In [1]:
```python
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [3]:
```python
import numpy as np
import pandas as pd
import os
import pickle
```

Objective 0: Data Load, EDA, Transformations

In [4]:
```python
amesDF=pd.read_pickle('amesDF.pickle')
```

In [5]:
```python
amesDF.dtypes.value_counts()
```

Out[5]:
```
object     40
int64      32
float64     2
dtype: int64
```

In [6]:
```python
amesNumDF=amesDF.select_dtypes(include=np.number).astype('float')
```

In [7]:
```python
amesNumDF.columns
```

Out[7]:
```
Index(['Lot_Frontage', 'Lot_Area', 'Year_Built', 'Year_Remod_Add',
       'Mas_Vnr_Area', 'BsmtFin_SF_1', 'BsmtFin_SF_2', 'Bsmt_Unf_SF',
       'Total_Bsmt_SF', 'First_Flr_SF', 'Second_Flr_SF', 'Gr_Liv_Area',
       'Bsmt_Full_Bath', 'Bsmt_Half_Bath', 'Full_Bath', 'Half_Bath',
       'Bedroom_AbvGr', 'Kitchen_AbvGr', 'TotRms_AbvGrd', 'Fireplaces',
       'Garage_Cars', 'Garage_Area', 'Wood_Deck_SF', 'Open_Porch_SF',
       'Enclosed_Porch', 'Three_season_porch', 'Screen_Porch', 'Pool_Area',
       'Misc_Val', 'Mo_Sold', 'Year_Sold', 'Sale_Price', 'Longitude',
       'Latitude'],
      dtype='object')
```

In [8]:
```python
amesNumDF.head()
```

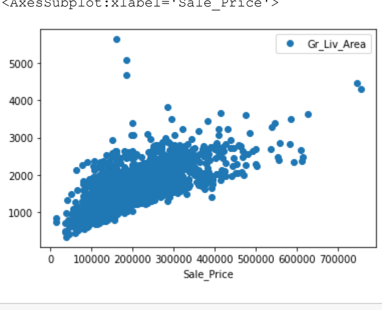
Out[8]:

	Lot_Frontage	Lot_Area	Year_Built	Year_Remod_Add	Mas_Vnr_Area	BsmtFin_SF_1	BsmtFin_SF_2	Bsmt_Unf_SF	Total_Bsmt_SF	Fir
0	141.0	31770.0	1960.0	1960.0	112.0	2.0	0.0	441.0	1080.0	
1	80.0	11622.0	1961.0	1961.0	0.0	6.0	144.0	270.0	882.0	
2	81.0	14267.0	1958.0	1958.0	108.0	1.0	0.0	406.0	1329.0	
3	93.0	11160.0	1968.0	1968.0	0.0	1.0	0.0	1045.0	2110.0	
4	74.0	13830.0	1997.0	1998.0	0.0	3.0	0.0	137.0	928.0	

5 rows × 34 columns

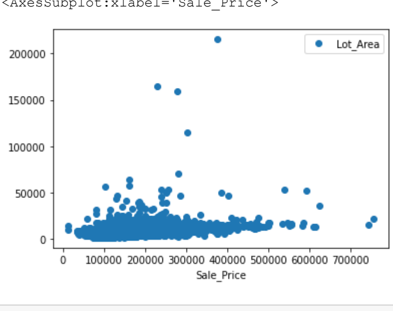
In [10]:
```python
amesNumDF.plot(x='Sale_Price', y='Gr_Liv_Area', style='o')
```

Out[10]: <AxesSubplot:xlabel='Sale_Price'>



In [11]:
```python
amesNumDF.plot(x='Sale_Price', y='Lot_Area', style='o')
```

Out[11]: <AxesSubplot:xlabel='Sale_Price'>

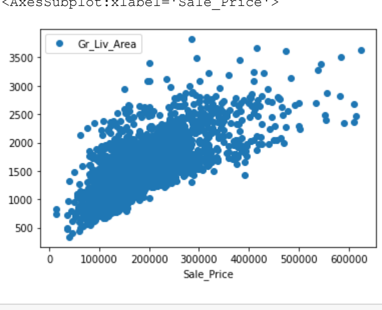


In [12]:
```python
amesNumDF.drop(amesNumDF[(amesNumDF.Gr_Liv_Area > 4000)].index, inplace=True)
```

remove house sizes with more than 4000 sf

In [13]:
```python
amesNumDF.plot(x='Sale_Price', y='Gr_Liv_Area', style='o')
```

Out[13]: <AxesSubplot:xlabel='Sale_Price'>

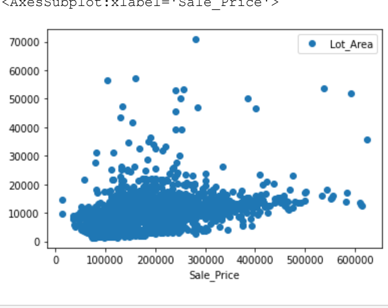


In [14]:
```python
amesNumDF.drop(amesNumDF[(amesNumDF.Lot_Area > 100000)].index, inplace=True)
```

remove lot sizes with more than 100000 sf

In [15]:
```python
amesNumDF.plot(x='Sale_Price', y='Lot_Area', style='o')
```

Out[15]: <AxesSubplot:xlabel='Sale_Price'>



In [16]:
```python
amesNumDF.to_pickle('amesNumDFclean.pickle')
```

In [18]:
```python
import pandas_profiling
from pandas_profiling import ProfileReport
amesProfile=ProfileReport(amesNumDF,'ames data',explorative=True)
```

In [20]:
```python
amesProfile.to_notebook_iframe()
```

Pandas Profiling Report   Overview   Variables

Overview

Overview   Warnings 17   Reproduction

Dataset statistics	
Number of variables	35
Number of observations	2921
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	798.8 KiB
Average record size in memory	280.0 B

Variable types	
Numeric	28
Categorical	7

