There are many different features of a house that go into its valuation, and this dataset makes that clear. Upon importing and inspecting the training data, several features had missing values marked as NA. Interestingly, for many of them, NA did not mean the data was missing—it meant the feature was not present in the house. For example, FireplaceQu is NA for houses without a fireplace, and PoolQC is NA for homes without a pool. A similar pattern was observed with basement and garage-related features.

An important first step was to distinguish between NA values that indicated non-existence versus those that were truly missing. Features like Alley, Fence, LotFrontage, and MiscFeature had many missing values with unclear interpretation, so they were dropped. Cleaning decisions for each feature were made deliberately and documented in the accompanying code.

Next, the focus shifted to identifying important features. Feature distributions were visualized for both numeric (continuous and discrete) and categorical variables. Correlations with SalePrice were calculated, and a simple linear regression model was later used to further highlight influential predictors. A scatterplot matrix helped detect multicollinearity among the features.

Several features were selected for deeper exploratory analysis, including square footage metrics, OverallQual, OverallCond, bathroom and bedroom counts, Neighborhood, and garage-related features.

Key takeaways from the analysis:

GrLivArea (above-ground living area) is the sum of 1stFlrSF, 2ndFlrSF, and LowQualFinSF. It is highly correlated with SalePrice. Combining it with TotalBsmtSF into a new feature, TotalLivArea, provided a more complete view of the livable space and improved correlation.

https://colab.research.google.com/drive/1gKRrXN0jYrheIwl3eefoSEj9gh3FwewI#scrollTo=5fpp35Eb1ycm

When plotting TotalLivArea vs. SalePrice and color-coding by OverallQual, it became clear that larger homes tend to also be higher quality, and both drive up home value.

A composite feature, TotalBaths, was created by summing full and half bathrooms (with half baths counted as 0.5), including those in the basement. This performed better than individual bathroom variables in predicting SalePrice.

Neighborhood analysis showed clear differences in average home value, with higher-value neighborhoods often having homes with higher OverallQual. These two variables were correlated, but each added unique information.

Combining OverallQual and OverallCond revealed strong interactions. A heatmap of median SalePrice and median PricePerSqFt by these two features showed a jump in value when both ratings exceeded 5.

SaleCondition provided insight into transaction types. Higher-quality homes were more likely to be sold as Normal or Partial. In contrast, low-quality homes were rarely sold as Partial, which makes intuitive sense—buyers are less likely to invest in partially built, low-quality homes.

A new feature, AvgRmSize (defined as GrLivArea divided by TotRmsAbvGrd), was used to capture room spaciousness. Larger average room sizes were associated with higher SalePrice, and even at the same room size, higher OverallQual homes sold for more.

As an exercise, SalePrice was scaled using both min-max and standard scaling. Standard scaling is best for variables with a roughly normal distribution, while min-max is more appropriate for bounded, non-normal variables. Categorical and low-cardinality discrete variables should generally remain unscaled.

https://colab.research.google.com/drive/1gKRrXN0jYrheIwl3eefoSEj9gh3FwewI#scrollTo=5fpp35Eb1ycm

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/

Houses in Ames, Iowa

Objective:

1. Provide appropriate descriptive statistics and visualizations to help understand the marginal distribution of the dependent variable.
2. Investigate missing data and outliers.
3. Investigate at least three potential predictors of the dependent variable and provide appropriate graphs / statistics to demonstrate the relationships.
4. Engage in feature creation by splitting, merging, or otherwise generating a new predictor.
5. Using the dependent variable, perform both min-max and standard scaling in Python.

Upload all data and modules

```
# Import modules
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import tree
from sklearn.metrics import accuracy_score

# 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)
```

Load data and analyze

Load data

```
df_train = pd.read_csv('train.csv')
df_test = pd.read_csv('test.csv')
```

```
df_train.head()
```

Show hidden output

```
df_test.head()
```

Show hidden output

```
# define a variable to show info() + levels information for categorical variables
def extended_info(df):
    df.info()

    # Add categorical levels information
    print("\nCategorical Levels:")
    for col in df.select_dtypes(include=['object']).columns:
        num_levels = df[col].nunique()
        print(f"  {col}: {num_levels} levels")

extended_info(df_train)
```

Show hidden output

Analysis of missing data part 1

I believe the following features have too few observations to work with and should be investigated further.

- Alley - "NA" implies no alley access, not missing data
- MasVnrType - Related to MasVnrArea, in most cases if Area is 0 MasVnrType is "None" (there are a few exceptions). There are 8 entries with "NA" that correspond to MasVnrArea "NA"
- FireplaceQu - could be related to Fireplaces
- PoolQC - could be related to PoolArea
- Fence - "NA" implies no fence, not missing data
- MiscFeature - "NA" implies no miscellaneous features, not missing data

Analysis of missing Alley data

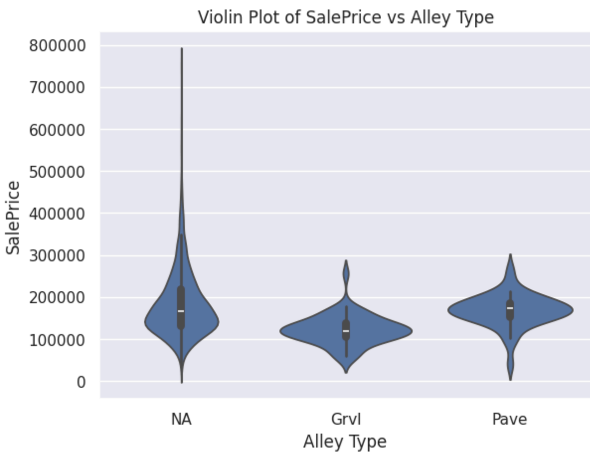
TAKEAWAY: best to exclude "Alley" from further analysis.

```
# does having an alley impact SalePrice?
df_train['Alley'] = df_train['Alley'].fillna('NA')
df_train.groupby('Alley')['SalePrice'].agg(['mean', 'median', 'min', 'max', 'count'])
```

	mean	median	min	max	count
Alley					
Grvl	122219.080000	119500.0	52500	256000	50
NA	183452.131483	165000.0	34900	755000	1369
Pave	168000.585366	172500.0	40000	265979	41

Only homes without an Alley have a chance of being ultra-luxury houses crossing the $300k threshold. This makes sense as suburbs where houses are spaced apart likely have a lot of space and don't need an alley. However, considering that the mean and median are somewhat on par with Alley type "Pave" we can conclude that the ultra-luxury houses could be an outlier and most houses fall under the "NA" category.

```
sns.violinplot(x='Alley', y='SalePrice', data=df_train)
plt.title('Violin Plot of SalePrice vs Alley Type')
plt.xlabel('Alley Type')
plt.ylabel('SalePrice')
plt.show()
```



The median house with no alley sits between gravel and paved houses. However, houses with alleys don't really cross the $300k threshold. This represents the outlier ultra high end houses that are probably in a suburb and spaced really far from each other not requiring alleys.

```
filtered_df = df_train[(df_train['Alley'] == 'NA') & (df_train['SalePrice'] > 300000)]
num_houses = len(filtered_df)
percentage1 = (num_houses / len(df_train)) * 100
print(f"Number of no alley houses worth more than $300k : {num_houses}, these correspond to {percentage1:.2f}% of all houses.")

gravel_count = df_train['Alley'].value_counts()['Grvl']
pave_count = df_train['Alley'].value_counts()['Pave']

# Print the results
print(f"Number of houses with gravel alley: {gravel_count}")
print(f"Number of houses with paved alley: {pave_count}")
```

```
Number of no alley houses worth more than $300k : 115, these correspond to 7.88% of all houses.
Number of houses with gravel alley: 50
Number of houses with paved alley: 41
```

The above is further evidence that a small portion of houses have alleys.

