Our exploratory data analysis examined the housing market in Ames, Iowa, using the house price [SalePrice] as the dependent variable of interest. The initial dataset includes 1460 data records and 81 fields. The test data has a mean sale price of $180,921.20 and a standard deviation of $79442.50. We cleaned the data by removing fields with null values and trimmed the data to only instances where house price falls within two standard deviations of the initial median value. The cleaned test data has 1,459 data records, 63 fields, a median sale price of $169,995.87, and a standard deviation of $58,943.80.

We found [OverallQual], [GrLivingArea], and [GarageCars] have correlation coefficients of 0.78, 0.66, and 0.63, respectively. We created two new features to be used as predictor variables. A total square foot variable [tot_sq] that measures the houses' entire square footage (including basements) and an overall quality variable [OverallQual] that multiplies [total_sq] by overall quality [OverallQual. [total_sq] has a correlation coefficient of 0.73 and [qual_space] has a correlation coefficient of 0.8.

Following our exploratory data analysis, we prepared our models using a cross-validation design. We divided our training data set into two groups: a training data set of 1167 (80%) and a testing data set of 292 (20%). Using a cross-validation design, we tested all of our models and used the one that best predicted the "test" training data to use with the "real" test data.

We built four regression models using the training data to predict the [SalePrice] for each house. The first model uses [OverallQual], [GrLivArea], and [GarageCars] as predictor variables. These variables showed the highest correlation with [SalePrice]. The second linear regression model uses [OverallQual], which has the highest correlation coefficient to predict [SalePrice].

The third linear regression model uses our new feature variable [qual_space] as the predictor variable. The fourth model uses both [qual_space] and [GarageCars] to predict [SalePrice].

The simple linear regression models in Model Two and Model Three show strong linear relationships. Therefore,no polynomial (non-linear) or piecewise (multiple different lines) components were observed in the plot of these linear models. Additionally, there were no dichotomous variables (two options) used in the models since all the indicator variables were numeric and none were binomial.

The models assume that the predictor variables are independent. Additionally, the models that use [qual_space] assume that the overall quality rating variable does not factor in square footage to the rating system. As we're using linear regression models, we also assume that the data is linear and that variance of residual is the same. After comparing the RMSE for each model, we have determined that Model Four, which uses [qual_space] and [GarageCars] to predict [SalePrice], produced the best fit based on having the lowest RMSE (27,482) using the cross-validation method.

The linear regression Model Four uses two independent variables ([qual_space] and [GarageCars]) to predict housing prices. The first feature variable quality of space [qual_space] allows us to analyze the overall quality rating with consideration to the usable living space of the house. The second feature variable we used in the model was [GarageCars], which is how many cars the garage can fit. (We replaced one [GarageCars] NaN value with zero for the testing dataset.) The model tells us that as the quality of space score and the number of cars the garage fits increases, so does the predicted sale price of the house.

MSDS 422-57 | Module 2 Assignment 1 | Group 5
Jul 2, 2022 | House Prices: Advanced Regression Techniques EDA | Scott Jue
Dr. Anil Chaturvedi | | Zach Watson

Kaggle user name for upload: ZachWat

YOUR RECENT SUBMISSION

group_5_msds_422_module_2.csv **Score: 0.20317**

Submitted by ZachWat · Submitted a minute ago

Jump to your leaderboard position

https://canvas.northwestern.edu/courses/167719/assignments/1078596

https://www.kaggle.com/c/house-prices-advanced-regression-techniques

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
```

In [2]:
```
#Import train.csv and test.csv from the Kaggle page linked above
df = pd.read_csv ('train.csv')
```

EDA

Intro Stats

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             1460 non-null   int64  
 1   MSSubClass     1460 non-null   int64  
 2   MSZoning       1460 non-null   object 
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64  
 5   Street         1460 non-null   object 
 6   Alley          91 non-null     object 
 7   LotShape       1460 non-null   object 
 8   LandContour    1460 non-null   object 
 9   Utilities      1460 non-null   object 
 10  LotConfig      1460 non-null   object 
 11  LandSlope      1460 non-null   object 
 12  Neighborhood   1460 non-null   object 
 13  Condition1     1460 non-null   object 
 14  Condition2     1460 non-null   object 
 15  BldgType       1460 non-null   object 
 16  HouseStyle     1460 non-null   object 
 17  OverallQual    1460 non-null   int64  
 18  OverallCond    1460 non-null   int64  
 19  YearBuilt      1460 non-null   int64  
 20  YearRemodAdd   1460 non-null   int64  
 21  RoofStyle      1460 non-null   object 
 22  RoofMatl       1460 non-null   object 
 23  Exterior1st    1460 non-null   object 
 24  Exterior2nd    1460 non-null   object 
 25  MasVnrType     1452 non-null   object 
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object 
 28  ExterCond      1460 non-null   object 
 29  Foundation     1460 non-null   object 
 30  BsmtQual       1423 non-null   object 
```

```
 31  BsmtCond       1423 non-null   object
 32  BsmtExposure   1422 non-null   object
 33  BsmtFinType1   1423 non-null   object
 34  BsmtFinSF1     1460 non-null   int64
 35  BsmtFinType2   1422 non-null   object
 36  BsmtFinSF2     1460 non-null   int64
 37  BsmtUnfSF      1460 non-null   int64
 38  TotalBsmtSF    1460 non-null   int64
 39  Heating        1460 non-null   object
 40  HeatingQC      1460 non-null   object
 41  CentralAir     1460 non-null   object
 42  Electrical     1459 non-null   object
 43  1stFlrSF       1460 non-null   int64
 44  2ndFlrSF       1460 non-null   int64
 45  LowQualFinSF   1460 non-null   int64
 46  GrLivArea      1460 non-null   int64
 47  BsmtFullBath   1460 non-null   int64
 48  BsmtHalfBath   1460 non-null   int64
 49  FullBath       1460 non-null   int64
 50  HalfBath       1460 non-null   int64
 51  BedroomAbvGr   1460 non-null   int64
 52  KitchenAbvGr   1460 non-null   int64
 53  KitchenQual    1460 non-null   object
 54  TotRmsAbvGrd   1460 non-null   int64
 55  Functional     1460 non-null   object
 56  Fireplaces     1460 non-null   int64
 57  FireplaceQu    770 non-null    object
 58  GarageType     1379 non-null   object
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

In [4]:

```
df.head()
```

Out[4]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	.
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	.
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	

5 rows × 81 columns

Provide appropriate descriptive statistics and visualizations to help understand the marginal distribution of the dependent variable.

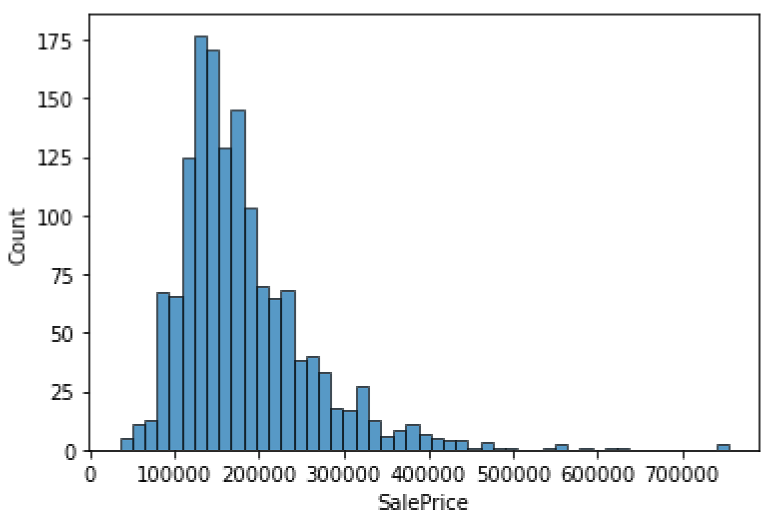
In [5]:
```
df["SalePrice"].describe()
```

Out[5]:
```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

EDA SalePrice Graphs

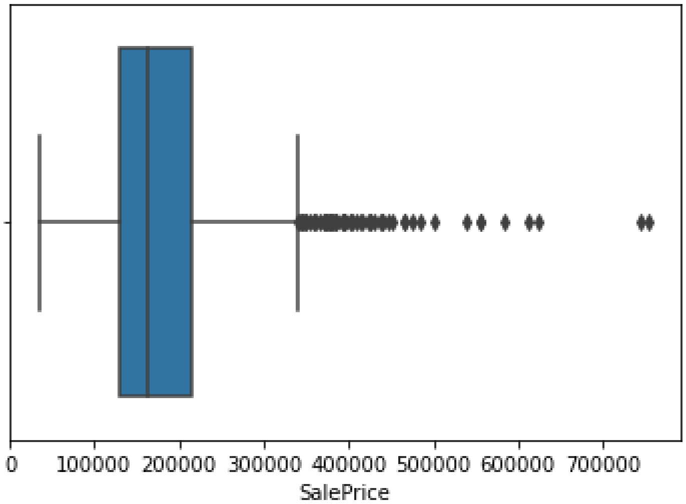
In [6]:
```
sns.histplot(x="SalePrice", data=df)
```

Out[6]: <AxesSubplot:xlabel='SalePrice', ylabel='Count'>



In [7]:
```
sns.boxplot(x="SalePrice", data=df)
```

Out[7]: <AxesSubplot:xlabel='SalePrice'>



Investigate Missing Data and Outliers

Missing Data:

In [8]:

```
df.isnull().sum()
```

Out[8]:

```
Id                 0
MSSubClass         0
MSZoning           0
LotFrontage      259
LotArea            0
                ... 
MoSold             0
YrSold             0
SaleType           0
SaleCondition      0
SalePrice          0
Length: 81, dtype: int64
```

The following categories have null values:

- LotFrontage
- Alley
- MasVnrType
- MasVnrArea
- BsmtQual
- BsmtCond
- BsmtExposure
- BsmtFinType1
- BsmtFinType2
- Electrical
- FireplaceQu
- GarageType
- GarageYrBlt
- GarageFinish
- GarageQual
- GarageCond

- PoolQC
- Fence
- MiscFeature

We're not concerned about most of these columns having null values. It makes sense that some of the data would be missing for each (if a house doesn't have a pool, for example). We're going to drop all of the columns that have null values with the exception of "Electrical." For 'Electrical,' we'll remove the row with the null value.

