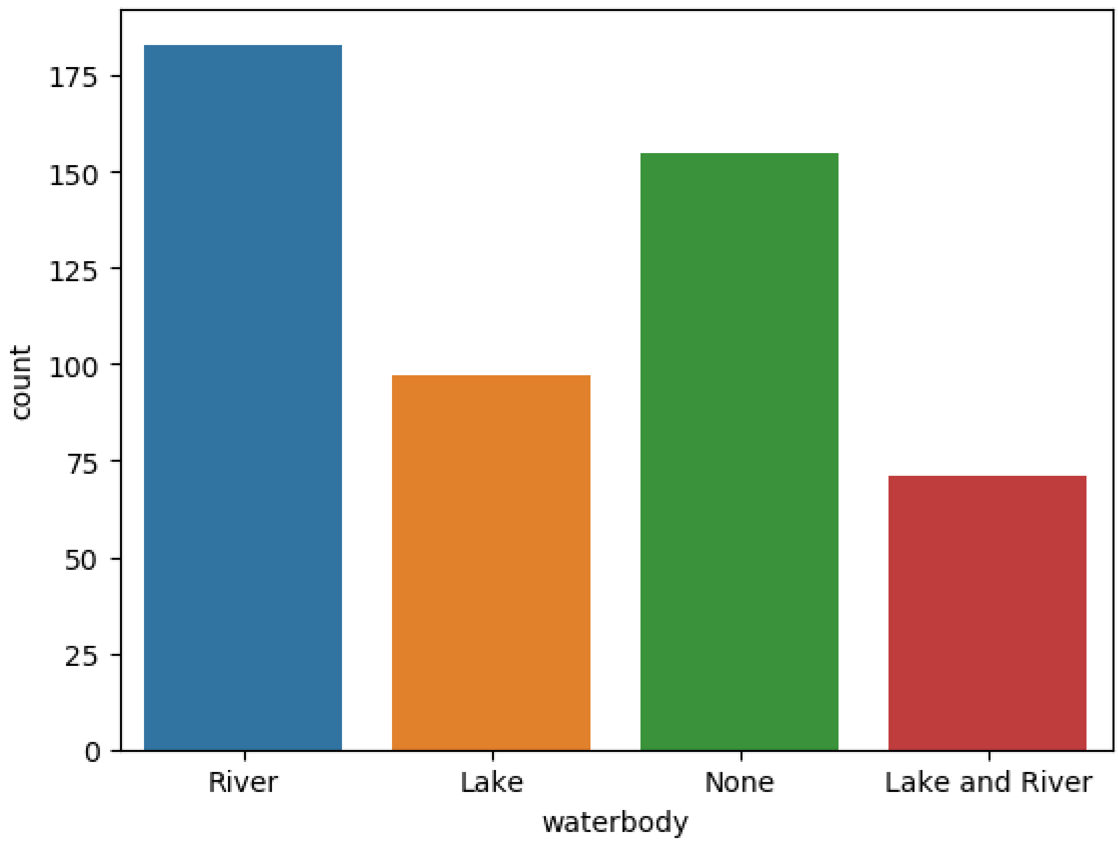
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x1a9f4634a08>



- there is nothing unusual with this data

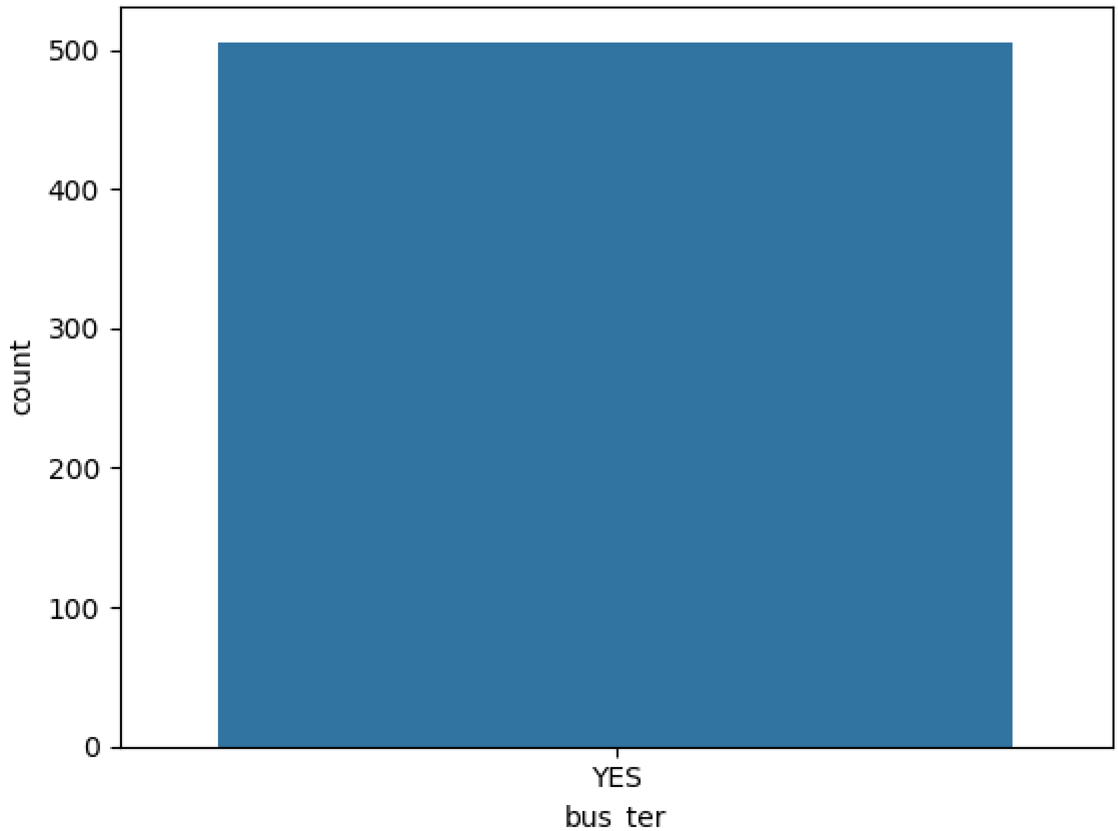
In [11]:
```
sns.countplot(x='waterbody', data=df)
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x2045df708c8>



In [12]:
```
sns.countplot(x='bus_ter', data=df)
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x2045e11ac48>



- my bus terminal is taking only one value, so this not be useful in my analysis since this will not provide any differentiation power for my dependent variable

Observations

1. Missing value in n_hos_beds
2. Skewness or outlier in crime rate
3. Outlier in n_hot_rooms and rainfall
4. Bus_ter is only 'YES'

In [13]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 19 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   price        506 non-null    float64
 1   crime_rate   506 non-null    float64
 2   resid_area   506 non-null    float64
 3   air_qual     506 non-null    float64
 4   room_num     506 non-null    float64
 5   age          506 non-null    float64
 6   dist1        506 non-null    float64
 7   dist2        506 non-null    float64
 8   dist3        506 non-null    float64
 9   dist4        506 non-null    float64
 10  teachers     506 non-null    float64
 11  poor_prop    506 non-null    float64
 12  airport      506 non-null    object 
 13  n_hos_beds   498 non-null    float64
 14  n_hot_rooms  506 non-null    float64
 15  waterbody    506 non-null    object 
 16  rainfall     506 non-null    int64  
 17  bus_ter      506 non-null    object 
 18  parks        506 non-null    float64
dtypes: float64(15), int64(1), object(3)
memory usage: 75.2+ KB
```

Outlier Treatment

In [16]:
```
# Outlier treatment(upper side max value is being larger)
np.percentile(df.n_hot_rooms,[99])
```

Out[16]:
```
array([15.39952])
```

In [17]:
```
np.percentile(df.n_hot_rooms,[99])[0]# to fetch the 1st element of the array
```

Out[17]:
```
15.39952
```

In [18]:
```
# to check the upper limit
uv=np.percentile(df.n_hot_rooms,[99])[0]
```

In [19]:
```
df[(df.n_hot_rooms>uv)] # to find how many values we have > the 99the percenti
le
```

Out[19]:

	price	crime_rate	resid_area	air_qual	room_num	age	dist1	dist2	dist3	dist4	teachers
2	34.7	0.02729	37.07	0.4690	7.185	61.1	5.03	4.86	5.01	4.97	22.2
166	50.0	2.01019	49.58	0.6050	7.929	96.2	2.11	1.91	2.31	1.86	25.3
204	50.0	0.02009	32.68	0.4161	8.034	31.9	5.41	4.80	5.28	4.99	25.3
267	50.0	0.57834	33.97	0.5750	8.297	67.0	2.60	2.13	2.43	2.52	27.0
369	50.0	5.66998	48.10	0.6310	6.683	96.8	1.55	1.28	1.65	0.94	19.8
423	13.4	7.05042	48.10	0.6140	6.103	85.1	2.08	1.80	2.34	1.87	19.8

In [20]:
```
# To caping values
# since 15.40 is close to my uv value i'll leave it as is and  I'm going to us
e 3std
df.n_hot_rooms[(df.n_hot_rooms)>3*uv] = 3*uv
```

