In [153…

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import SGDRegressor
```

In [ ]:

```
https://developers.google.com/machine-learning/data-prep/transform/normalization?hl=en
```

Feature Scaling

```
Before we dive right into Multiple Linear Regression, we have to tackle
an issue. This is an issue that wasn't as relevant in our previous
implementation of Regression. As we add in more and more features, the
range of those features, might vary, sometimes even wildly so. For
example let's take our housing data price again.
```

In [18]:

```
df = pd.read_csv('kc_house_data.csv')
```

In [19]:

```
df
```

Out[19]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wat
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	
...	...	...	...	...	...	...	...	...	
21608	263000018	20140521T000000	360000.0	3	2.50	1530	1131	3.0	
21609	6600060120	20150223T000000	400000.0	4	2.50	2310	5813	2.0	
21610	1523300141	20140623T000000	402101.0	2	0.75	1020	1350	2.0	
21611	291310100	20150116T000000	400000.0	3	2.50	1600	2388	2.0	
21612	1523300157	20141015T000000	325000.0	2	0.75	1020	1076	2.0	

21613 rows × 21 columns

In [20]:

```
df.corr()
```

Out[20]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
id	1.000000	-0.016762	0.001286	0.005160	-0.012258	-0.132109	0.018525	-0.002721
price	-0.016762	1.000000	0.308350	0.525138	0.702035	0.089661	0.256794	0.26636
bedrooms	0.001286	0.308350	1.000000	0.515884	0.576671	0.031703	0.175429	-0.00658
bathrooms	0.005160	0.525138	0.515884	1.000000	0.754665	0.087740	0.500653	0.06374
sqft_living	-0.012258	0.702035	0.576671	0.754665	1.000000	0.172826	0.353949	0.10381
sqft_lot	-0.132109	0.089661	0.031703	0.087740	0.172826	1.000000	-0.005201	0.02160
floors	0.018525	0.256794	0.175429	0.500653	0.353949	-0.005201	1.000000	0.02369
waterfront	-0.002721	0.266369	-0.006582	0.063744	0.103818	0.021604	0.023698	1.00000
view	0.011592	0.397293	0.079532	0.187737	0.284611	0.074710	0.029444	0.40185
condition	-0.023783	0.036362	0.028472	-0.124982	-0.058753	-0.008958	-0.263768	0.01665
grade	0.008130	0.667434	0.356967	0.664983	0.762704	0.113621	0.458183	0.08277
sqft_above	-0.010830	0.605567	0.477616	0.685363	0.876644	0.183511	0.523899	0.07207
sqft_basement	-0.005151	0.323816	0.303093	0.283770	0.435043	0.015286	-0.245705	0.08058
yr_built	0.021380	0.054012	0.154178	0.506019	0.318049	0.053080	0.489319	-0.026161
yr_renovated	-0.016907	0.126434	0.018841	0.050739	0.055363	0.007644	0.006338	0.09288
zipcode	-0.008224	-0.053203	-0.152668	-0.203866	-0.199430	-0.129574	-0.059121	0.03028
lat	-0.001891	0.307003	-0.008931	0.024573	0.052529	-0.085683	0.049614	-0.01427
long	0.020799	0.021626	0.129473	0.223042	0.240223	0.229521	0.125419	-0.04191
sqft_living15	-0.002901	0.585379	0.391638	0.568634	0.756420	0.144608	0.279885	0.08646
sqft_lot15	-0.138798	0.082447	0.029244	0.087175	0.183286	0.718557	-0.011269	0.03070

We can see that there seems to be a weak correlation with bedrooms and bathrooms and a strong with one sqft. Let's redefine our model to take in these factors as well. We now have 3 features that we are going to put in.

We can see that price is in 1000s and ranges from 100,000 to 1,000,000. Bedrooms and bathrooms range from 1-4, and then SQFT ranges from ~1000-3000.

Gradient Descent will run, but it might take an extremely long time for it to run. Let's get a feel for the shape of the bowl that we made. If we use the non-scaled features, the bowl might be extremely elongated, and extremely large. This means that in order to reach the middle, that our learning rate is going to have to be small, and that it's going to have to run many many times until it reaches the middle. Now let's see the shape if we were to normalize it. We would get this nice, easy to work with bowl shape, with a clear and easy path to the middle.

In [161…

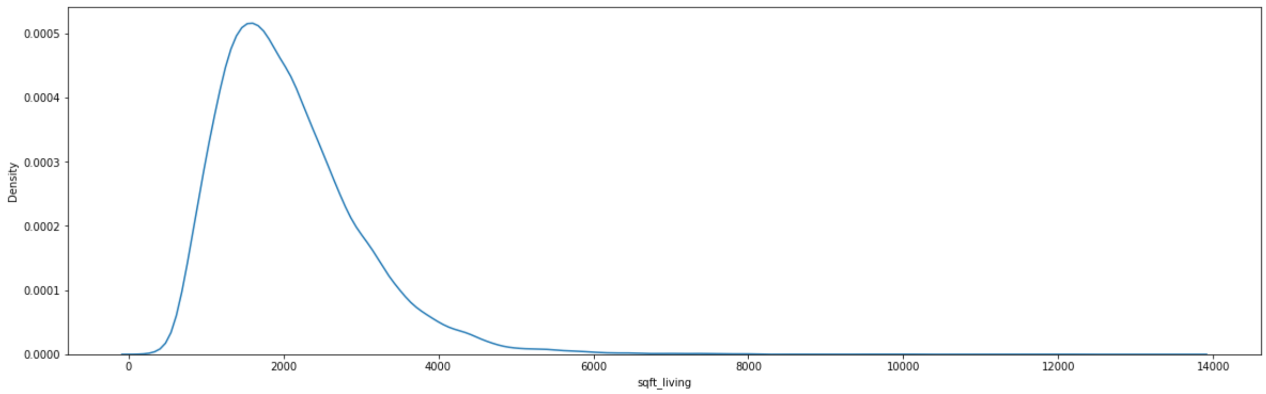
```
features = ['sqft_living','view','grade']
df_to_test = df[features]
```

In [162…