In [9]:

```
col_to_drop = ['LotFrontage','Alley','MasVnrType','MasVnrArea',
               'BsmtQual', 'BsmtCond', 'BsmtExposure','BsmtFinType1', 'BsmtFinType2', '
               'GarageFinish', 'GarageQual', 'GarageCond', 'PoolQC', 'Fence', 'MiscFeat

df2 = df.drop(columns=col_to_drop, inplace=False)
df3 = df2.dropna()
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1459 entries, 0 to 1459
Data columns (total 63 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Id             1459 non-null   int64 
 1   MSSubClass     1459 non-null   int64 
 2   MSZoning       1459 non-null   object
 3   LotArea        1459 non-null   int64 
 4   Street         1459 non-null   object
 5   LotShape       1459 non-null   object
 6   LandContour    1459 non-null   object
 7   Utilities      1459 non-null   object
 8   LotConfig      1459 non-null   object
 9   LandSlope      1459 non-null   object
 10  Neighborhood   1459 non-null   object
 11  Condition1     1459 non-null   object
 12  Condition2     1459 non-null   object
 13  BldgType       1459 non-null   object
 14  HouseStyle     1459 non-null   object
 15  OverallQual    1459 non-null   int64 
 16  OverallCond    1459 non-null   int64 
 17  YearBuilt      1459 non-null   int64 
 18  YearRemodAdd   1459 non-null   int64 
 19  RoofStyle      1459 non-null   object
 20  RoofMatl       1459 non-null   object
 21  Exterior1st    1459 non-null   object
 22  Exterior2nd    1459 non-null   object
 23  ExterQual      1459 non-null   object
 24  ExterCond      1459 non-null   object
 25  Foundation     1459 non-null   object
 26  BsmtFinSF1     1459 non-null   int64 
 27  BsmtFinSF2     1459 non-null   int64 
 28  BsmtUnfSF      1459 non-null   int64 
 29  TotalBsmtSF    1459 non-null   int64 
 30  Heating        1459 non-null   object
 31  HeatingQC      1459 non-null   object
 32  CentralAir     1459 non-null   object
 33  Electrical     1459 non-null   object
 34  1stFlrSF       1459 non-null   int64 
 35  2ndFlrSF       1459 non-null   int64 
 36  LowQualFinSF   1459 non-null   int64 
 37  GrLivArea      1459 non-null   int64 
```

```
 38  BsmtFullBath   1459 non-null   int64
 39  BsmtHalfBath   1459 non-null   int64
 40  FullBath       1459 non-null   int64
 41  HalfBath       1459 non-null   int64
 42  BedroomAbvGr   1459 non-null   int64
 43  KitchenAbvGr   1459 non-null   int64
 44  KitchenQual    1459 non-null   object
 45  TotRmsAbvGrd   1459 non-null   int64
 46  Functional     1459 non-null   object
 47  Fireplaces     1459 non-null   int64
 48  GarageCars     1459 non-null   int64
 49  GarageArea     1459 non-null   int64
 50  PavedDrive     1459 non-null   object
 51  WoodDeckSF     1459 non-null   int64
 52  OpenPorchSF    1459 non-null   int64
 53  EnclosedPorch  1459 non-null   int64
 54  3SsnPorch      1459 non-null   int64
 55  ScreenPorch    1459 non-null   int64
 56  PoolArea       1459 non-null   int64
 57  MiscVal        1459 non-null   int64
 58  MoSold         1459 non-null   int64
 59  YrSold         1459 non-null   int64
 60  SaleType       1459 non-null   object
 61  SaleCondition  1459 non-null   object
 62  SalePrice      1459 non-null   int64
dtypes: int64(35), object(28)
memory usage: 729.5+ KB
```

Outliers

In [10]:

```
df3['SalePrice'].describe(percentiles = [.25, .5, .75, .95])
```

Out[10]:

```
count      1459.000000
mean     180930.394791
std       79468.964025
min       34900.000000
25%      129950.000000
50%      163000.000000
75%      214000.000000
95%      326200.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

In [11]:

```
#This code trims data to a certain number of standard deviations from the mean. We went
#You can see in the graphs below that it removes many outliers and normalizes the data.

from scipy import stats
import numpy as np

df4 = df3[(np.abs(stats.zscore(df3['SalePrice'])) < 2)]
```

EDA SalePrice Stats/Graphs (Cleaned Data)

In [12]:

```
df4["SalePrice"].describe()
```

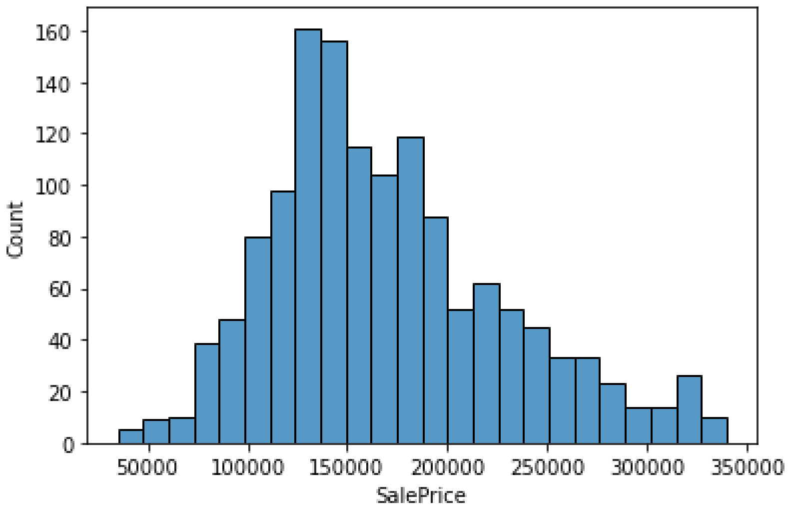
Out[12]:

```
count      1396.000000
mean     169995.874642
std       58943.796385
min       34900.000000
25%      128987.500000
```