```
C:\Users\dalaw\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports u
ntil
```

In [22]:
```
df[(df.n_hot_rooms)>uv]
```

Out[22]:

	price	crime_rate	resid_area	air_qual	room_num	age	dist1	dist2	dist3	dist4	teachers
2	34.7	0.02729	37.07	0.4690	7.185	61.1	5.03	4.86	5.01	4.97	22.2
166	50.0	2.01019	49.58	0.6050	7.929	96.2	2.11	1.91	2.31	1.86	25.3
204	50.0	0.02009	32.68	0.4161	8.034	31.9	5.41	4.80	5.28	4.99	25.3
267	50.0	0.57834	33.97	0.5750	8.297	67.0	2.60	2.13	2.43	2.52	27.0
369	50.0	5.66998	48.10	0.6310	6.683	96.8	1.55	1.28	1.65	0.94	19.8
423	13.4	7.05042	48.10	0.6140	6.103	85.1	2.08	1.80	2.34	1.87	19.8

In [24]:
```
# Treating outliers in lowerend - rainfall
np.percentile(df.rainfall,[1])[0]
```

Out[24]: 20.0

In [25]:
```
lv=np.percentile(df.rainfall,[1])[0]
```

In [26]:
```
df[(df.rainfall)<lv]
```
Out[26]:

	price	crime_rate	resid_area	air_qual	room_num	age	dist1	dist2	dist3	dist4	teachers
213	28.1	0.14052	40.59	0.489	6.375	32.3	4.11	3.92	4.18	3.57	21.4

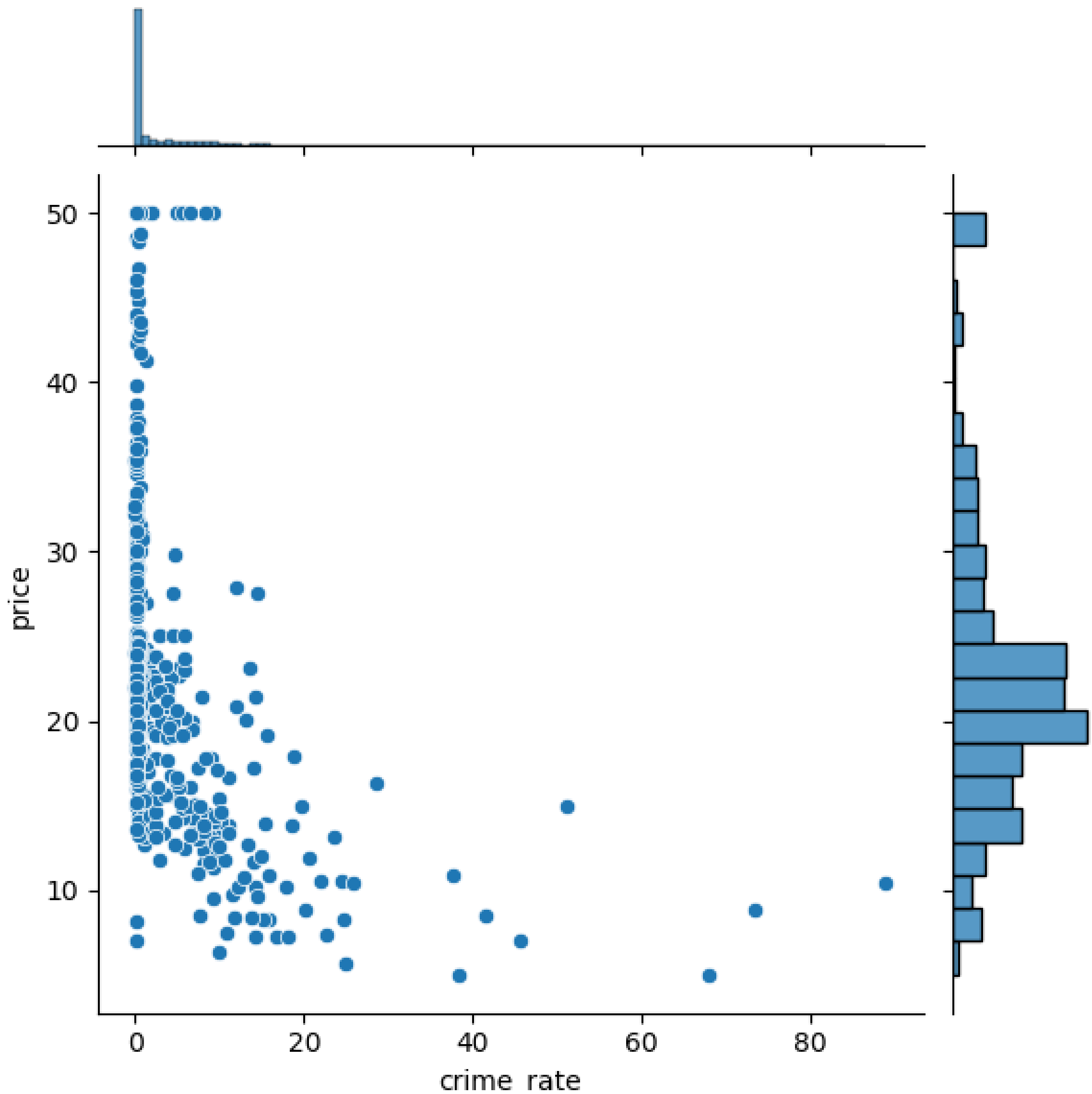
In [27]:
```
# lover values must be multiplte by decimal value
df.rainfall[(df.rainfall<0.3*lv)] = 0.3*lv
```
```
C:\Users\dalaw\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
```

In [28]:
```
df[(df.rainfall)<lv]
```
Out[28]:

	price	crime_rate	resid_area	air_qual	room_num	age	dist1	dist2	dist3	dist4	teachers
213	28.1	0.14052	40.59	0.489	6.375	32.3	4.11	3.92	4.18	3.57	21.4

In [25]:
```
sns.jointplot(x='crime_rate', y='price',data=df)
```

Out[25]: <seaborn.axisgrid.JointGrid at 0x2045e0148c8>



- most of the points are concentrated towards slow crime rate, whereas there are few values which have high crime rate and also I can see kind of polynimial relationship with Y (for low crime rate, the price is high, but as the crime rate is increasing the price rate is decreasing) it does not have linear relationship
- To make it more linear we can take log or exponancial or square root when we do that outliers will be automatically gone

In [26]:
```
# First I transform the values and then treat outliers
df.describe()
```

Out[26]:

	price	crime_rate	resid_area	air_qual	room_num	age	dist1	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.
mean	22.528854	3.613524	41.136779	0.554695	6.284634	68.574901	3.971996	3.
std	9.182176	8.601545	6.860353	0.115878	0.702617	28.148861	2.108532	2.
min	5.000000	0.006320	30.460000	0.385000	3.561000	2.900000	1.130000	0.
25%	17.025000	0.082045	35.190000	0.449000	5.885500	45.025000	2.270000	1.
50%	21.200000	0.256510	39.690000	0.538000	6.208500	77.500000	3.385000	3.
75%	25.000000	3.677083	48.100000	0.624000	6.623500	94.075000	5.367500	4.
max	50.000000	88.976200	57.740000	0.871000	8.780000	100.000000	12.320000	11.

Missing Value imputation

In [27]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 19 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   price        506 non-null    float64
 1   crime_rate   506 non-null    float64
 2   resid_area   506 non-null    float64
 3   air_qual     506 non-null    float64
 4   room_num     506 non-null    float64
 5   age          506 non-null    float64
 6   dist1        506 non-null    float64
 7   dist2        506 non-null    float64
 8   dist3        506 non-null    float64
 9   dist4        506 non-null    float64
 10  teachers     506 non-null    float64
 11  poor_prop    506 non-null    float64
 12  airport      506 non-null    object 
 13  n_hos_beds   498 non-null    float64
 14  n_hot_rooms  506 non-null    float64
 15  waterbody    506 non-null    object 
 16  rainfall     506 non-null    int64  
 17  bus_ter      506 non-null    object 
 18  parks        506 non-null    float64
dtypes: float64(15), int64(1), object(3)
memory usage: 75.2+ KB
```