Variables

df_index
Real number ($\mathbb{R}_{\geq 0}$)

Distinct	2921
Distinct (%)	100.0%

Mean	1463.501198
Minimum	0

Of the 30 or so features, I will not be transforming any.

In [ ]:

Onjective 1: K-Means Clustering

In [21]:
```python
import numpy as np
import pandas as pd
import os
from sklearn.cluster import KMeans, MiniBatchKMeans
%matplotlib inline
import matplotlib.pyplot as plt
import scikitplot as skplt
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_harabasz_score
```

In [22]:
```python
AmesSelDF=pd.read_pickle('amesSelDF.pickle')
AmesSelDF.columns
```

Out[22]:
```
Index(['Lot_Frontage', 'Lot_Area', 'Mas_Vnr_Area', 'Bsmt_Unf_SF',
       'Total_Bsmt_SF', 'First_Flr_SF', 'Second_Flr_SF', 'Gr_Liv_Area',
       'Bedroom_AbvGr', 'Kitchen_AbvGr', 'TotRms_AbvGrd', 'Fireplaces',
       'Garage_Area', 'Wood_Deck_SF', 'Open_Porch_SF', 'Sale_Price'],
      dtype='object')
```

In [23]:
```python
AmesSelDF.describe()
```

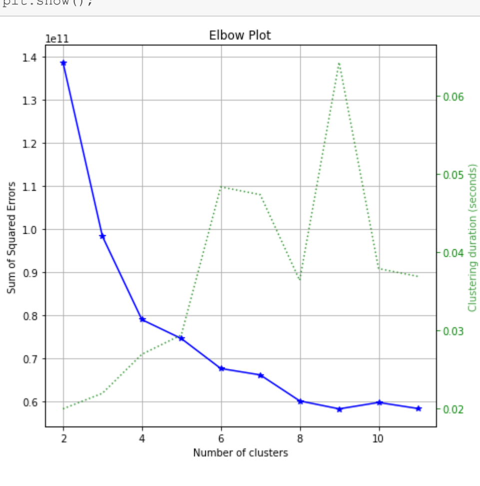
Out[23]:

	Lot_Frontage	Lot_Area	Mas_Vnr_Area	Bsmt_Unf_SF	Total_Bsmt_SF	First_Flr_SF	Second_Flr_SF	Gr_Liv_Area	Bedroom_A
count	2930.000000	2930.000000	2930.000000	2930.000000	2930.000000	2930.000000	2930.000000	2930.000000	2930.00
mean	57.647782	10147.921843	101.096928	559.071672	1051.255631	1159.557679	335.455973	1499.690444	2.85
std	33.499441	7880.017759	178.634545	439.540571	440.968018	391.890885	428.395715	505.508887	0.82
min	0.000000	1300.000000	0.000000	0.000000	0.000000	334.000000	0.000000	334.000000	0.00
25%	43.000000	7440.250000	0.000000	219.000000	793.000000	876.250000	0.000000	1126.000000	2.00
50%	63.000000	9436.500000	0.000000	465.500000	990.000000	1084.000000	0.000000	1442.000000	3.00
75%	78.000000	11555.250000	162.750000	801.750000	1301.500000	1384.000000	703.750000	1742.750000	3.00
max	313.000000	215245.000000	1600.000000	2336.000000	6110.000000	5095.000000	2065.000000	5642.000000	8.00

In [24]:
```python
AmesClusDF=AmesSelDF.loc[:,~(AmesSelDF.columns.isin(['Sale_Price']))].astype('float32')
AmesClusDF.dtypes
X=AmesClusDF.to_numpy(copy=True)
```

Out[24]:
```
Lot_Frontage     float32
Lot_Area         float32
Mas_Vnr_Area     float32
Bsmt_Unf_SF      float32
Total_Bsmt_SF    float32
First_Flr_SF     float32
Second_Flr_SF    float32
Gr_Liv_Area      float32
Bedroom_AbvGr    float32
Kitchen_AbvGr    float32
TotRms_AbvGrd    float32
Fireplaces       float32
Garage_Area      float32
Wood_Deck_SF     float32
Open_Porch_SF    float32
dtype: object
```

In [34]:
```python
kmMB=MiniBatchKMeans(random_state=88)
skplt.cluster.plot_elbow_curve(kmMB,X,n_jobs=-1,cluster_ranges=range(2,12),
                               figsize=(7,7))
plt.show();
```



From the elbow plot above, it appears the best number of cluters is 4

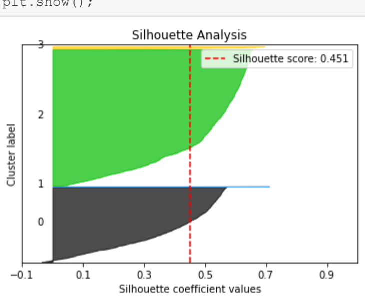
In [42]:
```python
km4=KMeans(n_clusters=4,random_state=33).fit(X)
label=km4.predict(X)
pd.Series(label).value_counts()
```

Out[42]:
```
0    1871
2    1022
3      33
1       4
dtype: int64
```

In [36]:
```python
print(f'CH Score: {calinski_harabasz_score(X,label)}')
print(f'DB Score: {davies_bouldin_score(X,label)}')
print(f'Sihouette Score: {silhouette_score(X,label)}')
```