```
50%       159467.000000
75%       203000.000000
max       339750.000000
Name: SalePrice, dtype: float64
```

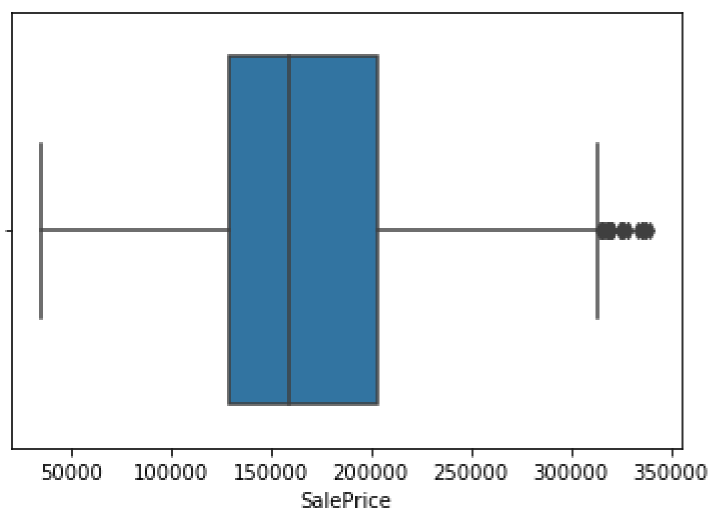
In [13]:
```
sns.histplot(x="SalePrice", data=df4)
```

Out[13]: <AxesSubplot:xlabel='SalePrice', ylabel='Count'>



In [14]:
```
sns.boxplot(x="SalePrice", data=df4)
```

Out[14]: <AxesSubplot:xlabel='SalePrice'>

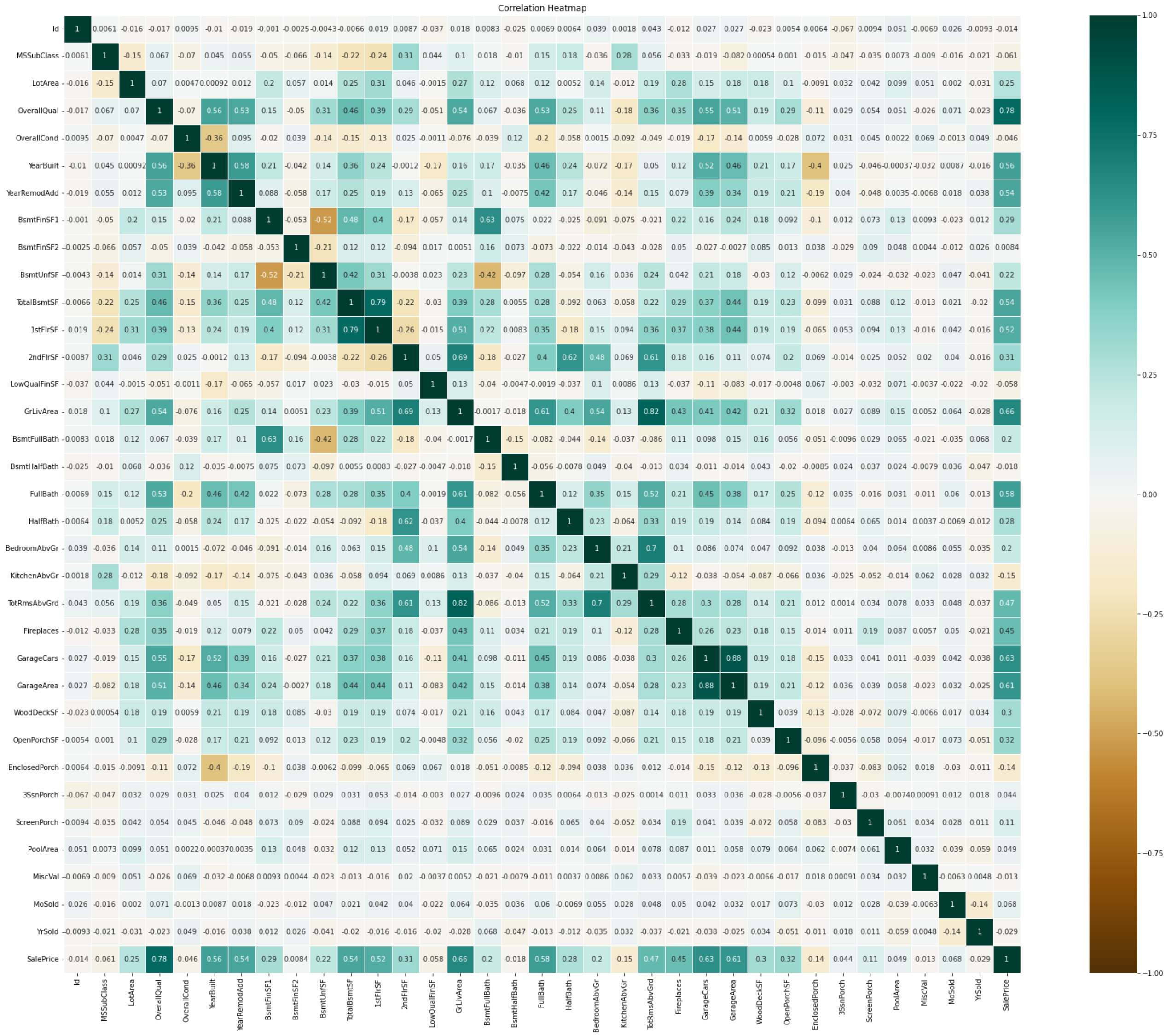


Correlations

Investigate at least three potential predictors of the dependent variable and provide appropriate graphs / statistics to demonstrate the relationships.

In [15]:
```
# creating correlation heatmap to determine potential predictor variables
corr_mat = df4.corr()
f, ax = plt.subplots(figsize=(35, 25))
sns.heatmap(corr_mat, vmin=-1, vmax=1, annot=True, square=True, linewidths=.5, cmap='Br
plt.title('Correlation Heatmap')
```

Out[15]: Text(0.5, 1.0, 'Correlation Heatmap')

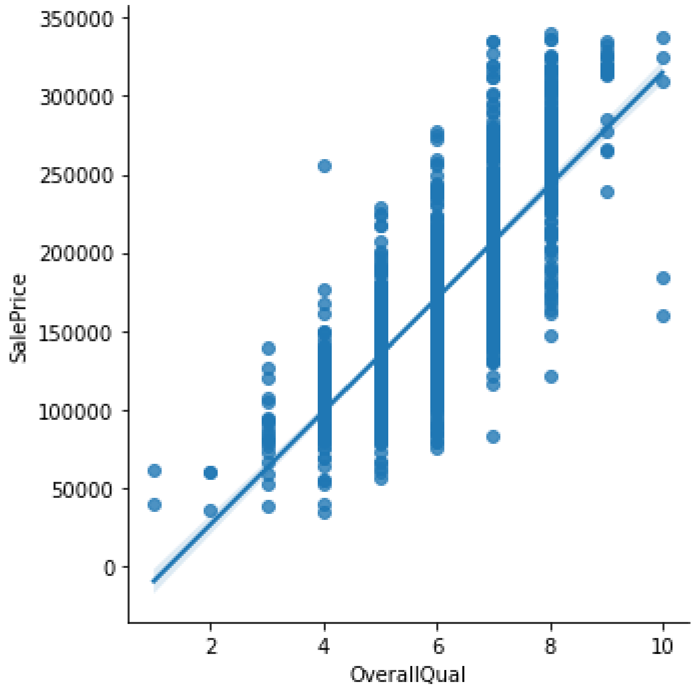


The 3 variables with the highest correlation to SalesPrice are OverallQual (0.78), GrLivArea (0.66), and GarageCars (0.63). These are potential predictor variables for SalesPrice which is our dependent variable.

Scatterplots for [OverallQual] [GrLivArea] [GarageArea]

In [16]:
```
#sns.scatterplot(x='OverallQual', y='SalePrice', data=df4)
sns.lmplot(x='OverallQual', y='SalePrice', data=df4)
```

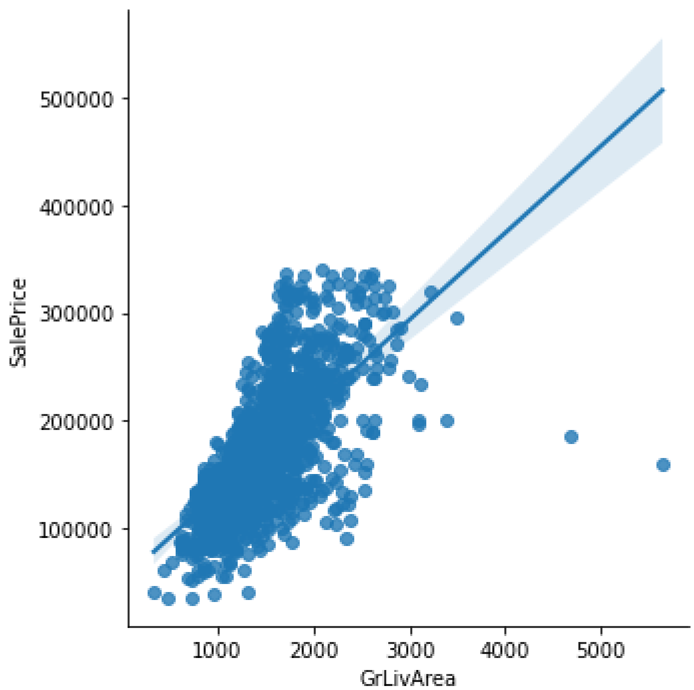
Out[16]: <seaborn.axisgrid.FacetGrid at 0x260da1a28b0>



In [17]:

```
#sns.scatterplot(x="GrLivArea", y="SalePrice", data=df4)
sns.lmplot(x="GrLivArea", y="SalePrice", data=df4)
```

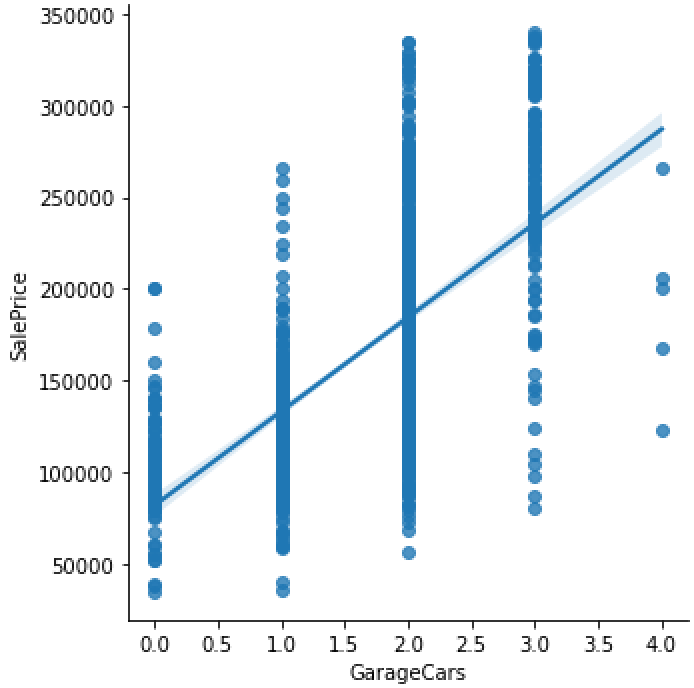
Out[17]: <seaborn.axisgrid.FacetGrid at 0x260da1bb6d0>



In [18]:

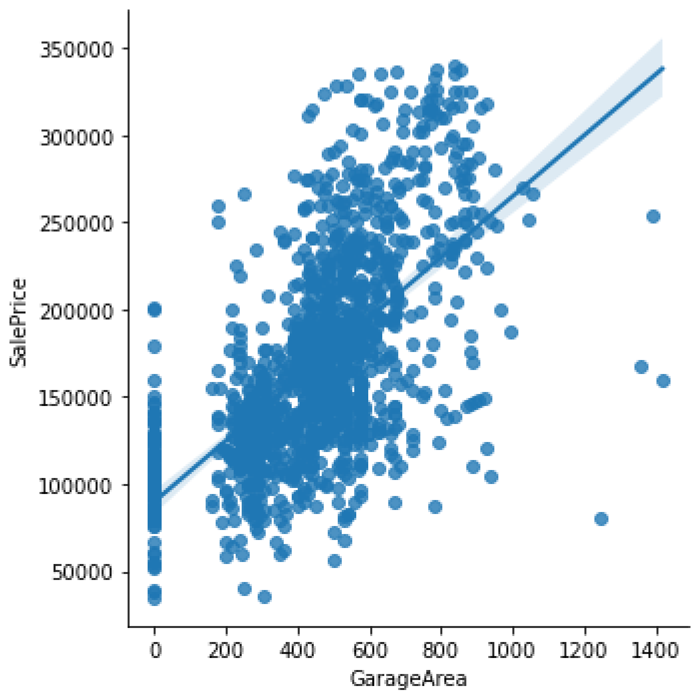
```
#sns.scatterplot(x="GarageCars", y="SalePrice", data=df4)
sns.lmplot(x="GarageCars", y="SalePrice", data=df4)
```

Out[18]: <seaborn.axisgrid.FacetGrid at 0x260da19b0a0>



In [19]:
```
#sns.scatterplot(x="GarageArea", y="SalePrice", data=df4)
sns.lmplot(x="GarageArea", y="SalePrice", data=df4)
```

Out[19]: <seaborn.axisgrid.FacetGrid at 0x260da028d30>



New Predictors (Total Square Feet [tot_sq] and Quality Space [qual_space]

Engage in feature creation by splitting, merging, or otherwise generating a new predictor.

In [20]:
```
# sum 1st floor, 2nd floor, and basement square footage to get total square footage
sum_column = df4['1stFlrSF'] + df4['2ndFlrSF'] + df4['TotalBsmtSF']

