Continuing from Module 1, I began by cleaning both the training and test datasets to prepare for regression modeling. Although my training data was mostly ready, the test set required significant work. I estimated missing values in the test set using related variables and predictive imputation.

After preprocessing, I conducted exploratory data analysis (EDA). I started with a correlation matrix to shortlist important numerical features. I then explored non-linear relationships: adding a quadratic term for TotalLivArea and a logarithmic transformation of AgeAtSale. I created the following new features:

AvgSalePrice: Median sale price in each neighborhood, capturing local pricing trends.

OpenSpaceRatio: (LotArea – 1stFlrSF)/LotArea, indicating how open a lot feels.

NbhoodSpace: Median OpenSpaceRatio in the neighborhood, reflecting neighborhood spaciousness.

I discovered that MSSubClass was improperly treated as a continuous variable. Recoding it as categorical and applying one-hot encoding to all categorical variables significantly improved model performance. With data cleaned and features finalized, I split the merged dataset back into training and test sets. I then built and compared multiple regression models:

Baseline Model): My first model used only 5 variables that were highly correlated with SalePrice. It achieved an RMSE of ~0.93—an acceptable baseline but far from ideal. I realized that by eliminating a lot of features, I was missing a lot of nuances.

All-Numeric Model: Using all cleaned numeric features, this model improved performance significantly. RMSE bumped down to 0.17.

https://colab.research.google.com/drive/1mryWq2_iZNdTU_TyRtH-5n162GsY_jpn?usp=sharing

Full Model (all features): Including both numeric and one-hot encoded categorical features worsened performance. This suggested that many categorical variables introduced noise and the model was overfitting the data. RMSE 0.19.

Refined Models: I applied feature selection by removing high p-value variables and using domain knowledge to retain interpretable, less collinear features. After multiple iterations, this model yielded my best score of 0.15.

Standardized Regression: I also tested a version with standardized numeric inputs. While this did not affect predictions, it made coefficients easier to compare and interpret.

Overall, I was happy to see that AvgSalePrice, OpenSpaceRatio, AvgRmSize, NbhoodSpace – features that I added in had low p-values and high coefficients.

For validation, I used multiple goodness-of-fit metrics on the training set: RMSE, MAE, correlation, and R^2 between actual and predicted SalePrice. I also implemented 5-fold cross-validation to assess model generalizability, focusing on RMSE and R^2 across folds.

In testing, simpler models with just a few well-chosen features underperformed. On the other hand, throwing in every variable led to overfitting—basically memorizing the training data without truly understanding it. The best-performing models struck a balance: use only the features that matter and generalize well to new data. Changing the set of variables between models helped me track the changes to p-values when variables were added and removed. The coefficients helped me understand what features influenced SalePrice the most.

Kaggle user name: vibhuvanjari

Kaggle name: Vibhu Vanjari

https://colab.research.google.com/drive/1mryWq2_iZNdTU_TyRtH-5n162GsY_jpn?usp=sharing

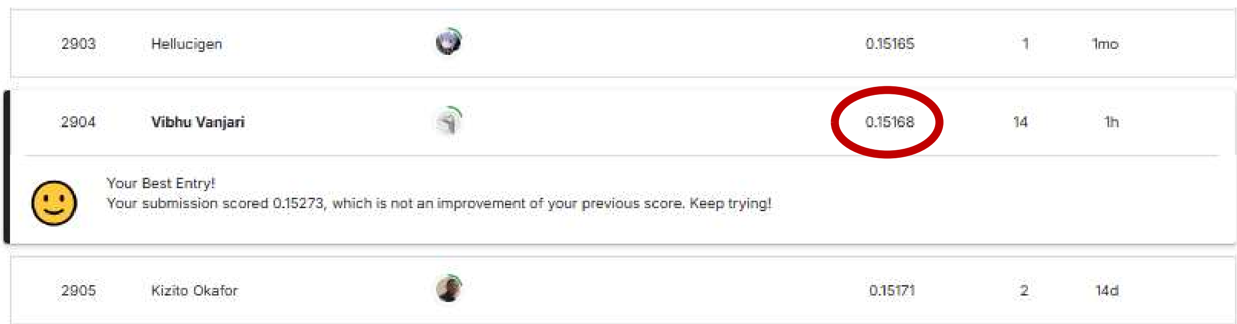
Vibhu Vanjari Module 2 Assignment 1 – House Prices

Initial EDA:

https://colab.research.google.com/drive/1Cg5IznYQKiKiPJjcd3S_TltXvsYuW5Cw#scrollTo=db95whIPqMGw

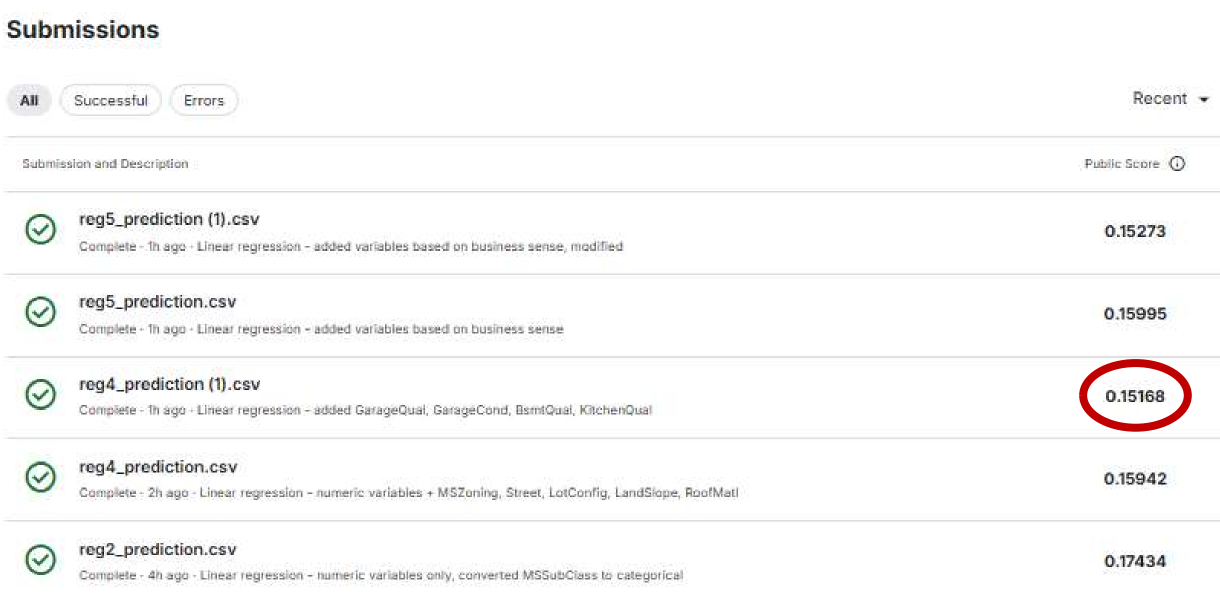
Final file (EDA + Data cleanup + Regressions):

https://colab.research.google.com/drive/1mryWq2_iZNdTU_TyRtH-5n162GsY_jpn#scrollTo=eUzhEbRHLkya



2903	Hellucigen	0.15165	1	1mo
2904	**Vibhu Vanjari**	0.15168	14	1h
2905	Kizito Okafor	0.15171	2	14d

Your Best Entry!
Your submission scored 0.15273, which is not an improvement of your previous score. Keep trying!



Submissions

All | Successful | Errors | Recent

Submission and Description	Public Score
reg5_prediction (1).csv Complete · 1h ago · Linear regression - added variables based on business sense, modified	**0.15273**
reg5_prediction.csv Complete · 1h ago · Linear regression - added variables based on business sense	**0.15995**
reg4_prediction (1).csv Complete · 1h ago · Linear regression - added GarageQual, GarageCond, BsmtQual, KitchenQual	**0.15168**
reg4_prediction.csv Complete · 2h ago · Linear regression - numeric variables + MSZoning, Street, LotConfig, LandSlope, RoofMatl	**0.15942**
reg2_prediction.csv Complete · 4h ago · Linear regression - numeric variables only, converted MSSubClass to categorical	**0.17434**

https://colab.research.google.com/drive/1mryWq2_iZNdTU_TyRtH-5n162GsY_jpn?usp=sharing

Introduction

Link to access this code - https://colab.research.google.com/drive/1gKRrXN0jYrheIwl3eefoSEj9gh3FwewI#scrollTo=OvyyD8iORVMK

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

Import modules and data files

```
# Import modules
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import numpy as np
from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, make_scorer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import cross_val_score, KFold
from sklearn.model_selection import cross_val_predict

# Figures inline and set visualization style
%matplotlib inline
sns.set()

# To ensure all columns are displayed when calling data
pd.set_option('display.max_columns', None)
```

```
df_train_original = pd.read_csv('train.csv')
df_test_original = pd.read_csv('test.csv')
```

```
df_train_original.info()
```

Show hidden output

```
#df_train_original.describe()
df_test_original.describe()
```

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF
count	1459.000000	1459.000000	1232.000000	1459.000000	1459.000000	1459.000000	1459.000000	1459.000000	1444.000000	1458.000000	1458.000000	1458.000000	1458.000000
mean	2190.000000	57.378341	68.580357	9819.161069	6.078821	5.553804	1971.357779	1983.662783	100.709141	439.203704	52.619342	554.294925	1046.117970
std	421.321334	42.746880	22.376841	4955.517327	1.436812	1.113740	30.390071	21.130467	177.625900	455.268042	176.753926	437.260486	442.898624
min	1461.000000	20.000000	21.000000	1470.000000	1.000000	1.000000	1879.000000	1950.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1825.500000	20.000000	58.000000	7391.000000	5.000000	5.000000	1953.000000	1963.000000	0.000000	0.000000	0.000000	219.250000	784.000000
50%	2190.000000	50.000000	67.000000	9399.000000	6.000000	5.000000	1973.000000	1992.000000	0.000000	350.500000	0.000000	460.000000	988.000000
75%	2554.500000	70.000000	80.000000	11517.500000	7.000000	6.000000	2001.000000	2004.000000	164.000000	753.500000	0.000000	797.750000	1305.000000
max	2919.000000	190.000000	200.000000	56600.000000	10.000000	9.000000	2010.000000	2010.000000	1290.000000	4010.000000	1526.000000	2140.000000	5095.000000

```
df_test_original.info()
```

Show hidden output

Merge and clean data

Merge data

```
# Create working datasets
df_train = df_train_original
df_test =  df_test_original

# Add a variable to both df_train and df_test indicating whether train or test dataset
df_train['TestYes'] = 0
df_test['TestYes'] = 1

# Create merged dataset used to cleanup data
df_merged = pd.concat([df_train, df_test])
```

```
df_merged.info()
```

Show hidden output

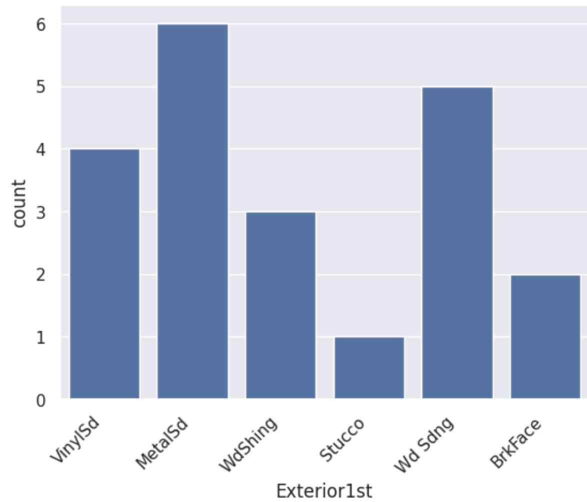
Checking missing data

Missing Exterior1st and Exterior2nd data

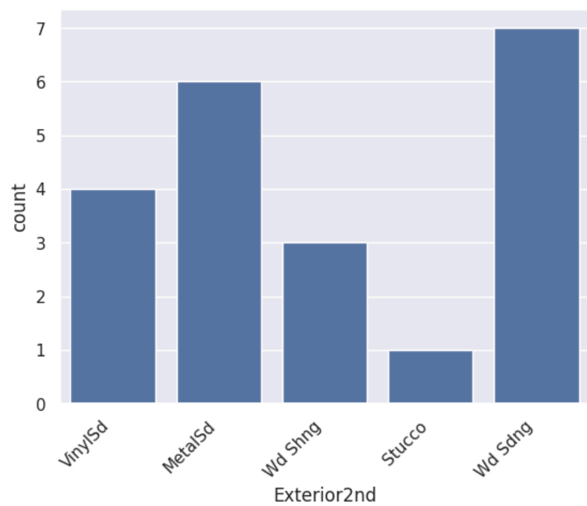
One observation ID 2152, has missing Exterior1st and Exterior2nd data. Hypothesis: the exterior covering can be determined by similar houses in the neighborhood.

```
# Filter to "Edwards" Neighborhood, "1Fam" BldgType houses, built in similar years
filtered_df = df_merged[(df_merged['Neighborhood'] == 'Edwards') & (df_merged['BldgType'] == '1Fam') & (df_merged['YearBuilt'] >= 1930) & (df_merged['YearBuilt'] <= 1945)]
```

```
# Plot distribution of Exterior1st and Exterior2nd for these
sns.countplot(x='Exterior1st', data=filtered_df.reset_index())
plt.xticks(rotation=45, ha='right')
plt.show()
```




```
# Plot distribution of Exterior1st and Exterior2nd for these
sns.countplot(x='Exterior2nd', data=filtered_df.reset_index())
plt.xticks(rotation=45, ha='right')
plt.show()
```