```
CH Score: 4835.062230207577
DB Score: 0.5722008624652534
Sihouette Score: 0.4511692225933075
```

In [37]:
```python
skplt.metrics.plot_silhouette(X, label)
plt.show();
```



If I decide to exclude any features from my cluster analysis, my criteria would be to make sure each feature included has no missing data or outliers

Objective 2: Rescaling Feature Data

In [48]:
```python
import numpy as np
import pandas as pd
import os
import joblib
import pickle
import pickleshare
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import math
```

In [49]:
```python
os.listdir()
```

Out[49]:
```
['.ipynb_checkpoints',
 'ames-data-info.zip',
 'amesDF.pickle',
 'amesNumDFclean.pickle',
 'amesSelDF.pickle',
 'DataDocumentation.txt',
 'MSDS422_Dantinne_Alex_Assignment 2.ipynb']
```

In [50]:
```python
amesDF=pd.read_pickle('amesSelDF.pickle')
amesDF.dtypes
```

Out[50]:
```
Lot_Frontage     int64
Lot_Area         int64
Mas_Vnr_Area     int64
Bsmt_Unf_SF      int64
Total_Bsmt_SF    int64
First_Flr_SF     int64
Second_Flr_SF    int64
Gr_Liv_Area      int64
Bedroom_AbvGr    int64
Kitchen_AbvGr    int64
TotRms_AbvGrd    int64
Fireplaces       int64
Garage_Area      int64
Wood_Deck_SF     int64
Open_Porch_SF    int64
Sale_Price       int64
dtype: object
```

In [51]:
```python
amesDF2=amesDF.astype('float32')
amesDF2.dtypes
```

Out[51]:
```
Lot_Frontage     float32
Lot_Area         float32
Mas_Vnr_Area     float32
Bsmt_Unf_SF      float32
Total_Bsmt_SF    float32
First_Flr_SF     float32
Second_Flr_SF    float32
Gr_Liv_Area      float32
Bedroom_AbvGr    float32
Kitchen_AbvGr    float32
TotRms_AbvGrd    float32
Fireplaces       float32
Garage_Area      float32
Wood_Deck_SF     float32
Open_Porch_SF    float32
Sale_Price       float32
dtype: object
```

In [52]:
```python
from sklearn.model_selection import train_test_split
X=amesDF.loc[:,~(amesDF.columns.isin(['Sale_Price']))].to_numpy(copy=True)
y=amesDF.Sale_Price.to_numpy(copy=True)
X.shape
y.shape
```

Out[52]: (2930, 15)

Out[52]: (2930,)

In [53]:
```python
trainX,testX,trainy, testy = train_test_split(X,y,test_size=0.15,random_state=33)
```

In [54]:
```python
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler.fit(trainX)
trainXS=scaler.transform(trainX)
testXS=scaler.transform(testX)
```

Out[54]: MinMaxScaler()

In [55]:
```python
pd.DataFrame(trainXS).describe()
pd.DataFrame(testXS).describe()
```

Out[55]:

	0	1	2	3	4	5	6	7	8	
count	2490.000000	2490.000000	2490.000000	2490.000000	2490.000000	2490.000000	2490.000000	2490.000000	2490.000000	2490.00000
mean	0.183168	0.054125	0.072252	0.237758	0.170941	0.173060	0.176757	0.218493	0.355773	0.3489
std	0.107049	0.044039	0.127108	0.188073	0.072825	0.082523	0.226640	0.094019	0.103490	0.0730
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	0.137380	0.037953	0.000000	0.092038	0.127823	0.113422	0.000000	0.148125	0.250000	0.3333
50%	0.201278	0.050064	0.000000	0.199486	0.161702	0.157635	0.000000	0.208082	0.375000	0.3333
75%	0.249201	0.063051	0.117562	0.341610	0.212439	0.220122	0.373665	0.263753	0.375000	0.3333
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0000

Out[55]:

	0	1	2	3	4	5	6	7	8	9	
count	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.0000
mean	0.189893	0.054373	0.079664	0.248216	0.178356	0.175322	0.193005	0.225935	0.362500	0.343182	0.3496
std	0.106842	0.067295	0.143081	0.188616	0.068090	0.081179	0.240741	0.101725	0.103264	0.060822	0.1223
min	0.000000	0.001077	0.000000	0.000000	0.000000	0.021844	0.000000	0.019593	0.000000	0.000000	0.0769
25%	0.140575	0.036609	0.000000	0.105415	0.136170	0.115522	0.000000	0.152176	0.250000	0.333333	0.2307
50%	0.202875	0.048641	0.000000	0.197132	0.163666	0.156585	0.000000	0.212980	0.375000	0.333333	0.3076
75%	0.249201	0.061485	0.125181	0.355522	0.217594	0.224375	0.391960	0.273267	0.375000	0.333333	0.3846
max	0.581470	1.309654	1.161103	0.907962	0.513584	0.588952	1.103098	0.818011	0.750000	0.666667	0.7692

Objective 3: PCA

In [69]:
```python
from sklearn.decomposition import PCA
pca75=PCA(n_components=0.75,svd_solver='full')
pca75.fit(trainX)
trainXPCA=pca75.transform(trainX)
testXPCA=pca75.transform(testX)
```

Out[69]: PCA(n_components=0.75, svd_solver='full')

In [70]:
```python
trainXPCA.shape
testXPCA.shape
```

Out[70]: (2490, 1)

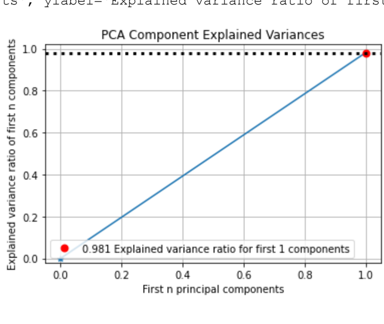
Out[70]: (440, 1)

In [71]:
```python
print(f'prop. of variance explained: {pca75.explained_variance_ratio_}')
```