# multiply total square footage by overall quality to generate new predictor variable q
```

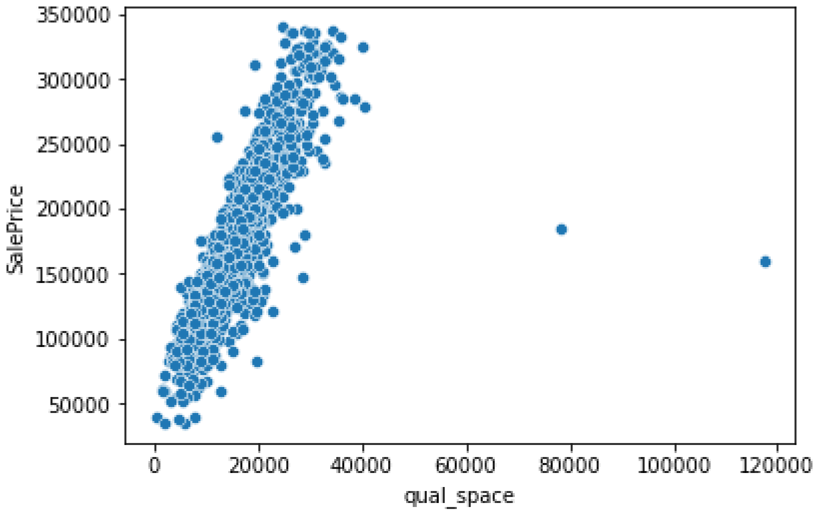
```
mult_column = sum_column*df4['OverallQual']

# add new predictor variables to dataframe
df4['tot_sq'] = sum_column
df4['qual_space'] = mult_column
print(df4)
```

```
        Id  MSSubClass MSZoning  LotArea Street LotShape LandContour  \
0        1          60       RL     8450   Pave      Reg         Lvl
1        2          20       RL     9600   Pave      Reg         Lvl
2        3          60       RL    11250   Pave      IR1         Lvl
3        4          70       RL     9550   Pave      IR1         Lvl
4        5          60       RL    14260   Pave      IR1         Lvl
...    ...         ...      ...      ...    ...      ...         ...
1455  1456          60       RL     7917   Pave      Reg         Lvl
1456  1457          20       RL    13175   Pave      Reg         Lvl
1457  1458          70       RL     9042   Pave      Reg         Lvl
1458  1459          20       RL     9717   Pave      Reg         Lvl
1459  1460          20       RL     9937   Pave      Reg         Lvl

     Utilities LotConfig LandSlope  ... ScreenPorch PoolArea MiscVal MoSold  \
0       AllPub    Inside       Gtl  ...           0        0       0      2
1       AllPub       FR2       Gtl  ...           0        0       0      5
2       AllPub    Inside       Gtl  ...           0        0       0      9
3       AllPub    Corner       Gtl  ...           0        0       0      2
4       AllPub       FR2       Gtl  ...           0        0       0     12
...        ...       ...       ...  ...         ...      ...     ...    ...
1455    AllPub    Inside       Gtl  ...           0        0       0      8
1456    AllPub    Inside       Gtl  ...           0        0       0      2
1457    AllPub    Inside       Gtl  ...           0        0    2500      5
1458    AllPub    Inside       Gtl  ...           0        0       0      4
1459    AllPub    Inside       Gtl  ...           0        0       0      6

     YrSold  SaleType  SaleCondition  SalePrice  tot_sq qual_space
0      2008        WD         Normal     208500    2566      17962
1      2007        WD         Normal     181500    2524      15144
2      2008        WD         Normal     223500    2706      18942
3      2006        WD        Abnorml     140000    2473      17311
4      2008        WD         Normal     250000    3343      26744
...     ...       ...            ...        ...     ...        ...
1455   2007        WD         Normal     175000    2600      15600
1456   2010        WD         Normal     210000    3615      21690
1457   2010        WD         Normal     266500    3492      24444
1458   2010        WD         Normal     142125    2156      10780
1459   2008        WD         Normal     147500    2512      12560

[1396 rows x 65 columns]
```

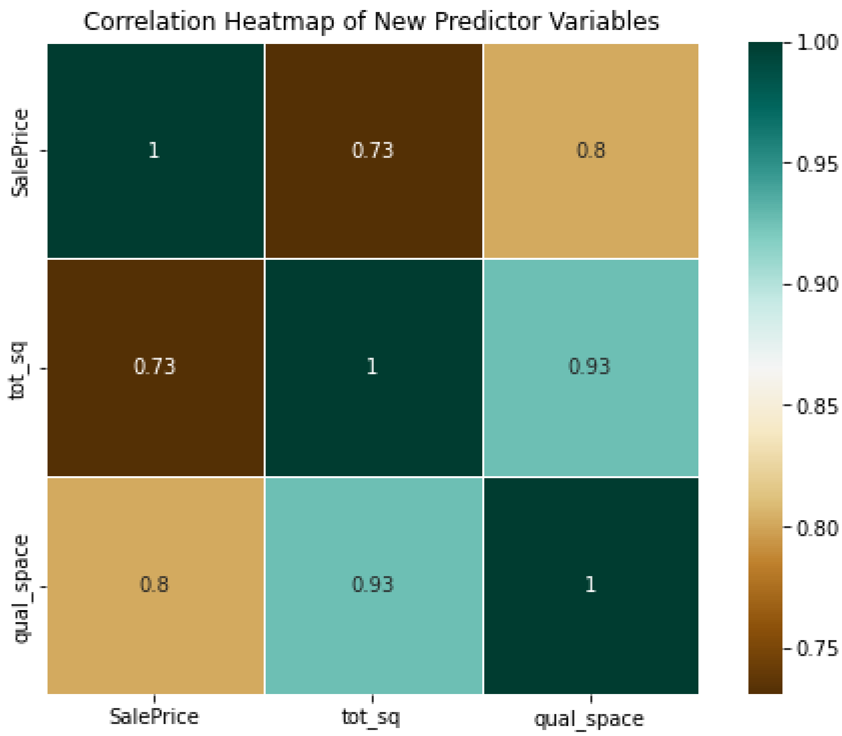
```
C:\Users\watsonz\AppData\Local\Temp\ipykernel_21836\239284426.py:8: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df4['tot_sq'] = sum_column
C:\Users\watsonz\AppData\Local\Temp\ipykernel_21836\239284426.py:9: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df4['qual_space'] = mult_column
```

In [21]:
```
sns.scatterplot(x="qual_space", y="SalePrice", data=df4)
```

Out[21]: <AxesSubplot:xlabel='qual_space', ylabel='SalePrice'>



In [22]:
```
# setting the columns to correlate
columns = ['SalePrice', 'tot_sq', 'qual_space']
df_corr = df4[columns]
# running the correlation
df_corr.corr()

