Import Packages

In [131]:

```
%load_ext autoreload
%autoreload 2
import glob
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import folium
import geopandas
import geopy
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.diagnostic import linear_rainbow, het_breuschpagan
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error, make_scorer
from geopy.geocoders import Nominatim
from geopy.extra.rate_limiter import RateLimiter
import folium.plugins as plugins
import math
from math import sin, cos, sqrt, atan2, radians
from haversine import haversine
from itertools import combinations
%matplotlib inline
pd.set_option("display.max_columns", 200)
pd.set_option("display.max_rows", 200)
import sys
import os
path_to_src = os.path.join('..\..', 'src')
sys.path.insert(0, path_to_src)
from useful_functions import *
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

Read in the data

In [132]:

```
files = glob.glob("..\..\data/raw/provided/*.csv")
names = ['lookup', 'parcel', 'resbldg', 'rpsale']
dict_dfs = {}
for x,y in zip(names, files):
    dict_dfs[x] = pd.read_csv(y, dtype=str)
lookup_df = dict_dfs['lookup']
parcel_df = dict_dfs['parcel']
resbldg_df = dict_dfs['resbldg']
rpsale_df = dict_dfs['rpsale']
```

First with an explore of the dataframes one by one and see which columns could potentially be dropped, important to remember this project is going to be focussed on home improvements. For that reason it will be important to keep any columns that could pertain to home improvements, but in order to make the conclusions reached as accurate as possible, it may be necessary to keep some columns not related to home improvements in order to improve the overall accuracy of the model i.e Zip code.

In [133]:

```
parcel_df.head()
```

Out[133]:

	Unnamed: 0	Major	Minor	PropName	PlatName	PlatLot	PlatBlock	Range	Township	Se
0	0	807841	0410		SUMMER RIDGE DIV NO. 02	41		6	25	
1	2	755080	0015		SANDER'S TO GILMAN PK & SALMON BAY	3	1	3	25	
2	3	888600	0135		VASHON GARDENS ADD	21		3	22	
3	6	022603	9181		NaN			3	26	
4	7	229670	0160		ELDORADO NORTH	16		5	26	

In [134]:

```
resbldg_df.head()
```

Out[134]:

	Major	Minor	BldgNbr	NbrLivingUnits	Address	BuildingNumber	Fraction	DirectionPrefi:
0	009800	0720	1	1	27719 SE 26TH WAY 98075	27719		SI
1	009802	0140	1	1	2829 277TH TER SE 98075	2829		
2	009830	0020	1	1	1715 298TH CRESENT SE	1715		
3	009830	0160	1	1	1861 297TH WAY SE 98024	1861		
4	010050	0180	1	1	35410 25TH PL S 98003	35410		

In [135]:

```
rpsale_df.head()
```

Out[135]:

	ExciseTaxNbr	Major	Minor	DocumentDate	SalePrice	RecordingNbr	Volume	Page	Pla
0	2857854	198920	1430	03/28/2017	0	20170410000541			
1	2743355	638580	0110	07/14/2015	190000	20150715002686			
2	2999169	919715	0200	07/08/2019	192000	20190712001080			
3	2841697	894677	0240	12/21/2016	818161	20161228000896			
4	2826129	445872	0260	10/03/2016	0	20161004000511			

In [136]:

```
lookup_df.head()
```

Out[136]:

	LUType	LUItem	LUDescription
0	1	1	LAND ONLY ...
1	1	10	Land with new building ...
2	1	11	Household, single family units ...
3	1	12	Multiple family residence (Residential, 2-4 un...
4	1	13	Multiple family residence (Residential, 5+ uni...

Looks like it will be possible to merge the first 3 dataframes on Major and Minor. The last dataframe is a look up table which contains important information pertaining to various features of the properties.

For each dataframe I will combine the Major and Minor columns, creating an 'id' column, I will then merge the dataframes on this.

In [137]:

```
for df in [parcel_df, resbldg_df, rpsale_df]:
    concat_col(df, 'id', 'Major', 'Minor')
```

Ensure this has worked successfully, print the first 3 entries for each df.

In [138]:

```
for df in [parcel_df, resbldg_df, rpsale_df]:
    print(df[['Major', 'Minor', 'id']][:3])
```

```
    Major Minor          id
0  807841  0410  8078410410
1  755080  0015  7550800015
2  888600  0135  8886000135
    Major Minor          id
0  009800  0720  0098000720
1  009802  0140  0098020140
2  009830  0020  0098300020
    Major Minor          id
0  198920  1430  1989201430
1  638580  0110  6385800110
2  919715  0200  9197150200
```

Time to merge the dataframes and start cleaning it as a whole

In [139]:

```
merge_df = resbldg_df.merge(parcel_df, on='id', how='inner')
total_df = merge_df.merge(rpsale_df, how='left', on='id')
```

Time to explore the merged dataframe

In [140]:

```
total_df.shape
```

Out[140]:

```
(251300, 157)
```

A lot of data! Hopefully some of these rows and columns can be cut down. First, remembering the brief of this project was to use data from 2019 to inform clients of home improvements. I will cut out any house sale that isn't from 2019.

In [141]:

```
total_df.head()
```

Out[141]:

	Major_x	Minor_x	BldgNbr	NbrLivingUnits	Address	BuildingNumber	Fraction	DirectionPre
0	009800	0720	1	1	27719 SE 26TH WAY 98075	27719		
1	009802	0140	1	1	2829 277TH TER SE 98075	2829		
2	009802	0140	1	1	2829 277TH TER SE 98075	2829		
3	009802	0140	1	1	2829 277TH TER SE 98075	2829		
4	009802	0140	1	1	2829 277TH TER SE 98075	2829		

In [142]:

```
total_df['Date'] = pd.to_datetime(total_df['DocumentDate'], format='%m/%d/%Y')
total_df['Date'] = pd.DatetimeIndex(total_df['Date']).year
total_df = total_df[total_df['Date']==2019]
```

In [143]:

```
total_df.shape
```

Out[143]:

```
(43838, 158)
```

Ok number of rows has been drastically reduced from ~251k to ~44k. Next I will filter out by property type, I want to focus on households. Using the look up table information, I know that property type 11 is household, single family unit. Property type 12 may be of interest too but depends on numbers.

In [144]:

```
total_df.PropertyType.value_counts()
```

Out[144]:

```
11    26510
3     13186
2      1612
10     1338
0       322
12      286
1       256
14      137
91       61
5        32
18       23
45       16
13       13
4        12
83        8
59        5
96        4
99        3
6         3
19        3
65        2
94        2
15        1
86        1
23        1
51        1
Name: PropertyType, dtype: int64
```

In [145]:

```
for col in lookup_df.columns[:-1]:
    lookup_df[col] = lookup_df[col].str.strip().astype(int)
lookup(lookup_df, 1)
```

Out[145]:

	LUType	LUItem	LUDescription
0	1	1	LAND ONLY ...
1	1	10	Land with new building ...
2	1	11	Household, single family units ...
3	1	12	Multiple family residence (Residential, 2-4 un...
4	1	13	Multiple family residence (Residential, 5+ uni...
5	1	14	Residential condominiums ...
6	1	15	Mobile home parks or courts ...
7	1	16	Hotels/motels ...
8	1	17	Institutional lodging ...
9	1	18	All other residential not elsewhere coded ...

Considering the overwhelming number of homes are type 11, the next two most populous categories refer to land sales. It makes sense to therefore restrict this analysis to property type 11.

In [146]:

```
total_df = total_df[total_df['PropertyType']=='11']
```

In [147]:

```
list(total_df.columns)
```

Out[147]:

```
['Major_x',
 'Minor_x',
 'BldgNbr',
 'NbrLivingUnits',
 'Address',
 'BuildingNumber',
 'Fraction',
 'DirectionPrefix',
 'StreetName',
 'StreetType',
 'DirectionSuffix',
 'ZipCode',
 'Stories',
 'BldgGrade',
 'BldgGradeVar',
 'SqFt1stFloor',
 'SqFtHalfFloor',
 'SqFt2ndFloor',
```

Ok, need to make this more useable, drop columns that will no longer be required.

Now I want to create an address column which can be used directly to find latitude and longitude of the property. The current Address column will not work with zip.

In [148]:

```
street_types = {'AVE': 'avenue', 'ST': 'street', 'PL': 'place', 'CT': 'court',\
                'DR': 'drive', 'LN': 'lane', 'RD':'road', 'BLVD': 'boulevard', 'PKWY': 'par
                'TER':'terrace', 'CRES': 'cresent', 'KY':'KY', 'WALK':'WALK'}
```

In [149]:

```
total_df.StreetType.str.strip().map(street_types)
```

Out[149]:

```
10           street
11           street
17           avenue
21        boulevard
28           avenue
            ...
251231       street
251258       avenue
251269        place
251295       street
251296       street
Name: StreetType, Length: 26510, dtype: object
```

In [150]:

```
total_df['address'] = total_df['BuildingNumber'].str.strip() + ' '+ total_df['DirectionPref
                            + total_df['StreetName'].str.strip() + ' ' + total_df['StreetTy
                            + ' ' + total_df['DirectionSuffix'].str.strip() + ',' + ' ' + t
                            + ', WA' + ', USA'
```

In [151]:

```
total_df.head()
```

Out[151]:

	Major_x	Minor_x	BldgNbr	NbrLivingUnits	Address	BuildingNumber	Fraction	Direction
10	010050	0380	1	1	2435 S 354TH ST 98003	2435		
11	010050	0380	1	1	2435 S 354TH ST 98003	2435		
17	017900	0315	1	1	12254 43RD AVE S 98178	12254		
21	018800	0095	1	1	1602 LAKEVIEW BLVD E 98102	1602		
28	019110	0310	1	1	4520 88TH AVE SE 98040	4520		

time to check for duplicates, check how many are duplicated on sale price and id.

In [152]:

```
total_df.duplicated(subset=['SalePrice','id'], keep='last').sum()
```

Out[152]:

```
1008
```

remove duplicates on Sale Price and id.

In [153]:

```
total_df.drop_duplicates(subset=['SalePrice','id'], keep='last', inplace=True)
```

Lets investigate the values in each column, this might aid me in deciding which ones to drop

In [154]:

```
for col in total_df.columns:
    print(col)
    print(total_df[col].value_counts())
```

```
Major_x
276760    90
762570    68
814136    63
510140    60
277060    57
          ..
107000     1
370890     1
715620     1
383060     1
082204     1
Name: Major_x, Length: 7494, dtype: int64
Minor_x
0040    484
0030    471
0020    438
0010    408
0060    407
```

In [155]:

```
total_df['SalePrice'] = total_df['SalePrice'].astype(int)
```

In [156]:

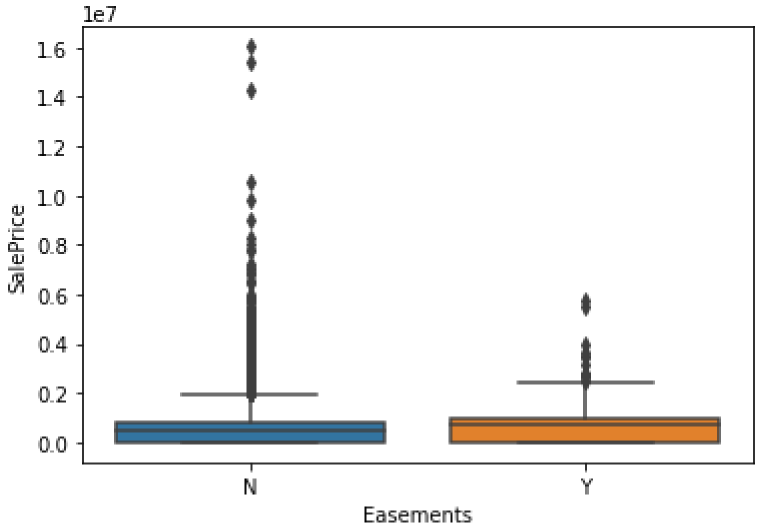
```
def show_box(df, col):
    return sns.boxplot(x=col, y="SalePrice", data=df, showfliers=False)
```

In [157]:

```
sns.boxplot(x='Easements', y="SalePrice", data=total_df, showfliers=True)
```

Out[157]:

```
<AxesSubplot:xlabel='Easements', ylabel='SalePrice'>
```



In [158]:

```
len(total_df)
```

Out[158]:

```
25502
```

In [159]:

```
total_df.PropertyType.value_counts()
```

Out[159]:

```
11    25502
Name: PropertyType, dtype: int64
```

In [160]:

```
total_df = total_df[total_df['SalePrice']!=0]
```

In [161]:

```
from geopy.geocoders import Nominatim
locator = Nominatim(user_agent='myGeocoder')
location = locator.geocode('2435  S  354TH                     ST       , KING COUNTY, WA,
print('Latitude = {}, Longitude = {}'.format(location.latitude, location.longitude))
```

```
Latitude = 47.28493, Longitude = -122.30216590825634
```

In [162]:

```
### commented out as running this will set off a long operation of fetching lat and long in

# # 1 - function to delay between geocoding calls
# geocode = RateLimiter(locator.geocode, min_delay_seconds=1)
# # 2- - create location column
# total_df['location'] = total_df['address'].apply(geocode)
# # 3 - create longitude, laatitude and altitude from location column (returns tuple)
# total_df['point'] = total_df['location'].apply(lambda loc: tuple(loc.point)\
#                                                               if loc else None)
# # 4 - split point column into latitude, longitude and altitude columns
# total_df[['latitude', 'longitude', 'altitude']] = pd.DataFrame(total_df['point'].\
#                                                                        tolist(), index=tot
#                                                                        index)
```

In [163]:

```
total_df.head()
```

Out[163]:

	Major_x	Minor_x	BldgNbr	NbrLivingUnits	Address	BuildingNumber	Fraction	Direction
10	010050	0380	1	1	2435 S 354TH ST 98003	2435		
11	010050	0380	1	1	2435 S 354TH ST 98003	2435		
17	017900	0315	1	1	12254 43RD AVE S 98178	12254		
21	018800	0095	1	1	1602 LAKEVIEW BLVD E 98102	1602		
28	019110	0310	1	1	4520 88TH AVE SE 98040	4520		

Just by chance I noticed the first two columns are for the same address but have different prices, also the only difference between them is the sale warning category. let's take a look at this category more closely...

In [164]:

```
total_df.SaleWarning.value_counts()
```

Out[164]:

```
                 17380
15                 242
26                 201
40                  99
41                  93
10                  51
15 51               49
15 46               45
51                  43
46                  38
15 26               25
12                  17
56                  16
18                  14
15 40               13
54                  13
34                   9
15 56                9
```

In [165]:

```
len(total_df[total_df['SaleWarning']==' '])
```

Out[165]:

```
17380
```

In [166]:

```
len(total_df[total_df['SaleWarning']!=' '])
```

Out[166]:

```
1111
```

In [167]:

```
total_df[total_df['SaleWarning']!=' ']['SalePrice'].mean()
```

Out[167]:

```
684651.8811881188
```

In [168]:

```
total_df[total_df['SaleWarning']==' ']['SalePrice'].mean()
```

Out[168]:

```
800842.2327387802
```

There is a clear difference in the average price of a home with a sale warning and a home without, this will be a feature worth keeping

In [169]:

```
total_df[total_df['Topography']=='0']['SalePrice'].mean()
```

Out[169]:

```
768093.3268439007
```

In [170]:

```
total_df[total_df['Topography']=='1']['SalePrice'].mean()
```

Out[170]:

```
1088088.315648086
```

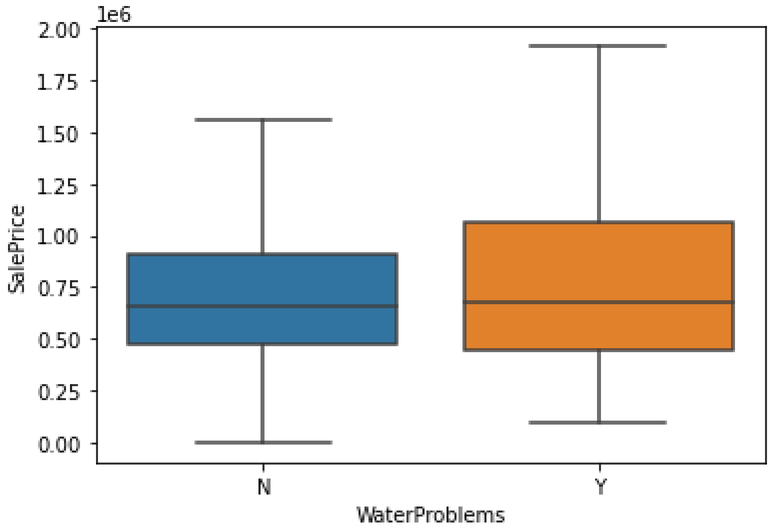
Likewise with topography, making use of landscape either with man made or natural features appears to make a difference, I will keep this

In [171]:

```
show_box(total_df, 'WaterProblems')
```

Out[171]:

```
<AxesSubplot:xlabel='WaterProblems', ylabel='SalePrice'>
```



Water problems, homes with water problems seem to have higher prices, this makes no sense and due to uneven spread (only 70 in over 20,000) I will drop this column

In [172]:

```
for col in total_df.columns:
    print(col)
    print(total_df[col].value_counts())
```

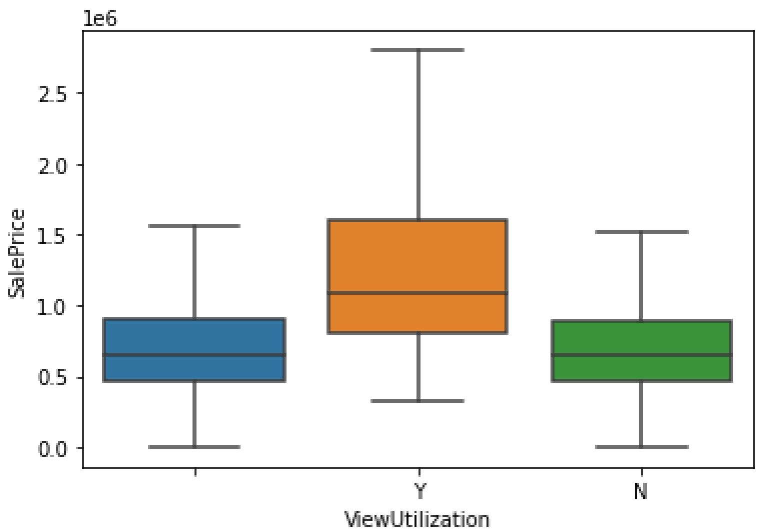
```
Major_x
276760    79
814136    62
762570    53
510140    42
277060    42
          ..
773240     1
213300     1
313730     1
783580     1
082204     1
Name: Major_x, Length: 6579, dtype: int64
Minor_x
0040    354
0030    330
0020    322
0060    314
0010    298
```

In [173]:

```
show_box(total_df, 'ViewUtilization')
```

Out[173]:

```
<AxesSubplot:xlabel='ViewUtilization', ylabel='SalePrice'>
```



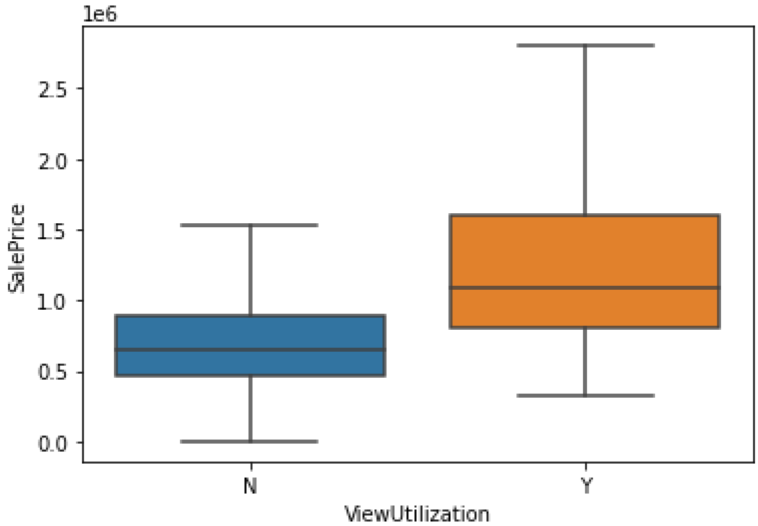
Another feature worth keeping, although I will assume a blank entry is N.

In [174]:

```
replace_val(total_df, 'ViewUtilization', ' ', 'N')

show_box(total_df, 'ViewUtilization')
```

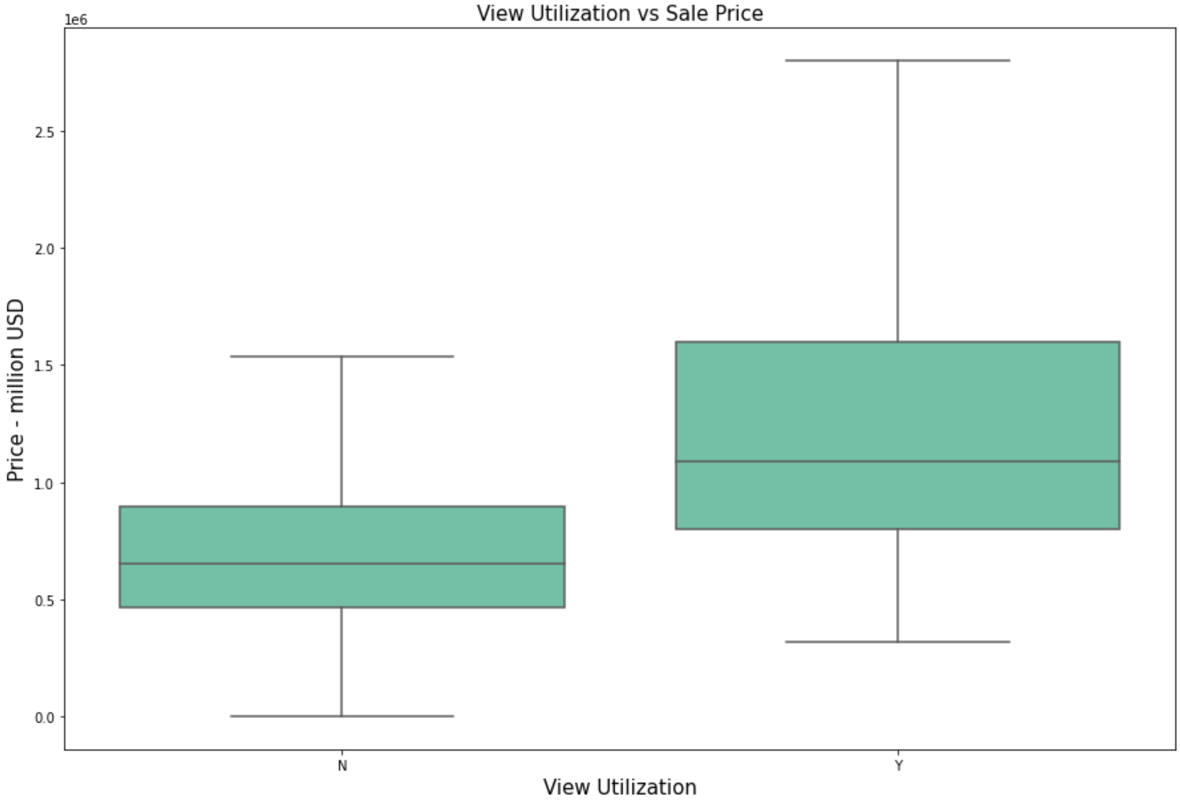
Out[174]:

```
<AxesSubplot:xlabel='ViewUtilization', ylabel='SalePrice'>
```



In [175]:

```
plt.figure(figsize=(15,10))
plt.title('View Utilization vs Sale Price', fontsize=15)
plt.ylabel('Price - million USD', fontsize=15)
plt.xlabel('Bedrooms minus Bathrooms', fontsize=15)
boxplot = sns.boxplot(x='ViewUtilization', y="SalePrice", data=total_df, color= 'mediumaqua
boxplot.set(xlabel='View Utilization',ylabel='Price - million USD');
plt.savefig('views.png', bbox_inches = 'tight')
```



In [177]:

```
# plt.figure(figsize=(15,10))
# plt.title('Square Feet Living Area vs Sale Price', fontsize=15)
# plt.ylabel('Price - million USD', fontsize=15)
# plt.xlabel('Square Foot Total Living', fontsize=15)