In [28]:
```
df[df.isna().any(axis=1)]
```

Out[28]:

	price	crime_rate	resid_area	air_qual	room_num	age	dist1	dist2	dist3	dist4	teachers
50	19.7	0.08873	35.64	0.439	5.963	45.7	7.08	6.55	7.00	6.63	23.2
112	18.8	0.12329	40.01	0.547	5.913	92.9	2.55	2.23	2.56	2.07	22.2
215	25.0	0.19802	40.59	0.489	6.182	42.4	4.15	3.81	3.96	3.87	21.4
260	33.8	0.54011	33.97	0.647	7.203	81.8	2.12	1.95	2.37	2.01	27.0
359	22.6	4.26131	48.10	0.770	6.112	81.3	2.78	2.38	2.56	2.31	19.8
403	8.3	24.80170	48.10	0.693	5.349	96.0	1.75	1.38	1.88	1.80	19.8
416	7.5	10.83420	48.10	0.679	6.782	90.8	1.90	1.54	2.04	1.80	19.8
496	19.7	0.28960	39.69	0.585	5.390	72.9	2.86	2.61	2.98	2.74	20.8

In [29]:
```
df.n_hos_beds = df.n_hos_beds.fillna(df.n_hos_beds.mean())
```

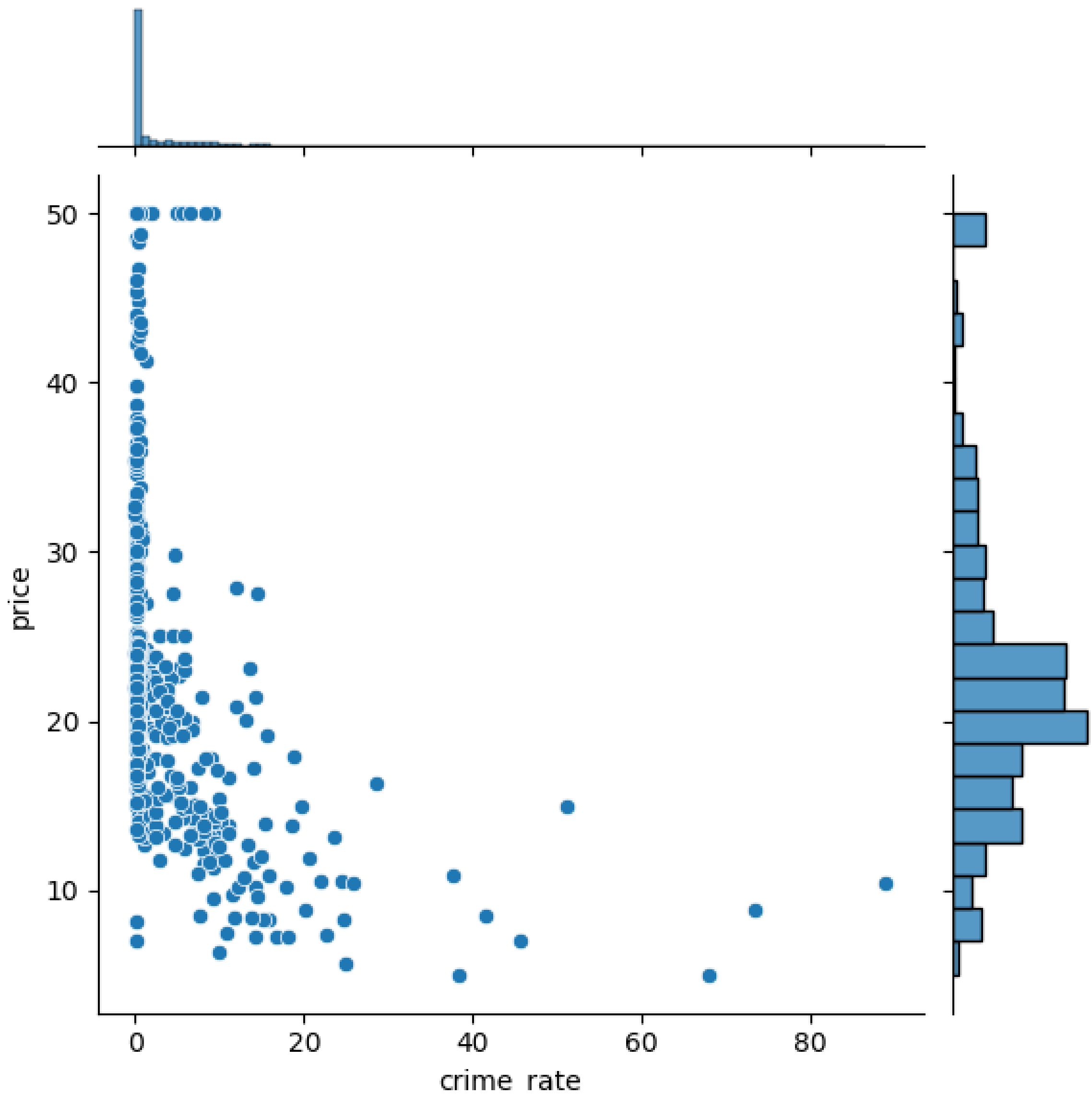
In [30]:
```
df[df.isna().any(axis=1)]
```

Out[30]:

price	crime_rate	resid_area	air_qual	room_num	age	dist1	dist2	dist3	dist4	teachers	po

In [31]:
```
#To transform crime rate variable
sns.jointplot(x='crime_rate', y = 'price', data=df)
```

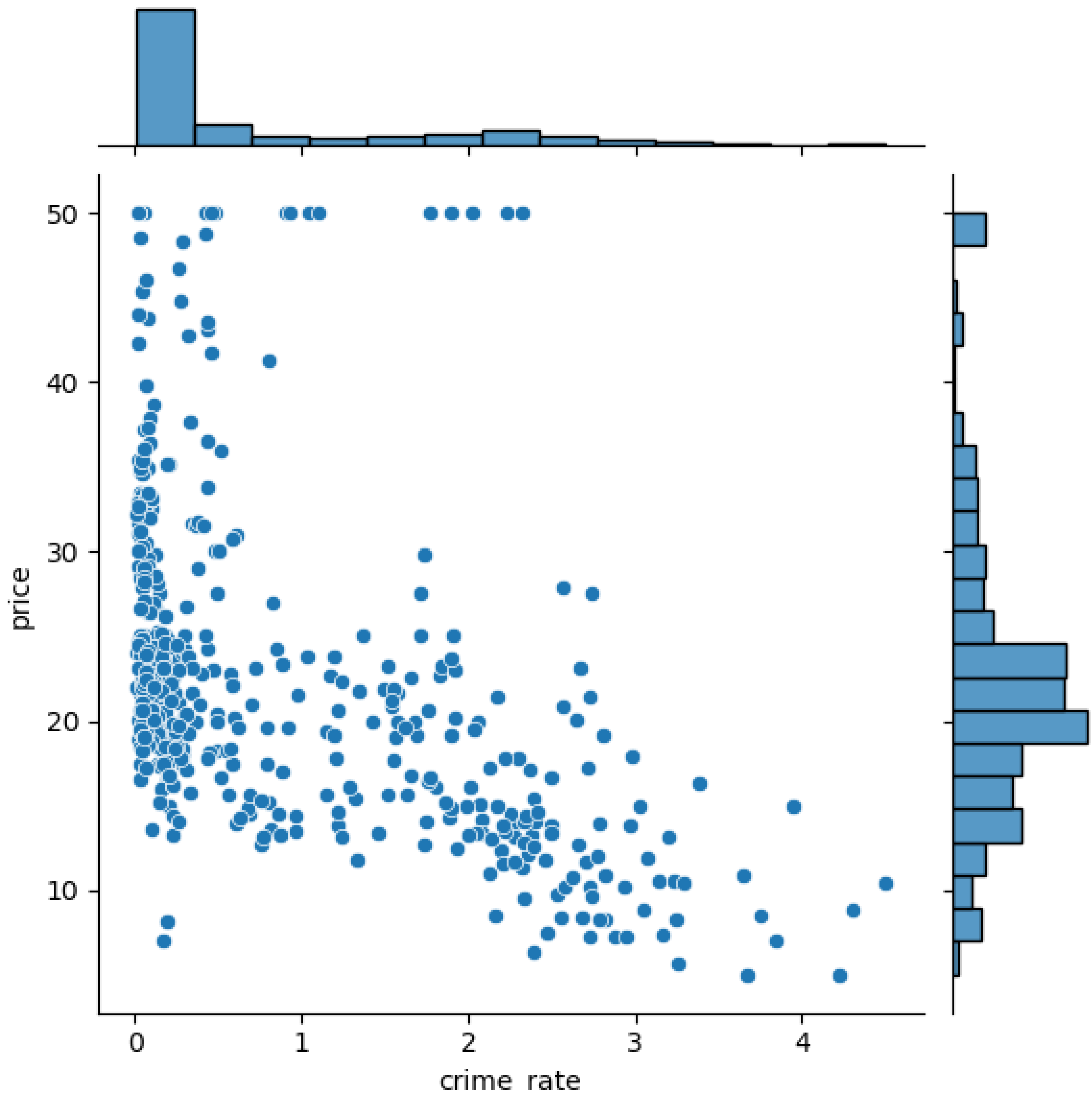
Out[31]: <seaborn.axisgrid.JointGrid at 0x2045e425e48>