# setting up the heatmap
corrmat = df_corr.corr()

# set the figure size
f, ax = plt.subplots(figsize=(9, 6))

# pass the data and set the parameters
sns.heatmap(corrmat, vmax=1, square=True, annot=True, cmap='BrBG', linewidths=.5 )
plt.title('Correlation Heatmap of New Predictor Variables')

# images can be saved - default is .png
# https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.savefig.html
plt.savefig('Correlation Heatmap of New Predictor Variables')
```



Min-Max Scaling for Pricing

In [23]:

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
mm_scaler = MinMaxScaler()
std_scaler = StandardScaler()

# reshape to SalePrice 2D array and perform min-max scaling
mmscaled_data = mm_scaler.fit_transform(df4['SalePrice'].values.reshape(-1,1))

print(mmscaled_data)
```

```
[[0.56946039]
 [0.48089224]
 [0.61866492]
 ...
 [0.75971789]
 [0.35173036]
 [0.36936198]]
```

In [24]:

```
# checking min and max
print(mmscaled_data.min())
print(mmscaled_data.max())
```

```
0.0
1.0
```

In [25]:

```
# reshape to SalePrice 2D array and perform standard scaling
stdscaled_data = std_scaler.fit_transform(df4['SalePrice'].values.reshape(-1,1))

print(stdscaled_data)
```

```
[[ 0.65346866]
 [ 0.19524104]
 [ 0.90803956]
 ...
```

```
[ 1.63780948]
[-0.47300758]
[-0.38178634]]
```

In [26]:

```
# checking mean and standard deviation
print(stdscaled_data.mean())
print(stdscaled_data.std())
```

```
2.1440983340898438e-16
1.0
```

In [27]:

```
# add scaled sale price to dataframe
df4['MinMaxScaled_SalePrice'] = mmscaled_data
df4['StdScaled_SalePrice'] = stdscaled_data
```

```
C:\Users\watsonz\AppData\Local\Temp\ipykernel_21836\647750776.py:2: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df4['MinMaxScaled_SalePrice'] = mmscaled_data
C:\Users\watsonz\AppData\Local\Temp\ipykernel_21836\647750776.py:3: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df4['StdScaled_SalePrice'] = stdscaled_data
```

Regressions

Liner Regression Model One ([OverallQual] [GRliveArea] [GarageCars])

In [33]:

```
# assigning the 3 predictor variables with the highest correlation coefficient
features = ['OverallQual', 'GrLivArea', 'GarageCars']
```

In [34]:

```
X = df4[features]
```

In [35]:

```
y = df4['SalePrice']
```

In [36]:

```
# split data in to training and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
```

In [37]:

```
# creating linear regression model
model_1 = LinearRegression().fit(X_train, y_train)
```

In [38]:

```
# display model coefficients and r-sqaured scores
print('Coefficient:', model_1.coef_)
```

```
print('Scores:', model_1.score(X_train, y_train), model_1.score(X_test, y_test))
```

```
Coefficient: [22662.77685508    35.27537731 19875.9200822 ]
Scores: 0.7286420265854436 0.7442832958076211
```

In [39]:

```
# predicted housing prices
y_prediction = model_1.predict(X_test)
```

In [40]:

```
# model RMSE
RMSE = sqrt(mean_squared_error(y_true = y_test, y_pred = y_prediction))
print(RMSE)
```

```
28262.71855795518
```

Liner Regression Model Two ([OverallQual])

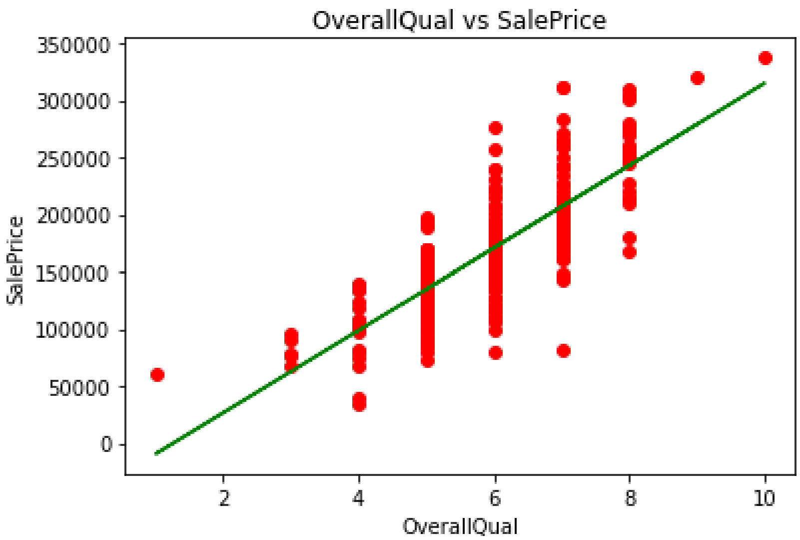
In [41]:

```
x = df4['OverallQual'].values.reshape((-1, 1))
y = df4['SalePrice']

# split data in to training and test data
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=4

# creating linear regression model
model_2 = LinearRegression().fit(X_train, y_train)

# display model coefficients and r-sqaured scores
print('Coefficient:', model_2.coef_)
print('Scores:', model_2.score(X_train, y_train), model_2.score(X_test, y_test))

# predicted housing prices
y_prediction = model_2.predict(X_test)

# model RMSE
RMSE = sqrt(mean_squared_error(y_true = y_test, y_pred = y_prediction))
print('RMSE:', RMSE)
```

```
Coefficient: [36054.40278892]
Scores: 0.6128972608954553 0.6187936765567974
RMSE: 34507.574371186674
```