# ax.set_ylabel('amplitude')
```

```
---------------------------------------------------------------------------
-
UFuncTypeError                            Traceback (most recent call las
t)
<ipython-input-177-abfdcc25f098> in <module>
      3 # plt.ylabel('Price - million USD', fontsize=15)
      4 # plt.xlabel('Square Foot Total Living', fontsize=15)
----> 5 ax = sns.lmplot(x="SqFtTotLiving", y="SalePrice", data=total_df, s
catter_kws={'color': 'mediumaquamarine'}, height = 7,\
      6                   aspect=1.5, line_kws={'color': 'green'});
      7 ax.fig.suptitle('Square Foot Living vs Sale Price', fontsize=15)

~\anaconda3\envs\geo-env\lib\site-packages\seaborn\_decorators.py in inner
_f(*args, **kwargs)
     44             )
     45         kwargs.update({k: arg for k, arg in zip(sig.parameters, ar
gs)})
---> 46         return f(**kwargs)
     47     return inner_f
```

That's better. Now I think all the columns worth keeping have been highlighted, its time to drop the rest, keeping some that I could be useful at some point - I don't know what I don't know yet!

In [178]:

```
cols_to_drop = ['Fraction','BldgGradeVar','AddnlCost', 'Unnamed: 0', 'Major_y', 'Minor_y',
               'PlatLot_x', 'PlatBlock_x', 'Range', 'SpecArea', 'SubArea', 'SpecSubArea', '
               'HBUAsIfVacant', 'HBUAsImproved', 'PresentUse','WaterSystem', 'SewerSystem',
               'PcntUnusable', 'WfntBank','WfntPoorQuality', 'WfntRestrictedAccess', 'WfntA
                'WfntProximityInfluence', 'TidelandShoreland','LotDepthFactor', 'AirportNoi
                'NbrBldgSites', 'Contamination', 'DNRLease','AdjacentGolfFairway', 'Histori
                'CurrentUseDesignation', 'NativeGrowthProtEsmt', 'OtherDesignation', 'DeedR
                'DevelopmentRightsPurch', 'CoalMineHazard', 'CriticalDrainage','ErosionHaza
                'HundredYrFloodPlain', 'SeismicHazard', 'LandslideHazard','SteepSlopeHazard
                'SpeciesOfConcern', 'SensitiveAreaTract', 'WaterProblems', 'TranspConcurren
                'PlatNbr', 'PlatType','PlatLot_y', 'PlatBlock_y', 'SellerName', 'BuyerName'
                'AFForestLand', 'AFCurrentUseLand', 'AFNonProfitUse', 'AFHistoricProperty',
               'NbrLivingUnits', 'SqFtUnfinFull', 'SqFtUnfinHalf','FpSingleStory','FpMultiS
               'PcntComplete','Obsolescence','PcntNetCondition', 'PropType', 'Unbuildable',
                'Minor', 'RecordingNbr', 'PropertyType', 'PropertyClass', 'Date']
```

In [179]:

```
total_df.drop(columns=cols_to_drop, inplace=True)
```

lets take a look at the new dataframe

In [180]:

```
total_df.head()
```

Out[180]:

	Major_x	Minor_x	BldgNbr	Address	BuildingNumber	DirectionPrefix	StreetName	StreetType	Directic
10	010050	0380	1	2435 S 354TH ST 98003	2435	S	354TH	ST	
11	010050	0380	1	2435 S 354TH ST 98003	2435	S	354TH	ST	
17	017900	0315	1	12254 43RD AVE S 98178	12254		43RD	AVE	

Ok, it is starting to take shape, I want to transform the yes no columns into ones and zeroes though for analysis.

In [181]:

```
cols_to_encode = ['PowerLines','OtherNuisances','AdjacentGreenbelt','Easements', 'DaylightB
# lets check the value counts first
```

In [182]:

```
for col in cols_to_encode:
    total_df[col] = total_df[col].str.strip()
    print(total_df[col].value_counts())
```

```
N    18276
Y      215
Name: PowerLines, dtype: int64
N    17931
Y      560
Name: OtherNuisances, dtype: int64
N    17957
Y      534
Name: AdjacentGreenbelt, dtype: int64
N    18098
Y      393
Name: Easements, dtype: int64
N    7543
     6114
Y    4831
y       3
Name: DaylightBasement, dtype: int64
```

In [183]:

```
# clean columns so they are either Y or N
replace_val(total_df, 'DaylightBasement', '', 'N')
replace_val(total_df, 'DaylightBasement', 'y', 'Y')
for col in cols_to_encode:
    total_df[col] = total_df[col].str.strip()
    print(total_df[col].value_counts())
```

```
N    18276
Y      215
Name: PowerLines, dtype: int64
N    17931
Y      560
Name: OtherNuisances, dtype: int64
N    17957
Y      534
Name: AdjacentGreenbelt, dtype: int64
N    18098
Y      393
Name: Easements, dtype: int64
N    13657
Y     4834
Name: DaylightBasement, dtype: int64
```

In [184]:

```
# creating instance of labelencoder
labelencoder = LabelEncoder()
# Replacing Y/N with numerical
for col in cols_to_encode:
    total_df[col] = labelencoder.fit_transform(total_df[col])
    print(total_df[col].value_counts())
```

```
0    18276
1      215
Name: PowerLines, dtype: int64
0    17931
1      560
Name: OtherNuisances, dtype: int64
0    17957
1      534
Name: AdjacentGreenbelt, dtype: int64
0    18098
1      393
Name: Easements, dtype: int64
0    13657
1     4834
Name: DaylightBasement, dtype: int64
```

In [185]:

```
total_df['ViewUtilization'].value_counts()
```

Out[185]:

```
N    18074
Y      417
Name: ViewUtilization, dtype: int64
```

In [186]:

```
total_df['ViewUtilization'] = labelencoder.fit_transform(total_df['ViewUtilization'])
total_df['ViewUtilization'].value_counts()
```

Out[186]:

```
0    18074
1      417
Name: ViewUtilization, dtype: int64
```

In [187]:

```
total_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18491 entries, 10 to 251295
Data columns (total 68 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Major_x              18491 non-null  object
 1   Minor_x              18491 non-null  object
 2   BldgNbr              18491 non-null  object
 3   Address              18491 non-null  object
 4   BuildingNumber       18491 non-null  object
 5   DirectionPrefix      18477 non-null  object
 6   StreetName           18491 non-null  object
 7   StreetType           18491 non-null  object
 8   DirectionSuffix      18477 non-null  object
 9   ZipCode              16072 non-null  object
 10  Stories              18491 non-null  object
 11  BldgGrade            18491 non-null  object
 12  SqFt1stFloor         18491 non-null  object
 13  SqFtHalfFloor        18491 non-null  object
```

I still need to convert alot of these columns before they will be usable in a model.

In [188]:

```
for col in total_df.columns:
    print(total_df[col].value_counts())
```

```
276760    79
814136    62
762570    53
510140    42
277060    42
          ..
773240     1
213300     1
313730     1
783580     1
082204     1
Name: Major_x, Length: 6579, dtype: int64
0040    354
0030    330
0020    322
0060    314
0010    298
       ...
0413      1
```

In [189]:

```
cols_to_int = ['SqFt1stFloor', 'SqFtHalfFloor', 'SqFt2ndFloor', 'SqFtUpperFloor', 'SqFtTotL
               'SqFtFinBasement', 'FinBasementGrade', 'SqFtGarageBasement', 'SqFtGarageAtt
              'Topography','WfntLocation', 'WfntFootage', 'MtRainier', 'Olympics', 'Cascad
              'PugetSound','LakeWashington', 'LakeSammamish', 'SmallLakeRiverCreek', 'Othe
```

In [190]:

```
total_df.columns
```

Out[190]:

```
Index(['Major_x', 'Minor_x', 'BldgNbr', 'Address', 'BuildingNumber',
       'DirectionPrefix', 'StreetName', 'StreetType', 'DirectionSuffix',
       'ZipCode', 'Stories', 'BldgGrade', 'SqFt1stFloor', 'SqFtHalfFloor',
       'SqFt2ndFloor', 'SqFtUpperFloor', 'SqFtTotLiving', 'SqFtTotBasement',
       'SqFtFinBasement', 'FinBasementGrade', 'SqFtGarageBasement',
       'SqFtGarageAttached', 'DaylightBasement', 'SqFtOpenPorch',
       'SqFtEnclosedPorch', 'SqFtDeck', 'HeatSystem', 'HeatSource',
       'BrickStone', 'ViewUtilization', 'Bedrooms', 'BathHalfCount',
       'Bath3qtrCount', 'BathFullCount', 'YrBuilt', 'YrRenovated', 'Conditio
n',
       'id', 'Township', 'Section', 'QuarterSection', 'Area', 'DistrictNam
e',
       'SqFtLot', 'Access', 'Topography', 'InadequateParking', 'MtRainier',
       'Olympics', 'Cascades', 'Territorial', 'SeattleSkyline', 'PugetSoun
d',
       'LakeWashington', 'LakeSammamish', 'SmallLakeRiverCreek', 'OtherVie
w',
       'WfntLocation', 'WfntFootage', 'TrafficNoise', 'PowerLines',
       'OtherNuisances', 'AdjacentGreenbelt', 'Easements', 'DocumentDate',
       'SalePrice', 'SaleWarning', 'address'],
      dtype='object')
```

It is starting to take shape and resemble something that could be useable for analysis, however, remembering this data was imported as string. I will need to convert columns that should be integer.

In [191]:

```
# convert columns to integer type
for col in cols_to_int:
    total_df[col] = total_df[col].astype(int)
```

In [192]:

```
total_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18491 entries, 10 to 251295
Data columns (total 68 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   Major_x             18491 non-null  object
 1   Minor_x             18491 non-null  object
 2   BldgNbr             18491 non-null  object
 3   Address             18491 non-null  object
 4   BuildingNumber      18491 non-null  object
 5   DirectionPrefix     18477 non-null  object
 6   StreetName          18491 non-null  object
 7   StreetType          18491 non-null  object
 8   DirectionSuffix     18477 non-null  object
 9   ZipCode             16072 non-null  object
 10  Stories             18491 non-null  object
 11  BldgGrade           18491 non-null  object
 12  SqFt1stFloor        18491 non-null  int32 
 13  SqFtHalfFloor       18491 non-null  int32 
 14  SqFt2ndFloor        18491 non-null  int32 
 15  SqFtUpperFloor      18491 non-null  int32 
 16  SqFtTotLiving       18491 non-null  int32 
 17  SqFtTotBasement     18491 non-null  int32 
 18  SqFtFinBasement     18491 non-null  int32 
 19  FinBasementGrade    18491 non-null  int32 
 20  SqFtGarageBasement  18491 non-null  int32 
 21  SqFtGarageAttached  18491 non-null  int32 
 22  DaylightBasement    18491 non-null  int32 
 23  SqFtOpenPorch       18491 non-null  object
 24  SqFtEnclosedPorch   18491 non-null  object
 25  SqFtDeck            18491 non-null  int32 
 26  HeatSystem          18491 non-null  object
 27  HeatSource          18491 non-null  object
 28  BrickStone          18491 non-null  object
 29  ViewUtilization     18491 non-null  int32 
 30  Bedrooms            18491 non-null  object
 31  BathHalfCount       18491 non-null  object
 32  Bath3qtrCount       18491 non-null  object
 33  BathFullCount       18491 non-null  object
 34  YrBuilt             18491 non-null  int32 
 35  YrRenovated         18491 non-null  object
 36  Condition           18491 non-null  object
 37  id                  18491 non-null  object
 38  Township            18491 non-null  object
 39  Section             18491 non-null  object
 40  QuarterSection      18491 non-null  object
 41  Area                18491 non-null  object
 42  DistrictName        18491 non-null  object
 43  SqFtLot             18491 non-null  int32 
 44  Access              18491 non-null  object
 45  Topography          18491 non-null  int32 
 46  InadequateParking   18491 non-null  object
 47  MtRainier           18491 non-null  int32 
 48  Olympics            18491 non-null  int32 
 49  Cascades            18491 non-null  int32 
 50  Territorial         18491 non-null  int32 
```

```
 51  SeattleSkyline       18491 non-null  int32
 52  PugetSound           18491 non-null  int32
 53  LakeWashington       18491 non-null  int32
 54  LakeSammamish        18491 non-null  int32
 55  SmallLakeRiverCreek  18491 non-null  int32
 56  OtherView            18491 non-null  int32
 57  WfntLocation         18491 non-null  int32
 58  WfntFootage          18491 non-null  int32
 59  TrafficNoise         18491 non-null  object
 60  PowerLines           18491 non-null  int32
 61  OtherNuisances       18491 non-null  int32
 62  AdjacentGreenbelt    18491 non-null  int32
 63  Easements            18491 non-null  int32
 64  DocumentDate         18491 non-null  object
 65  SalePrice            18491 non-null  int32
 66  SaleWarning          18491 non-null  object
 67  address              18477 non-null  object
dtypes: int32(33), object(35)
memory usage: 7.9+ MB
```

In [193]:

```
#check for null values
total_df.isna().sum()
```

Out[193]:

```
Major_x                  0
Minor_x                  0
BldgNbr                  0
Address                  0
BuildingNumber           0
DirectionPrefix         14
StreetName               0
StreetType               0
DirectionSuffix         14
ZipCode               2419
Stories                  0
BldgGrade                0
SqFt1stFloor             0
SqFtHalfFloor            0
SqFt2ndFloor             0
SqFtUpperFloor           0
SqFtTotLiving            0
SqFtTotBasement          0
```

In [194]:

```
len(total_df)
```

Out[194]:

```
18491
```

In [195]:

```
# checking duplicates where only the price has changed, if it has changed and nothing else
# one of them should be dropped
columns_check_duplicates = ['Major_x', 'Minor_x', 'BldgNbr', 'Address', 'BuildingNumber',
       'DirectionPrefix', 'StreetName', 'StreetType', 'DirectionSuffix',
       'ZipCode', 'Stories', 'BldgGrade', 'SqFt1stFloor', 'SqFtHalfFloor',
       'SqFt2ndFloor', 'SqFtUpperFloor', 'SqFtTotLiving', 'SqFtTotBasement',
       'SqFtFinBasement', 'FinBasementGrade', 'SqFtGarageBasement',
       'SqFtGarageAttached', 'DaylightBasement', 'SqFtOpenPorch',
       'SqFtEnclosedPorch', 'SqFtDeck', 'HeatSystem', 'HeatSource',
       'BrickStone', 'ViewUtilization', 'Bedrooms', 'BathHalfCount',
       'Bath3qtrCount', 'BathFullCount', 'YrBuilt', 'YrRenovated', 'Condition',
       'id', 'Township', 'Section', 'QuarterSection', 'Area', 'DistrictName',
       'SqFtLot', 'Access', 'Topography', 'InadequateParking', 'MtRainier',
       'Olympics', 'Cascades', 'Territorial', 'SeattleSkyline', 'PugetSound',
       'LakeWashington', 'LakeSammamish', 'SmallLakeRiverCreek', 'OtherView',
       'WfntLocation', 'WfntFootage', 'TrafficNoise', 'PowerLines',
       'OtherNuisances', 'AdjacentGreenbelt', 'Easements', 'DocumentDate','SaleWarning', 'a
```

In [196]:

```
# removing duplicates where only price has changed. keeping the highest value
total_df[total_df.sort_values('SalePrice').duplicated(subset=columns_check_duplicates, keep
```

```
<ipython-input-196-7c07cd8313d9>:2: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  total_df[total_df.sort_values('SalePrice').duplicated(subset=columns_check
_duplicates, keep=False)]
```

Out[196]:

	Major_x	Minor_x	BldgNbr	Address	BuildingNumber	DirectionPrefix	StreetName
59507	312206	9048	1	18203 SE 272ND ST 98042	18203	SE	272ND
59508	312206	9048	1	18203 SE 272ND ST 98042	18203	SE	272ND
59509	312206	9048	1	18203 SE 272ND ST 98042	18203	SE	272ND
183159	781280	0105	1	7469 S 116TH ST 98178	7469	S	116TH
183160	781280	0105	1	7469 S 116TH ST 98178	7469	S	116TH
203156	927420	3841	1	2008 A CALIFORNIA AVE SW	2008		CALIFORNIA
203158	927420	3841	1	2008 A CALIFORNIA AVE SW	2008		CALIFORNIA

In [197]:

```
# remove these duplicates so only the highest price is kept.
total_df = total_df.sort_values('SalePrice')
total_df.drop_duplicates(subset=columns_check_duplicates, keep='last', inplace=True)
```

appears to be a lot of zip codes missing, there may be a way to find these from the Address column

In [198]:

```
# lets make the SaleWarning column more user friendly
total_df.SaleWarning.value_counts()
```

Out[198]:

```
                 17376
15                 242
26                 201
40                  99
41                  93
10                  51
15 51               49
15 46               45
51                  43
46                  38
15 26               25
12                  17
56                  16
18                  14
15 40               13
54                  13
15 56                9
12 15                9
34                   9
10 15                8
24                   6
10 56                6
35                   6
5 51                 5
29                   5
49                   4
15 26 46             4
18 51                4
15 36 56             4
15 24                3
10 12                3
10 29                3
13 15                2
60                   2
18 22                2
15 36                2
15 46 51             2
26 46                2
10 15 56             2
13                   2
15 18                2
30                   2
7                    1
38                   1
45                   1
7 20                 1
10 15 34             1
3 26                 1
23 51                1
3                    1
22 24                1
26 51                1
26 56                1
```

```
5                   1
52                  1
10 36               1
12 26 51            1
15 46 56            1
*                   1
18 22 51            1
10 11 15            1
10 15 29            1
12 15 26            1
15 18 51            1
57                  1
12 22 51            1
46 56               1
13 26               1
10 11 56            1
10 15 46            1
15 22 26 51         1
15 26 51            1
12 26               1
15 18 46            1
80                  1
5 15                1
15 22 51            1
12 46 51            1
15 36 51            1
15 26 56            1
10 51               1
3 15 26 29          1
12 15 46 58         1
12 15 51            1
26 38 46            1
13 23               1
Name: SaleWarning, dtype: int64
```

In [199]:

```
len(total_df[total_df['SaleWarning']== ' '])
```

Out[199]:

```
17376
```

In [200]:

```
total_df.loc[total_df['SaleWarning']!= ' ', 'SaleWarning'] = 1
total_df.loc[total_df['SaleWarning'] == ' ', 'SaleWarning'] = 0
total_df.SaleWarning.value_counts()
```

Out[200]:

```
0    17376
1     1111
Name: SaleWarning, dtype: int64
```

Now it is time to map the columns that have references that are related to in the look up table provided. Replacing these numbers with their actual values will make the values easier to interpret when it comes to making them dummy variables

In [201]:

```
# convert heat system column as per values in lookup table
heating_dict = get_dict(108, lookup_df)
total_df.HeatSystem = total_df.HeatSystem.str.strip().astype(int).map(heating_dict)
total_df.HeatSystem.value_counts()
```

Out[201]:

```
Forced Air    14392
Heat Pump      1595
Elec BB        1150
Floor-Wall      565
Hot Water       461
Radiant         258
Gravity          38
Other            11
Name: HeatSystem, dtype: int64
```

In [202]:

```
heatsource_dict = get_dict(84, lookup_df)
total_df.HeatSource = total_df.HeatSource.str.strip().astype(int).map(heatsource_dict)
total_df.HeatSource.value_counts()
```

Out[202]:

```
Gas                  13352
Electricity           3257
Oil                   1792
Gas/Solar               39
Other                   17
Electricity/Solar       11
Oil/Solar                3
Name: HeatSource, dtype: int64
```

lets review the status of the df now a lot has changed

In [203]:

```
total_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18487 entries, 62686 to 153622
Data columns (total 68 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   Major_x              18487 non-null  object
 1   Minor_x              18487 non-null  object
 2   BldgNbr              18487 non-null  object
 3   Address              18487 non-null  object
 4   BuildingNumber       18487 non-null  object
 5   DirectionPrefix      18473 non-null  object
 6   StreetName           18487 non-null  object
 7   StreetType           18487 non-null  object
 8   DirectionSuffix      18473 non-null  object
 9   ZipCode              16069 non-null  object
 10  Stories              18487 non-null  object
 11  BldgGrade            18487 non-null  object
 12  SqFt1stFloor         18487 non-null  int32 
 13  SqFtHalfFloor        18487 non-null  int32 
 14  SqFt2ndFloor         18487 non-null  int32 
 15  SqFtUpperFloor       18487 non-null  int32 
 16  SqFtTotLiving        18487 non-null  int32 
 17  SqFtTotBasement      18487 non-null  int32 
 18  SqFtFinBasement      18487 non-null  int32 
 19  FinBasementGrade     18487 non-null  int32 
 20  SqFtGarageBasement   18487 non-null  int32 
 21  SqFtGarageAttached   18487 non-null  int32 
 22  DaylightBasement     18487 non-null  int32 
 23  SqFtOpenPorch        18487 non-null  object
 24  SqFtEnclosedPorch    18487 non-null  object
 25  SqFtDeck             18487 non-null  int32 
 26  HeatSystem           18470 non-null  object
 27  HeatSource           18471 non-null  object
 28  BrickStone           18487 non-null  object
 29  ViewUtilization      18487 non-null  int32 
 30  Bedrooms             18487 non-null  object
 31  BathHalfCount        18487 non-null  object
 32  Bath3qtrCount        18487 non-null  object
 33  BathFullCount        18487 non-null  object
 34  YrBuilt              18487 non-null  int32 
 35  YrRenovated          18487 non-null  object
 36  Condition            18487 non-null  object
 37  id                   18487 non-null  object
 38  Township             18487 non-null  object
 39  Section              18487 non-null  object
 40  QuarterSection       18487 non-null  object
 41  Area                 18487 non-null  object
 42  DistrictName         18487 non-null  object
 43  SqFtLot              18487 non-null  int32 
 44  Access               18487 non-null  object
 45  Topography           18487 non-null  int32 
 46  InadequateParking    18487 non-null  object
 47  MtRainier            18487 non-null  int32 
 48  Olympics             18487 non-null  int32 
 49  Cascades             18487 non-null  int32 
 50  Territorial          18487 non-null  int32 
```

```
 51  SeattleSkyline       18487 non-null  int32
 52  PugetSound           18487 non-null  int32
 53  LakeWashington       18487 non-null  int32
 54  LakeSammamish        18487 non-null  int32
 55  SmallLakeRiverCreek  18487 non-null  int32
 56  OtherView            18487 non-null  int32
 57  WfntLocation         18487 non-null  int32
 58  WfntFootage          18487 non-null  int32
 59  TrafficNoise         18487 non-null  object
 60  PowerLines           18487 non-null  int32
 61  OtherNuisances       18487 non-null  int32
 62  AdjacentGreenbelt    18487 non-null  int32
 63  Easements            18487 non-null  int32
 64  DocumentDate         18487 non-null  object
 65  SalePrice            18487 non-null  int32
 66  SaleWarning          18487 non-null  object
 67  address              18473 non-null  object
dtypes: int32(33), object(35)
memory usage: 7.4+ MB
```

In [204]:

```python
for col in total_df.columns:
    print('\n')
    print(total_df[col].value_counts())
    print('\n')
```

```
276760    79
814136    62
762570    53
277060    42
510140    42
          ..
814200     1
219160     1
405080     1
177423     1
082204     1
Name: Major_x, Length: 6579, dtype: int64




0040    354
```

some more tidying required.

In [205]:

```
# convert wfntlocation to binary column
total_df.loc[total_df['WfntLocation']!= 0, 'WfntLocation'] = 1
total_df.WfntLocation.value_counts()
```

Out[205]:

```
0    18192
1      295
Name: WfntLocation, dtype: int64
```

time to drop more columns

In [206]:

```
total_df.columns
```

Out[206]:

```
Index(['Major_x', 'Minor_x', 'BldgNbr', 'Address', 'BuildingNumber',
       'DirectionPrefix', 'StreetName', 'StreetType', 'DirectionSuffix',
       'ZipCode', 'Stories', 'BldgGrade', 'SqFt1stFloor', 'SqFtHalfFloor',
       'SqFt2ndFloor', 'SqFtUpperFloor', 'SqFtTotLiving', 'SqFtTotBasement',
       'SqFtFinBasement', 'FinBasementGrade', 'SqFtGarageBasement',
       'SqFtGarageAttached', 'DaylightBasement', 'SqFtOpenPorch',
       'SqFtEnclosedPorch', 'SqFtDeck', 'HeatSystem', 'HeatSource',
       'BrickStone', 'ViewUtilization', 'Bedrooms', 'BathHalfCount',
       'Bath3qtrCount', 'BathFullCount', 'YrBuilt', 'YrRenovated', 'Conditio
n',
       'id', 'Township', 'Section', 'QuarterSection', 'Area', 'DistrictNam
e',
       'SqFtLot', 'Access', 'Topography', 'InadequateParking', 'MtRainier',
       'Olympics', 'Cascades', 'Territorial', 'SeattleSkyline', 'PugetSoun
d',
       'LakeWashington', 'LakeSammamish', 'SmallLakeRiverCreek', 'OtherVie
w',
       'WfntLocation', 'WfntFootage', 'TrafficNoise', 'PowerLines',
       'OtherNuisances', 'AdjacentGreenbelt', 'Easements', 'DocumentDate',
       'SalePrice', 'SaleWarning', 'address'],
      dtype='object')
```

In [207]:

```
more_drops = ['Major_x', 'Minor_x', 'BldgNbr', 'WfntFootage']
total_df.drop(columns=more_drops, inplace=True)
```

In [208]:

```
total_df['SqFtOpenPorch'] = total_df['SqFtOpenPorch'].str.strip().astype(int)
total_df['SqFtEnclosedPorch'] = total_df['SqFtEnclosedPorch'].str.strip().astype(int)
```

do access and inadequate parking, then make column for excellent view.

In [209]:

```
view_columns = ['MtRainier','Olympics', 'Cascades', 'Territorial', 'SeattleSkyline', 'Puget
                'LakeSammamish', 'SmallLakeRiverCreek', 'OtherView']

total_df[(total_df['MtRainier']==4)| (total_df['Olympics']==4) | (total_df['Cascades']==4)
        | (total_df['SeattleSkyline']==4) | (total_df['PugetSound']==4) | (total_df['LakeWa
        (total_df['LakeSammamish']==4)|(total_df['SmallLakeRiverCreek']==4) | (total_df['Ot
```

Out[209]:

	Address	BuildingNumber	DirectionPrefix	StreetName	StreetType	DirectionSuffix	ZipCode
7464	10831 SE LAKE RD 98004	10831	SE	LAKE	RD		98004
10233	11065 SE LAKE RD 98004	11065	SE	LAKE	RD		98004
137867	30726 270TH AVE SE 98010	30726		270TH	AVE	SE	98010
26556	11610 DOLPHIN	11610		DOLPHIN	TRL	SW	98070

In [210]:

```
total_df['excellent_view'] = 0
```

In [211]:

```
total_df.loc[(total_df['MtRainier']==4)| (total_df['Olympics']==4) | (total_df['Cascades']=
        | (total_df['SeattleSkyline']==4) | (total_df['PugetSound']==4) | (total_df['LakeWa
        (total_df['LakeSammamish']==4)|(total_df['SmallLakeRiverCreek']==4) | (total_df['Ot
```

created a new column just for excellent views, I will now make the rest of the view columns binary.

In [212]:

```
total_df.head()
```

Out[212]:

	Address	BuildingNumber	DirectionPrefix	StreetName	StreetType	DirectionSuffix	Zip(
62686	17701 185TH AVE NE 98072	17701		185TH	AVE	NE	9
157183	9508 167TH AVE NE 98052	9508		167TH	AVE	NE	9
876	19361 61ST AVE NE 98028	19361		61ST	AVE	NE	9
247014	15915 VASHON HWY SW 98070	15915		VASHON	HWY	SW	9
188918	10616 SW 133RD ST 98070	10616	SW	133RD	ST		9

In [213]:

```
for col in view_columns:
    total_df.loc[total_df[col]!= 0, col] = 1
```

In [214]:

```
total_df.Access.value_counts()
```

Out[214]:

```
4    17329
3     1120
1       18
5       11
0        7
2        2
Name: Access, dtype: int64
```

In [215]:

```
access_dict = get_dict(55, lookup_df)
access_dict
```

Out[215]:

```
{1: 'RESTRICTED',
 2: 'LEGAL/UNDEVELOPED',
 3: 'PRIVATE',
 4: 'PUBLIC',
 5: 'WALK IN'}
```

In [216]:

```
total_df.Access = total_df.Access.str.strip().astype(int).map(access_dict)
```

In [217]:

```
total_df.Access.value_counts()
```

Out[217]:

```
PUBLIC               17329
PRIVATE               1120
RESTRICTED              18
WALK IN                 11
LEGAL/UNDEVELOPED        2
Name: Access, dtype: int64
```

In [218]:

```
total_df.InadequateParking.value_counts()
```

Out[218]:

```
2    11477
0     6992
1       18
Name: InadequateParking, dtype: int64
```

I am going to make an assumption that since 2 represents aqequate parking that 0 and 1 will represent inadequate parking

In [219]:

```
replace_val(total_df, 'InadequateParking', '1', 0)
total_df.InadequateParking.value_counts()
```

Out[219]:

```
2    11477
0     6992
0       18
Name: InadequateParking, dtype: int64
```

In [220]:

```
replace_val(total_df, 'InadequateParking', '0', 0)
replace_val(total_df, 'InadequateParking', '1', 0)
replace_val(total_df, 'InadequateParking', '2', 1)
total_df.InadequateParking.value_counts()
```

Out[220]:

```
1    11477
0     7010
Name: InadequateParking, dtype: int64
```

I will now join the table I created to get latitude and longitude information

In [221]:

```
longslats_df = pd.read_csv('..\..\data/raw/longslats.csv', dtype='str')
```

In [222]:

```
longslats_df = longslats_df[['id', 'latitude', 'longitude']]
```

In [223]:

```
total_df.to_csv('pre-merge.csv')
```

In [224]:

```
total_df = total_df.merge(longslats_df, how='left', on='id')
```

In [225]:

```
total_df['latitude'] = total_df['latitude'].astype(float)
total_df['longitude'] = total_df['longitude'].astype(float)
```

In [226]:

```
total_df.isna().sum()
```

Out[226]:

```
Address                    0
BuildingNumber             0
DirectionPrefix           14
StreetName                 0
StreetType                 0
DirectionSuffix           14
ZipCode                 2441
Stories                    0
BldgGrade                  0
SqFt1stFloor               0
SqFtHalfFloor              0
SqFt2ndFloor               0
SqFtUpperFloor             0
SqFtTotLiving              0
SqFtTotBasement            0
SqFtFinBasement            0
FinBasementGrade           0
SqFtGarageBasement         0
SqFtGarageAttached         0
DaylightBasement           0
SqFtOpenPorch              0
SqFtEnclosedPorch          0
SqFtDeck                   0
HeatSystem                17
HeatSource                16
BrickStone                 0
ViewUtilization            0
Bedrooms                   0
BathHalfCount              0
Bath3qtrCount              0
BathFullCount              0
YrBuilt                    0
YrRenovated                0
Condition                  0
id                         0
Township                   0
Section                    0
QuarterSection             0
Area                       0
DistrictName               0
SqFtLot                    0
Access                     7
Topography                 0
InadequateParking          0
MtRainier                  0
Olympics                   0
Cascades                   0
Territorial                0
SeattleSkyline             0
PugetSound                 0
LakeWashington             0
LakeSammamish              0
SmallLakeRiverCreek        0
OtherView                  0
WfntLocation               0
```

```
TrafficNoise          0
PowerLines            0
OtherNuisances        0
AdjacentGreenbelt     0
Easements             0
DocumentDate          0
SalePrice             0
SaleWarning           0
address              14
excellent_view        0
latitude            291
longitude           291
dtype: int64
```

In [227]:

```
len(total_df)
```

Out[227]:

```
18797
```

In [228]:

```
total_df.dropna(subset=['ZipCode'], inplace=True)
```

In [229]:

```
len(total_df)
```

Out[229]:

```
16356
```

In [230]:

```
total_df.isna().sum()
```

Out[230]:

```
Address                 0
BuildingNumber          0
DirectionPrefix         0
StreetName              0
StreetType              0
DirectionSuffix         0
ZipCode                 0
Stories                 0
BldgGrade               0
SqFt1stFloor            0
SqFtHalfFloor           0
SqFt2ndFloor            0
SqFtUpperFloor          0
SqFtTotLiving           0
SqFtTotBasement         0
SqFtFinBasement         0
FinBasementGrade        0
SqFtGarageBasement      0
SqFtGarageAttached      0
DaylightBasement        0
SqFtOpenPorch           0
SqFtEnclosedPorch       0
SqFtDeck                0
HeatSystem             16
HeatSource             16
BrickStone              0
ViewUtilization         0
Bedrooms                0
BathHalfCount           0
Bath3qtrCount           0
BathFullCount           0
YrBuilt                 0
YrRenovated             0
Condition               0
id                      0
Township                0
Section                 0
QuarterSection          0
Area                    0
DistrictName            0
SqFtLot                 0
Access                  5
Topography              0
InadequateParking       0
MtRainier               0
Olympics                0
Cascades                0
Territorial             0
SeattleSkyline          0
PugetSound              0
LakeWashington          0
LakeSammamish           0
SmallLakeRiverCreek     0
OtherView               0
WfntLocation            0
```