TAKEAWAY: might be best to exclude "Alley" from further analysis.

Analysis of missing MasVnrType data

TAKEAWAY 1: Add an interaction variable between MasVnrType and MasVnrArea into the analysis if using regression models.

TAKEAWAY 2: There are 4 inconsistent entries where MasVnrArea is not 0 when MasVnrType is None. Something to note for later.

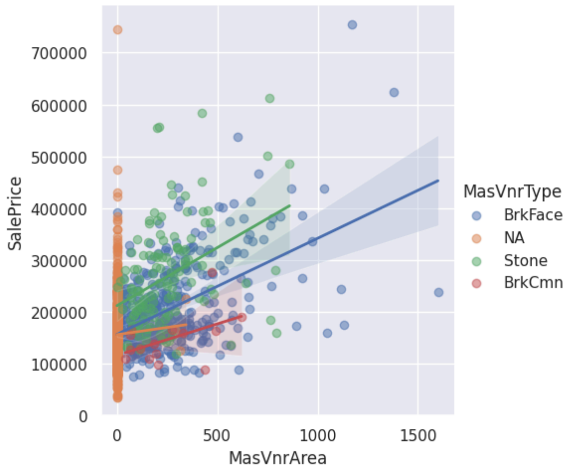
TAKEAWAY 3: there are 8 entires where MasVnrArea and MasVnrType are NA, for consistency these will be modified to be 0 and None respectively.

```
# does having an MasVnrType impact SalePrice?
df_train['MasVnrType'] = df_train['MasVnrType'].fillna('NA')
df_train.groupby('MasVnrType')['SalePrice'].agg(['mean', 'median', 'min', 'max', 'count'])
```

	mean	median	min	max	count
MasVnrType					
BrkCmn	146318.066667	139000.0	89471	277000	15
BrkFace	204691.871910	181000.0	75000	755000	445
NA	156958.243119	143125.0	34900	745000	872
Stone	265583.625000	246839.0	119000	611657	128

```
# does having an MasVnrArea impact SalePrice?
sns.lmplot(x='MasVnrArea', y='SalePrice', hue='MasVnrType', data=df_train,
           scatter_kws={'alpha': 0.5},
           line_kws={'linewidth': 2})
```

```
# Set the y-axis limits using Matplotlib
plt.ylim(0, None)

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



TAKEAWAY: The MasVnrType changes the strength and direction of the correlation between MasVnrArea and SalePrice. It might be best to add an interaction variable into the analysis.

Analysis of missing Fireplace QC

TAKEAWAY: modify Fireplace QC column to accurate represent "NA" columns as homes with "NF" (no fireplaces) as this has an impact on SalesPrice.

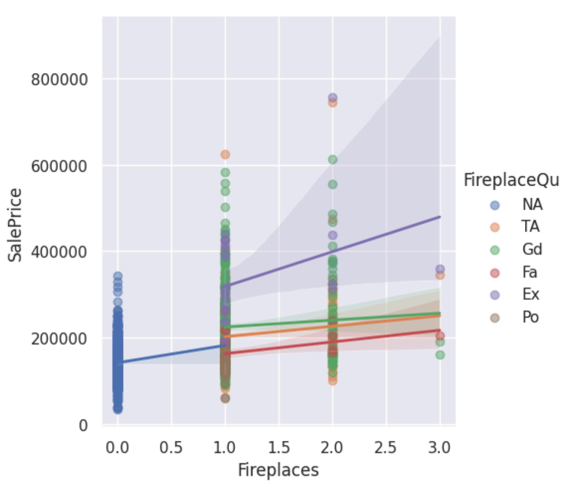
```
# does "FireplaceQu" impact SalePrice?
df_train['FireplaceQu'] = df_train['FireplaceQu'].fillna('NA')
df_train.groupby('FireplaceQu')['SalePrice'].agg(['mean', 'median', 'min', 'max', 'count']).sort_values(by=['median'], ascending=False)
```

FireplaceQu	mean	median	min	max	count
Ex	337712.500000	314250.0	130500	755000	24
Gd	226351.415789	206950.0	90350	611657	380
TA	205801.128205	187500.0	82500	745000	312
Fa	167298.484848	158000.0	117000	262000	33
NA	141389.613603	135000.0	34900	342643	691
Po	129764.150000	131500.0	60000	172000	20

```
# checking number of Fireplaces when FireplaceQu is "NA"
filtered_df = df_train[df_train['FireplaceQu'] == 'NA']
average_fireplaces = filtered_df['Fireplaces'].mean()
print(f"Average Fireplaces when FirepalceQu is NA: {average_fireplaces}")
```

```
Average Fireplaces when FirepalceQu is NA: 0.001447178002894356
```

```
sns.lmplot(x='Fireplaces', y='SalePrice', hue='FireplaceQu', data=df_train,
           scatter_kws={'alpha': 0.5},  # Adjust transparency of scatter points
           line_kws={'linewidth': 2});  # Adjust line width
```



TAKEAWAY: Even with the low number of samples, homes with better quality fireplaces sell at a significantly higher price. Poor quality fireplace homes sell at a lower price than homes with no fireplaces. Best to accurately reflect "NF" (no fireplaces) in the "FireplaceQu" column.

```
# Replace 'NA' with 'NF' in the 'FireplaceQu' column
df_train['FireplaceQu'] = df_train['FireplaceQu'].replace('NA', 'NF')
df_train.groupby('FireplaceQu')['SalePrice'].agg(['mean', 'median', 'min', 'max', 'count']).sort_values(by=['median'], ascending=False)
```

FireplaceQu	mean	median	min	max	count
Ex	337712.500000	314250.0	130500	755000	24
Gd	226351.415789	206950.0	90350	611657	380
TA	205801.128205	187500.0	82500	745000	312
Fa	167298.484848	158000.0	117000	262000	33
NF	141389.613603	135000.0	34900	342643	691
Po	129764.150000	131500.0	60000	172000	20

Analysis of missing PoolQC data

TAKEAWAY: best to exclude "PoolQC" from further analysis.