```
prop. of variance explained: [0.98086147]
```

In [72]:
```python
import scikitplot as skplt
skplt.decomposition.plot_pca_component_variance(pca75)
plt.show()
```

Out[72]: <AxesSubplot:title={'center':'PCA Component Explained Variances'}, xlabel='First n principal components', ylabel='Explained variance ratio of first n components'>



Objective 4: Training RF regressors using Original and Rescaled Features

In [73]:
```python
import numpy as np
import pandas as pd
import os
from pickleshare import *
import re
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
```

In [74]:
```python
aDF=pd.read_pickle('amesNumDFclean.pickle')
```

In [75]:
```python
print(aDF.describe())
```

```
       Lot_Frontage      Lot_Area   Year_Built  Year_Remod_Add  Mas_Vnr_Area  \
count   2921.000000   2921.000000  2921.000000     2921.000000   2921.000000   
mean      57.488189   9893.504964  1971.311879     1984.233824     99.773708   
std       33.002562   5160.828715    30.261729       20.863510    175.470332   
min        0.000000   1300.000000  1872.000000     1950.000000      0.000000   
25%       43.000000   7436.000000  1954.000000     1965.000000      0.000000   
50%       63.000000   9405.000000  1973.000000     1993.000000      0.000000   
75%       78.000000  11500.000000  2001.000000     2004.000000    162.000000   
max      313.000000  70761.000000  2010.000000     2010.000000   1600.000000   

       BsmtFin_SF_1  BsmtFin_SF_2  Bsmt_Unf_SF  Total_Bsmt_SF  First_Flr_SF  \
count   2921.000000   2921.000000  2921.000000    2921.000000   2921.000000   
mean       4.181445     49.527559   559.057857    1045.626155   1154.673400   
std        2.234370    168.774481   439.777661     420.995545    376.157585   
min        0.000000      0.000000     0.000000       0.000000    334.000000   
25%        3.000000      0.000000   219.000000     792.000000    876.000000   
50%        3.000000      0.000000   465.000000     988.000000   1082.000000   
75%        7.000000      0.000000   801.000000    1297.000000   1382.000000   
max        7.000000   1526.000000  2336.000000    3206.000000   3820.000000   

       ...  Enclosed_Porch  Three_season_porch  Screen_Porch    Pool_Area  \
count  ...     2921.000000         2921.000000   2921.000000  2921.000000   
mean   ...       23.082506            2.600479     16.051352     1.895926   
std    ...       64.225083           25.179634     56.166693    32.986768   
min    ...        0.000000            0.000000      0.000000     0.000000   
25%    ...        0.000000            0.000000      0.000000     0.000000   
50%    ...        0.000000            0.000000      0.000000     0.000000   
75%    ...        0.000000            0.000000      0.000000     0.000000   
max    ...     1012.000000          508.000000    576.000000   800.000000   