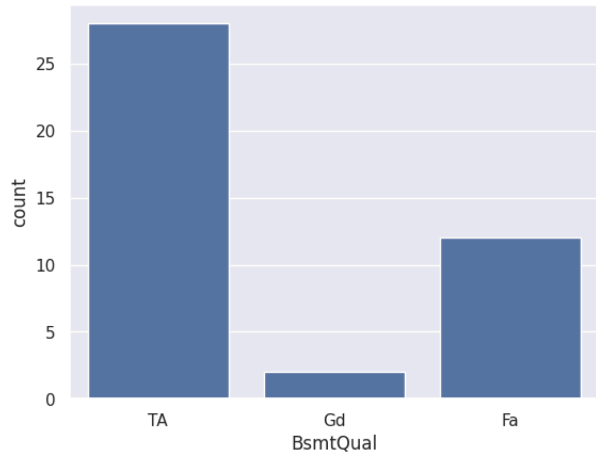
Assume that the Exterior1st is MetalSd and Exterior2nd is Wd Sdng

Missing BsmtQual data

Observation ID 2218 and 2219 have missing BsmtQual data. Hypothesis: the BsmtQual can be determined by houses built/remodeled in a similar time frame.

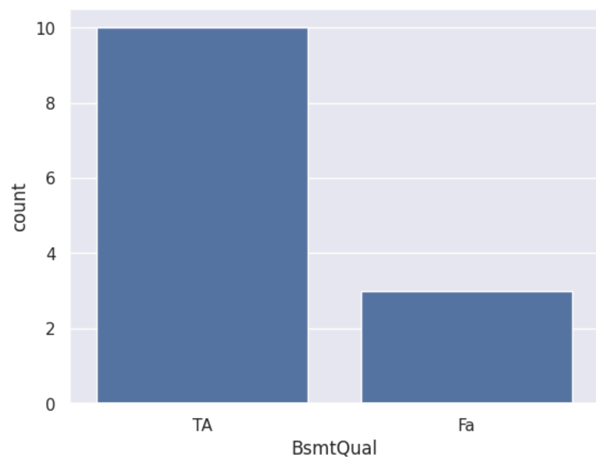
```
# checking 2218 based on YearBuilt
filtered_df = df_merged[(df_merged['YearBuilt'] >= 1890) & (df_merged['YearBuilt'] <= 1900)]
sns.countplot(x='BsmtQual', data=filtered_df.reset_index())
```

```
<Axes: xlabel='BsmtQual', ylabel='count'>
```



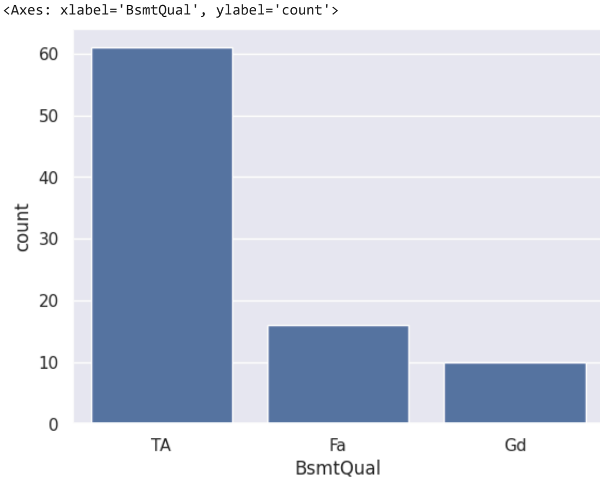

```
# checking 2218 based on YearBuilt and YearRemodAdd
filtered_df = df_merged[(df_merged['YearBuilt'] >= 1890) & (df_merged['YearBuilt'] <= 1900) & (df_merged['YearRemodAdd'] >= 1945) & (df_merged['YearRemodAdd'] <= 1955)]
sns.countplot(x='BsmtQual', data=filtered_df.reset_index())
```

```
<Axes: xlabel='BsmtQual', ylabel='count'>
```



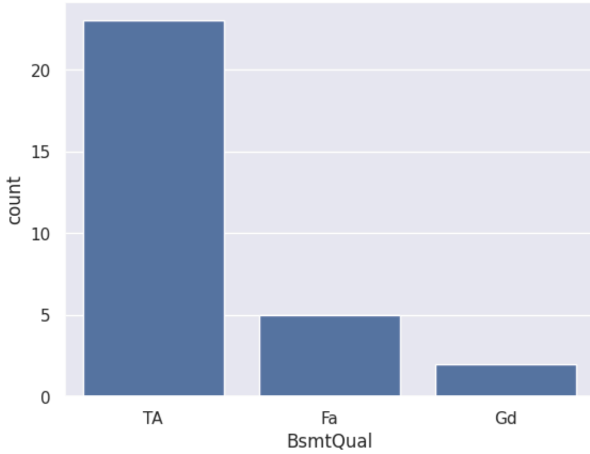

```
# checking 2219 based on YearBuilt
filtered_df = df_merged[(df_merged['YearBuilt'] >= 1905) & (df_merged['YearBuilt'] <= 1915)]
sns.countplot(x='BsmtQual', data=filtered_df.reset_index())
```

```
<Axes: xlabel='BsmtQual', ylabel='count'>
```




```
# checking 2219 based on YearBuilt and YearRemodAdd
filtered_df = df_merged[(df_merged['YearBuilt'] >= 1905) & (df_merged['YearBuilt'] <= 1915) & (df_merged['YearRemodAdd'] >= 1995) & (df_merged['YearRemodAdd'] <= 2005)]
sns.countplot(x='BsmtQual', data=filtered_df.reset_index())
```

<Axes: xlabel='BsmtQual', ylabel='count'>



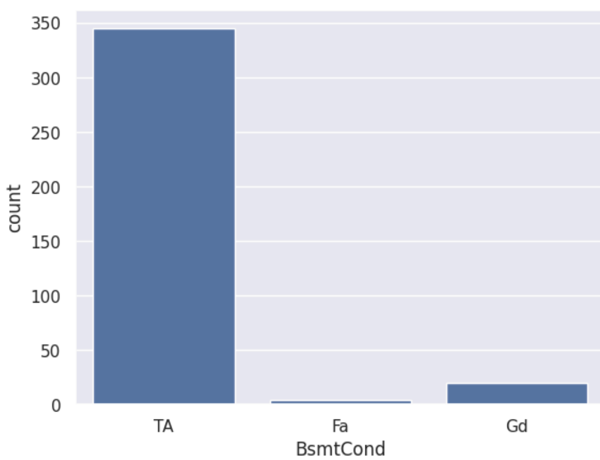
For ID 2218 and 2219, assume that the BsmtQual is TA.

Missing BsmtCond data

Observation ID 2041, 2186 and 2525 have missing BsmtCond data. Hypothesis: the BsmtQual can be determined by houses built/remodeled in a similar time frame.

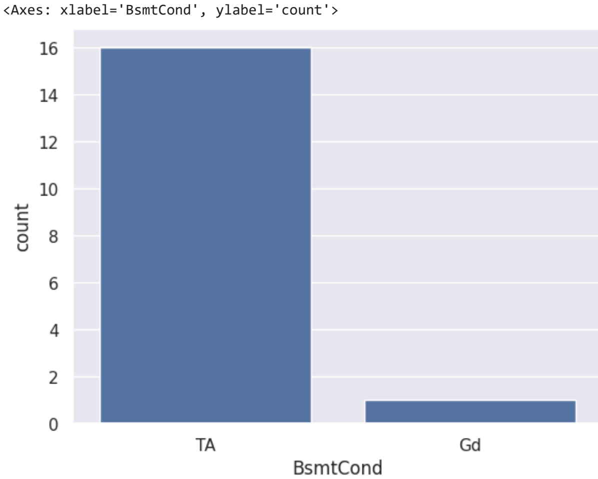
```
# checking 2041, 2186 and 2525 based on YearBuilt
filtered_df = df_merged[(df_merged['YearBuilt'] >= 1970) & (df_merged['YearBuilt'] <= 1980)]
sns.countplot(x='BsmtCond', data=filtered_df.reset_index())
```

<Axes: xlabel='BsmtCond', ylabel='count'>



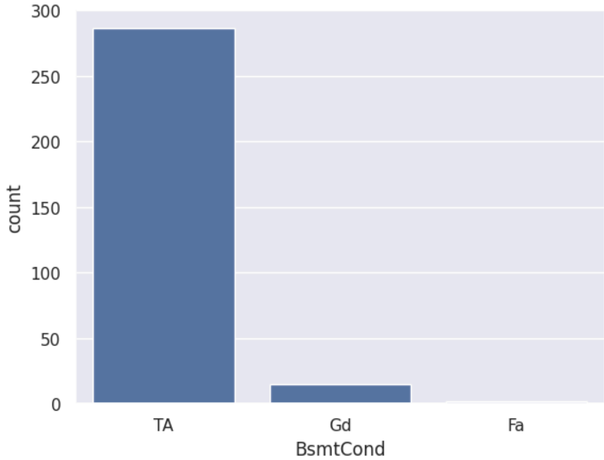

```
# checking 2041 based on YearBuilt and YearRemodAdd
filtered_df = df_merged[(df_merged['YearBuilt'] >= 1970) & (df_merged['YearBuilt'] <= 1980) & (df_merged['YearRemodAdd'] >= 2005) & (df_merged['YearRemodAdd'] <= 2010)]
sns.countplot(x='BsmtCond', data=filtered_df.reset_index())
```

<Axes: xlabel='BsmtCond', ylabel='count'>



```
# checking 2186 and 2525 based on YearBuilt and YearRemodAdd
filtered_df = df_merged[(df_merged['YearBuilt'] >= 1970) & (df_merged['YearBuilt'] <= 1980) & (df_merged['YearRemodAdd'] >= 1970) & (df_merged['YearRemodAdd'] <= 1980)]
sns.countplot(x='BsmtCond', data=filtered_df.reset_index())
```

```
<Axes: xlabel='BsmtCond', ylabel='count'>
```

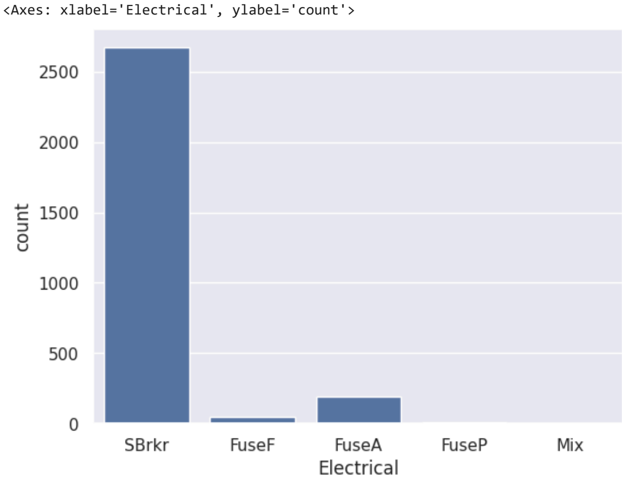


For ID 2041, 2186 and 2525 assume that the BsmtCond is TA.