```
# does "PoolQC" impact SalePrice?
df_train['PoolQC'] = df_train['PoolQC'].fillna('NA')
df_train.groupby('PoolQC')['SalePrice'].agg(['mean', 'median', 'min', 'max', 'count'])
```

PoolQC	mean	median	min	max	count
Ex	490000.000000	490000.0	235000	745000	2
Fa	215500.000000	215500.0	181000	250000	2
Gd	201990.000000	171000.0	160000	274970	3
NA	180404.663455	162900.0	34900	755000	1453

```
# checking average PoolArea when PoolQC is "NA"
filtered_df = df_train[df_train['PoolQC'] == 'NA']
average_pool_area = filtered_df['PoolArea'].mean()
print(f"Average Pool Area when PoolQC is NA: {average_pool_area}")
```

```
Average Pool Area when PoolQC is NA: 0.0
```

TAKEAWAY: most homes, don't have a pool might be best to exclude. PoolQC variable will also partially be captured in PoolArea, as when PoolQC is NA, PoolArea is 0.

Analysis of missing Fence data

TAKEAWAY: best to exclude "Fence" from further analysis.

```
# does having a fence impact SalePrice?
df_train['Fence'] = df_train['Fence'].fillna('NA')
df_train.groupby('Fence')['SalePrice'].agg(['mean', 'median', 'min', 'max', 'count'])
```

Fence	mean	median	min	max	count
GdPrv	178927.457627	167500.0	108000	475000	59
GdWo	140379.314815	138750.0	34900	381000	54
MnPrv	148751.089172	137450.0	40000	745000	157
MnWw	134286.363636	130000.0	110000	187000	11
NA	187596.837998	173000.0	35311	755000	1179

TAKEAWAY: since the percentage of missing values is high, it is best to drop this feature.

Analysis of missing MiscFeature data

TAKEAWAY: best to exclude "MiscFeature" from further analysis.

```
miscfeature_na_count = df_train['MiscFeature'].isna().sum()
miscfeature_na_percentage = (miscfeature_na_count / len(df_train)) * 100
print(f"Number of missing values in 'MiscFeature': {miscfeature_na_count}")
print(f"Percentage of missing values: {miscfeature_na_percentage:.2f}%")
```

```
Number of missing values in 'MiscFeature': 1406
Percentage of missing values: 96.30%
```

TAKEAWAY: since the percentage of missing values is high, it is best to drop this feature.

Clean up data

Cleaning up data as per analysis of missing data part 1

```
# reloading data from csvs
df_train_original = pd.read_csv('train.csv')
df_test_original = pd.read_csv('test.csv')


# reloading train and test data
df_train = df_train_original.copy()
df_test = df_test_original.copy()

# removing Alley, PoolQC, Fence, MiscFeature
df_train = df_train_original.drop(['Alley', 'PoolQC', 'Fence', 'MiscFeature'], axis=1)
df_test = df_test_original.drop(['Alley', 'PoolQC', 'Fence', 'MiscFeature'], axis=1)

# modifying FireplaceQu missing values to indicate "NF" or no fireplace
df_train['FireplaceQu'] = df_train['FireplaceQu'].fillna('NF')
df_test['FireplaceQu'] = df_test['FireplaceQu'].fillna('NF')


# modifying MasVnrArea and MasVnrType missing values to indicate "0" and "None" respectively, this is an assumption.
df_train['MasVnrType'] = df_train['MasVnrType'].fillna('None')
df_train['MasVnrArea'] = df_train['MasVnrArea'].fillna(0)
df_test['MasVnrType'] = df_test['MasVnrType'].fillna('None')
df_test['MasVnrArea'] = df_test['MasVnrArea'].fillna(0)
```

```
extended_info(df_train)
```

Show hidden output

There seems to be a big set of missing data related to basements and garages, this can be investigated next.

Analysis and cleanup of garage related data

Assumption: if GarageArea is 0, there is no garage.

```
# Check how many observations have GarageArea = 0, and make a dataset
df_GarageArea_0 = df_train[df_train['GarageArea'] == 0]
len(df_GarageArea_0)
```

```
81
```

```
# Check how many values GarageType, GarageYrBlt, GarageFinish, GarageQual and GarageCond are not null when GarageArea is 0
df_GarageArea_0['GarageType'].notnull().sum() + df_GarageArea_0['GarageYrBlt'].notnull().sum() + df_GarageArea_0['GarageFinish'].notnull().sum() + df_GarageArea_0['GarageQual'].notnull().su
```

```
np.int64(0)
```

The above check confirms that when GarageArea is 0, data for GarageType, GarageYrBlt, GarageFinish, GarageQual and GarageCond is missing.

```
# do the same for test data
df_GarageArea_0 = df_test[df_test['GarageArea'] == 0]
len(df_GarageArea_0)
```

```
76
```

```
df_GarageArea_0['GarageType'].notnull().sum() + df_GarageArea_0['GarageYrBlt'].notnull().sum() + df_GarageArea_0['GarageFinish'].notnull().sum() + df_GarageArea_0['GarageQual'].notnull().su
```

```
np.int64(0)
```

Even in test data this information is missing.

```
# 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_train.loc[df_train['GarageArea'] == 0, columns_to_update] = 'NG'
df_test.loc[df_test['GarageArea'] == 0, columns_to_update] = 'NG'

# When GarageArea is 0 set GarageYrBlt to 0
df_train.loc[df_train['GarageArea'] == 0, 'GarageYrBlt'] = 0
df_test.loc[df_test['GarageArea'] == 0, 'GarageYrBlt'] = 0
```

Analysis and cleanup of basement related data

There are 9 features related to basements. BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinSF1, BsmtFinType2, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF

Out of these features, BsmtFinSF1, BsmtFinSF2, BsmtUnfSF and TotalBsmtSF have no missing data. These also happen to be the continuous/numerical features.

For the categorical variables, missing data doesn't indicate missing data but rather lack of a basement so this needs to be represented accurately.

ASSUMPTION: BsmtFinSF1 + BsmtFinSF2 + BsmtUnfSF = TotalBsmtSF

NOTE: BsmtExposure has one extra missing data cell than the rest of the categorical features, this must be investigated further. Needs further investigation.

ASSUMPTION: If TotalBsmtSF is 0 then categorical variables should indicate no basement.