           Misc_Val      Mo_Sold    Year_Sold     Sale_Price    Longitude  \
count   2921.000000  2921.000000  2921.000000    2921.000000  2921.000000   
mean      44.560425     6.216364  2007.791510  180253.648066   -93.642820   
std      472.329137     2.713188     1.317952   78468.221621     0.025695   
min        0.000000     1.000000  2006.000000   12789.000000   -93.693153   
25%        0.000000     4.000000  2007.000000  129400.000000   -93.660168   
50%        0.000000     6.000000  2008.000000  160000.000000   -93.641781   
75%        0.000000     8.000000  2009.000000  213133.000000   -93.622025   
max    15500.000000    12.000000  2010.000000  625000.000000   -93.577427   

          Latitude  
count  2921.000000  
mean     42.034516  
std       0.018400  
min      41.986498  
25%      42.022089  
50%      42.034673  
75%      42.049860  
max      42.063388  

[8 rows x 34 columns]
```

In [76]:
```python
aDF.isna().sum().sum()
```

Out[76]: 0

In [77]:
```python
aDF2=aDF[aDF.Gr_Liv_Area<4000].copy()
aDF2.shape
```

Out[77]: (2921, 34)

In [78]:
```python
((aDF2.Year_Built-aDF2.Year_Sold)>0.00).value_counts()
```

Out[78]:
```
False    2921
dtype: int64
```

No homes sold before they are build

In [79]:
```python
((aDF2.Year_Built-aDF2.Year_Remod_Add)>0.00).value_counts()
```

Out[79]:
```
False    2920
True        1
dtype: int64
```

1 home remodeeled before build; to be removed

In [80]:
```python
((aDF2.Year_Remod_Add-aDF2.Year_Sold)>0.00).value_counts()
```

Out[80]:
```
False    2920
True        1
dtype: int64
```

1 home remodeled after sold; to be removed

In [81]:
```python
aDF3=aDF2.loc[~((aDF2.Year_Built-aDF2.Year_Remod_Add)>0.00),:].copy()
aDF3.shape
```

Out[81]: (2920, 34)

In [82]:
```python
aDF4=aDF3.loc[~((aDF2.Year_Remod_Add-aDF2.Year_Sold)>0.00),:].copy()
aDF4.shape
```

Out[82]: (2919, 34)

In [83]:
```python
moDummies=pd.get_dummies(aDF4.Mo_Sold,prefix="sales_mo").astype(int)
moDummies.columns
```

Out[83]:
```
Index(['sales_mo_1.0', 'sales_mo_2.0', 'sales_mo_3.0', 'sales_mo_4.0',
       'sales_mo_5.0', 'sales_mo_6.0', 'sales_mo_7.0', 'sales_mo_8.0',
       'sales_mo_9.0', 'sales_mo_10.0', 'sales_mo_11.0', 'sales_mo_12.0'],
      dtype='object')
```

In [84]:
```python
aDF5=aDF4.loc[:,~(aDF4.columns.isin(['Sale_Month']))]
aDF6=pd.concat([aDF5,moDummies],axis=1,ignore_index=False)
aDF6.columns
```

Out[84]:
```
Index(['Lot_Frontage', 'Lot_Area', 'Year_Built', 'Year_Remod_Add',
       'Mas_Vnr_Area', 'BsmtFin_SF_1', 'BsmtFin_SF_2', 'Bsmt_Unf_SF',
       'Total_Bsmt_SF', 'First_Flr_SF', 'Second_Flr_SF', 'Gr_Liv_Area',
       'Bsmt_Full_Bath', 'Bsmt_Half_Bath', 'Full_Bath', 'Half_Bath',
       'Bedroom_AbvGr', 'Kitchen_AbvGr', 'TotRms_AbvGrd', 'Fireplaces',
       'Garage_Cars', 'Garage_Area', 'Wood_Deck_SF', 'Open_Porch_SF',
       'Enclosed_Porch', 'Three_season_porch', 'Screen_Porch', 'Pool_Area',
       'Misc_Val', 'Mo_Sold', 'Year_Sold', 'Sale_Price', 'Longitude',
       'Latitude', 'sales_mo_1.0', 'sales_mo_2.0', 'sales_mo_3.0',
       'sales_mo_4.0', 'sales_mo_5.0', 'sales_mo_6.0', 'sales_mo_7.0',
       'sales_mo_8.0', 'sales_mo_9.0', 'sales_mo_10.0', 'sales_mo_11.0',
       'sales_mo_12.0'],
      dtype='object')
```

In [85]:
```python
aDF6.dtypes.value_counts()
```

Out[85]:
```
float64    34
int32      12
dtype: int64
```

In [86]:
```python
years=['Year_Built','Year_Remod_Add','Year_Sold']
yearsDF=aDF6.loc[:,years]
aDF7=aDF6.loc[:,~(aDF6.columns.isin(years))]
```

In [87]:
```python
yearsDF=yearsDF.transform(lambda x: 2010-x)
yearsDF.head()
```

Out[87]:

	Year_Built	Year_Remod_Add	Year_Sold
0	50.0	50.0	0.0
1	49.0	49.0	0.0
2	52.0	52.0	0.0
3	42.0	42.0	0.0
4	13.0	12.0	0.0

In [88]:
```python
aDF8=pd.concat([aDF7,yearsDF],axis=1)
aDF8.columns
```

Out[88]:
```
Index(['Lot_Frontage', 'Lot_Area', 'Mas_Vnr_Area', 'BsmtFin_SF_1',
       'BsmtFin_SF_2', 'Bsmt_Unf_SF', 'Total_Bsmt_SF', 'First_Flr_SF',
       'Second_Flr_SF', 'Gr_Liv_Area', 'Bsmt_Full_Bath', 'Bsmt_Half_Bath',
       'Full_Bath', 'Half_Bath', 'Bedroom_AbvGr', 'Kitchen_AbvGr',
       'TotRms_AbvGrd', 'Fireplaces', 'Garage_Cars', 'Garage_Area',
       'Wood_Deck_SF', 'Open_Porch_SF', 'Enclosed_Porch', 'Three_season_porch',
       'Screen_Porch', 'Pool_Area', 'Misc_Val', 'Mo_Sold', 'Sale_Price',
       'Longitude', 'Latitude', 'sales_mo_1.0', 'sales_mo_2.0', 'sales_mo_3.0',
       'sales_mo_4.0', 'sales_mo_5.0', 'sales_mo_6.0', 'sales_mo_7.0',
       'sales_mo_8.0', 'sales_mo_9.0', 'sales_mo_10.0', 'sales_mo_11.0',
       'sales_mo_12.0', 'Year_Built', 'Year_Remod_Add', 'Year_Sold'],
      dtype='object')
```

In [89]:
```python
y=aDF8.Sale_Price.to_numpy(copy=True)
SFfeats=aDF8.columns.to_list()[:10]
salesFeats=list(filter(lambda s: s.startswith('sales_'),
                       aDF8.columns.to_list()))