Missing Electrical data

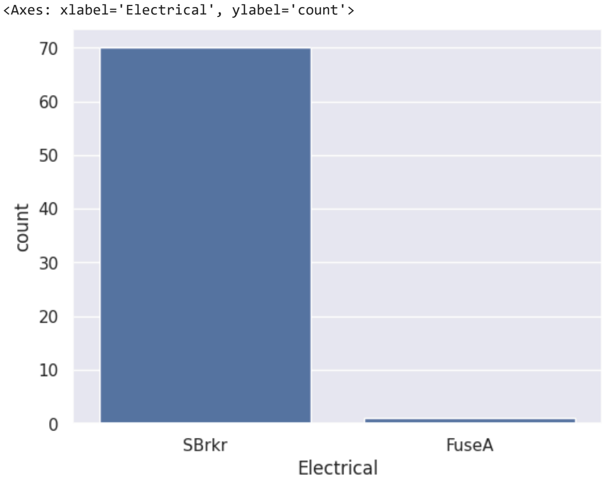
```
#Plot Electrical data
sns.countplot(x='Electrical', data=df_merged.reset_index())
```

```
<Axes: xlabel='Electrical', ylabel='count'>
```




```
#Plot Electrical data filter to neighborhood "Timber"
filtered_df = df_merged[df_merged['Neighborhood'] == 'Timber']
sns.countplot(x='Electrical', data=filtered_df.reset_index())
```

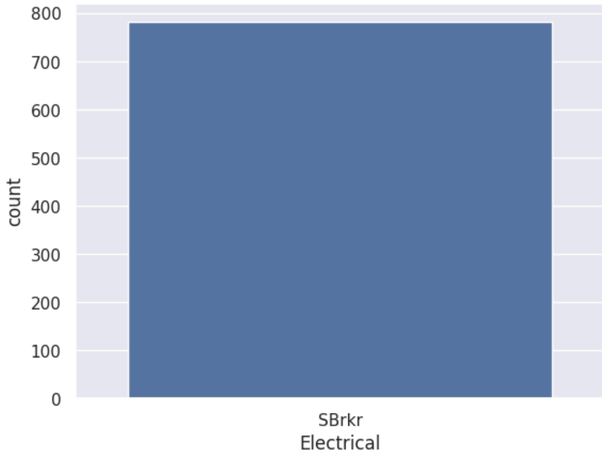
```
<Axes: xlabel='Electrical', ylabel='count'>
```




```
#Plot Electrical data filter to YearBuilt 2000-2010
filtered_df = df_merged[(df_merged['YearBuilt'] >= 2000) & (df_merged['YearBuilt'] <= 2010)]
```

```
sns.countplot(x='Electrical', data=filtered_df.reset_index())
```

<Axes: xlabel='Electrical', ylabel='count'>

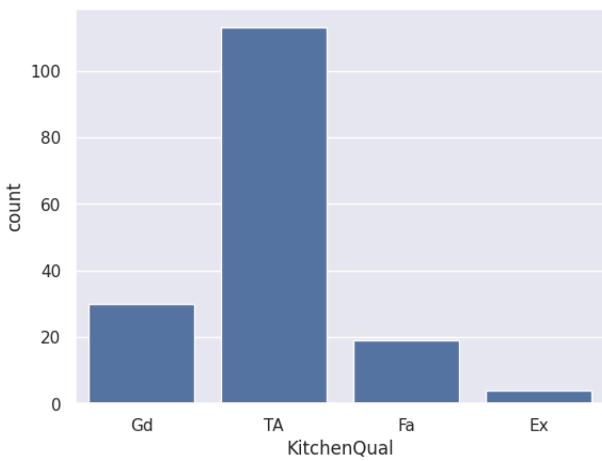


For ID 1380 assume that the Electrical is SBrkr.

Missing KitchenQual data

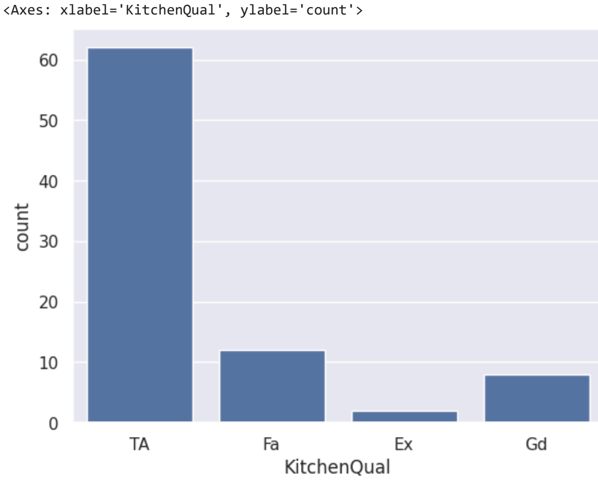
```
# checking 1556 based on YearBuilt
filtered_df = df_merged[(df_merged['YearBuilt'] >= 1910) & (df_merged['YearBuilt'] <= 1920)]
sns.countplot(x='KitchenQual', data=filtered_df.reset_index())
```

<Axes: xlabel='KitchenQual', ylabel='count'>




```
# checking 1556 based on YearBuilt and YearRemodAdd
filtered_df = df_merged[(df_merged['YearBuilt'] >= 1910) & (df_merged['YearBuilt'] <= 1920) & (df_merged['YearRemodAdd'] >= 1945) & (df_merged['YearRemodAdd'] <= 1955)]
sns.countplot(x='KitchenQual', data=filtered_df.reset_index())
```

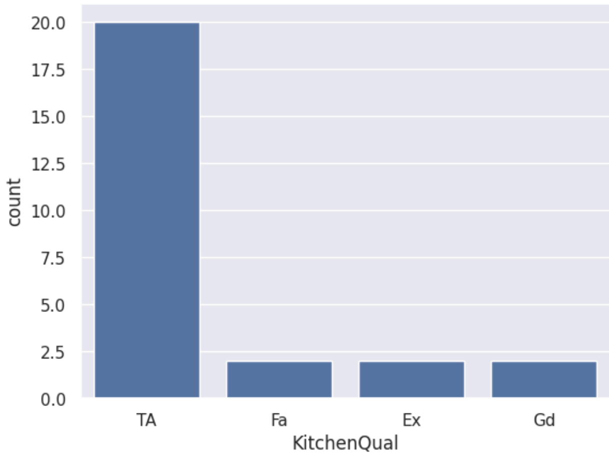
<Axes: xlabel='KitchenQual', ylabel='count'>




```
# checking 1556 based on YearBuilt, YearRemodAdd and OverallQual
filtered_df = df_merged[(df_merged['YearBuilt'] >= 1910) & (df_merged['YearBuilt'] <= 1920) & (df_merged['YearRemodAdd'] >= 1945) & (df_merged['YearRemodAdd'] <= 1955) & (df_mer
```

```
sns.countplot(x='KitchenQual', data=filtered_df.reset_index())
```

```
<Axes: xlabel='KitchenQual', ylabel='count'>
```

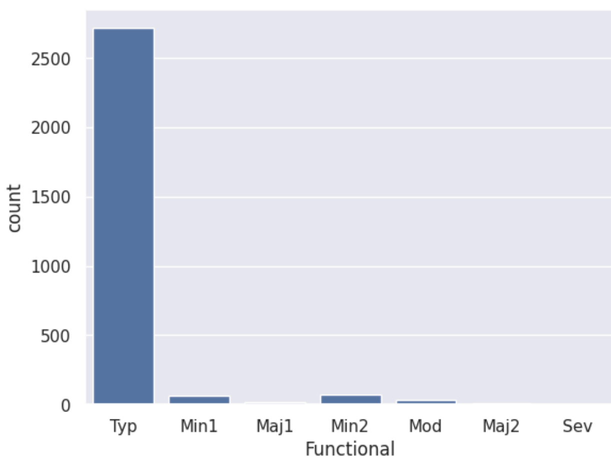


For ID 1556 assume that the KitchenQual is TA.

Missing Functional data

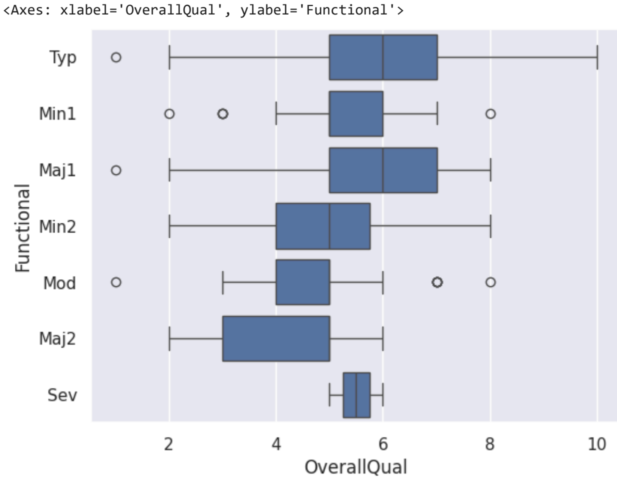
```
# check distribution of Functional data
sns.countplot(x='Functional', data=df_merged.reset_index())
```

```
<Axes: xlabel='Functional', ylabel='count'>
```




```
# plot Functional vs OverallQual
sns.boxplot(x='OverallQual', y='Functional', data=df_merged.reset_index())
```

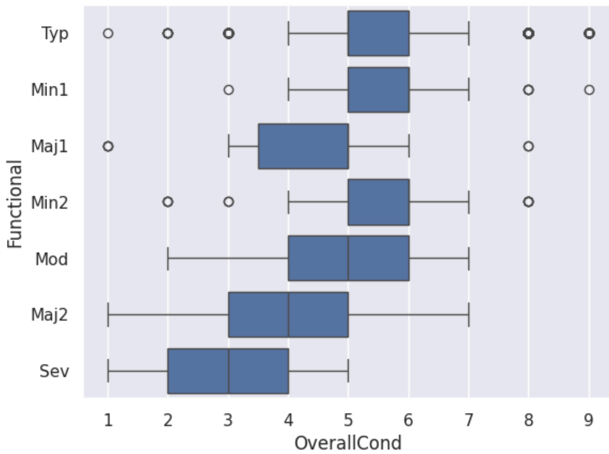
```
<Axes: xlabel='OverallQual', ylabel='Functional'>
```



Doesn't seem like a strong correlation between Functional and OverallQual.

```
# plot Functional vs OverallCond
sns.boxplot(x='OverallCond', y='Functional', data=df_merged.reset_index())
```

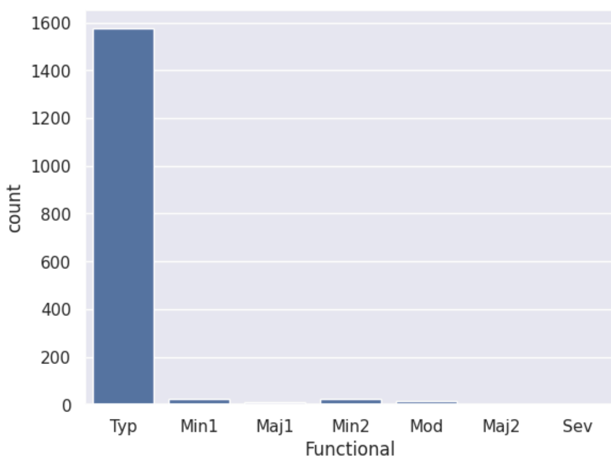
<Axes: xlabel='OverallCond', ylabel='Functional'>



Seems like a stronger correlation between Functional and OverallCond.

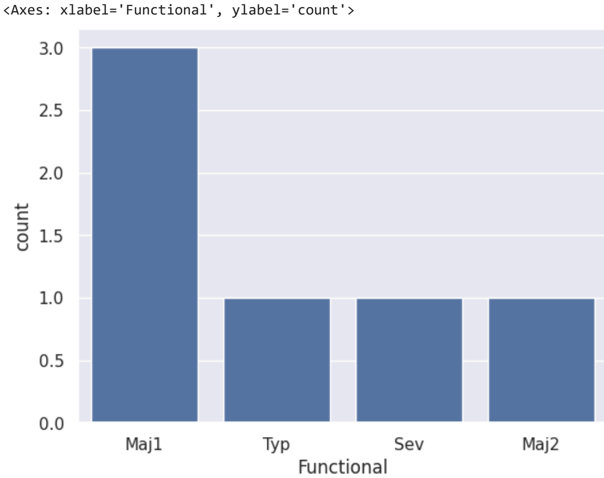
```
# check distribution of Functional data when OverallCond == 5
filtered_df = df_merged[df_merged['OverallCond'] == 5]
sns.countplot(x='Functional', data=filtered_df.reset_index())
```

<Axes: xlabel='Functional', ylabel='count'>




```
# check distribution of Functional data when OverallCond == 1
filtered_df = df_merged[df_merged['OverallCond'] == 1]
sns.countplot(x='Functional', data=filtered_df.reset_index())
```

<Axes: xlabel='Functional', ylabel='count'>



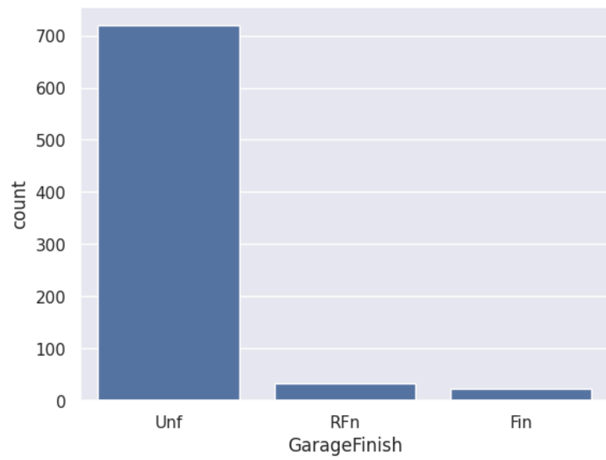
For ID 2217 assume that the Functional is "Typ" for ID 2474 assume that Functional is "Maj1".