```
# verify that BsmtFinSF1 + BsmtFinSF2 + BsmtUnfSF = TotalBsmtSF for all rows
df_train['TotalBsmtSF_check'] = df_train['BsmtFinSF1'] + df_train['BsmtFinSF2'] + df_train['BsmtUnfSF']
```

```
df_train['TotalBsmtSF_check'].equals(df_train['TotalBsmtSF'])
```

True

```
# do the same for test
df_test['TotalBsmtSF_check'] = df_test['BsmtFinSF1'] + df_test['BsmtFinSF2'] + df_test['BsmtUnfSF']
df_test['TotalBsmtSF_check'].equals(df_test['TotalBsmtSF'])
```

True

```
df_train = df_train.drop('TotalBsmtSF_check', axis=1)
df_test = df_test.drop('TotalBsmtSF_check', axis=1)
```

```
# find the data rows that has missing BsmTExpsoure but all other data is present
df_train[df_train['BsmtExposure'].isna() & df_train['BsmtFinType1'].notna()]
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType	HouseStyle	OverallQu
948	949	60	RL	65.0	14006	Pave	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm	1Fam	2Story	

There is some inconsistency with this row of data. The correct BsmtExpsoure value should be No (No Exposure) instead of NA (No Basement), because all other Basement features indicate that there is a basement.

```
# do the same for test
df_test[df_test['BsmtExposure'].isna() & df_test['BsmtFinType1'].notna()]
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType	HouseStyle	OverallQ
27	1488	20	RL	73.0	8987	Pave	Reg	Lvl	AllPub	Inside	Gtl	Somerst	Norm	Norm	1Fam	1Story	
888	2349	60	FV	81.0	10411	Pave	Reg	Lvl	AllPub	Corner	Gtl	Somerst	Norm	Norm	1Fam	2Story	

```
# find "Id" values for these rows
id_values = df_train.loc[df_train['BsmtExposure'].isna() & df_train['BsmtFinType1'].notna(), 'Id'].tolist()
print(id_values)
```

[949]

```
# edit this data to fix inconsistency
for id_val in id_values:
    df_train.loc[df_train['Id'] == id_val, 'BsmtExposure'] = "No"
```

```
# check row
df_train[df_train['Id'] == 949]
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType	HouseStyle	OverallQu
948	949	60	RL	65.0	14006	Pave	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm	1Fam	2Story	

```
# do the same for test data
id_values = df_test.loc[df_test['BsmtExposure'].isna() & df_test['BsmtFinType1'].notna(), 'Id'].tolist()
print(id_values)

# edit this data to fix inconsistency
for id_val in id_values:
    df_test.loc[df_test['Id'] == id_val, 'BsmtExposure'] = "No"
```

[1488, 2349]

```
# check rows
df_test[df_test['Id'] == 1488]
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType	HouseStyle	OverallQu
27	1488	20	RL	73.0	8987	Pave	Reg	Lvl	AllPub	Inside	Gtl	Somerst	Norm	Norm	1Fam	1Story	

```
df_test[df_test['Id'] == 2349]
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType	HouseStyle	OverallQ
888	2349	60	FV	81.0	10411	Pave	Reg	Lvl	AllPub	Corner	Gtl	Somerst	Norm	Norm	1Fam	2Story	

```
# 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_train.loc[df_train['TotalBsmtSF'] == 0, columns_to_update] = 'NB'
df_test.loc[df_test['TotalBsmtSF'] == 0, columns_to_update] = 'NB'
```

Checking data again for more missing observations

```
extended_info(df_train)
```

Show hidden output

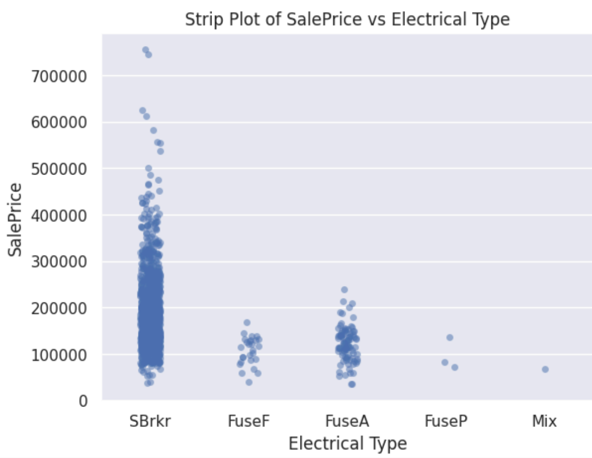
In training data there is considerable missing data for LotFrontage. One missing data for each of Electrical and BsmtFinType2.

Handling missing Electrical data

```
# investigate
df_train.groupby('Electrical')['SalePrice'].agg(['mean', 'median', 'min', 'max', 'count'])
```

Electrical	mean	median	min	max	count
FuseA	122196.893617	121250.0	34900	239000	94
FuseF	107675.444444	115000.0	39300	169500	27
FuseP	97333.333333	82000.0	73000	137000	3
Mix	67000.000000	67000.0	67000	67000	1
SBrkr	186825.113193	170000.0	37900	755000	1334

```
sns.stripplot(x='Electrical', y='SalePrice', data=df_train, jitter=True, alpha=0.5)
plt.title('Strip Plot of SalePrice vs Electrical Type')
plt.xlabel('Electrical Type')
plt.ylabel('SalePrice')
plt.show()
```



Considering that there are mostly SBrkr houses, let's make the assumption that the missing data is also Electrical type SBrkr.

```
# modifying Electrical missing values to indicate "SBrkr"
df_train['Electrical'] = df_train['Electrical'].fillna('SBrkr')
df_test['Electrical'] = df_test['Electrical'].fillna('SBrkr')
```

Handling missing BsmtFinType2 data

```
# finding row with missing data
df_train[df_train['BsmtFinType2'].isna()]
```

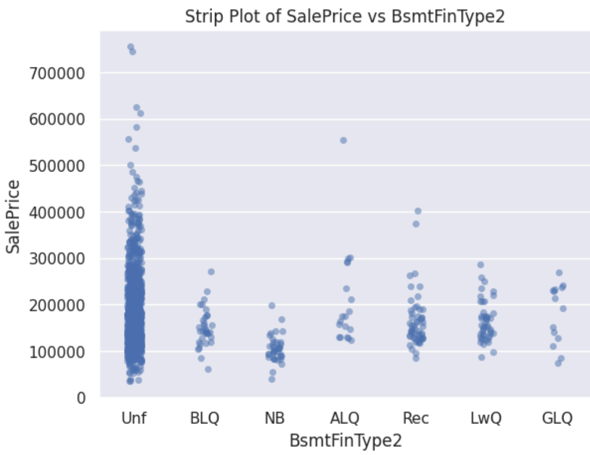
	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType	HouseStyle	OverallQu
332	333	20	RL	85.0	10655	Pave	IR1	Lvl	AllPub	Inside	Gtl	NridgHt	Norm	Norm	1Fam	1Story	

There is a basement here, just missing the BsmtFinType2 data.

```
# investigate
df_train.groupby('BsmtFinType2')['SalePrice'].agg(['mean', 'median', 'min', 'max', 'count'])
```

BsmtFinType2	mean	median	min	max	count
ALQ	209942.105263	174900.0	123500	555000	19
BLQ	151101.000000	143000.0	62383	271900	33
GLQ	180982.142857	203125.0	75500	270000	14
LwQ	164364.130435	154000.0	88000	287000	46
NB	105652.891892	101800.0	39300	198500	37
Rec	164917.129630	148750.0	85000	402000	54
Unf	184694.690287	167000.0	34900	755000	1256

```
sns.stripplot(x='BsmtFinType2', y='SalePrice', data=df_train, jitter=True, alpha=0.5)
plt.title('Strip Plot of SalePrice vs BsmtFinType2')
plt.xlabel('BsmtFinType2')
plt.ylabel('SalePrice')
plt.show()
```



Considering that there are mostly Unf houses, let's make the assumption that the missing data is also BsmtFinType2 Unf.

```
# modifying BsmtFinType2 missing values to indicate "Unf"
df_train['BsmtFinType2'] = df_train['BsmtFinType2'].fillna('Unf')
df_test['BsmtFinType2'] = df_test['BsmtFinType2'].fillna('Unf')
```

Handling missing LotFrontage data

Only 1201/1460 samples have LotFrontage listed. LotFrontage could be correlated with LotArea.