In [32]:
```
df.crime_rate = np.log(1+df.crime_rate)
```

In [33]:
```
sns.jointplot(x='crime_rate', y = 'price', data=df)
```

Out[33]: <seaborn.axisgrid.JointGrid at 0x2045e932e48>



- now we are getting a somewhat linear plot and the outliers are already trated

In [34]:
```python
# creating average variable for 4 distances from the properties to the work pl
ace(dist1, dist2,dist3,dist4)
df['avg_dist'] = (df.dist1+df.dist2+df.dist3+df.dist4)/4
df.describe()
```

Out[34]:

	price	crime_rate	resid_area	air_qual	room_num	age	dist1	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.
mean	22.528854	0.813418	41.136779	0.554695	6.284634	68.574901	3.971996	3.
std	9.182176	1.022731	6.860353	0.115878	0.702617	28.148861	2.108532	2.
min	5.000000	0.006300	30.460000	0.385000	3.561000	2.900000	1.130000	0.
25%	17.025000	0.078853	35.190000	0.449000	5.885500	45.025000	2.270000	1.
50%	21.200000	0.228336	39.690000	0.538000	6.208500	77.500000	3.385000	3.
75%	25.000000	1.542674	48.100000	0.624000	6.623500	94.075000	5.367500	4.
max	50.000000	4.499545	57.740000	0.871000	8.780000	100.000000	12.320000	11.

In [35]:
```python
# removing 4 variables(dist1 dist2 dist3 dist4)
del df['dist1']
```

In [36]:
```python
del df['dist2']
```

In [37]:
```python
del df['dist3']
```

In [38]:
```python
del df['dist4']
```

In [39]:
```python
df.describe()
```

Out[39]:

	price	crime_rate	resid_area	air_qual	room_num	age	teachers	poo
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.
mean	22.528854	0.813418	41.136779	0.554695	6.284634	68.574901	21.544466	12.
std	9.182176	1.022731	6.860353	0.115878	0.702617	28.148861	2.164946	7.
min	5.000000	0.006300	30.460000	0.385000	3.561000	2.900000	18.000000	1.
25%	17.025000	0.078853	35.190000	0.449000	5.885500	45.025000	19.800000	6.
50%	21.200000	0.228336	39.690000	0.538000	6.208500	77.500000	20.950000	11.
75%	25.000000	1.542674	48.100000	0.624000	6.623500	94.075000	22.600000	16.
max	50.000000	4.499545	57.740000	0.871000	8.780000	100.000000	27.400000	37.

In [40]:
```python
del df['bus_ter']
```

In [42]:
```python
df.head()
```

Out[42]:

	price	crime_rate	resid_area	air_qual	room_num	age	teachers	poor_prop	airport	n_hos_l
0	24.0	0.006300	32.31	0.538	6.575	65.2	24.7	4.98	YES	5
1	21.6	0.026944	37.07	0.469	6.421	78.9	22.2	9.14	NO	7
2	34.7	0.026924	37.07	0.469	7.185	61.1	22.2	4.03	NO	7
3	33.4	0.031857	32.18	0.458	6.998	45.8	21.3	2.94	YES	9
4	36.2	0.066770	32.18	0.458	7.147	54.2	21.3	5.33	NO	8

In [43]:
```python
# To transform  caregorical variables to numerical, crating dummy variables
df=pd.get_dummies(df)
```

In [44]:
```python
df.head()
```

Out[44]:

	price	crime_rate	resid_area	air_qual	room_num	age	teachers	poor_prop	n_hos_beds	n_
0	24.0	0.006300	32.31	0.538	6.575	65.2	24.7	4.98	5.480	
1	21.6	0.026944	37.07	0.469	6.421	78.9	22.2	9.14	7.332	
2	34.7	0.026924	37.07	0.469	7.185	61.1	22.2	4.03	7.394	
3	33.4	0.031857	32.18	0.458	6.998	45.8	21.3	2.94	9.268	
4	36.2	0.066770	32.18	0.458	7.147	54.2	21.3	5.33	8.824	

In [45]:
```python
# I'm deleting airport no variable becase one variable gives me the info
del df['airport_NO']
```

In [46]:
```python
# I'm deleting waterbody_none it a redundent variable
del df['waterbody_None']
```

In [47]:
```python
df.head()
```

Out[47]:

	price	crime_rate	resid_area	air_qual	room_num	age	teachers	poor_prop	n_hos_beds	n_
0	24.0	0.006300	32.31	0.538	6.575	65.2	24.7	4.98	5.480	
1	21.6	0.026944	37.07	0.469	6.421	78.9	22.2	9.14	7.332	
2	34.7	0.026924	37.07	0.469	7.185	61.1	22.2	4.03	7.394	
3	33.4	0.031857	32.18	0.458	6.998	45.8	21.3	2.94	9.268	
4	36.2	0.066770	32.18	0.458	7.147	54.2	21.3	5.33	8.824	

In [48]:
```python
# to find out which variables to select
df.corr()
```

Out[48]:

	price	crime_rate	resid_area	air_qual	room_num	age	teachers	p
price	1.000000	-0.466527	-0.484754	-0.429300	0.696304	-0.377999	0.505655	
crime_rate	-0.466527	1.000000	0.660283	0.707587	-0.288784	0.559591	-0.390052	
resid_area	-0.484754	0.660283	1.000000	0.763651	-0.391676	0.644779	-0.383248	
air_qual	-0.429300	0.707587	0.763651	1.000000	-0.302188	0.731470	-0.188933	
room_num	0.696304	-0.288784	-0.391676	-0.302188	1.000000	-0.240265	0.355501	
age	-0.377999	0.559591	0.644779	0.731470	-0.240265	1.000000	-0.261515	
teachers	0.505655	-0.390052	-0.383248	-0.188933	0.355501	-0.261515	1.000000	
poor_prop	-0.740836	0.608970	0.603800	0.590879	-0.613808	0.602339	-0.374044	
n_hos_beds	0.108880	-0.004089	0.005799	-0.049553	0.032009	-0.021012	-0.008056	
n_hot_rooms	0.017007	0.056570	-0.003761	0.007238	0.014583	0.013918	-0.037007	
rainfall	-0.047200	0.082151	0.055845	0.091956	-0.064718	0.074684	-0.045928	
parks	-0.391574	0.638951	0.707635	0.915544	-0.282817	0.673850	-0.187004	
avg_dist	0.249289	-0.586371	-0.708022	-0.769247	0.205241	-0.747906	0.232452	
airport_YES	0.182867	-0.134486	-0.115401	-0.073903	0.163774	0.005101	0.069437	
waterbody_Lake	0.036233	-0.025390	-0.026590	-0.046393	-0.004195	0.003452	0.048717	
waterbody_Lake and River	-0.037497	0.009076	0.051649	0.013849	0.010554	-0.004354	-0.046981	
waterbody_River	0.071751	-0.060099	-0.098976	-0.037772	0.046251	-0.088609	0.094256	

- room_num - is close to 1 and it is important variable for my analysis
- teachers - is close to 1 and it is important variable for my analysis
- poor_prop (poor population) close to -1 which means that price and poor population are highly corelated with each other

Multicollinearity

- The high correlation between 2 independent variables leads to a problem try to identifing values which are greater than 0.8 and less than -0.8
- Parks and air_qul has a high correlation with each other so I need to delete one of them, to do so I'm checking the correlation between my dependent variable(Price) air_qual has strong relationship comparet to park so I'm going delete park

In [49]:
```python
del df['parks']
```

In [50]:
```
df.head()
```

Out[50]:

	price	crime_rate	resid_area	air_qual	room_num	age	teachers	poor_prop	n_hos_beds	n_
0	24.0	0.006300	32.31	0.538	6.575	65.2	24.7	4.98	5.480	
1	21.6	0.026944	37.07	0.469	6.421	78.9	22.2	9.14	7.332	
2	34.7	0.026924	37.07	0.469	7.185	61.1	22.2	4.03	7.394	
3	33.4	0.031857	32.18	0.458	6.998	45.8	21.3	2.94	9.268	
4	36.2	0.066770	32.18	0.458	7.147	54.2	21.3	5.33	8.824	