Missing garage data

ID 2127 and 2577 have multiple missing garage data elements

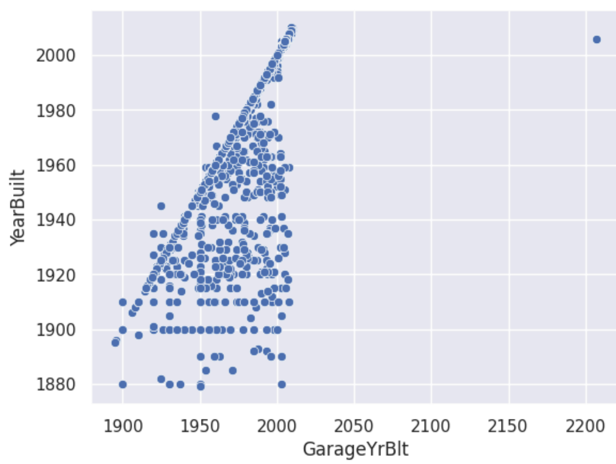
```
# Check distribution of GarageFinish for "Detchd" GarageType
filtered_df = df_merged[df_merged['GarageType'] == 'Detchd']
sns.countplot(x='GarageFinish', data=filtered_df.reset_index())
```

```
<Axes: xlabel='GarageFinish', ylabel='count'>
```




```
# Check relationship between GarageYrBlt and YearBuilt, exclude observations with no garage
filtered_df = df_merged[df_merged['GarageType'] != 'NG']
sns.scatterplot(x='GarageYrBlt', y='YearBuilt', data=filtered_df.reset_index())
```

```
<Axes: xlabel='GarageYrBlt', ylabel='YearBuilt'>
```




```
# Filtering down to houses that were remodeled and have a detached garage
filtered_df = df_merged[df_merged['GarageType'] != 'NG']
filtered_df = filtered_df[filtered_df['YearBuilt'] != filtered_df['YearRemodAdd']]
filtered_df = filtered_df[filtered_df['GarageType'] == 'Detchd']

# check what percentage of filtered_df have GarageYrBlt == YearBuilt vs whath percentage have GarageYrBlt == YearRemodAdd
total = len(filtered_df)

same_as_yearbuilt = (filtered_df['GarageYrBlt'] == filtered_df['YearBuilt']).sum()
same_as_remod = (filtered_df['GarageYrBlt'] == filtered_df['YearRemodAdd']).sum()
different_from_both = ((filtered_df['GarageYrBlt'] != filtered_df['YearBuilt']) & (filtered_df['GarageYrBlt'] != filtered_df['YearRemodAdd'])).sum()

print(f"GarageYrBlt == YearBuilt: {same_as_yearbuilt / total:.2%}")
print(f"GarageYrBlt == YearRemodAdd: {same_as_remod / total:.2%}")
print(f"GarageYrBlt != YearBuilt and != YearRemodAdd: {different_from_both / total:.2%}")
```

```
GarageYrBlt == YearBuilt: 44.21%
GarageYrBlt == YearRemodAdd: 4.25%
GarageYrBlt != YearBuilt and != YearRemodAdd: 51.54%
```

```
# Hard to accurately predict when the garages were built, to make further estimates assume the garage was built somewhere in between when the house was built and remodelled.

# Drop rows from filtered_df with missing GarageYrBlt
filtered_df = filtered_df.dropna(subset=['GarageYrBlt'])

# Calculate age of house at sale
filtered_df['AgeAtSale'] = filtered_df['YrSold'] - filtered_df['YearBuilt']

# Calculate age of house when garage was built
filtered_df['AgeAtGarBlt'] = filtered_df['GarageYrBlt'] - filtered_df['YearBuilt']

# Calcualte % of house age passed when garage was built
filtered_df['%AgeAtGarBlt'] = filtered_df['AgeAtGarBlt'] / filtered_df['AgeAtSale']
```

```
# Provide descriptive statistics for %AgeAtGarBlt include median
filtered_df['%AgeAtGarBlt'].describe()
```

	%AgeAtGarBlt
count	515.000000
mean	0.243797
std	0.310625
min	-0.202703
25%	0.000000
50%	0.056338
75%	0.482140
max	0.989899

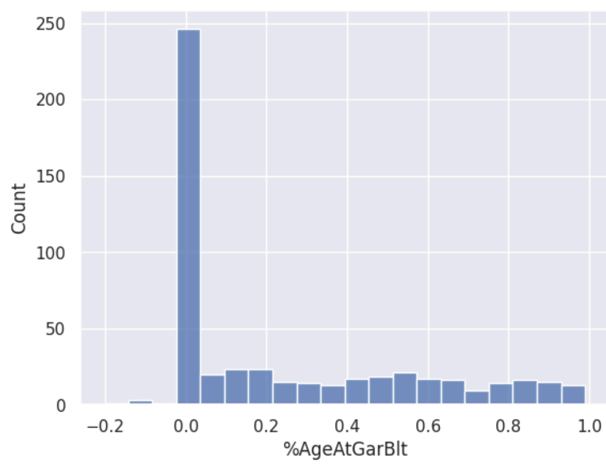
dtype: float64

```
# Find median of %AgeAtGarBlt
med_GartoHouseBlt = filtered_df['%AgeAtGarBlt'].median()
med_GartoHouseBlt
```

```
0.056338028169014086
```

```
# Plot %AgeAtGarBlt, divide into 20 bars
sns.histplot(filtered_df['%AgeAtGarBlt'], bins=20)
```

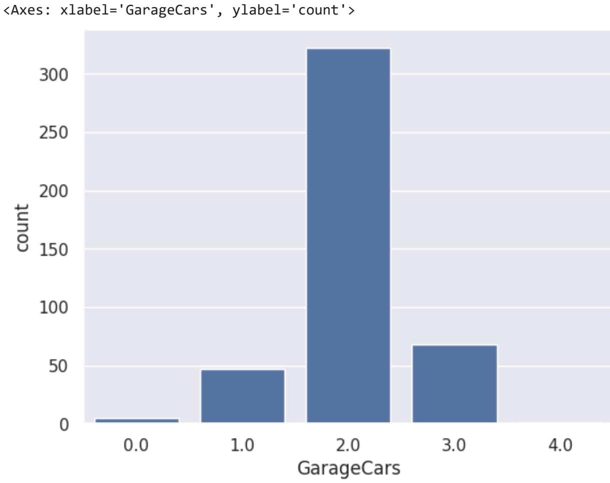
```
<Axes: xlabel='%AgeAtGarBlt', ylabel='Count'>
```



The median garage is built (5.6% of age of house at sale) years after the house is built.

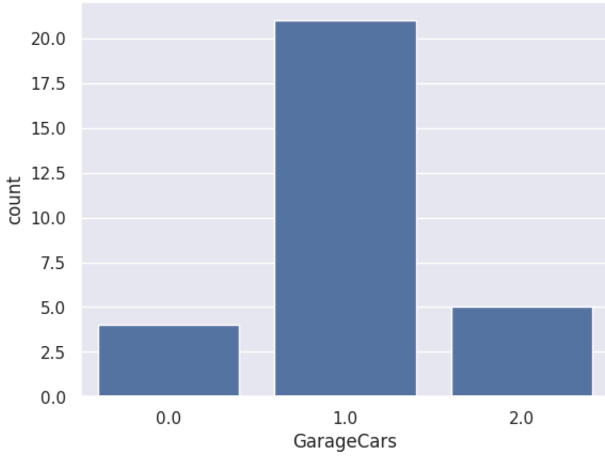
```
# Estimating number of GarageCars for ID 2577

# Distribution of GarageCars for BldgType == "1Fam", HouseStyle == "2Story", BedroomAbvGr == 3
filtered_df = df_merged[(df_merged['BldgType'] == '1Fam') & (df_merged['HouseStyle'] == '2Story') & (df_merged['BedroomAbvGr'] == 3)]
sns.countplot(x='GarageCars', data=filtered_df.reset_index())
```

```
<Axes: xlabel='GarageCars', ylabel='count'>
```




```
# Distribution of GarageCars for BldgType == "1Fam", HouseStyle == "2Story", BedroomAbvGr == 3, Neighborhood == "IDOTRR"
filtered_df = df_merged[(df_merged['BldgType'] == '1Fam') & (df_merged['BedroomAbvGr'] == 3) & (df_merged['Neighborhood'] == 'IDOTRR')]
sns.countplot(x='GarageCars', data=filtered_df.reset_index())
```

<Axes: xlabel='GarageCars', ylabel='count'>




```
# Estimating area of a single car garage in IDOTRR
filtered_df = df_merged[(df_merged['BldgType'] == '1Fam') & (df_merged['BedroomAbvGr'] == 3) & (df_merged['Neighborhood'] == 'IDOTRR')]
filtered_df['GarageArea'].describe()
```

	GarageArea
count	30.000000
mean	318.566667
std	200.683948
min	0.000000
25%	216.000000
50%	284.000000
75%	388.500000
max	720.000000

dtype: float64

```
filtered_df['GarageArea'].median()
```

284.0

For ID 2127 and 2577:

- Set GarageFinish to "Unf" since that is most popular for "Detchd" GarageType
- Set GarageYrBlt to be YearBuilt + 5.6%*(YrSold - YearBuilt) - this is based on the median value med_GartoHouseBlt
- Set GarageQual and GarageCond to "TA"

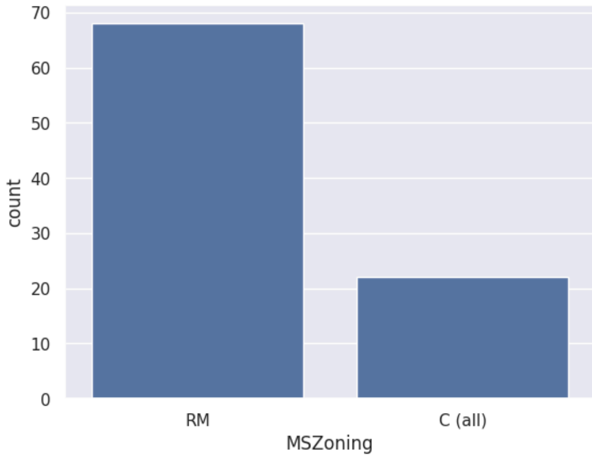
For ID 2577:

- Set GarageCars to 1
- Set GarageArea to 300 (between mean and median)

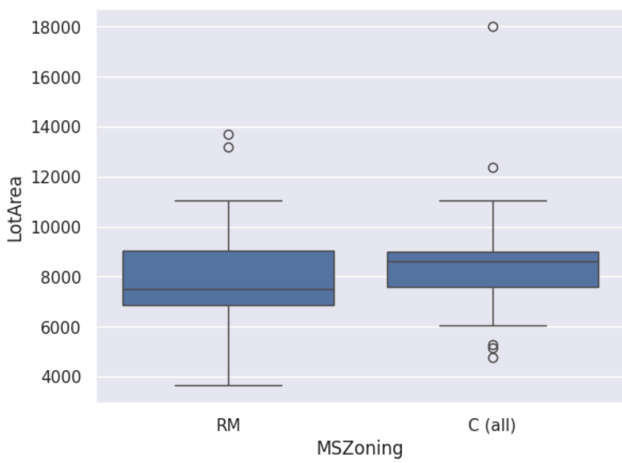
Missing MSZoning data

```
# Check distribution of MSZoning data in IDOTRR neighborhood
filtered_df = df_merged[df_merged['Neighborhood'] == 'IDOTRR']
sns.countplot(x='MSZoning', data=filtered_df.reset_index())
```

<Axes: xlabel='MSZoning', ylabel='count'>



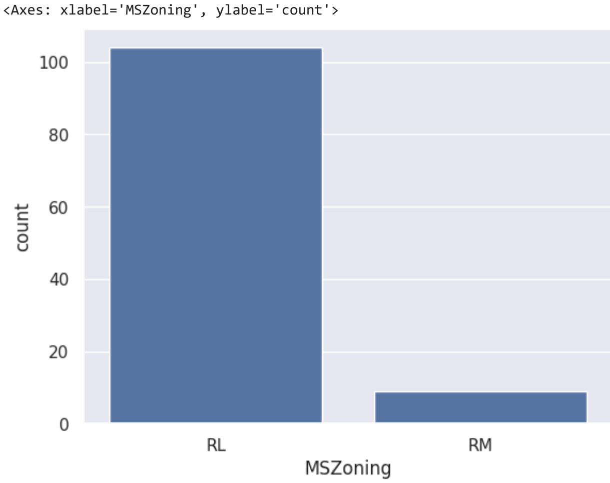

```
# In IDOTRR neighborhood, find distribution of LotArea for MSZoning type "RM" and "C (all)"
filtered_df = df_merged[(df_merged['Neighborhood'] == 'IDOTRR')]
sns.boxplot(x='MSZoning', y='LotArea', data=filtered_df.reset_index())
plt.show()
```



Id 2251, 1916 and 2217 are some of the largest lots in the IDOTRR neighborhood. Looking at two metrics of population density 1) BedroomAbvGr/LotArea and 2) 1stFlrSF/LotArea, they should be classified as "C (all)".

```
# Check distribution of MSZoning data in Mitchel neighborhood
filtered_df = df_merged[df_merged['Neighborhood'] == 'Mitchel']
sns.countplot(x='MSZoning', data=filtered_df.reset_index())
```

<Axes: xlabel='MSZoning', ylabel='count'>



Looking at two metrics of population density 1) BedroomAbvGr/LotArea and 2) 1stFlrSF/LotArea, Id 2905 should be classified "RL".