TAKEAWAY: Exclude LotFrontage from dataset, LotArea will capture most of this effect.

```
# investigate
df_train.describe()
```

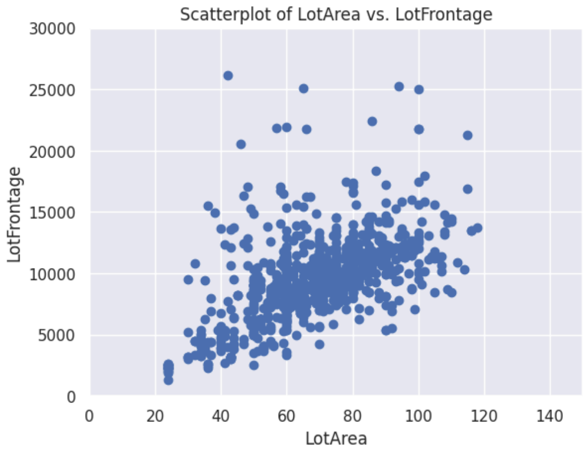
	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.00000
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.117123	443.639726	46.549315	567.240411	1057.429452	1162.62671
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	180.731373	456.098091	161.319273	441.866955	438.705324	386.58773
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.000000	0.000000	0.000000	0.000000	334.00000
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.000000	0.000000	0.000000	223.000000	795.750000	882.00000
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000	383.500000	0.000000	477.500000	991.500000	1087.00000
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	164.250000	712.250000	0.000000	808.000000	1298.250000	1391.25000
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	1474.000000	2336.000000	6110.000000	4692.00000

```
plt.scatter(df_train['LotFrontage'], df_train['LotArea'])
plt.xlabel('LotArea')  # Label for the x-axis
plt.ylabel('LotFrontage') # Label for the y-axis
plt.title('Scatterplot of LotArea vs. LotFrontage') # Title of the plot
plt.show()
```

Show hidden output

There are a lot of outliers, let's limit data to 2 St Devs from median.

```
#filtered_df includes data that is 2 StDevs from mean for each of LotFrontage, LotArea and SalePrice

# Calculate the mean and standard deviation for each column
lot_frontage_mean = df_train['LotFrontage'].mean()
lot_frontage_std = df_train['LotFrontage'].std()
lot_area_mean = df_train['LotArea'].mean()
lot_area_std = df_train['LotArea'].std()
sale_price_mean = df_train['SalePrice'].mean()
sale_price_std = df_train['SalePrice'].std()

# Create the filtered DataFrame
filtered_df = df_train[
    (df_train['LotFrontage'] >= lot_frontage_mean - 2 * lot_frontage_std) &
    (df_train['LotFrontage'] <= lot_frontage_mean + 2 * lot_frontage_std) &
    (df_train['LotArea'] >= lot_area_mean - 2 * lot_area_std) &
    (df_train['LotArea'] <= lot_area_mean + 2 * lot_area_std) &
    (df_train['SalePrice'] >= sale_price_mean - 2 * sale_price_std) &
    (df_train['SalePrice'] <= sale_price_mean + 2 * sale_price_std)
]


plt.scatter(filtered_df['LotFrontage'], filtered_df['LotArea'])
plt.xlabel('LotArea')  # Label for the x-axis
plt.xlim (0,150)
plt.ylabel('LotFrontage') # Label for the y-axis
plt.ylim (0,30000)
plt.title('Scatterplot of LotArea vs. LotFrontage') # Title of the plot
plt.show()
```



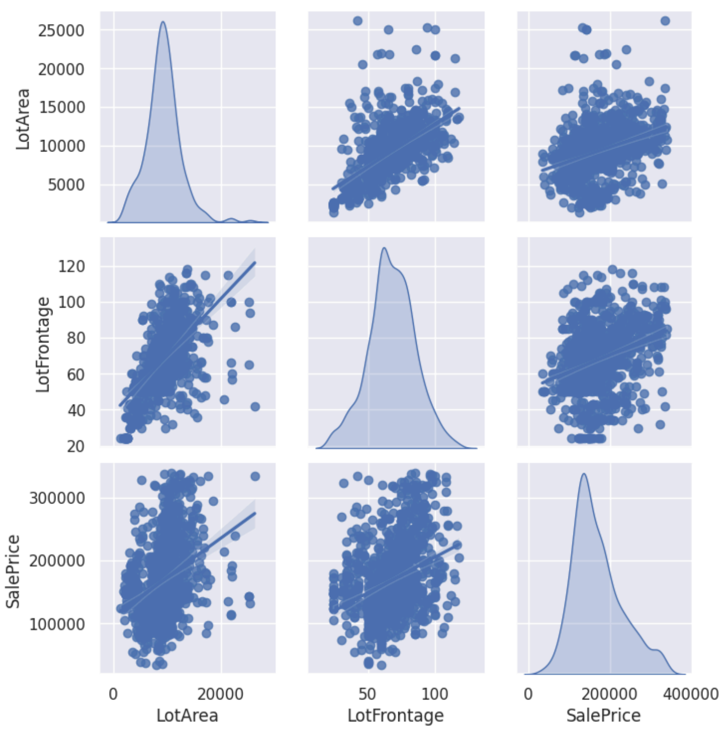

```
correlation = df_train['LotFrontage'].corr(df_train['LotArea'])
print(correlation)
```

```
0.42609501877180833
```

```
correlation = filtered_df['LotFrontage'].corr(filtered_df['LotArea'])
print(correlation)
```

```
0.5935208365539371
```

```
sns.pairplot(filtered_df[['LotArea', 'LotFrontage', 'SalePrice']], kind='reg', diag_kind='kde')
plt.show()
```




```
len(filtered_df)
```

```
1080
```

Excluding the outliers, the correlation imporves significantly.