In [51]:
```
# Building a simple linear regresion model in python
import statsmodels.api as sn
x = sn.add_constant(df['room_num'])
```

```
C:\Users\dalaw\Anaconda3\lib\site-packages\statsmodels\tools\_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the pu
blic API at pandas.testing instead.
  import pandas.util.testing as tm
C:\Users\dalaw\Anaconda3\lib\site-packages\statsmodels\tsa\tsatools.py:117: F
utureWarning: In a future version of pandas all arguments of concat except fo
r the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

In [52]:
```
lm = sn.OLS(df['price'], x).fit()
```

In [53]:
```
lm.summary()
```

Out[53]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.485
Model:	OLS	**Adj. R-squared:**	0.484
Method:	Least Squares	**F-statistic:**	474.3
Date:	Wed, 13 Sep 2023	**Prob (F-statistic):**	1.31e-74
Time:	12:01:46	**Log-Likelihood:**	-1671.6
No. Observations:	506	**AIC:**	3347.
Df Residuals:	504	**BIC:**	3356.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-34.6592	2.642	-13.118	0.000	-39.850	-29.468
room_num	9.0997	0.418	21.779	0.000	8.279	9.921

Omnibus:	103.753	**Durbin-Watson:**	0.681
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	633.429
Skew:	0.729	**Prob(JB):**	2.84e-138
Kurtosis:	8.284	**Cond. No.**	58.4

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

- I can see here the p value is 0 which means that there is a significant relationship between room number and price variable
- R squred also nearly 0.5 which means that its correct to run a linear regression model

In [54]:

```python
from sklearn.linear_model import LinearRegression
```

```
C:\Users\dalaw\Anaconda3\lib\site-packages\sklearn\linear_model\least_angle.p
y:30: DeprecationWarning: `np.float` is a deprecated alias for the builtin `f
loat`. To silence this warning, use `float` by itself. Doing this will not mo
dify any behavior and is safe. If you specifically wanted the numpy scalar ty
pe, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/de
vdocs/release/1.20.0-notes.html#deprecations
  method='lar', copy_X=True, eps=np.finfo(np.float).eps,
C:\Users\dalaw\Anaconda3\lib\site-packages\sklearn\linear_model\least_angle.p
y:167: DeprecationWarning: `np.float` is a deprecated alias for the builtin `
float`. To silence this warning, use `float` by itself. Doing this will not m
odify any behavior and is safe. If you specifically wanted the numpy scalar t
ype, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/de
vdocs/release/1.20.0-notes.html#deprecations
  method='lar', copy_X=True, eps=np.finfo(np.float).eps,
C:\Users\dalaw\Anaconda3\lib\site-packages\sklearn\linear_model\least_angle.p
y:284: DeprecationWarning: `np.float` is a deprecated alias for the builtin `
float`. To silence this warning, use `float` by itself. Doing this will not m
odify any behavior and is safe. If you specifically wanted the numpy scalar t
ype, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/de
vdocs/release/1.20.0-notes.html#deprecations
  eps=np.finfo(np.float).eps, copy_Gram=True, verbose=0,
C:\Users\dalaw\Anaconda3\lib\site-packages\sklearn\linear_model\least_angle.p
y:862: DeprecationWarning: `np.float` is a deprecated alias for the builtin `
float`. To silence this warning, use `float` by itself. Doing this will not m
odify any behavior and is safe. If you specifically wanted the numpy scalar t
ype, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/de
vdocs/release/1.20.0-notes.html#deprecations
  eps=np.finfo(np.float).eps, copy_X=True, fit_path=True,
C:\Users\dalaw\Anaconda3\lib\site-packages\sklearn\linear_model\least_angle.p
y:1101: DeprecationWarning: `np.float` is a deprecated alias for the builtin
`float`. To silence this warning, use `float` by itself. Doing this will not
modify any behavior and is safe. If you specifically wanted the numpy scalar
type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/de
vdocs/release/1.20.0-notes.html#deprecations
  eps=np.finfo(np.float).eps, copy_X=True, fit_path=True,
C:\Users\dalaw\Anaconda3\lib\site-packages\sklearn\linear_model\least_angle.p
y:1127: DeprecationWarning: `np.float` is a deprecated alias for the builtin
`float`. To silence this warning, use `float` by itself. Doing this will not
modify any behavior and is safe. If you specifically wanted the numpy scalar
type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/de
vdocs/release/1.20.0-notes.html#deprecations
  eps=np.finfo(np.float).eps, positive=False):
C:\Users\dalaw\Anaconda3\lib\site-packages\sklearn\linear_model\least_angle.p
y:1362: DeprecationWarning: `np.float` is a deprecated alias for the builtin
`float`. To silence this warning, use `float` by itself. Doing this will not
modify any behavior and is safe. If you specifically wanted the numpy scalar
type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/de
vdocs/release/1.20.0-notes.html#deprecations
  max_n_alphas=1000, n_jobs=None, eps=np.finfo(np.float).eps,
C:\Users\dalaw\Anaconda3\lib\site-packages\sklearn\linear_model\least_angle.p
```

```
y:1602: DeprecationWarning: `np.float` is a deprecated alias for the builtin
`float`. To silence this warning, use `float` by itself. Doing this will not
modify any behavior and is safe. If you specifically wanted the numpy scalar
type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/de
vdocs/release/1.20.0-notes.html#deprecations
  max_n_alphas=1000, n_jobs=None, eps=np.finfo(np.float).eps,
C:\Users\dalaw\Anaconda3\lib\site-packages\sklearn\linear_model\least_angle.p
y:1738: DeprecationWarning: `np.float` is a deprecated alias for the builtin
`float`. To silence this warning, use `float` by itself. Doing this will not
modify any behavior and is safe. If you specifically wanted the numpy scalar
type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/de
vdocs/release/1.20.0-notes.html#deprecations
  eps=np.finfo(np.float).eps, copy_X=True, positive=False):
```

In [55]:
```
y=df['price']
```

In [56]:
```
x=df[['room_num']]  # to make this 2 dimensional array I use 2 square bracket
```

In [60]:

```
help(lm)
```

```
Help on RegressionResultsWrapper in module statsmodels.regression.linear_mode
l object:

class RegressionResultsWrapper(statsmodels.base.wrapper.ResultsWrapper)
 |  RegressionResultsWrapper(results)
 |
 |  Class which wraps a statsmodels estimation Results class and steps in to
 |  reattach metadata to results (if available)
 |
 |  Method resolution order:
 |      RegressionResultsWrapper
 |      statsmodels.base.wrapper.ResultsWrapper
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  conf_int(self, alpha=0.05, cols=None)
 |      conf_int(self, alpha=0.05, cols=None)
 |
 |      Returns the confidence interval of the fitted parameters.
 |
 |      Parameters
 |      ----------
 |      alpha : float, optional
 |          The `alpha` level for the confidence interval.
 |          ie., The default `alpha` = .05 returns a 95% confidence interval.
 |      cols : array-like, optional
 |          `cols` specifies which confidence intervals to return
 |
 |      Notes
 |      -----
 |      The confidence interval is based on Student's t-distribution.
 |
 |  cov_params(self, r_matrix=None, column=None, scale=None, cov_p=None, othe
r=None)
 |      cov_params(self, r_matrix=None, column=None, scale=None, cov_p=None,
other=None)
 |
 |      Returns the variance/covariance matrix.
 |
 |      The variance/covariance matrix can be of a linear contrast
 |      of the estimates of params or all params multiplied by scale which
 |      will usually be an estimate of sigma^2.  Scale is assumed to be
 |      a scalar.
 |
 |      Parameters
 |      ----------
 |      r_matrix : array-like
 |          Can be 1d, or 2d.  Can be used alone or with other.
 |      column :  array-like, optional
 |          Must be used on its own.  Can be 0d or 1d see below.
 |      scale : float, optional
 |          Can be specified or not.  Default is None, which means that
 |          the scale argument is taken from the model.
 |      other : array-like, optional
 |          Can be used when r_matrix is specified.
 |
```

```
 |      Returns
 |      -------
 |      cov : ndarray
 |          covariance matrix of the parameter estimates or of linear
 |          combination of parameter estimates. See Notes.
 |
 |      Notes
 |      -----
 |      (The below are assumed to be in matrix notation.)
 |
 |      If no argument is specified returns the covariance matrix of a model
 |      ``(scale)*(X.T X)^(-1)``
 |
 |      If contrast is specified it pre and post-multiplies as follows
 |      ``(scale) * r_matrix (X.T X)^(-1) r_matrix.T``
 |
 |      If contrast and other are specified returns
 |      ``(scale) * r_matrix (X.T X)^(-1) other.T``
 |
 |      If column is specified returns
 |      ``(scale) * (X.T X)^(-1)[column,column]`` if column is 0d
 |
 |      OR
 |
 |      ``(scale) * (X.T X)^(-1)[column][:,column]`` if column is 1d
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from statsmodels.base.wrapper.ResultsWrapper:
 |
 |  __dir__(self)
 |      Default dir() implementation.
 |
 |  __getattribute__(self, attr)
 |      Return getattr(self, name).
 |
 |  __getstate__(self)
 |
 |  __init__(self, results)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __setstate__(self, dict_)
 |
 |  save(self, fname, remove_data=False)
 |      save a pickle of this instance
 |
 |      Parameters
 |      ----------
 |      fname : string or filehandle
 |          fname can be a string to a file path or filename, or a filehandl
e.
 |      remove_data : bool
 |          If False (default), then the instance is pickled without changes.
 |          If True, then all arrays with length nobs are set to None before
 |          pickling. See the remove_data method.
 |          In some cases not all arrays will be set to None.
 |
 |  ----------------------------------------------------------------------
```