Clean data

```
# Removing features where data is hard to predict
df_merged = df_merged.drop(['Alley', 'PoolQC', 'Fence', 'MiscFeature', 'Utilities', 'LotFrontage'], axis=1)
# Small number have observations for Alley, PoolQC, Fence and MiscFeatures, doesn't make sense to predict these for all the rest
# LotFrontage is a numeric variable that has lots of missing values
# Removed Utilities because all but one observation had Utilities = AllPub, wouldn't make a significant difference on prediciton

# Modifying FireplaceQu missing values to indicate "NF" to indicate "No Fireplace"
df_merged['FireplaceQu'] = df_merged['FireplaceQu'].fillna('NF')

# Modifying MasVnrArea and MasVnrType missing values to indicate "0" and "None" respectively.
# Assumption: if these values are missing there is no masonry veneer.
df_merged['MasVnrType'] = df_merged['MasVnrType'].fillna('None')
df_merged['MasVnrArea'] = df_merged['MasVnrArea'].fillna(0)

# When GarageArea is 0 categorical variables GarageType, GarageFinish, GarageQual and GarageCond should be set to "NG" to indicate "No Garage"
columns_to_update = ['GarageType', 'GarageFinish', 'GarageQual', 'GarageCond']
df_merged.loc[df_merged['GarageArea'] == 0, columns_to_update] = 'NG'

# When GarageArea is 0 set GarageYrBlt to 0
df_merged.loc[df_merged['GarageArea'] == 0, 'GarageYrBlt'] = 0

# Set BsmTExpsoure to "No" for missing values if all square footage in the basement is unfinished
df_merged.loc[(df_merged['BsmtFinSF1'] == 0) & (df_merged['BsmtFinSF2'] == 0) & (df_merged['BsmtUnfSF'] > 0) & (df_merged['BsmtExposure'].isna()), 'BsmtExposure'] = 'No'

# Assume if TotalBsmtSF is missing, there is no basement
df_merged['TotalBsmtSF'] = df_merged['TotalBsmtSF'].fillna(0)

# When TotalBsmtSF is 0 categorical variables BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1 and BsmtFinType2 should be set to "NB" to indicate "No Basement"
columns_to_update = ['BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2']
df_merged.loc[df_merged['TotalBsmtSF'] == 0, columns_to_update] = 'NB'

# When TotalBsmtSF is 0 numerical variables BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, BsmtFullBath and BsmtHalfBath should be set to 0.
columns_to_update = ['BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'BsmtFullBath', 'BsmtHalfBath']
df_merged.loc[df_merged['TotalBsmtSF'] == 0, columns_to_update] = 0

# Set missing values of BsmtFinType2 to "Unf" if BsmtFinType1 isn't "NB"
df_merged.loc[(df_merged['BsmtFinType1'] != 'NB') & (df_merged['BsmtFinType2'].isna()), 'BsmtFinType2'] = 'Unf'
```

Specific fixes in test data based on "Checking missing data" section

```
# For Id = 2152, set Exterior1st to "MetalSd" and Exterior2nd to "Wd Sdng"
df_merged.loc[df_merged['Id'] == 2152, 'Exterior1st'] = 'MetalSd'
df_merged.loc[df_merged['Id'] == 2152, 'Exterior2nd'] = 'Wd Sdng'

# For Id = 2218 and 2219, set BsmtQual to "TA"
df_merged.loc[df_merged['Id'] == 2218, 'BsmtQual'] = 'TA'
df_merged.loc[df_merged['Id'] == 2219, 'BsmtQual'] = 'TA'

# For Id = 2041, 2186 and 2525, set BsmtCond to "TA"
df_merged.loc[df_merged['Id'] == 2041, 'BsmtCond'] = 'TA'
df_merged.loc[df_merged['Id'] == 2186, 'BsmtCond'] = 'TA'
df_merged.loc[df_merged['Id'] == 2525, 'BsmtCond'] = 'TA'

# For Id = 1380, set Electrical to "SBrkr"
df_merged.loc[df_merged['Id'] == 1380, 'Electrical'] = 'SBrkr'

# For Id = 1556, set KitchenQual to "TA"
df_merged.loc[df_merged['Id'] == 1556, 'KitchenQual'] = 'TA'

# For Id 2217 set Functional to "Typ" for ID 2474 set Functional to "Maj1".
df_merged.loc[df_merged['Id'] == 2217, 'Functional'] = 'Typ'
df_merged.loc[df_merged['Id'] == 2474, 'Functional'] = 'Maj1'

# For Id 2593 set GarageYrBlt to 2007 same as YearRemodAdd, data initially has this as 2207.
df_merged.loc[df_merged['Id'] == 2593, 'GarageYrBlt'] = 2007

# For Id 2127 and 2577 set GarageFinish to "Unf"
df_merged.loc[df_merged['Id'] == 2127, 'GarageFinish'] = 'Unf'
df_merged.loc[df_merged['Id'] == 2577, 'GarageFinish'] = 'Unf'

# For Id 2127 and 2577 set GarageQual to "TA" and set GarageCond to "TA"
df_merged.loc[df_merged['Id'] == 2127, 'GarageQual'] = 'TA'
df_merged.loc[df_merged['Id'] == 2127, 'GarageCond'] = 'TA'
df_merged.loc[df_merged['Id'] == 2577, 'GarageQual'] = 'TA'
df_merged.loc[df_merged['Id'] == 2577, 'GarageCond'] = 'TA'

# For Id 2577 set GarageCars to 1 and GarageArea to 300
df_merged.loc[df_merged['Id'] == 2577, 'GarageCars'] = 1
df_merged.loc[df_merged['Id'] == 2577, 'GarageArea'] = 300

# For Id 2127 and 2577 set GarageYrBlt to be YearBuilt + 5.6%*(YrSold - YearBuilt) - this is based on the median value med_GartoHouseBlt
df_merged.loc[df_merged['Id'] == 2127, 'GarageYrBlt'] = df_merged.loc[df_merged['Id'] == 2127, 'YearBuilt'] + med_GartoHouseBlt * (df_merged.loc[df_merged['Id'] == 2127, 'YrSol
df_merged.loc[df_merged['Id'] == 2577, 'GarageYrBlt'] = df_merged.loc[df_merged['Id'] == 2577, 'YearBuilt'] + med_GartoHouseBlt * (df_merged.loc[df_merged['Id'] == 2577, 'YrSol

# For Id 2490 set SaleType to WD
df_merged.loc[df_merged['Id'] == 2490, 'SaleType'] = 'WD'

# For Id 2905 set MSZoning to "RL"
df_merged.loc[df_merged['Id'] == 2905, 'MSZoning'] = 'RL'

# For Id 1916, 2217, 2251 set MSZoning to "C (all)"
df_merged.loc[df_merged['Id'] == 1916, 'MSZoning'] = 'C (all)'
```

```
df_merged.loc[df_merged['Id'] == 2217, 'MSZoning'] = 'C (all)'
df_merged.loc[df_merged['Id'] == 2251, 'MSZoning'] = 'C (all)'


# Saving cleaned up data to a file
df_merged.to_csv('df_merged', index=False)
```

Creating additional features

```
# Variable TotalLivArea is a measure of all living area under and above ground level
df_merged['TotalLivArea'] = df_merged['GrLivArea'] + df_merged['TotalBsmtSF']

# Variable TotalBaths is a cumulative variable of all baths, assuming half baths are equivalent to 0.5x a full bathroom
df_merged['TotalBaths'] = df_merged['BsmtFullBath'] + (df_merged['BsmtHalfBath'] / 2) + df_merged['FullBath'] + (df_merged['HalfBath'] / 2)

# AvgRmSize provides information on how spacious rooms above ground are, this can be used as a proxy for general spaciousness
df_merged['AvgRmSize'] = df_merged['GrLivArea'] / df_merged['TotRmsAbvGrd']

# AgeAtSale provides information on how old the house was when the sale occurred
df_merged['AgeAtSale'] = df_merged['YrSold'] - df_merged['YearBuilt']

# AgeSinceRemod provides information on how many years since last remodel
df_merged['AgeSinceRemod'] = df_merged['YrSold'] - df_merged['YearRemodAdd']


# AvgSalePrice provides information on the median sale price of houses in the neighborhood
df_merged['AvgSalePrice'] = df_merged.groupby('Neighborhood')['SalePrice'].transform('median')

# OpenSpaceRatio provides information on how much open space each house has, higher is more space
df_merged['OpenSpaceRatio'] = (df_merged['LotArea'] - df_merged['1stFlrSF']) / df_merged['LotArea']

#NbhoodSpace provides the median OpenSpaceRatio per neighborhood, higher is more space
df_merged['NbhoodSpace'] = df_merged.groupby('Neighborhood')['OpenSpaceRatio'].transform('median')


# Adding a quadratic term for TotalLivArea
df_merged['TotalLivArea_squared'] = df_merged['TotalLivArea'] ** 2

#Adding a logarithmic term for AgeAtSale
epsilon = 0.001
df_merged['AgeAtSale_log'] = np.log1p(df_merged['AgeAtSale'] + epsilon)


# MSSubClass Dtype should be changed to object
df_merged['MSSubClass'] = df_merged['MSSubClass'].astype(str)


# Create a list of numeric_features
numeric_features = df_merged.select_dtypes(include=['number']).columns.tolist()
numeric_features.remove('Id')
numeric_features.remove('SalePrice')
numeric_features.remove('TestYes')
print(numeric_features)
```

```
['LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQua
```

```
# Create a list of categorical_features
categorical_features = df_merged.select_dtypes(include=['object']).columns.tolist()
print(categorical_features)
```

```
['MSSubClass', 'MSZoning', 'Street', 'LotShape', 'LandContour', 'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'RoofStyle',
```

```
len(numeric_features)  + len(categorical_features)
```

```
83
```

```
df_merged.info()
```

Show hidden output

```
# One-hot encode categorical data
df_merged_encoded = pd.get_dummies(df_merged, columns=categorical_features, drop_first=True)
df_merged_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2919 entries, 0 to 1458
Columns: 271 entries, Id to SaleCondition_Partial
dtypes: bool(224), float64(19), int64(28)
memory usage: 1.7 MB
```

```
len(numeric_features)  + len(df_merged_encoded.select_dtypes(include=['bool']).columns.tolist())
```

```
268
```

```
# Create a list of boolean_features representing the categorical variables
boolean_features = df_merged_encoded.select_dtypes(include=['bool']).columns.tolist()
print(boolean_features)
```

```
['MSSubClass_150', 'MSSubClass_160', 'MSSubClass_180', 'MSSubClass_190', 'MSSubClass_20', 'MSSubClass_30', 'MSSubClass_40', 'MSSubClass_45', 'MSSubClass_50', 'MSSubClass_60
```

Split merged and cleaned data into test and train datasets

```
# Split merged dataset df_merged into df_train and df_test based on the TestYes feature
df_train = df_merged_encoded[df_merged_encoded['TestYes'] == 0]
df_test = df_merged_encoded[df_merged_encoded['TestYes'] == 1]

# Dropping TestYes feature from both datasets
df_train = df_train.drop(['TestYes'], axis=1)
df_test = df_test.drop(['TestYes'], axis=1)
```