TAKEAWAY: Exclude LotFrontage from dataset, LotArea will capture most of this effect.

```
# removing LotFrontage
df_train = df_train.drop(['LotFrontage'], axis=1)
df_test = df_test.drop(['LotFrontage'], axis=1)
```

```
extended_info(df_train)
```

Show hidden output

Data is now clean

Creating a list of categorical data types

```
categorical_features = df_train.select_dtypes(include=['object']).columns.tolist()
print(categorical_features)
```

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

```
len(categorical_features)
```

```
39
```

Creating a list of numeric data types

```
numeric_features = df_train.select_dtypes(include=['number']).columns.tolist()
print(numeric_features)
```

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

```
len(numeric_features)
```

```
37
```

```
extended_info(df_test)
```

Show hidden output

```
df_train.describe()
```

Show hidden output

```
df_test.describe()
```

Show hidden output

Visualizing variable distributions

Continuous variables

```
%matplotlib inline
df_train.hist(bins=10, figsize=(20,15))
plt.tight_layout()
plt.show()
```

Show hidden output

Categorical variables

```
num_plots = len(categorical_features)
num_cols = 3  # Adjust as needed
num_rows = (num_plots + num_cols - 1) // num_cols  # Calculate the number of rows needed

# Handle NaN values (replace with 'Unknown' or remove rows)
for column in categorical_features:
    df_train[column] = df_train[column].fillna('Unknown')


# Create a figure and a grid of subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5 * num_rows))  # Adjust figsize as needed

# Iterate and plot on subplots
for i, column in enumerate(categorical_features):
    row = i // num_cols
    col = i % num_cols
    sns.countplot(x=column, data=df_train, ax=axes[row, col])
    axes[row, col].set_title(f"Countplot for {column}")
    axes[row, col].tick_params(axis='x', rotation=90)  # Rotate x-axis labels

plt.tight_layout()  # Adjust subplot parameters for a tight layout
plt.show()
```

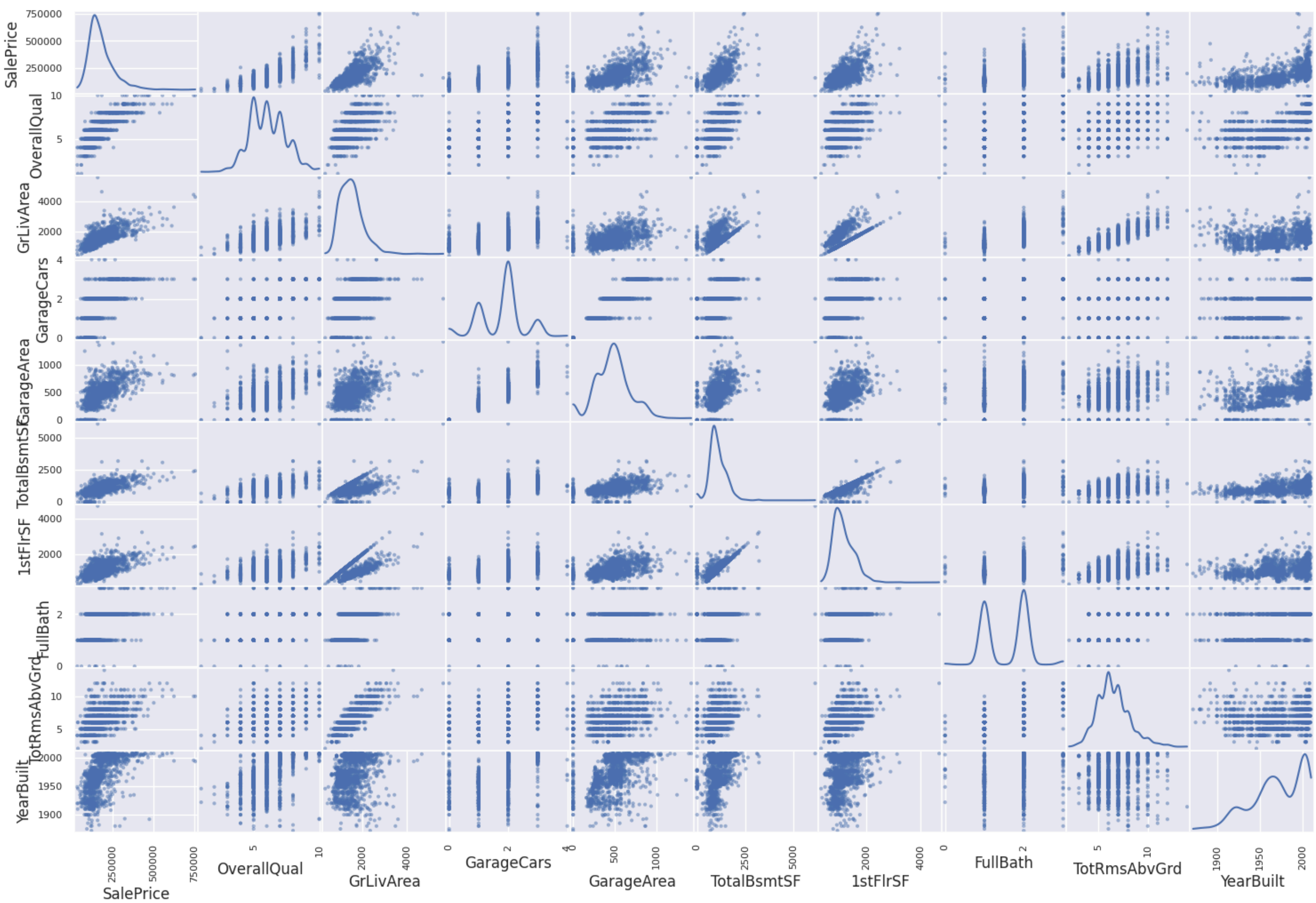
Show hidden output

Correlations

```
corr_matrix = df_train[numeric_features].corr()
corr_matrix["SalePrice"].sort_values(ascending=False)
```

Show hidden output

```
# picking the variables with the highest correlations
from pandas.plotting import scatter_matrix
attributes = ["SalePrice","OverallQual", "GrLivArea", "GarageCars", "GarageArea", "TotalBsmtSF", "1stFlrSF", "FullBath", "TotRmsAbvGrd", "YearBuilt" ]
scatter_matrix(df_train[attributes], figsize=(18, 12), diagonal='kde');
```



Exploratory Data Analysis

Analysis of various square footage features

Investigation of how GrLivArea, TotalBsmtSF, 1stFlrSF and 2ndFlrSF correlate with each other and how they correlate with SalePrice

Assumption: GrLivArea = 1stFlrSF + 2ndFlrSF + LowQualFin SF

TAKEAWAY: TotalLivArea = GrLivArea + TotalBsmtSF is more strongly correlated with SalePrice than any individual square footage feature.

```
# verify that 1stFlrSF + 2ndFlrSF = GrLivArea for all rows
df_train['GrLivArea_check'] = df_train['1stFlrSF'] + df_train['2ndFlrSF']
df_train['GrLivArea_check'].equals(df_train['GrLivArea'])
```

```
False
```

```
count = df_train[(df_train['GrLivArea'] == (df_train['1stFlrSF'] + df_train['2ndFlrSF']))].shape[0]
print(f"Number of rows where GrLivArea = 1stFlrSF + 2ndFlrSF: {count}")
```

```
Number of rows where GrLivArea = 1stFlrSF + 2ndFlrSF: 1434
```