RFfeatures=SFfeats+salesFeats
X=aDF8.loc[:,RFfeatures].to_numpy(copy=True)
X.shape
y.shape
```

Out[89]: (2919, 22)

Out[89]: (2919,)

In [90]:
```python
len(RFfeatures)
X.shape
```

Out[90]: 22

Out[90]: (2919, 22)

In [91]:
```python
XTrain, XTest, yTrain, yTest = train_test_split(X,y, random_state=11)
trainTestData=[XTrain,XTest, yTrain, yTest]
XTrain.shape, XTest.shape
```

Out[91]: ((2189, 22), (730, 22))

In [92]:
```python
RFdb=PickleShareDB('RF.pickleDB')
RFdb.clear()
```

In [93]:
```python
RFdb['trainTestData']=trainTestData
```

In [94]:
```python
RFregr=RandomForestRegressor(max_features=None,oob_score=True,n_jobs=-1,
                             random_state=11)
RFregr.fit(XTrain,yTrain)
```

Out[94]:
```
RandomForestRegressor(max_features=None, n_jobs=-1, oob_score=True,
                      random_state=11)
```

In [95]:
```python
print(f'RF R Squared, Training: {RFregr.score(XTrain,yTrain):5.3f}')
print(f'RF R Squared, OOB: {RFregr.oob_score_:5.3f}')
```

```
RF R Squared, Training: 0.974
RF R Squared, OOB: 0.807
```

In [96]:
```python
predTesty=RFregr.predict(XTest)
print(f'Test Data R Squared: {r2_score(yTest,predTesty):4.3f}')
```

```
Test Data R Squared: 0.795
```

In [97]:
```python
RFregr.feature_importances_
```

Out[97]:
```
array([0.02795252, 0.04094854, 0.07355294, 0.0446128 , 0.00474938,
       0.03522424, 0.32445487, 0.02901625, 0.06714273, 0.33030169,
       0.00217363, 0.00125253, 0.00186306, 0.00136103, 0.0015559 ,
       0.00221504, 0.00230911, 0.00168549, 0.00170402, 0.00217387,
       0.00232162, 0.00142873])
```

In [98]:
```python
RFFeatImpDF=pd.DataFrame({'feature':RFfeatures,
                          'importance':RFregr.feature_importances_})
```

In [99]:
```python
print('Feature importances')
RFFeatImpDF.sort_values('importance',ascending=False)
```

```
Feature importances
```

Out[99]:

	feature	importance
9	Gr_Liv_Area	0.330302
6	Total_Bsmt_SF	0.324455
2	Mas_Vnr_Area	0.073553
8	Second_Flr_SF	0.067143
3	BsmtFin_SF_1	0.044613
1	Lot_Area	0.040949
5	Bsmt_Unf_SF	0.035224
7	First_Flr_SF	0.029016
0	Lot_Frontage	0.027953
4	BsmtFin_SF_2	0.004749
20	sales_mo_11.0	0.002322
16	sales_mo_7.0	0.002309
15	sales_mo_6.0	0.002215
19	sales_mo_10.0	0.002174
10	sales_mo_1.0	0.002174
12	sales_mo_3.0	0.001863
18	sales_mo_9.0	0.001704
17	sales_mo_8.0	0.001685
14	sales_mo_5.0	0.001556
21	sales_mo_12.0	0.001429
13	sales_mo_4.0	0.001361
11	sales_mo_2.0	0.001253

In [100]:
```python
RFregr2=RandomForestRegressor(max_features='log2',oob_score=True,n_jobs=-1,
                              random_state=11)
RFregr2.fit(XTrain,yTrain)
```

Out[100]:
```
RandomForestRegressor(max_features='log2', n_jobs=-1, oob_score=True,
                      random_state=11)
```

In [101]:
```python
print(f'RF R Squared, Training: {RFregr2.score(XTrain,yTrain):4.3f}')
print(f'RF R Squared, OOB: {RFregr2.oob_score_:4.3f}')
```