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1460 entries, 0 to 1459
Columns: 270 entries, Id to SaleCondition_Partial
dtypes: bool(224), float64(19), int64(27)
memory usage: 855.5 KB
```

Regression Analysis

Simple linear regression 2

All numeric features

[ ] ↳ 4 cells hidden

Simple linear regression 3

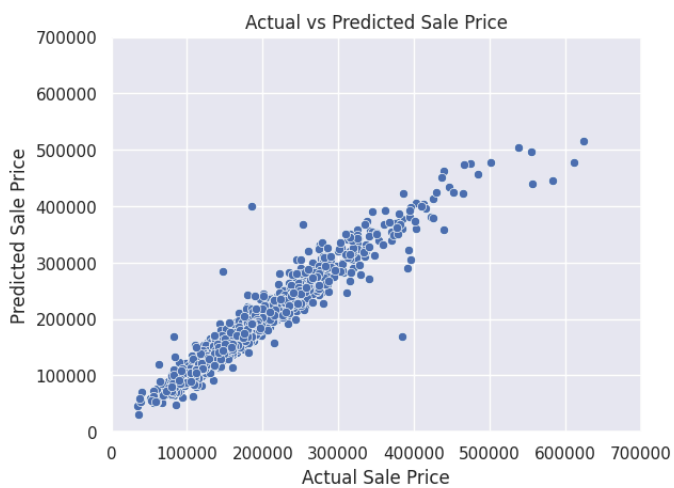
All features - numeric and categorical

```
# Select the features and target variable
features = numeric_features + boolean_features
target = 'SalePrice'

# Create feature and target DataFrames for training
X = df_train[features].copy()
y = df_train[target]

# Convert boolean features to int (0 or 1)
for feature in boolean_features:
    X[feature] = X[feature].astype(int)

# Add constant term
X = sm.add_constant(X)

# Create and fit model
reg3 = sm.OLS(y, X)
results = reg3.fit()

print(results.summary())
```

LotConfig_Inside	-1655.4995	1759.431	-0.941	0.347	-5107.377	1796.378
LandSlope_Mod	8906.4127	4016.638	2.217	0.027	1026.058	1.68e+04
LandSlope_Sev	-3.614e+04	1.16e+04	-3.107	0.002	-5.9e+04	-1.33e+04
Neighborhood_Blueste	-5.227e+04	6.29e+04	-0.831	0.406	-1.76e+05	7.11e+04
Neighborhood_BrDale	-9.619e+04	1.05e+05	-0.919	0.358	-3.01e+05	1.09e+05
Neighborhood_BrkSide	-4.551e+04	4.95e+04	-0.919	0.358	-1.43e+05	5.17e+04
Neighborhood_ClearCr	5.87e+04	7.31e+04	0.803	0.422	-8.47e+04	2.02e+05
Neighborhood_CollgCr	4.898e+04	6.19e+04	0.792	0.429	-7.24e+04	1.7e+05
Neighborhood_Crawfor	7.734e+04	6.91e+04	1.120	0.263	-5.82e+04	2.13e+05
Neighborhood_Edwards	-5.979e+04	4.94e+04	-1.209	0.227	-1.57e+05	3.72e+04
Neighborhood_Gilbert	3.284e+04	4.27e+04	0.769	0.442	-5.1e+04	1.17e+05
Neighborhood_IDOTRR	-7.47e+04	7.62e+04	-0.980	0.327	-2.24e+05	7.49e+04
Neighborhood_MeadowV	-1.4e+05	1.39e+05	-1.007	0.314	-4.13e+05	1.33e+05
Neighborhood_Mitchel	-1.826e+04	4598.245	-3.970	0.000	-2.73e+04	-9234.671
Neighborhood_NAmes	-3.257e+04	2.37e+04	-1.375	0.169	-7.9e+04	1.39e+04

```
# Add predictions to the training dataset
df_train['reg3_SalePrice'] = results.predict(sm.add_constant(df_train[features]))

# Calculate correlation between actual and predicted SalePrice
correlation = df_train['SalePrice'].corr(df_train['reg3_SalePrice'])
rmse = np.sqrt(mean_squared_error(df_train['SalePrice'], df_train['reg3_SalePrice']))
mae = mean_absolute_error(df_train['SalePrice'], df_train['reg3_SalePrice'])
r2 = r2_score(df_train['SalePrice'], df_train['reg3_SalePrice'])

# Print the correlation
print(f"Correlation: {correlation}")
print(f"RMSE: {rmse:,.0f}")
print(f"MAE: {mae:,.0f}")
print(f"R² Score: {r2:.4f}")

# Create a scatter plot to visualize the relationship
sns.scatterplot(x='SalePrice', y='reg3_SalePrice', data=df_train)
plt.title('Actual vs Predicted Sale Price')
plt.xlabel('Actual Sale Price')
plt.ylabel('Predicted Sale Price')
plt.xlim(0, 700000)
plt.ylim(0, 700000)
plt.show()
```

```
Correlation: 0.9659937165044892
RMSE: 20,534
MAE: 12,933
R² Score: 0.9331
```




```
# Add predictions to test dataset
df_test['reg3_SalePrice'] = results.predict(sm.add_constant(df_test[features]))


# saving prediction into another dataset
df_test[['Id', 'reg3_SalePrice']].to_csv('reg3_prediction.csv', index=False)
```

Simple linear regression 3 (updated) for cross-validation

```
# Prepare data (same as before)
features = numeric_features + boolean_features
target = 'SalePrice'

X = df_train[features].copy()
y = df_train[target]

for feature in boolean_features:
    X[feature] = X[feature].astype(int)

# Initialize the model
model = LinearRegression()

# Use 5-fold cross-validation
cv = KFold(n_splits=5, shuffle=True, random_state=42)

# Cross-validated RMSE (negative means "loss function" style, so we take the negative back)
```

```
rmse_scores = cross_val_score(model, X, y, cv=cv, scoring='neg_root_mean_squared_error')
mae_scores = cross_val_score(model, X, y, cv=cv, scoring='neg_mean_absolute_error')
r2_scores = cross_val_score(model, X, y, cv=cv, scoring='r2')

# Print results
print(f"Cross-validated RMSE: {-rmse_scores.mean():,.0f}")
print(f"Cross-validated MAE: {-mae_scores.mean():,.0f}")
print(f"Cross-validated R² Score: {r2_scores.mean():.4f}")
```

```
Cross-validated RMSE: 55,989
Cross-validated MAE: 19,240
Cross-validated R² Score: 0.3870
```

Simple linear regression 3 (updated) with standardized numeric_features

```
from sklearn.preprocessing import StandardScaler
import statsmodels.api as sm

# Select features and target
features = numeric_features + boolean_features
target = 'SalePrice'

# === PREPARE TRAINING DATA ===
X_train = df_train[features].copy()
y_train = df_train[target]

# Standard scale numeric features
scaler = StandardScaler()
X_train[numeric_features] = scaler.fit_transform(X_train[numeric_features])

# Convert boolean features to int
for feature in boolean_features:
    X_train[feature] = X_train[feature].astype(int)

# Add constant term
X_train = sm.add_constant(X_train, has_constant='add')

# Fit model
reg_std = sm.OLS(y_train, X_train)
results_std = reg_std.fit()
print(results_std.summary())
```

```
    Electrical_SBrkr       -796.6432   2969.916     -0.268      0.789   -6623.405    5030.119
```

```
# === PREPARE TEST DATA ===
X_test = df_test[features].copy()

# Apply same scaling to test numeric features
X_test[numeric_features] = scaler.transform(X_test[numeric_features])

# Convert boolean features to int
for feature in boolean_features:
    X_test[feature] = X_test[feature].astype(int)

# Add constant term
X_test = sm.add_constant(X_test, has_constant='add')

# Ensure column alignment with training data
X_test = X_test[X_train.columns]

# Predict on test data
df_test['reg_std_SalePrice'] = results_std.predict(X_test)


# saving prediction into another dataset
df_test[['Id', 'reg_std_SalePrice']].to_csv('reg_std_prediction.csv', index=False)
```

Standardization improves readability but prediciton using OLS does not improve.

Simple linear regression 5

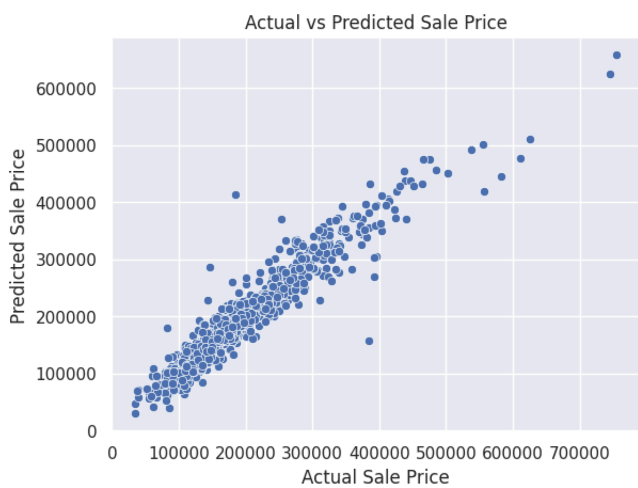
All features - numeric and business sense

```
# Select the features and target variable
subset_bool = ['MSZoning_FV', 'MSZoning_RH', 'MSZoning_RL', 'MSZoning_RM', 'Street_Pave', 'LotConfig_CulDSac', 'LotConfig_FR2', 'LotConfig_FR3', 'LotConfig_Inside',
               'LandSlope_Mod', 'LandSlope_Sev', 'RoofMatl_CompShg', 'RoofMatl_Membran', 'RoofMatl_Metal', 'RoofMatl_Roll', 'RoofMatl_Tar&Grv', 'RoofMatl_WdShake',
               'RoofMatl_WdShngl', 'BsmtQual_Fa', 'BsmtQual_Gd', 'BsmtQual_NB', 'BsmtQual_TA', 'KitchenQual_Fa', 'KitchenQual_Gd', 'KitchenQual_TA', 'GarageQual_Fa',
               'GarageQual_Gd', 'GarageQual_NG', 'GarageQual_Po', 'GarageQual_TA', 'GarageCond_Fa', 'GarageCond_Gd', 'GarageCond_NG', 'GarageCond_Po', 'GarageCond_TA',
               'Condition1_PosA', 'Condition1_PosN', 'Condition2_PosA', 'Condition2_PosN', 'BldgType_2fmCon', 'BldgType_Duplex', 'BldgType_Twnhs', 'BldgType_TwnhsE',
               'ExterQual_Fa', 'ExterQual_Gd', 'ExterQual_TA', 'BsmtExposure_Gd', 'BsmtExposure_Mn', 'BsmtExposure_NB', 'BsmtExposure_No', 'BsmtFinType1_GLQ',
               'BsmtFinType1_LwQ', 'BsmtFinType2_GLQ', 'BsmtFinType2_LwQ', 'SaleCondition_Normal', 'SaleCondition_Partial']
features = numeric_features + subset_bool
target = 'SalePrice'

# Create feature and target DataFrames for training
X = df_train[features].copy()
y = df_train[target]

# Convert boolean features to int (0 or 1)
for feature in subset_bool:
    X[feature] = X[feature].astype(int)

# Add constant term
X = sm.add_constant(X)

# Create and fit model
reg5 = sm.OLS(y, X)
results = reg5.fit()

print(results.summary())
```

GarageCond_Fa	8.945e+04	3.36e+04	2.662	0.008	2.35e+04	1.55e+05
GarageCond_Gd	8.912e+04	3.46e+04	2.574	0.010	2.12e+04	1.57e+05
GarageCond_NG	-7.045e+04	5.66e+04	-1.246	0.213	-1.81e+05	4.05e+04
GarageCond_Po	8.831e+04	3.57e+04	2.474	0.013	1.83e+04	1.58e+05
GarageCond_TA	9.259e+04	3.33e+04	2.783	0.005	2.73e+04	1.58e+05
Condition1_PosA	-6718.8608	8792.256	-0.764	0.445	-2.4e+04	1.05e+04
Condition1_PosN	-926.8855	6067.530	-0.153	0.879	-1.28e+04	1.1e+04
Condition2_PosA	2.885e+04	2.63e+04	1.098	0.272	-2.27e+04	8.04e+04
Condition2_PosN	-2.756e+05	1.99e+04	-13.843	0.000	-3.15e+05	-2.37e+05
BldgType_2fmCon	-6702.5625	5450.759	-1.230	0.219	-1.74e+04	3990.197
BldgType_Duplex	-3802.8915	5655.309	-0.672	0.501	-1.49e+04	7291.133
BldgType_Twnhs	-1.923e+04	5496.159	-3.498	0.000	-3e+04	-8444.214
BldgType_TwnhsE	-1.552e+04	3792.970	-4.092	0.000	-2.3e+04	-8081.521
ExterQual_Fa	-1.575e+04	9351.681	-1.684	0.092	-3.41e+04	2594.917
ExterQual_Gd	-1.916e+04	4678.865	-4.095	0.000	-2.83e+04	-9981.539
ExterQual_TA	-2.268e+04	5187.054	-4.372	0.000	-3.29e+04	-1.25e+04
BsmtExposure_Gd	1.523e+04	2909.405	5.234	0.000	9520.641	2.09e+04
BsmtExposure_Mn	32.0111	2879.878	0.011	0.991	-5617.448	5681.470
BsmtExposure_NB	-5313.8945	3813.021	-1.394	0.164	-1.28e+04	2166.112
BsmtExposure_No	-2614.0615	1978.277	-1.321	0.187	-6494.850	1266.727