```
 |  Class methods inherited from statsmodels.base.wrapper.ResultsWrapper:
 |  
 |  load(fname) from builtins.type
 |  
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from statsmodels.base.wrapper.ResultsWrapper:
 |  
 |  __dict__
 |      dictionary for instance variables (if defined)
 |  
 |  __weakref__
 |      list of weak references to the object (if defined)
```

In [61]:
```
lm2 = LinearRegression()
```

In [62]:
```
lm2.fit(x,y)
```

Out[62]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fals
e)
```

In [63]:
```
print(lm2.intercept_, lm2.coef_)
```

```
-34.65924312309721 [9.09966966]
```

In [64]:

```
lm2.predict(x)
```

```
Out[64]: array([25.17108491, 23.76973578, 30.72188341, 29.02024518, 30.37609596,
       23.85163281, 20.04797089, 21.50391804, 16.58099675, 19.97517353,
       23.36935032, 20.02067188, 18.92871152, 19.4746917 , 20.81234314,
       18.42822969, 19.34729633, 19.84777816, 14.98855456, 17.45456504,
       16.0259169 , 19.62028642, 21.23092795, 18.23713663, 19.24719996,
       16.28980732, 18.23713663, 20.36645933, 24.44311134, 26.07195221,
       17.32716966, 20.59395107, 19.48379137, 17.21797363, 20.81234314,
       19.32909699, 18.49192738, 18.57382441, 19.62938609, 25.3530783 ,
       29.25683659, 26.9455205 , 21.47661903, 21.85880515, 20.56665206,
       17.0450799 , 17.99144555, 20.21176495, 14.46987339, 16.31710633,
       19.60208708, 20.98523687, 24.58870605, 19.92057552, 18.91961185,
       31.30426226, 23.42394834, 27.3641053 , 21.25822696, 19.27449897,
       17.58196041, 19.62938609, 24.08822422, 26.87272314, 29.98481016,
       22.57767906, 18.00054522, 18.82861516, 16.24430897, 18.89231284,
       23.7333371 , 19.58388774, 20.53025338, 22.16819392, 22.42298467,
       22.54128038, 22.47758269, 21.21272861, 22.04989822, 18.79221648,
       26.5542347 , 25.57147038, 22.68687509, 21.45841969, 23.47854635,
       25.67156674, 20.0752699 , 21.03983488, 29.10214221, 29.75731842,
       23.7333371 , 23.62414107, 23.96082885, 21.85880515, 22.2045926 ,
       25.62606839, 21.42202101, 38.76599139, 36.50017364, 32.8239071 ,
       26.5542347 , 27.04561686, 23.62414107, 21.1854296 , 21.45841969,
       18.58292408, 18.44642903, 21.0944329 , 24.25201828, 22.02259921,
       21.71321044, 26.44503866, 19.14710359, 20.77594446, 22.25009095,
       19.28359864, 21.54031672, 20.12986792, 18.77401714, 17.49096372,
       18.7558178 , 19.97517353, 19.58388774, 18.62842242, 18.83771483,
       19.81137948, 16.4172027 , 17.14517627, 23.86073248, 16.63559477,
       24.10642356, 22.90526717, 23.32385197, 18.31903366, 17.72755513,
       22.98716419, 19.41099401, 24.07002488, 18.63752209, 21.31282497,
       21.52211738, 11.01199892, 14.50627207, 15.09775059,  9.95643723,
       21.12173191, 16.55369774, 10.16572964, 12.53164375, 16.27160798,
       21.04893455, 14.51537174, 10.94830123, 17.29077098, 21.11263224,
       21.32192464, 13.31421534, 28.51976335, 20.53935305, 24.57960638,
       22.21369227, 33.48818298, 36.33637959, 41.55049031, 18.61022308,
       20.85784149, 37.49203764, 18.81951549, 22.84156948, 23.59684206,
       18.80131615, 18.8468145 , 16.04411624, 23.72423744, 18.65572143,
       24.90719449, 20.12076825, 22.8051708 , 27.76449077, 28.85645113,
       35.99969181, 21.24912729, 30.44889332, 25.06188888, 16.33530567,
       21.33102431, 36.60027001, 27.05471653, 24.99819119, 30.72188341,
       28.5925607 , 26.66343074, 30.65818572, 27.21851059, 25.43497533,
       37.00065547, 31.65004971, 30.01210917, 31.53175401, 28.81095278,
       30.26689992, 21.41292134, 34.58924301, 36.80046274, 38.44750295,
       18.94691086, 22.90526717, 17.96414654, 20.52115371, 13.96939156,
       19.57478807, 14.51537174, 18.18253861, 23.35115098, 14.58816909,
       21.59491473, 18.91961185, 25.78076278, 19.49289104, 23.33295164,
       28.5925607 , 21.43112068, 27.93738449, 25.56237071, 40.55862631,
       44.73537469, 38.50210097, 30.52169067, 35.28081791, 24.96179251,
       19.76588113, 32.78750842, 41.20470286, 40.38573259, 26.54513503,
       20.72134645, 25.68066641, 32.29612626, 24.31571596, 25.45317467,
       28.10117854, 20.80324347, 23.19645659, 23.51494503, 16.2352093 ,
       16.34440534, 20.92153918, 21.9953002 , 23.87893182, 26.47233767,
       24.37031398, 23.92443017, 28.64715872, 40.49492862, 20.92153918,
       18.81041582, 33.16969455, 44.54428162, 32.06863452, 27.60069671,
       30.88567746, 33.77027274, 41.75978271, 32.0140365 , 30.91297647,
       15.9349202 , 29.16583989, 40.84071607, 33.31528926, 19.21080128,
       18.62842242, 22.12269557, 24.83439713, 35.32631626, 26.83632446,
       27.70989275, 31.46805632, 27.455102  , 24.32481563, 27.32770662,
       36.50017364, 28.74725509, 34.90773145, 37.43743962, 29.83921545,
```

```
24.06092521, 22.03169888, 21.84060581, 22.8051708 , 25.08008821,
27.77359044, 30.38519563, 25.67156674, 21.0944329 , 20.02067188,
26.10835089, 24.9344935 , 18.02784423, 23.07816089, 29.41153097,
27.86458713, 25.30757996, 24.44311134, 28.87465046, 31.18596656,
25.54417137, 32.86030578, 27.6643944 , 25.71706509, 19.6839841 ,
10.59341411, 21.04893455, 20.14806726, 22.35928699, 25.09828755,
17.2543723 , 19.15620326, 17.95504687, 23.41484867, 20.96703753,
23.81523413, 23.36025065, 20.31186131, 17.28167131, 23.71513777,
23.86073248, 22.77787179, 20.69404744, 18.73761846, 22.96896485,
21.24912729, 17.26347197, 20.22086461, 22.81427047, 22.75967245,
20.27546263, 18.74671813, 18.98330954, 20.47565537, 19.80227981,
19.64758543, 31.23146491, 24.85259647, 26.27214494, 27.89188614,
20.06617023, 19.01060855, 24.6342044 , 25.71706509, 28.48336467,
24.39761299, 25.20748359, 18.88321317, 26.56333437, 16.87218618,
19.356396  , 21.86790482, 23.53314437, 21.0944329 , 20.95793786,
23.56044338, 22.22279194, 14.13318561, 18.14613993, 45.23585652,
-2.25531945, 10.50241741,  0.49278079, 10.5661151 , 26.15384924,
29.18403923, 21.9043035 , 18.80131615,  9.98373624,  2.99518994,
31.88664112, 25.84446047, 27.16391257, 23.39664933, 21.96800119,
28.74725509, 24.89809482, 15.71652813, 15.57093342,  5.08811397,
13.35971369,  7.67242015, 10.83910519,  9.74714483, 14.38797636,
17.32716966, 20.40285801, 11.1666933 , 21.6950111 , 18.91051218,
24.22471927, 23.62414107, 17.63655843, 14.96125555, 18.59202375,
19.82047915, 23.05996155, 23.6150414 , 14.0148899 , 15.67102978,
17.05417957,  2.99518994, 16.37170435, 16.45360137, 27.69169341,
17.72755513, 25.91725782,  7.45402808, 12.24955399,  6.46216408,
23.88803149, 27.05471653, 13.60540477, 19.54748906, 27.43690266,
23.67873909, 19.99337287, 16.73569113, 20.87604083, 15.98041855,
18.99240921, 18.4555287 , 21.77690813, 21.6950111 , 23.39664933,
23.1054599 , 27.51879968, 23.80613446, 23.90623083, 21.83150615,
25.66246707, 24.13372257, 21.32192464, 19.34729633, 16.54459807,
18.28263498, 23.63324074, 21.93160251, 24.35211464, 18.61022308,
24.11552323, 23.04176221, 22.22279194, 21.62221374, 23.7333371 ,
26.75442743, 25.89905848, 22.64137675, 32.6146147 , 26.56333437,
24.71610143, 19.72038278, 19.356396  , 22.67777542, 20.6758481 ,
26.31764329, 23.36025065, 22.82337014, 24.60690539, 21.84060581,
17.74575447, 19.50199071, 19.96607386, 19.2653993 , 17.32716966,
21.45841969, 22.02259921, 23.9153305 , 28.85645113, 14.72466414,
21.41292134, 24.34301497, 13.60540477, 21.62221374, 22.02259921,
22.14089491, 26.7635271 , 29.59352437, 17.77305348, 18.76491747,
22.77787179, 20.9761372 , 19.07430624, 14.97035522, 14.60636843,
11.68537447, 19.78408047, 19.78408047, 17.27257164, 19.2653993 ,
16.93588387, 14.38797636, 18.0642429 , 20.11166858, 16.01681723,
20.18446594, 25.33487897, 21.03073521, 28.82005245, 27.16391257,
20.21176495])
```