```
fig,ax = plt.subplots(figsize=(20,6))
sns.kdeplot(data=df_to_test['sqft_living'])
```

Out[162…

```
<AxesSubplot:xlabel='sqft_living', ylabel='Density'>
```

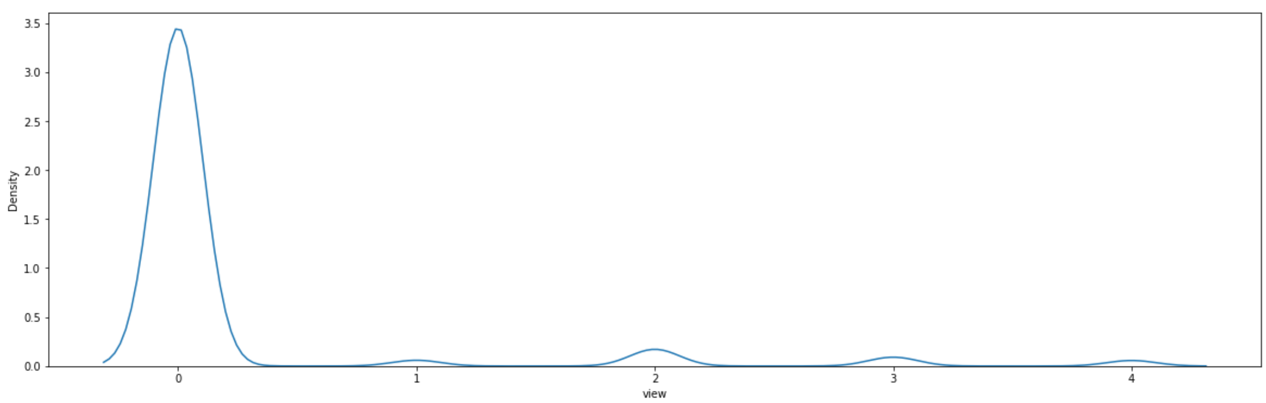


In [165…

```
fig,ax = plt.subplots(figsize=(20,6))
sns.kdeplot(data=df_to_test['view'])
```

Out[165…

```
<AxesSubplot:xlabel='view', ylabel='Density'>
```

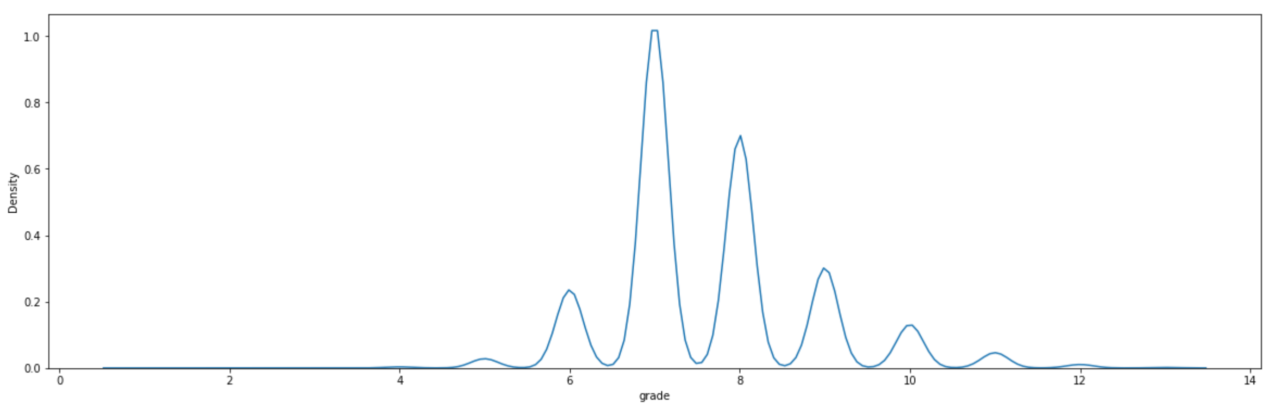


In [166…