```
# Extract regression results into a DataFrame
summary_df = pd.DataFrame({
    'coef': results.params,
    'std_err': results.bse,
    't': results.tvalues,
    'p_value': results.pvalues
})

# Save to CSV
summary_df.to_csv('regression_summary.csv')


# Add predictions to the training dataset
df_train['reg5_SalePrice'] = results.predict(sm.add_constant(df_train[features]))

# Calculate correlation between actual and predicted SalePrice
correlation = df_train['SalePrice'].corr(df_train['reg5_SalePrice'])
rmse = np.sqrt(mean_squared_error(df_train['SalePrice'], df_train['reg5_SalePrice']))
mae = mean_absolute_error(df_train['SalePrice'], df_train['reg5_SalePrice'])
r2 = r2_score(df_train['SalePrice'], df_train['reg5_SalePrice'])

# Print the correlation
print(f"Correlation: {correlation}")
print(f"RMSE: {rmse:,.0f}")
print(f"MAE: {mae:,.0f}")
print(f"R² Score: {r2:.4f}")

# Create a scatter plot to visualize the relationship
sns.scatterplot(x='SalePrice', y='reg5_SalePrice', data=df_train)
plt.title('Actual vs Predicted Sale Price')
plt.xlabel('Actual Sale Price')
plt.ylabel('Predicted Sale Price')
plt.show()
```

```
Correlation: 0.9576084348504664
RMSE: 22,877
MAE: 14,741
R² Score: 0.9170
```




```
# Add predictions to test dataset
df_test['reg5_SalePrice'] = results.predict(sm.add_constant(df_test[features]))


# saving prediction into another dataset
df_test[['Id', 'reg5_SalePrice']].to_csv('reg5_prediction.csv', index=False)
```

Simple linear regression 4

All features - numeric and low p-value categorical

```python
# Select the features and target variable
subset_bool = ['MSZoning_FV', 'MSZoning_RH', 'MSZoning_RL', 'MSZoning_RM', 'Street_Pave', 'LotConfig_CulDSac', 'LotConfig_FR2', 'LotConfig_FR3',
               'LotConfig_Inside', 'LandSlope_Mod', 'LandSlope_Sev', 'RoofMatl_CompShg', 'RoofMatl_Membran', 'RoofMatl_Metal', 'RoofMatl_Roll',
               'RoofMatl_Tar&Grv', 'RoofMatl_WdShake', 'RoofMatl_WdShngl', 'BsmtQual_Fa', 'BsmtQual_Gd', 'BsmtQual_NB', 'BsmtQual_TA', 'KitchenQual_Fa',
               'KitchenQual_Gd', 'KitchenQual_TA', 'GarageQual_Fa', 'GarageQual_Gd', 'GarageQual_NG', 'GarageQual_Po', 'GarageQual_TA', 'GarageCond_Fa',
               'GarageCond_Gd', 'GarageCond_NG', 'GarageCond_Po', 'GarageCond_TA']
features = numeric_features + subset_bool
target = 'SalePrice'

# Create feature and target DataFrames for training
X = df_train[features].copy()
y = df_train[target]

# Convert boolean features to int (0 or 1)
for feature in subset_bool:
    X[feature] = X[feature].astype(int)

# Add constant term
X = sm.add_constant(X)

# Create and fit model
reg4 = sm.OLS(y, X)
results = reg4.fit()

print(results.summary())
```

```
TotRmsAbvGrd        -2373.8839   3093.009     -0.767      0.443   -8441.364    3693.596
Fireplaces           4314.0872   1390.492      3.103      0.002    1586.393    7041.781
GarageYrBlt           -49.8709     62.232     -0.801      0.423    -171.950      72.208
GarageCars           5632.6781   2381.628      2.365      0.018     960.697    1.03e+04
GarageArea              7.0462      8.154      0.864      0.388      -8.949      23.042
WoodDeckSF             11.1643      6.248      1.787      0.074      -1.092      23.420
OpenPorchSF           -17.3673     11.946     -1.454      0.146     -40.801       6.067
EnclosedPorch           5.1060     12.953      0.394      0.693     -20.303      30.515
3SsnPorch              33.9078     24.085      1.408      0.159     -13.340      81.155
ScreenPorch            33.9878     13.441      2.529      0.012       7.620      60.355
PoolArea               70.9876     19.165      3.704      0.000      33.391     108.584
MiscVal                -0.8357      1.411     -0.592      0.554      -3.604       1.933
MoSold               -105.7037    263.186     -0.402      0.688    -621.990     410.583
YrSold                164.3912    276.061      0.595      0.552    -377.151     705.934
TotalLivArea           19.7024      6.546      3.010      0.003       6.861      32.544
TotalBaths           1919.6374   1238.037      1.551      0.121    -508.990    4348.265
AvgRmSize            -137.2726     81.174     -1.691      0.091    -296.510      21.965
AgeAtSale              40.5756    144.971      0.280      0.780    -243.810     324.961
AgeSinceRemod          53.8429    142.647      0.377      0.706    -225.984     333.670
AvgSalePrice            0.2539      0.022     11.479      0.000       0.210       0.297
OpenSpaceRatio       1.024e+05   1.81e+04      5.656      0.000    6.69e+04    1.38e+05
NbhoodSpace         -8.078e+04   2.13e+04     -3.800      0.000   -1.22e+05   -3.91e+04
TotalLivArea_squared    0.0013      0.001      1.547      0.122      -0.000       0.003
AgeAtSale_log       -6764.2391   1647.502     -4.106      0.000   -9996.104   -3532.374
MSZoning_FV           1.67e+04   9835.649      1.697      0.090   -2599.212     3.6e+04
MSZoning_RH          1.092e+04   1.12e+04      0.979      0.328    -1.1e+04    3.28e+04
MSZoning_RL          1.051e+04   9153.917      1.148      0.251   -7450.297    2.85e+04
MSZoning_RM          1.388e+04   9131.347      1.520      0.129   -4031.306    3.18e+04
Street_Pave          2.282e+04    1.17e+04     1.953      0.051    -103.250    4.57e+04
LotConfig_CulDSac    7502.4473   3333.483      2.251      0.025     963.235     1.4e+04
LotConfig_FR2       -7333.7123   4271.056     -1.717      0.086   -1.57e+04    1044.715
LotConfig_FR3        -3.03e+04   1.35e+04     -2.237      0.025   -5.69e+04   -3725.690
LotConfig_Inside    -1614.2785   1874.944     -0.861      0.389   -5292.312    2063.755
LandSlope_Mod         1.25e+04   3528.731      3.543      0.000    5579.893    1.94e+04
LandSlope_Sev       -1.456e+04   1.06e+04     -1.368      0.172   -3.54e+04    6319.396
RoofMatl_CompShg     7.525e+05   6.21e+04     12.114      0.000    6.31e+05    8.74e+05
RoofMatl_Membran     7.871e+05   6.87e+04     11.456      0.000    6.52e+05    9.22e+05
RoofMatl_Metal       7.678e+05   6.83e+04     11.245      0.000    6.34e+05    9.02e+05
RoofMatl_Roll        7.487e+05   6.77e+04     11.057      0.000    6.16e+05    8.81e+05
RoofMatl_Tar&Grv     7.424e+05   6.28e+04     11.816      0.000    6.19e+05    8.66e+05
RoofMatl_WdShake     7.507e+05   6.33e+04     11.855      0.000    6.26e+05    8.75e+05
RoofMatl_WdShngl     8.093e+05   6.15e+04     13.165      0.000    6.89e+05     9.3e+05
BsmtQual_Fa         -2.266e+04   6452.265     -3.512      0.000   -3.53e+04      -1e+04
BsmtQual_Gd          -2.72e+04   3373.754     -8.062      0.000   -3.38e+04   -2.06e+04
BsmtQual_NB          -100.4970   8128.937     -0.012      0.990    -1.6e+04    1.58e+04
BsmtQual_TA         -2.418e+04   4230.655     -5.716      0.000   -3.25e+04   -1.59e+04
KitchenQual_Fa      -2.103e+04   6095.660     -3.450      0.001    -3.3e+04   -9072.609
KitchenQual_Gd      -2.399e+04   3526.926     -6.803      0.000   -3.09e+04   -1.71e+04
KitchenQual_TA       -2.65e+04   3997.986     -6.629      0.000   -3.43e+04   -1.87e+04
GarageQual_Fa       -8.766e+04    3.18e+04    -2.754      0.006    -1.5e+05   -2.52e+04
GarageQual_Gd       -7.391e+04    3.24e+04    -2.282      0.023   -1.37e+05   -1.04e+04
GarageQual_NG       -5.312e+04    6.21e+04    -0.856      0.392   -1.75e+05    6.86e+04
GarageQual_Po       -9.077e+04    3.79e+04    -2.395      0.017   -1.65e+05   -1.64e+04
GarageQual_TA       -8.384e+04    3.15e+04    -2.664      0.008   -1.46e+05   -2.21e+04
GarageCond_Fa        6.418e+04    3.69e+04     1.739      0.082   -8212.271    1.37e+05
GarageCond_Gd        5.962e+04     3.8e+04     1.568      0.117    -1.5e+04    1.34e+05
GarageCond_NG       -5.312e+04    6.21e+04    -0.856      0.392   -1.75e+05    6.86e+04
GarageCond_Po         6.22e+04    3.92e+04     1.586      0.113   -1.47e+04    1.39e+05
GarageCond_TA        6.609e+04    3.66e+04     1.807      0.071   -5671.814    1.38e+05
```

```python
# Extract regression results into a DataFrame
summary_df = pd.DataFrame({
    'coef': results.params,
    'std_err': results.bse,
    't': results.tvalues,
    'p_value': results.pvalues
})

# Save to CSV
summary_df.to_csv('regression_summary.csv')


# Add predictions to the training dataset
df_train['reg4_SalePrice'] = results.predict(sm.add_constant(df_train[features]))

# Calculate correlation between actual and predicted SalePrice
```

```
correlation = df_train['SalePrice'].corr(df_train['reg4_SalePrice'])
rmse = np.sqrt(mean_squared_error(df_train['SalePrice'], df_train['reg4_SalePrice']))
mae = mean_absolute_error(df_train['SalePrice'], df_train['reg4_SalePrice'])
r2 = r2_score(df_train['SalePrice'], df_train['reg4_SalePrice'])

# Print the correlation
print(f"Correlation: {correlation}")
print(f"RMSE: {rmse:,.0f}")
print(f"MAE: {mae:,.0f}")
print(f"R² Score: {r2:.4f}")

# Create a scatter plot to visualize the relationship
sns.scatterplot(x='SalePrice', y='reg4_SalePrice', data=df_train)
plt.title('Actual vs Predicted Sale Price')
plt.xlabel('Actual Sale Price')
plt.ylabel('Predicted Sale Price')
plt.show()
```

```
Correlation: 0.946757589412793
RMSE: 25,568
MAE: 15,784
R² Score: 0.8963
```

Actual vs Predicted Sale Price

Introduction

Link to access this code - https://colab.research.google.com/drive/1gKRrXN0jYrheIwl3eefoSEj9gh3FwewI#scrollTo=OvyyD8iORVMK

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

Import modules and data files

[ ] ↳ 3 cells hidden

Clean Data

[ ] ↳ 6 cells hidden

Create additional variables

[ ] ↳ 10 cells hidden

Exploratory Data Analysis

Identifying important numerical features

↳ 3 cells hidden

Scaling important numerical features

TotalLivArea

Added a quadratic term "TotalLivArea_squared" after analysis.