There could be an extra variable, this happens to be LowQualFinSF

```
# verify that 1stFlrSF + 2ndFlrSF + LowQualFinSF = GrLivArea for all rows
df_train['GrLivArea_check'] = df_train['1stFlrSF'] + df_train['2ndFlrSF'] + df_train['LowQualFinSF']
df_train['GrLivArea_check'].equals(df_train['GrLivArea'])
```

```
True
```

```
df_train = df_train.drop('GrLivArea_check', axis=1)
```

```
# let's create another variable TotalLivArea = GrLivArea + TotalBsmtSF
df_train['TotalLivArea'] = df_train['GrLivArea'] + df_train['TotalBsmtSF']
```

```
attributes = ["SalePrice","TotalLivArea", "GrLivArea", "TotalBsmtSF", "1stFlrSF", "2ndFlrSF"]

corr_matrix = df_train[attributes].corr()
corr_matrix["SalePrice"].sort_values(ascending=False)
```

	SalePrice
SalePrice	1.000000
TotalLivArea	0.778959
GrLivArea	0.708624
TotalBsmtSF	0.613581
1stFlrSF	0.605852
2ndFlrSF	0.319334

dtype: float64

```
plt.figure(figsize=(10, 6))  # Adjust figure size as needed
sns.scatterplot(x='TotalLivArea', y='SalePrice', hue='OverallQual', data=df_train, palette='viridis')
plt.title('Total Living Area vs. Sale Price (Color-coded by Overall Quality)')
plt.xlabel('Total Living Area (sqft)')
plt.ylabel('Sale Price')
plt.show()
```

Larger homes tend to also have higher OverallQual

Larger homes tend to have higher quality

```
plt.figure(figsize=(10, 6))
sns.pointplot(x='OverallQual', y='TotalLivArea', data=df_train, linestyles='none', capsize=.2)  # Changed 'join' to 'linestyles'
plt.title('Mean and Standard Deviation of Total Living Area by Overall Quality')
plt.xlabel('Overall Quality')
plt.ylabel('Total Living Area (sqft)')
plt.show()
```

Analysis of various number of bathrooms metrics

BsmtFullBath, BsmtHalfBath, FullBath, HalfBath

```
df_train['TotalBaths'] = df_train['BsmtFullBath'] + (df_train['BsmtHalfBath']/2) + df_train['FullBath'] + (df_train['HalfBath']/2)
```

```
for feature in ["BsmtFullBath", "BsmtHalfBath", "FullBath", "HalfBath", "TotalBaths"]:
    correlation = df_train[feature].corr(df_train["SalePrice"])
    print(f"Correlation between {feature} and SalePrice: {correlation}")
```

```
Correlation between BsmtFullBath and SalePrice: 0.22712223313149382
Correlation between BsmtHalfBath and SalePrice: -0.016844154297359016
Correlation between FullBath and SalePrice: 0.5606637627484449
Correlation between HalfBath and SalePrice: 0.2841076755947831
Correlation between TotalBaths and SalePrice: 0.6317310679319873
```

Running a simple linear regression to evaluate which variables are most important

TAKEAWAY: Investigate neighborhood next

```
df_model = pd.get_dummies(df_train, drop_first=True)
df_model.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Columns: 249 entries, Id to SaleCondition_Partial
dtypes: bool(210), float64(3), int64(36)
memory usage: 744.4 KB
```

```
from sklearn.linear_model import RidgeCV
from sklearn.model_selection import train_test_split

X = df_model.drop('SalePrice', axis=1)  # All columns except 'SalePrice'
Y = df_model['SalePrice']

ridge = RidgeCV(alphas=[0.1, 1.0, 10.0], cv=5)  # Adjust alphas and cv as needed
ridge.fit(X, Y)

pd.set_option("display.max_rows", None)

print(pd.Series(ridge.coef_, index=X.columns).sort_values(key=abs, ascending=False))
```

Show hidden output

TAKEAWAY: Neighborhood seems to be very important.

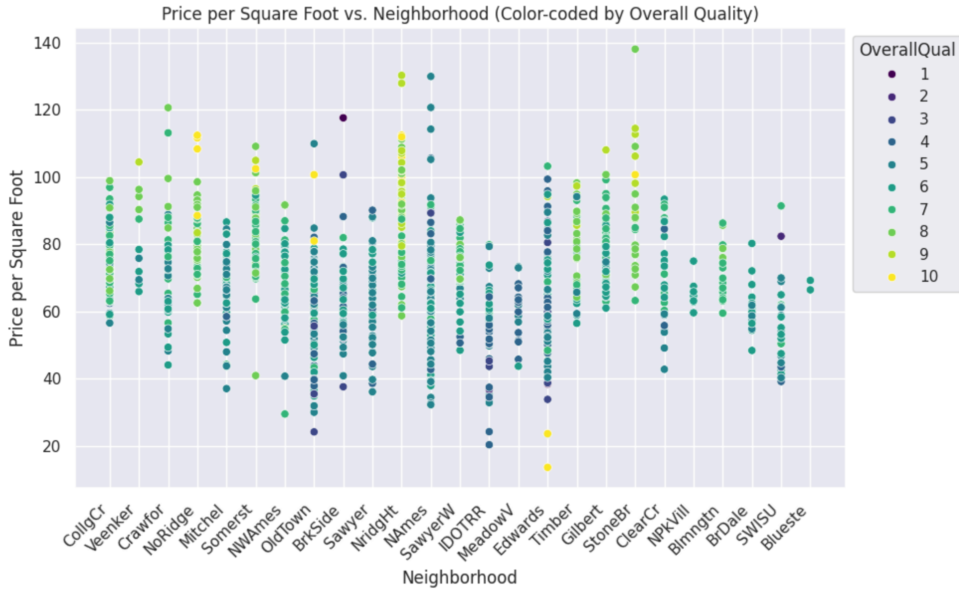
Analysis of importance of neighborhood

```
# To avoid TotalLivArea from impacting this analysis creat a feature called PricePerSqFt
df_train['PricePerSqFt'] = df_train['SalePrice']/df_train['TotalLivArea']
```

```
plt.figure(figsize=(12, 8))  # Adjust size as needed
sns.boxplot(x='Neighborhood', y='PricePerSqFt', data=df_train)
plt.title('Price per Square Foot by Neighborhood')
plt.xlabel('Neighborhood')
plt.ylabel('Price per Square Foot')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for readability
plt.show()
```