```
RF R Squared, Training: 0.972
RF R Squared, OOB: 0.792
```

In [102]:
```python
predTesty=RFregr2.predict(XTest)
print(f'Test Data R Squared: {r2_score(yTest,predTesty):4.3f}')
```

```
Test Data R Squared: 0.797
```

In [103]:
```python
RFregr2.feature_importances_
```

Out[103]:
```
array([0.0535019 , 0.06997874, 0.08884824, 0.05215604, 0.00931006,
       0.04323747, 0.21488312, 0.1266394 , 0.08520959, 0.22201245,
       0.00316596, 0.00244874, 0.00289621, 0.00208326, 0.00281202,
       0.00362867, 0.00356433, 0.00295096, 0.00280518, 0.00280413,
       0.00295332, 0.00211018])
```

In [104]:
```python
RFFeatImpDF=pd.DataFrame({'feature':RFfeatures,
                          'importance':RFregr2.feature_importances_})
```

```
In [105]: print('Feature importances')
          RFFeatImpDF.sort_values('importance',ascending=False)
```

Feature importances

Out[105]:

	feature	importance
9	Gr_Liv_Area	0.222012
6	Total_Bsmt_SF	0.214883
7	First_Flr_SF	0.126639
2	Mas_Vnr_Area	0.088848
8	Second_Flr_SF	0.085210
1	Lot_Area	0.069979
0	Lot_Frontage	0.053502
3	BsmtFin_SF_1	0.052156
5	Bsmt_Unf_SF	0.043237
4	BsmtFin_SF_2	0.009310
15	sales_mo_6.0	0.003629
16	sales_mo_7.0	0.003564
10	sales_mo_1.0	0.003166
20	sales_mo_11.0	0.002953
17	sales_mo_8.0	0.002951
12	sales_mo_3.0	0.002896
14	sales_mo_5.0	0.002812
18	sales_mo_9.0	0.002805
19	sales_mo_10.0	0.002804
11	sales_mo_2.0	0.002449
21	sales_mo_12.0	0.002110
13	sales_mo_4.0	0.002083

Between the two, the R squared of the training set only lowered by 0.002 when a log2 was used as max_features.
The order of importance of some features changed once log 2 was added and the top two in both, gr_liv_area & total_bsmt_sf, were the most important features

Objective 5: Training Boosted Regression Models Using Original Features And Using Principal Components as Features

```
In [106]: from sklearn.ensemble import AdaBoostRegressor
          from sklearn.datasets import make_regression

In [170]: Adaregr = AdaBoostRegressor(random_state=0, n_estimators=100)
          Adaregr.fit(trainX, trainy)

Out[170]: AdaBoostRegressor(n_estimators=100, random_state=0)

In [171]: print(f'Ada R Squared, Training: {Adaregr.score(trainX,trainy):5.3f}')

          Ada R Squared, Training: 0.794

In [173]: predTesty=Adaregr.predict(testX)
          print(f'Test Data R Squared: {r2_score(testy,predTesty):4.3f}')

          Test Data R Squared: 0.743
```

Ada and test data have similar R Squareds

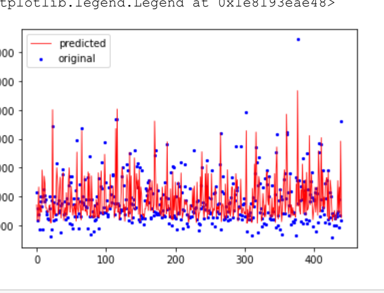
```
In [174]: Adaregr.feature_importances_

Out[174]: array([0.03029882, 0.03472769, 0.08322295, 0.01228706, 0.27660396,
                 0.07581501, 0.12986852, 0.15998078, 0.0051923 , 0.00116811,
                 0.00973074, 0.03035259, 0.07195087, 0.01176578, 0.06703482])

In [204]: AdaFeatImpDF=pd.DataFrame({'importance':Adaregr.feature_importances_})

In [205]: print('Feature importances')
          AdaFeatImpDF.sort_values('importance',ascending=False)
```

Feature importances

Out[205]:

	importance
4	0.276604
7	0.159981
6	0.129869
2	0.083223
5	0.075815
12	0.071951
14	0.067035
1	0.034728
11	0.030353
0	0.030299
3	0.012287
13	0.011766
10	0.009731
8	0.005192
9	0.001168