```
# Create the scatter plot using Seaborn, with hue based on 'HasBasement'
sns.scatterplot(x='TotalLivArea', y='SalePrice', data=df_train)

# Customize the plot (optional)
plt.title('Total Living Area vs Sale Price')
plt.xlabel('Total Living Area (sqft)')
plt.ylabel('Sale Price ($)')

# Show the plot
plt.show()
```


```
# Create n-tiles (20 n-tiles) based on TotalLivArea
df_train['TotalLivArea_ntile'] = pd.qcut(df_train['TotalLivArea'], q=20, labels=False)

# Calculate the median SalePrice for each n-tile
median_prices_by_ntile = df_train.groupby('TotalLivArea_ntile')['SalePrice'].median().reset_index()

# Create the scatter plot using the median prices by n-tile
sns.scatterplot(x='TotalLivArea_ntile', y='SalePrice', data=median_prices_by_ntile)  # Change here

# Customize the plot
plt.title('Median Sale Price vs Total Living Area N-tile')  # Change here
plt.xlabel('Total Living Area N-tile')  # Change here
plt.ylabel('Median Sale Price ($)')

# Show the plot
plt.show()
```

The above plot shows a potential quadratic effect.

```
#Add a quadratic term for TotalLivArea
df_train['TotalLivArea_squared'] = df_train['TotalLivArea'] ** 2
df_test['TotalLivArea_squared'] = df_test['TotalLivArea'] ** 2


# Create the scatter plot using Seaborn, with hue based on 'HasBasement'
sns.scatterplot(x='TotalLivArea_squared', y='SalePrice', data=df_train)

# Customize the plot (optional)
plt.title('Total Living Area (squared) vs Sale Price')
plt.xlabel('Total Living Area (sqft)')
plt.ylabel('Sale Price ($)')
plt.xlim(right=50000000)

# Show the plot
plt.show()
```

Looks more linear when plotting against quadratic term.

AvgRmSize

Has a linear relationship with SalePrice

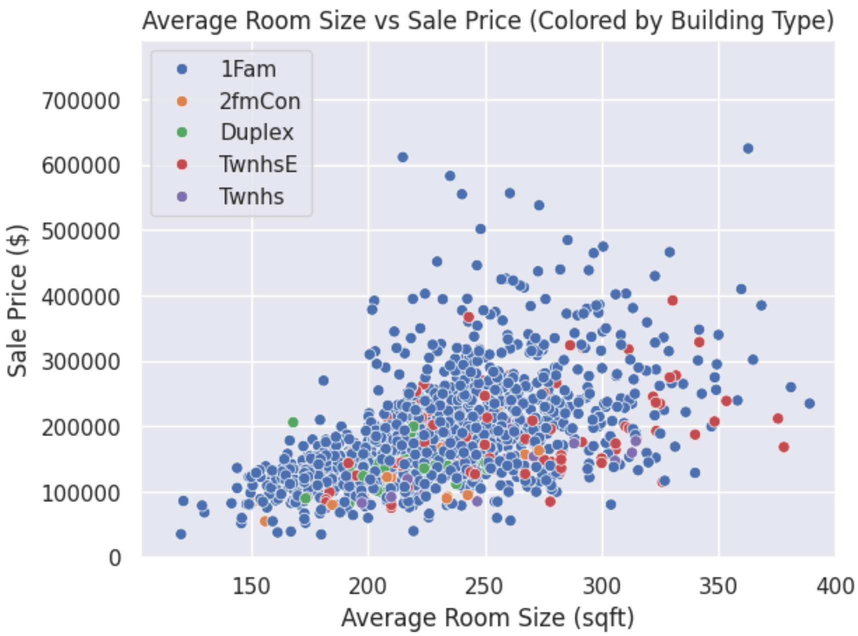
```
df_train.groupby('BldgType')[['SalePrice', 'AvgRmSize']].describe()
```

	SalePrice								AvgRmSize			
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min
BldgType												
1Fam	1220.0	185763.807377	82648.502922	34900.0	131475.0	167900.0	222000.0	755000.0	1220.0	230.178106	44.676834	120.0000
2fmCon	31.0	128432.258065	35458.545158	55000.0	106875.0	127500.0	142500.0	228950.0	31.0	215.107401	33.628369	150.8000
Duplex	52.0	133541.076923	27833.249197	82000.0	118375.0	135980.0	145000.0	206300.0	52.0	205.434188	25.878809	163.2000
Twnhs	43.0	135911.627907	41013.222080	75000.0	95750.0	137500.0	168750.0	230000.0	43.0	231.042968	38.560803	174.0000
TwnhsE	114.0	181959.342105	60626.108918	75500.0	143187.5	172200.0	207375.0	392500.0	114.0	254.550731	47.125055	176.6666

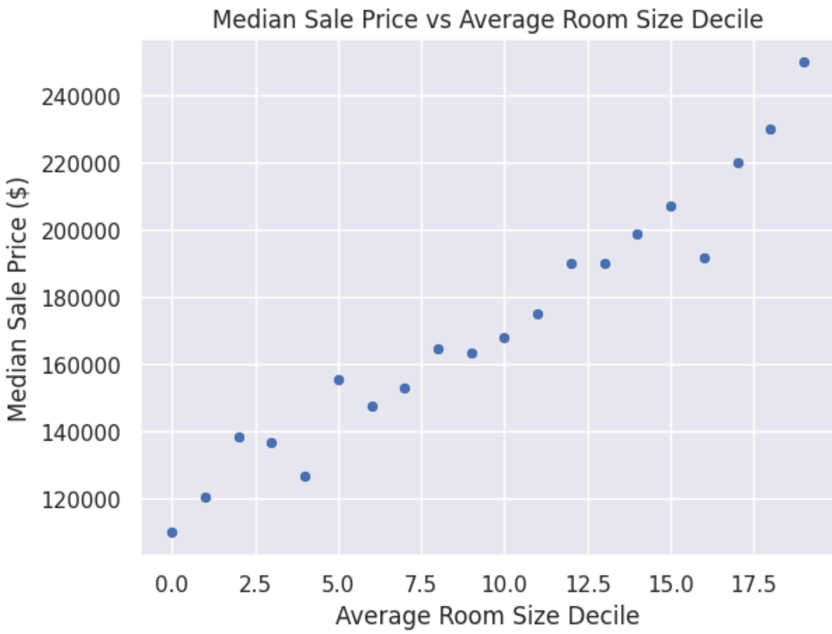
TAKEAWAY: BldgType, doesn't seem to have a major impact on the AvgRmSize.

```
# Create the scatter plot using Seaborn, with hue based on 'BldgType'
sns.scatterplot(x='AvgRmSize', y='SalePrice', hue='BldgType', data=df_train)
```

```
# Customize the plot (optional)
plt.title('Average Room Size vs Sale Price (Colored by Building Type)')  # Updated title
plt.xlabel('Average Room Size (sqft)')  # Updated x-axis label
plt.ylabel('Sale Price ($)')
plt.legend()  # Show legend for BldgType
plt.xlim(right=400)

# Show the plot
plt.show()
```



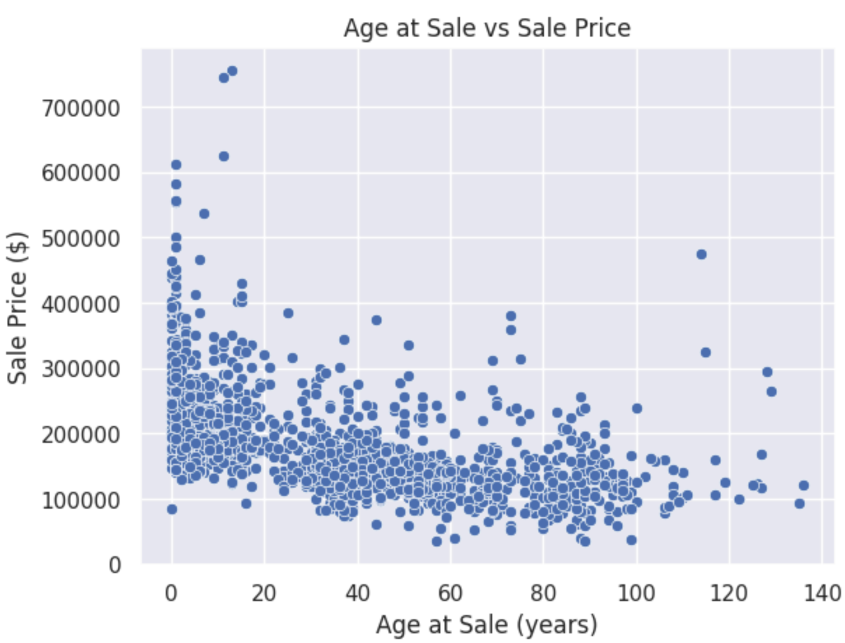

```
# Create deciles (20 deciles) based on AvgRmSize
df_train['AvgRmSize_decile'] = pd.qcut(df_train['AvgRmSize'], q=20, labels=False)

# Calculate the median SalePrice for each decile
median_prices_by_decile = df_train.groupby('AvgRmSize_decile')['SalePrice'].median().reset_index()

# Create the scatter plot using the median prices by decile
sns.scatterplot(x='AvgRmSize_decile', y='SalePrice', data=median_prices_by_decile)

# Customize the plot
plt.title('Median Sale Price vs Average Room Size Decile')  # Updated title
plt.xlabel('Average Room Size Decile')
plt.ylabel('Median Sale Price ($)')

# Show the plot
plt.show()
```



AgeAtSale

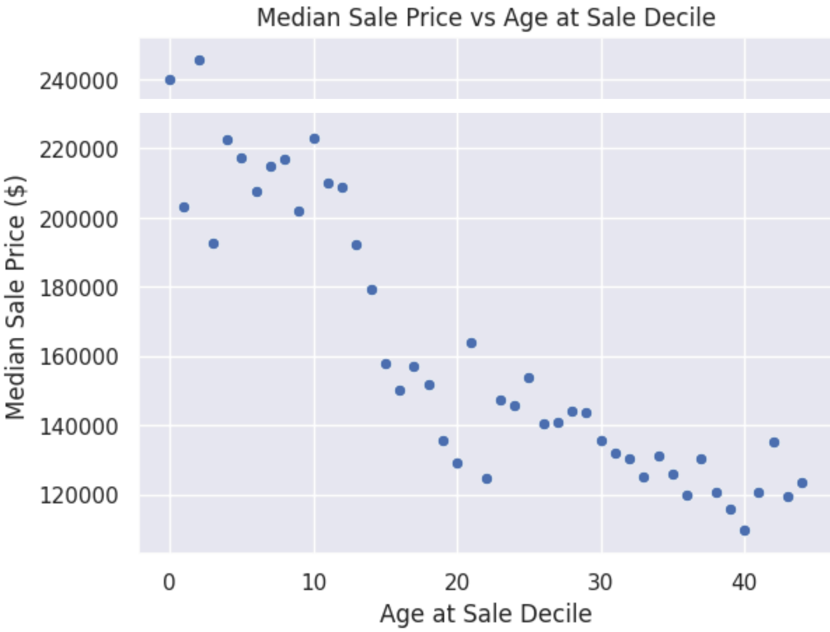
Added a log term "AgeAtSale_log" after analysis.*italicized text*

```
# Create the scatter plot using Seaborn
sns.scatterplot(x='AgeAtSale', y='SalePrice', data=df_train)

# Customize the plot (optional)
plt.title('Age at Sale vs Sale Price')  # Updated title
plt.xlabel('Age at Sale (years)')  # Updated x-axis label
plt.ylabel('Sale Price ($)')

# Show the plot
plt.show()
```




```
# Create deciles (50 deciles) based on AgeAtSale
df_train['AgeAtSale_decile'] = pd.qcut(df_train['AgeAtSale'], q=50, labels=False, duplicates='drop'

# Calculate the median SalePrice for each decile
```

```
median_prices_by_decile = df_train.groupby('AgeAtSale_decile')['SalePrice'].median().reset_index()

# Create the scatter plot using the median prices by decile
sns.scatterplot(x='AgeAtSale_decile', y='SalePrice', data=median_prices_by_decile)

# Customize the plot
plt.title('Median Sale Price vs Age at Sale Decile')  # Updated title
plt.xlabel('Age at Sale Decile')  # Updated x-axis label
plt.ylabel('Median Sale Price ($)')

# Show the plot
```



Looks like there is a logarithmic effect.

```
#Add a logarithmic term for AgeAtSale
epsilon = 0.001
df_train['AgeAtSale_log'] = np.log1p(df_train['AgeAtSale'] + epsilon)
df_test['AgeAtSale_log'] = np.log1p(df_test['AgeAtSale'] + epsilon)


# Create the scatter plot using Seaborn
sns.scatterplot(x='AgeAtSale_log', y='SalePrice', data=df_train)

# Customize the plot (optional)
plt.title('Log of Age at Sale vs Sale Price')  # Updated title
plt.xlabel('Log of Age at Sale (years)')  # Updated x-axis label
plt.ylabel('Sale Price ($)')

# Show the plot
plt.show()
```