```
fig,ax = plt.subplots(figsize=(20,6))
sns.kdeplot(data=df_to_test['grade'])
```

Out[166…

```
<AxesSubplot:xlabel='grade', ylabel='Density'>
```



In [167…

```
#Regression using unscaled data

X_train,X_test,y_train,y_test = train_test_split(df_to_test,df['price'],test_size=0.3,r
```

```
unscaled_Regressor = SGDRegressor(loss='huber')
unscaled_Regressor.fit(X_train,y_train)
print(unscaled_Regressor.score(X_test,y_test))
print(unscaled_Regressor.predict([[2000,2,7]]))
```

```
0.469471723230939
[484767.86170374]
C:\Users\axlcr\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\base.py:
450: UserWarning: X does not have valid feature names, but SGDRegressor was fitted with
feature names
  warnings.warn(
```

What are some formulas that we can use to Scale these features?

Min-Max Normalization (Scaling)

the equation is as follows

$$x^{'} = \frac{x - min(x)}{max(x) - min(x)}$$

The idea behind this is that we try to get the feature into a range between -1,+1 or 0,1. Remember, there is infinite space between 2 numbers in a range, so between -1 and +1 is plenty. If the range is too large, than our scaling hasn't done much to help us.

Why would we take this approach in normalizing the data? The reason is beacuse we want to preserve distance, while not necessarily changing the distribution. We are not changing the shape of our distribution when we make this type of change. This is great when we have algorithms that do not assume distribution, but DO care about distance, such as K Nearest Neighbhors.

In [170…

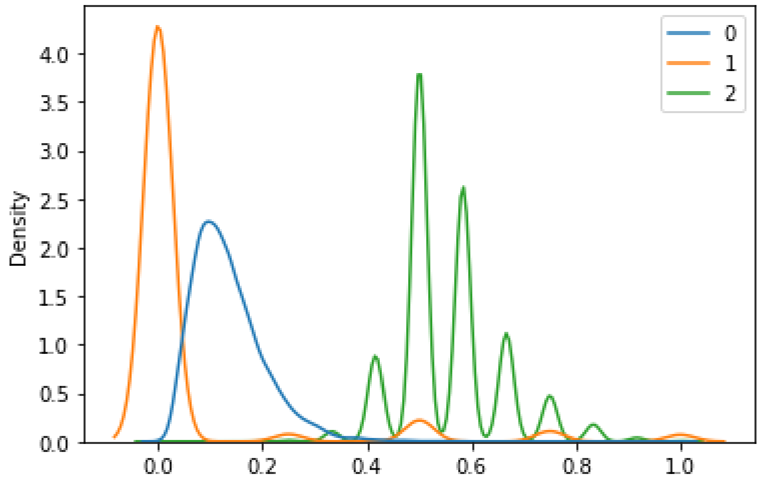
```
#Fit the scaler
scaled_minmax = MinMaxScaler().fit(df_to_test)


#Transform using the scaler
minmax_scaled_data = scaled_minmax.transform(df_to_test)


#Split the Data
X_train,X_test,y_train,y_test = train_test_split(minmax_scaled_data,df['price'],test_si
```

In [171…

```
sns.kdeplot(data=X_train)
```

Out[171…

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



In [129…

```
# Regression using minmax normalized data

minmax_Regressor = SGDRegressor(max_iter=1000)
minmax_Regressor.fit(X_train,y_train)

score = minmax_Regressor.score(X_train,y_train)
test = scaled_minmax.transform([[2000,2,7]])
print(minmax_Regressor.predict(test))
print(score)
```

```
[642457.06236371]
0.5734566778638793
```

```
C:\Users\axlcr\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\base.py:
450: UserWarning: X does not have valid feature names, but MinMaxScaler was fitted with
feature names
  warnings.warn(
```

Z-Score Normalization (Also called Standardization)

The equation is as follows

$X' = \frac{X-\mu}{\sigma}$

This is a type of feature scaling that has two goals. Not only does it aim to move values into a smaller range, but it seeks to fit them across a normal distribution. This is extremely helpful when our algorithm expects the data to be normally distributed. This also retains the importance of outliers. This is particularly useful for SVM, Logistic Regression, and Neural Networks.

In [173…