In [65]:

```
help(sns.jointplot)
```

```
Help on function jointplot in module seaborn.axisgrid:

jointplot(data=None, *, x=None, y=None, hue=None, kind='scatter', height=6, r
atio=5, space=0.2, dropna=False, xlim=None, ylim=None, color=None, palette=No
ne, hue_order=None, hue_norm=None, marginal_ticks=False, joint_kws=None, marg
inal_kws=None, **kwargs)
    Draw a plot of two variables with bivariate and univariate graphs.

    This function provides a convenient interface to the :class:`JointGrid`
    class, with several canned plot kinds. This is intended to be a fairly
    lightweight wrapper; if you need more flexibility, you should use
    :class:`JointGrid` directly.

    Parameters
    ----------
    data : :class:`pandas.DataFrame`, :class:`numpy.ndarray`, mapping, or seq
uence
        Input data structure. Either a long-form collection of vectors that c
an be
        assigned to named variables or a wide-form dataset that will be inter
nally
        reshaped.
    x, y : vectors or keys in ``data``
        Variables that specify positions on the x and y axes.
    hue : vector or key in ``data``
        Semantic variable that is mapped to determine the color of plot eleme
nts.
        Semantic variable that is mapped to determine the color of plot eleme
nts.
    kind : { "scatter" | "kde" | "hist" | "hex" | "reg" | "resid" }
        Kind of plot to draw. See the examples for references to the underlyi
ng functions.
    height : numeric
        Size of the figure (it will be square).
    ratio : numeric
        Ratio of joint axes height to marginal axes height.
    space : numeric
        Space between the joint and marginal axes
    dropna : bool
        If True, remove observations that are missing from ``x`` and ``y``.
    {x, y}lim : pairs of numbers
        Axis limits to set before plotting.
    color : :mod:`matplotlib color <matplotlib.colors>`
        Single color specification for when hue mapping is not used. Otherwis
e, the
        plot will try to hook into the matplotlib property cycle.
    palette : string, list, dict, or :class:`matplotlib.colors.Colormap`
        Method for choosing the colors to use when mapping the ``hue`` semant
ic.
        String values are passed to :func:`color_palette`. List or dict value
s
        imply categorical mapping, while a colormap object implies numeric ma
pping.
    hue_order : vector of strings
        Specify the order of processing and plotting for categorical levels o
f the
        ``hue`` semantic.
```

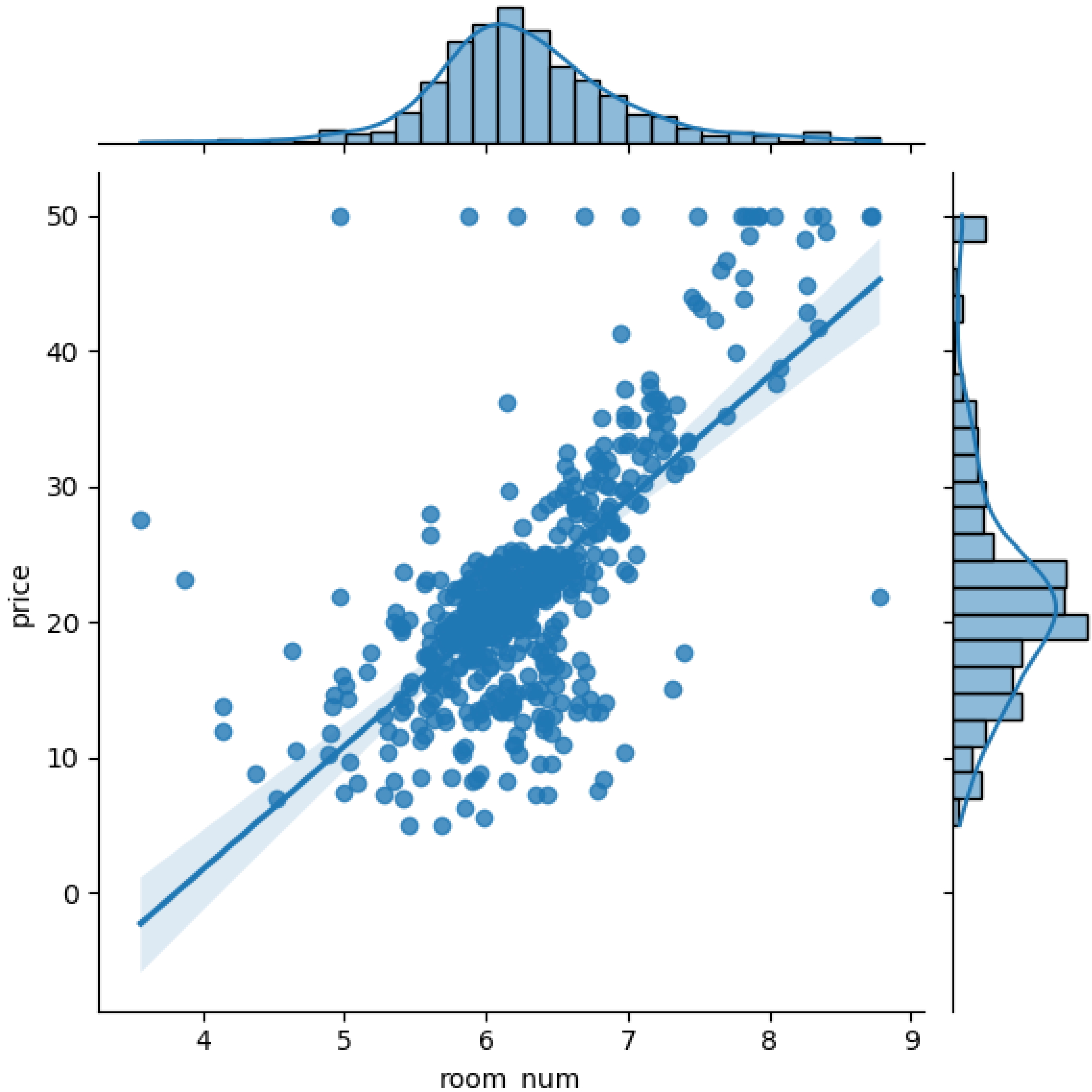
```
    hue_norm : tuple or :class:`matplotlib.colors.Normalize`
        Either a pair of values that set the normalization range in data unit
s
        or an object that will map from data units into a [0, 1] interval. Us
age
        implies numeric mapping.
    marginal_ticks : bool
        If False, suppress ticks on the count/density axis of the marginal pl
ots.
    {joint, marginal}_kws : dicts
        Additional keyword arguments for the plot components.
    kwargs
        Additional keyword arguments are passed to the function used to
        draw the plot on the joint Axes, superseding items in the
        ``joint_kws`` dictionary.

    Returns
    -------
    :class:`JointGrid`
        An object managing multiple subplots that correspond to joint and mar
ginal axes
        for plotting a bivariate relationship or distribution.

    See Also
    --------
    JointGrid : Set up a figure with joint and marginal views on bivariate da
ta.
    PairGrid : Set up a figure with joint and marginal views on multiple vari
ables.
    jointplot : Draw multiple bivariate plots with univariate marginal distri
butions.

    Examples
    --------

    .. include:: ../docstrings/jointplot.rst
```