```
In [133]: from sklearn.ensemble import AdaBoostRegressor
          from sklearn.datasets import load_boston
          from sklearn.model_selection import train_test_split
          from sklearn.model_selection import cross_val_score, KFold
          from sklearn.metrics import mean_squared_error
          import matplotlib.pyplot as plt

In [176]: scores = cross_val_score(Adaregr, trainX, trainy,cv=5)
          print("Mean cross-validataion score: %.2f" % scores.mean())

          Mean cross-validataion score: 0.74

In [177]: kfold = KFold(n_splits=10, shuffle=True)
          kf_cv_scores = cross_val_score(Adaregr, trainX, trainy, cv=kfold )
          print("K-fold CV average score: %.2f" % kf_cv_scores.mean())

          K-fold CV average score: 0.74

In [178]: X_ax = range(len(testy))
          plt.scatter(X_ax, testy, s=5, color="blue", label="original")
          plt.plot(X_ax, predTesty, lw=0.8, color="red", label="predicted")
          plt.legend()
          plt.show()

Out[178]: <matplotlib.collections.PathCollection at 0x1e819a64a88>

Out[178]: [<matplotlib.lines.Line2D at 0x1e819768448>]

Out[178]: <matplotlib.legend.Legend at 0x1e8193eae48>
```



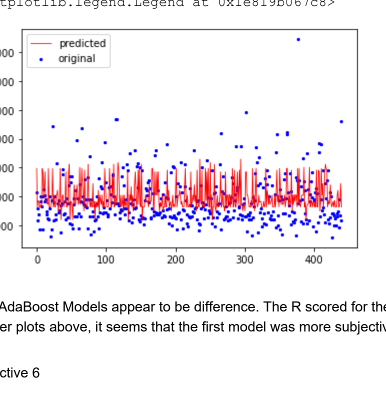

```
In [184]: Ada2regr = AdaBoostRegressor(random_state=0, n_estimators=100)
          Ada2regr.fit(trainXPCA,trainy)

Out[184]: AdaBoostRegressor(n_estimators=100, random_state=0)

In [185]: print(f'Ada R Squared, Training: {Ada2regr.score(trainXPCA,trainy):5.3f}')

          Ada R Squared, Training: 0.142

In [187]: predTesty=Ada2regr.predict(testXPCA)
          print(f'Test Data R Squared: {r2_score(testy,predTesty):4.3f}')

          Test Data R Squared: 0.151

In [194]: Ada2regr.feature_importances_

Out[194]: array([1.])

In [195]: Ada2FeatImpDF=pd.DataFrame({'importance':Ada2regr.feature_importances_})

In [196]: print('Feature importances')
          Ada2FeatImpDF.sort_values('importance',ascending=False)
```

Feature importances

Out[196]:

	importance
0	1.0

```
In [188]: scores = cross_val_score(Ada2regr, trainXPCA, trainy,cv=5)
          print("Mean cross-validataion score: %.2f" % scores.mean())

          Mean cross-validataion score: 0.08

In [189]: kfold = KFold(n_splits=10, shuffle=True)
          kf_cv_scores = cross_val_score(Ada2regr, trainXPCA, trainy, cv=kfold )
          print("K-fold CV average score: %.2f" % kf_cv_scores.mean())

          K-fold CV average score: 0.07

In [191]: X2_ax = range(len(testy))
          plt.scatter(X2_ax, testy, s=5, color="blue", label="original")
          plt.plot(X2_ax, predTesty, lw=0.8, color="red", label="predicted")
          plt.legend()
          plt.show()

Out[191]: <matplotlib.collections.PathCollection at 0x1e819905048>

Out[191]: [<matplotlib.lines.Line2D at 0x1e819947148>]

Out[191]: <matplotlib.legend.Legend at 0x1e819b067c8>
```



The AdaBoost Models appear to be difference. The R scored for the second model is much lower then the frist. From the predicted model scatter plots above, it seems that the first model was more subjective to outliers whole the second model was not.

Objective 6

1. Suppose that the cluster analysis you did indicates that the data fall into a particular number of clusters, like 4, or 7, or whatever. Describe how you could determine the ways that the clusters differ from each other?

A few ways to tell how the clusters differ from each other would to produce and visually inspect an elbow plot, a silhouette plot, or to used the gap statistic method.

1. Why not just apply minmax rescaling, or standardization rescaling, to all of your features at once, rather than rescaling your training and your validation, separately?

Applying minmax or standardization to all the feautures all at once rather than rescaling the training and validation separately would affect the clusters and cause validation issues .

1. The tree-based ensemble methods Random Forest classification and regression models, and boosted regression (e.g., AdaBoost) and classification models, are believed to be better at either reducing variance, or reducing bias. Which is more true for each of these two types of ensemble methods, and why? That is is RF better at reducing variance or reducing bias, and why? Which one are boosted ensemble models, and why?

Random Forest is better at reducing variance becasue the models are in parallel, but this leads to a higher bias. On the other hand, boosted models have a lower bias because the models are in series. Meaning that the first subset is trained, then that trained subset is trianed again and learns from the previous subset of trained data. Since booseted models are in series, this leads to a higher variance.

```
In [ ]:
```