```
from sklearn.preprocessing import StandardScaler


#Fit the scaler
scaled_std = StandardScaler().fit(df_to_test)


#Transform using the scaler
std_scaled_data = scaled_std.transform(df_to_test)
```

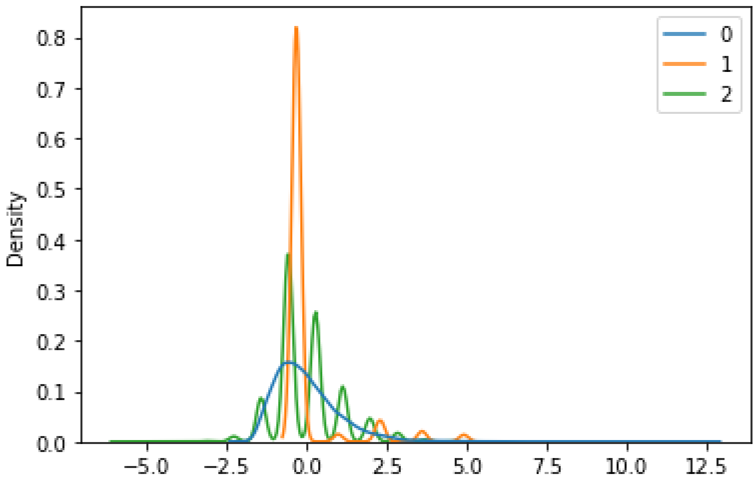
```
#Split the Data
X_train,X_test,y_train,y_test = train_test_split(std_scaled_data,df['price'],test_size=
```

In [174...

```
sns.kdeplot(data=X_train)
```

Out[174...

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



In [182...

```
# Regression using standardized  data

std_Regressor = SGDRegressor(max_iter=1000)
std_Regressor.fit(X_train,y_train)
std_Regressor.score(X_train,y_train)


score = std_Regressor.score(X_train,y_train)
test = scaled_std.transform([[2000,2,7]])
print(std_Regressor.predict(test))
print(score)

std_Regressor.score(X_test,y_test)
```

```
[577915.02914217]
0.5757386804105766
C:\Users\axlcr\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\base.py:
450: UserWarning: X does not have valid feature names, but StandardScaler was fitted wit
h feature names
  warnings.warn(
```

Out[182...

```
0.5641642734550231
```

Robust Scaler

This is great when we want to scale while being robust to outliers. Instead of using Standard Dev as the means for scaling, we use median and IQR.

In [175...

```
from sklearn.preprocessing import RobustScaler
```

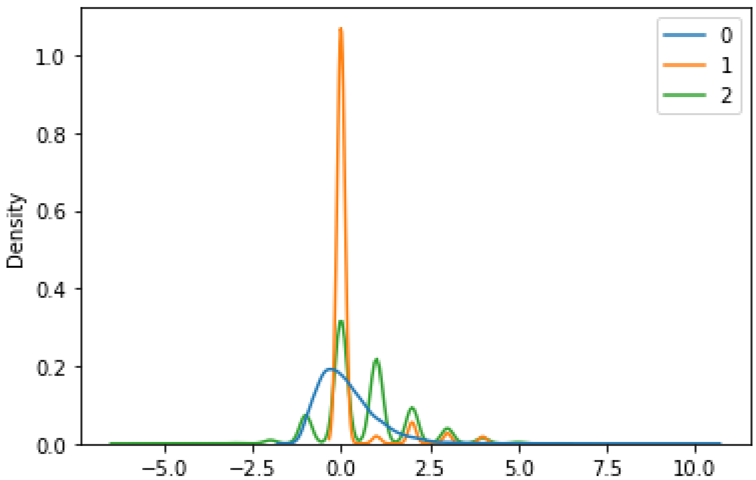
```
#Fit the scaler
scaled_robust = RobustScaler().fit(df_to_test)


#Transform using the scaler
scaled_robust_data = scaled_robust.transform(df_to_test)

#Split the Data
X_train,X_test,y_train,y_test = train_test_split(scaled_robust_data,df['price'],test_si
```

In [176…

```
sns.kdeplot(data=X_train)
```

Out[176…

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



In [159…

```
robust_Regressor = SGDRegressor(max_iter=1000)
robust_Regressor.fit(X_train,y_train)
robust_Regressor.score(X_train,y_train)

score = robust_Regressor.score(X_train,y_train)
test = scaled_robust.transform([[2000,2,7]])
print(robust_Regressor.predict(test))
print(score)
```

```
[627190.84768228]
0.5757703772744562
```

```
C:\Users\axlcr\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\base.py:
450: UserWarning: X does not have valid feature names, but RobustScaler was fitted with
feature names
  warnings.warn(
```