In [66]:

```
###To plot the regression line
sns.jointplot(x=df['room_num'], y=df['price'], data=df, kind ='reg')
```

```
C:\Users\dalaw\Anaconda3\lib\site-packages\matplotlib\cbook\__init__.py:1402:
FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) i
s deprecated and will be removed in a future version.  Convert to a numpy arr
ay before indexing instead.
  x[:, None]
C:\Users\dalaw\Anaconda3\lib\site-packages\matplotlib\axes\_base.py:276: Futu
reWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is de
precated and will be removed in a future version.  Convert to a numpy array b
efore indexing instead.
  x = x[:, np.newaxis]
C:\Users\dalaw\Anaconda3\lib\site-packages\matplotlib\axes\_base.py:278: Futu
reWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is de
precated and will be removed in a future version.  Convert to a numpy array b
efore indexing instead.
  y = y[:, np.newaxis]
C:\Users\dalaw\Anaconda3\lib\site-packages\matplotlib\cbook\__init__.py:1402:
FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) i
s deprecated and will be removed in a future version.  Convert to a numpy arr
ay before indexing instead.
  x[:, None]
C:\Users\dalaw\Anaconda3\lib\site-packages\matplotlib\axes\_base.py:276: Futu
reWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is de
precated and will be removed in a future version.  Convert to a numpy array b
efore indexing instead.
  x = x[:, np.newaxis]
C:\Users\dalaw\Anaconda3\lib\site-packages\matplotlib\axes\_base.py:278: Futu
reWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is de
precated and will be removed in a future version.  Convert to a numpy array b
efore indexing instead.
  y = y[:, np.newaxis]
```

Out[66]:
```
<seaborn.axisgrid.JointGrid at 0x204614024c8>
```



- supose we increase my x value from 4 to 5 (1 value) my y value will incrase from 0 to 10 I can say that roughly it will increase by 9 units

Building a multiple linear model

- crathing a X and y variables for my model

In [67]:
```
# Since price variable is my  dependent variable, I need all the variables exc
ept price variable from
x_multi = df.drop('price', axis=1).reset_index(drop=True)
```

In [68]:
```
x_multi.head()
```

Out[68]:

	crime_rate	resid_area	air_qual	room_num	age	teachers	poor_prop	n_hos_beds	n_hot_ro
0	0.006300	32.31	0.538	6.575	65.2	24.7	4.98	5.480	11.19
1	0.026944	37.07	0.469	6.421	78.9	22.2	9.14	7.332	12.17
2	0.026924	37.07	0.469	7.185	61.1	22.2	4.03	7.394	46.19
3	0.031857	32.18	0.458	6.998	45.8	21.3	2.94	9.268	11.26
4	0.066770	32.18	0.458	7.147	54.2	21.3	5.33	8.824	11.28

In [69]:
```
# to create dependent variable
y_multi = df['price']
```

In [70]:
```
y_multi.head()
```

Out[70]:
```
0    24.0
1    21.6
2    34.7
3    33.4
4    36.2
Name: price, dtype: float64
```

In [71]:
```
# To add a constant to my dependent variable
x_multi_cons = sn.add_constant(x_multi)
```

```
C:\Users\dalaw\Anaconda3\lib\site-packages\statsmodels\tsa\tsatools.py:117: F
utureWarning: In a future version of pandas all arguments of concat except fo
r the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

In [72]:
```
x_multi_cons.head()
```

Out[72]:

	const	crime_rate	resid_area	air_qual	room_num	age	teachers	poor_prop	n_hos_beds	n_
0	1.0	0.006300	32.31	0.538	6.575	65.2	24.7	4.98	5.480	
1	1.0	0.026944	37.07	0.469	6.421	78.9	22.2	9.14	7.332	
2	1.0	0.026924	37.07	0.469	7.185	61.1	22.2	4.03	7.394	
3	1.0	0.031857	32.18	0.458	6.998	45.8	21.3	2.94	9.268	
4	1.0	0.066770	32.18	0.458	7.147	54.2	21.3	5.33	8.824	

In [73]:
```
# to fit my model
lm_multi = sn.OLS(y_multi, x_multi_cons).fit()
```

In [85]:
```
x_multi.shape
```
Out[85]:
```
(506, 15)
```

In [74]:

```
lm_multi.summary()
```

Out[74]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.721
Model:	OLS	**Adj. R-squared:**	0.712
Method:	Least Squares	**F-statistic:**	84.34
Date:	Wed, 13 Sep 2023	**Prob (F-statistic):**	4.19e-125
Time:	12:03:29	**Log-Likelihood:**	-1516.6
No. Observations:	506	**AIC:**	3065.
Df Residuals:	490	**BIC:**	3133.
Df Model:	15		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-6.4986	5.264	-1.235	0.218	-16.842	3.844
crime_rate	0.0097	0.348	0.028	0.978	-0.674	0.694
resid_area	-0.0409	0.058	-0.710	0.478	-0.154	0.072
air_qual	-15.8974	4.004	-3.971	0.000	-23.764	-8.031
room_num	4.0190	0.427	9.421	0.000	3.181	4.857
age	-0.0057	0.014	-0.420	0.675	-0.032	0.021
teachers	1.0070	0.122	8.247	0.000	0.767	1.247
poor_prop	-0.5773	0.053	-10.955	0.000	-0.681	-0.474
n_hos_beds	0.3292	0.152	2.163	0.031	0.030	0.628
n_hot_rooms	0.0919	0.082	1.118	0.264	-0.070	0.253
rainfall	0.0161	0.018	0.904	0.367	-0.019	0.051
avg_dist	-1.2186	0.189	-6.450	0.000	-1.590	-0.847
airport_YES	1.1315	0.454	2.491	0.013	0.239	2.024
waterbody_Lake	0.2641	0.642	0.411	0.681	-0.997	1.525
waterbody_Lake and River	-0.6876	0.714	-0.963	0.336	-2.090	0.715
waterbody_River	-0.2913	0.547	-0.533	0.594	-1.365	0.783

Omnibus:	182.596	**Durbin-Watson:**	0.990
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	826.137
Skew:	1.554	**Prob(JB):**	4.04e-180
Kurtosis:	8.434	**Cond. No.**	2.37e+03

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.37e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

- R-squared:0.721 the value is >.5 which is good
- Prob (F-statistic):4.19e-125 is very low so, I can say with confidence that my independent variables have some impact on my dependent variables
- toal No. Observations:506
- Degrees of freedom Df Residuals:490
- lower the p-value the more significant lowest p value - air quality, room numbers, number of teachers in the area, poor population, average distant to the work
- when I check the coefficient for the room numbers I can see (4.0190) when I increase 1 unit my price is going to increase by 4 units
    - If all other variables are constant for 2 houses, if oe of house has more rooms than the other , then the price is going to be 4 units more
- coefficient for airquality (-15.8974) which means that if I am inceasing the value of air quality the price is going to decrease
- corficiaent for airport is 1.1315 if the airport is present in any city the price is going to be increase in 1 unit

Observations

- The house prices are depent mainly on:

  more room Numbers will positively effect to the price Airport present will positively effect to the price Teachers in the area Avg. distant to the work place Poor population has negatively effect to the price

- Same data with a sklearn -

In [75]:
```
lm3 = LinearRegression()
```

In [77]:
```
lm3.fit(x_multi, y_multi)
```

Out[77]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fals
e)
```

In [79]:
```
print(lm3.intercept_, lm3.coef_)
```

```
-6.498625198419436 [ 9.70998193e-03 -4.08746495e-02 -1.58973999e+01  4.019016
76e+00
 -5.71475069e-03  1.00700068e+00 -5.77271243e-01  3.29221139e-01
  9.18675603e-02  1.61185504e-02 -1.21863952e+00  1.13151586e+00
  2.64086064e-01 -6.87555889e-01 -2.91318712e-01]
```

In [81]:
```
# To create test and train split
from sklearn.model_selection import train_test_split
```

In [82]:
```
# need to define my 4 variables 1.independent test & train, dependent test & t
rain and in the btacket indepent & Dependent variable,under the test size 0.2
(20% of data, 80% goes to train and I use random number for just to get a same
number
# everytime when I use random_state = 1 or zero everytime my test and training
sample  remain same )
x_train, x_test, y_train, y_test = train_test_split(x_multi, y_multi, test_siz
e = 0.2, random_state = 0)
```

In [83]:
```
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(404, 15) (102, 15) (404,) (102,)
```

- 404 (80%) Observations are in my trainning set and 102 (20%) observations are in my test

Creating a linear regression model

In [86]:
```
# Creating my object
lm_a = LinearRegression()
```

In [87]:
```
# Going to train my model
lm_a.fit(x_train, y_train)
```

Out[87]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fals
e)
```

In [88]:
```
# creating predicted  value of y using the above value
y_test_a = lm_a.predict(x_test)
```

In [90]:
```
y_train_a = lm_a.predict(x_train)
```

In [91]:
```
# to check the R-squre I'll import r2_score
from sklearn.metrics import r2_score
```

In [92]:
```
# To get the help
r2_score?
```

In [93]:
```
# y_test = original value , y_test_a is the predicted value
r2_score(y_test, y_test_a)
```

Out[93]:
```
0.549646828820567
```

In [94]:
```
r2_score(y_train, y_train_a)
```

Out[94]:
```
0.756463540591123
```

- R-square value for the test data is less than the r-square value of the train, but test data is most important I need compared to trainning set so I need to look at test instead of trainning set's R- square value to evaluate the performance of the model.