```
TrafficNoise           0
PowerLines             0
OtherNuisances         0
AdjacentGreenbelt      0
Easements              0
DocumentDate           0
SalePrice              0
SaleWarning            0
address                0
excellent_view         0
latitude             206
longitude            206
dtype: int64
```

In [231]:

```
total_df.dropna(subset=['latitude', 'longitude', 'HeatSource', 'HeatSystem', 'Access'], inp
```

In [232]:

```
len(total_df)
```

Out[232]:

```
16131
```

In [233]:

```
total_df.isna().sum()
```

Out[233]:

```
Address                0
BuildingNumber         0
DirectionPrefix        0
StreetName             0
StreetType             0
DirectionSuffix        0
ZipCode                0
Stories                0
BldgGrade              0
SqFt1stFloor           0
SqFtHalfFloor          0
SqFt2ndFloor           0
SqFtUpperFloor         0
SqFtTotLiving          0
SqFtTotBasement        0
SqFtFinBasement        0
FinBasementGrade       0
SqFtGarageBasement     0
SqFtGarageAttached     0
DaylightBasement       0
SqFtOpenPorch          0
SqFtEnclosedPorch      0
SqFtDeck               0
HeatSystem             0
HeatSource             0
BrickStone             0
ViewUtilization        0
Bedrooms               0
BathHalfCount          0
Bath3qtrCount          0
BathFullCount          0
YrBuilt                0
YrRenovated            0
Condition              0
id                     0
Township               0
Section                0
QuarterSection         0
Area                   0
DistrictName           0
SqFtLot                0
Access                 0
Topography             0
InadequateParking      0
MtRainier              0
Olympics               0
Cascades               0
Territorial            0
SeattleSkyline         0
PugetSound             0
LakeWashington         0
LakeSammamish          0
SmallLakeRiverCreek    0
OtherView              0
WfntLocation           0
```

```
TrafficNoise           0
PowerLines             0
OtherNuisances         0
AdjacentGreenbelt      0
Easements              0
DocumentDate           0
SalePrice              0
SaleWarning            0
address                0
excellent_view         0
latitude               0
longitude              0
dtype: int64
```

In [234]:

```
total_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16131 entries, 0 to 18796
Data columns (total 67 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Address               16131 non-null  object
 1   BuildingNumber        16131 non-null  object
 2   DirectionPrefix       16131 non-null  object
 3   StreetName            16131 non-null  object
 4   StreetType            16131 non-null  object
 5   DirectionSuffix       16131 non-null  object
 6   ZipCode               16131 non-null  object
 7   Stories               16131 non-null  object
 8   BldgGrade             16131 non-null  object
 9   SqFt1stFloor          16131 non-null  int32
 10  SqFtHalfFloor         16131 non-null  int32
 11  SqFt2ndFloor          16131 non-null  int32
 12  SqFtUpperFloor        16131 non-null  int32
 13  SqFtTotLiving         16131 non-null  int32
```

In [235]:

```
#drop duplicates with same address and same price
total_df.drop_duplicates(subset=['Address', 'SalePrice'], keep='last', inplace=True)
```

In [236]:

```
#total_df.to_csv('cleaned_data.csv')
```

that was hard work, a lot of columns required attention but now its time to start modelling. This data will require further cleaning iterations and feature engineering but this is a good starting point

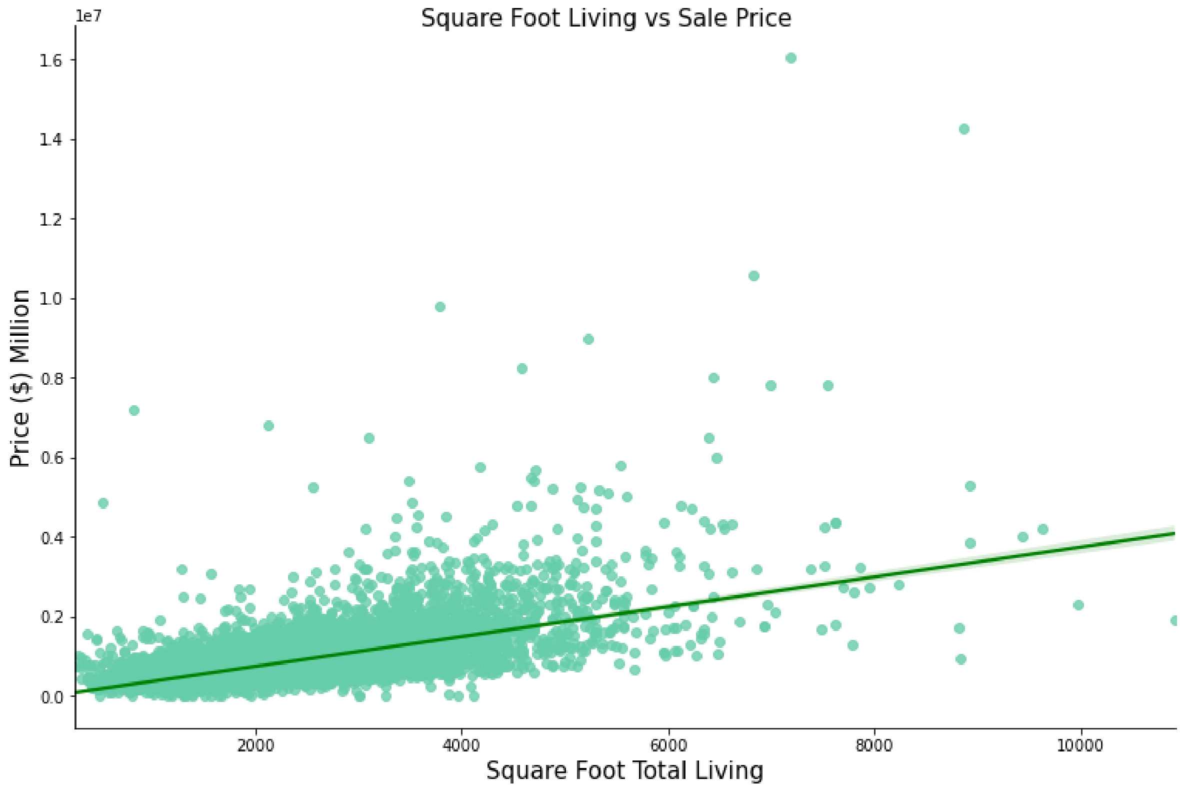
In [237]:

```
#Create base map zoomed in to seattle
map3=folium.Map(location=[47.5837012,-122.3984634],  tiles=None, zoom_start=7)
folium.TileLayer('cartodbpositron', name='King County House Prices').add_to(map3)

#Make Marker Cluster Group layer
mcg = folium.plugins.MarkerCluster(control=False)
map3.add_child(mcg)

#Create layer of markers
#Set marker popups to display name and address of service
for row in total_df.iterrows():
    row_values=row[1]
    location=[row_values['latitude'], row_values['longitude']]
    popup=popup=('$' + str(row_values['SalePrice'])+'<br>'+'<br>'+ row_values['Address']+
                 '<br>'+'<br>'+row_values['DistrictName'])
    marker=folium.Marker(location=location, popup=popup, min_width=2000)
    marker.add_to(mcg)

#Add layer control
folium.LayerControl().add_to(map3)

map3
```

Out[237]:

In [238]:

```
ax = sns.lmplot(x="SqFtTotLiving", y="SalePrice", data=total_df, scatter_kws={'color': 'med
                aspect=1.5, line_kws={'color': 'green'});
ax.fig.suptitle('Square Foot Living vs Sale Price', fontsize=15)
ax.ax.set_xlabel('Square Foot Total Living', fontsize=15)
ax.ax.set_ylabel('Price ($) Million', fontsize=15)
ax.fig.savefig('sqft.png', bbox_inches = 'tight')
```



In [ ]:

```

```

In [ ]:

```

```