In [42]:

```
plt.scatter(X_test, y_test, color = "red")
plt.plot(X_train, model_2.predict(X_train), color = "green")
plt.title("OverallQual vs SalePrice")
plt.xlabel("OverallQual")
plt.ylabel("SalePrice")
plt.show()
```



Liner Regression Model Three ([qual_space])

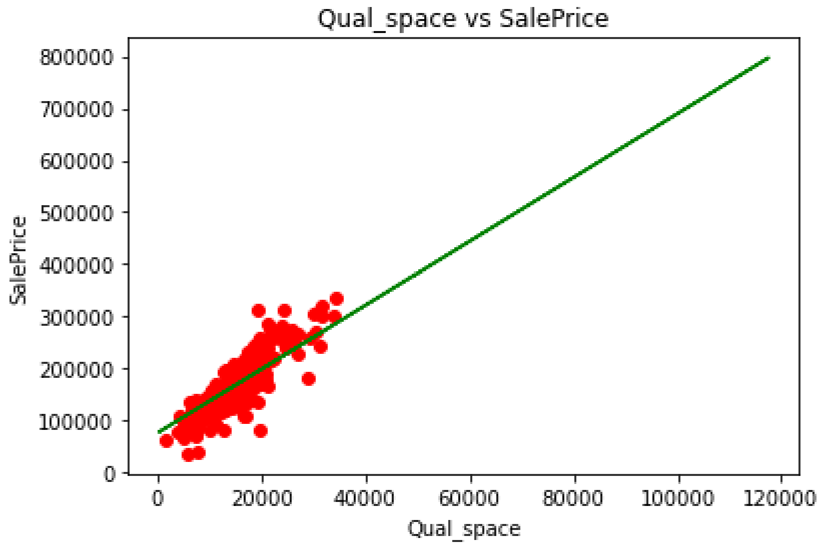
In [43]:

```
x = df4['qual_space'].values.reshape((-1, 1))
y = df4['SalePrice']

# split data in to training and test data
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=4

# creating linear regression model
model_3 = LinearRegression().fit(X_train, y_train)

# display model coefficients and r-sqaured scores
print('Coefficient:', model_3.coef_)
print('Scores:', model_3.score(X_train, y_train), model_3.score(X_test, y_test))

# predicted housing prices
y_prediction = model_3.predict(X_test)

# model RMSE
RMSE = sqrt(mean_squared_error(y_true = y_test, y_pred = y_prediction))
print(RMSE)
```

```
Coefficient: [6.14656749]
Scores: 0.6236730363612033 0.7226340157102138
29434.79286723823
```

In [44]:

```
plt.scatter(X_test, y_test, color = "red")
plt.plot(X_train, model_3.predict(X_train), color = "green")
plt.title("Qual_space vs SalePrice")
plt.xlabel("Qual_space")
plt.ylabel("SalePrice")
plt.show()
```



Liner Regression Model Four ([qual_space] and [GarageCars])

In [45]:
```
features_2 = ['qual_space', 'GarageCars']
```

In [46]:
```
x = df4[features_2]
```

In [47]:
```
# split data in to training and test data
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=4
```

In [48]:
```
# creating linear regression model
model_4 = LinearRegression().fit(X_train, y_train)

# display model coefficients and r-sqaured scores
print('Coefficient:', model_4.coef_)
print('Scores:', model_4.score(X_train, y_train), model_4.score(X_test, y_test))
```

```
Coefficient: [4.96444291e+00 2.38195881e+04]
Scores: 0.6834845496665918 0.7582239554953171
```

In [49]:
```
# predicted housing prices
y_prediction = model_4.predict(X_test)

# model RMSE
RMSE = sqrt(mean_squared_error(y_true = y_test, y_pred = y_prediction))
print(RMSE)
```

```
27481.537013139783
```

When comparing the linear regression models above, the final model using [OverallQual] and [GarageCars] produced the best fit as shown by having the lowest Root Mean Square Error (RMSE).

Testing

In [50]:
```
# create dataframe using test data from kaggle
```

```
df_test = pd.read_csv("test.csv")
```

In [51]:
```
# replace NaN values with zero for the test data
df_test = df_test.fillna(0)
```

In [52]:
```
#Repeat the process to create the 'Overall Quality' variable for the testing dataframe.

# sum 1st floor, 2nd floor, and basement square footage to get total square footage
sum_column = df_test['1stFlrSF'] + df_test['2ndFlrSF'] + df_test['TotalBsmtSF']

# multiply total square footage by overall quality to generate new predictor variable q
mult_column = sum_column*df_test['OverallQual']

# add new predictor variables to dataframe
df_test['tot_sq'] = sum_column
df_test['qual_space'] = mult_column
```

In [53]:
```
features_test = ['qual_space', 'GarageCars']
```

In [54]:
```
X = df_test[features_test]
```

In [55]:
```
test_prediction = model_4.predict(X)
```

In [56]:
```
df = pd.DataFrame({'id':df_test['Id'], 'SalePrice':test_prediction})
df.to_csv('group_5_msds_422_module_2.csv', index=False)
```