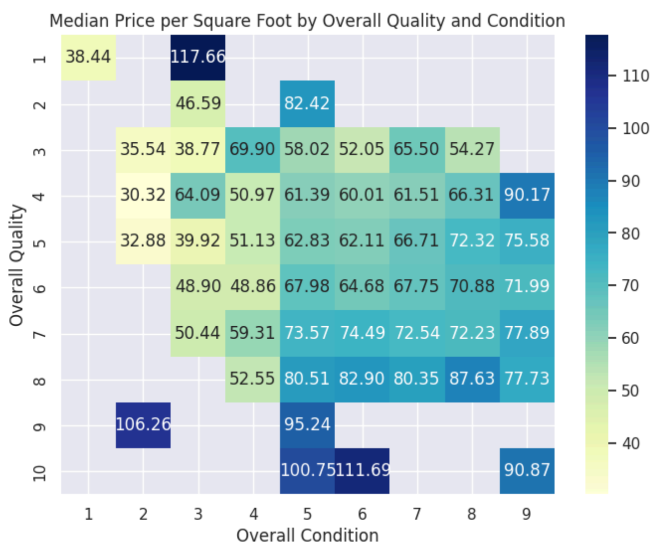

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Neighborhood', y='PricePerSqFt', hue='OverallQual', data=df_train, palette='viridis', legend = 'full') # Updated scatterplot
plt.title('Price per Square Foot vs. Neighborhood (Color-coded by Overall Quality)')
plt.xlabel('Neighborhood')
```

```
plt.xlabel('Neighborhood')
plt.ylabel('Price per Square Foot')
plt.xticks(rotation=45, ha='right')
sns.move_legend(plt.gca(), "upper left", bbox_to_anchor=(1, 1))
plt.show()
```



OverallQual vs OverallCond

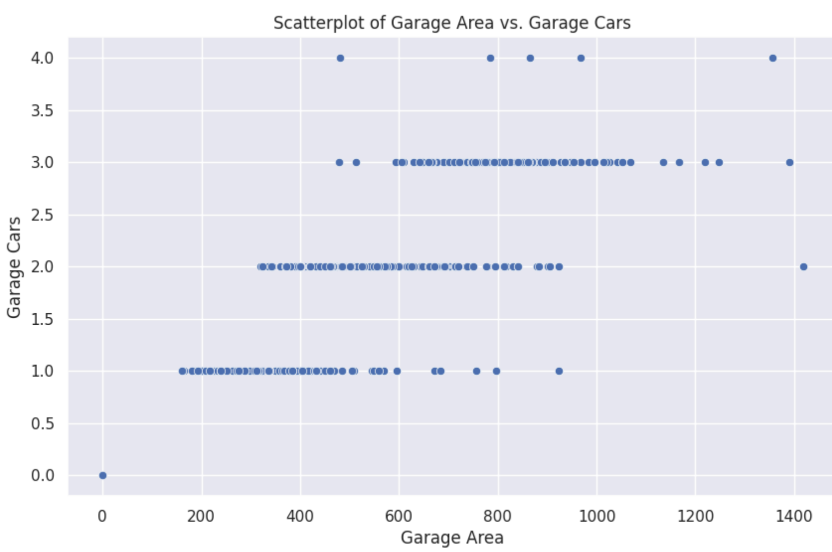
```
# Calculate median PricePerSqFt by OverallQual and OverallCond
median_price_per_sqft = df_train.groupby(['OverallQual', 'OverallCond'])['PricePerSqFt'].median().reset_index()

# Create the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(pd.pivot_table(median_price_per_sqft, values='PricePerSqFt', index='OverallQual', columns='OverallCond'),
            annot=True, fmt=".2f", cmap="YlGnBu")
plt.title('Median Price per Square Foot by Overall Quality and Condition')
plt.xlabel('Overall Condition')
plt.ylabel('Overall Quality')
plt.show()
```




```
# Calculate median SalePrice by OverallQual and OverallCond
median_sale_price = df_train.groupby(['OverallQual', 'OverallCond'])['SalePrice'].median().reset_index()

# Divide SalePrice by 1000 to display in thousands
median_sale_price['SalePrice'] = median_sale_price['SalePrice'] / 1000

# Create the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(pd.pivot_table(median_sale_price, values='SalePrice', index='OverallQual', columns='OverallCond'),
            annot=True, fmt=".0f", cmap="YlGnBu")
plt.title('Median Sale Price (in Thousands) by Overall Quality and Condition')
plt.xlabel('Overall Condition')
plt.ylabel('Overall Quality')
plt.show()
```

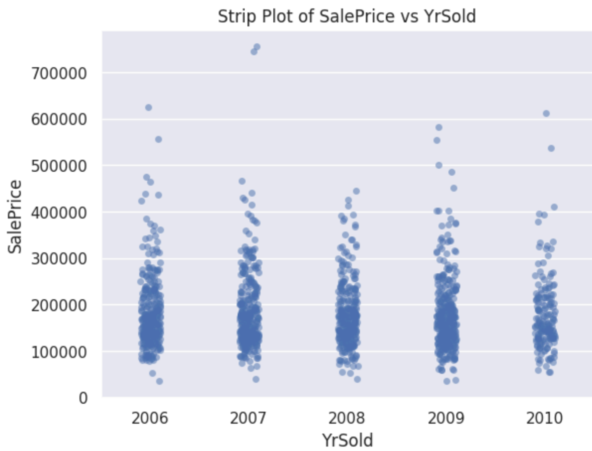
GarageArea vs GarageCars

```
# scatterplot for mapping correlation between GarageArea and GarageCars
plt.figure(figsize=(10, 6))
sns.scatterplot(x='GarageArea', y='GarageCars', data=df_train)
plt.title('Scatterplot of Garage Area vs. Garage Cars')
plt.xlabel('Garage Area')
plt.ylabel('Garage Cars')
plt.show()
```



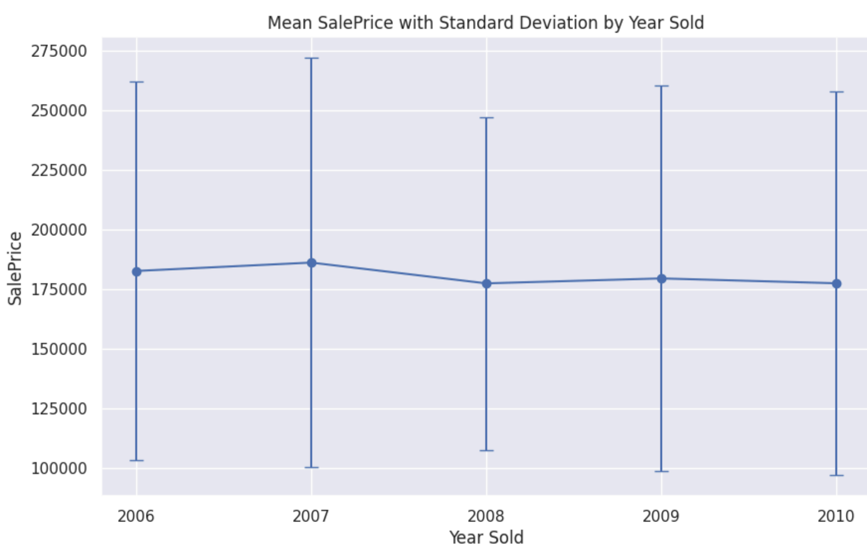
YrSold vs SalePrice

```
sns.stripplot(x='YrSold', y='SalePrice', data=df_train, jitter=True, alpha=0.5)
plt.title('Strip Plot of SalePrice vs YrSold')
plt.xlabel('YrSold')
plt.ylabel('SalePrice')
plt.show()
```



There is data across all years.

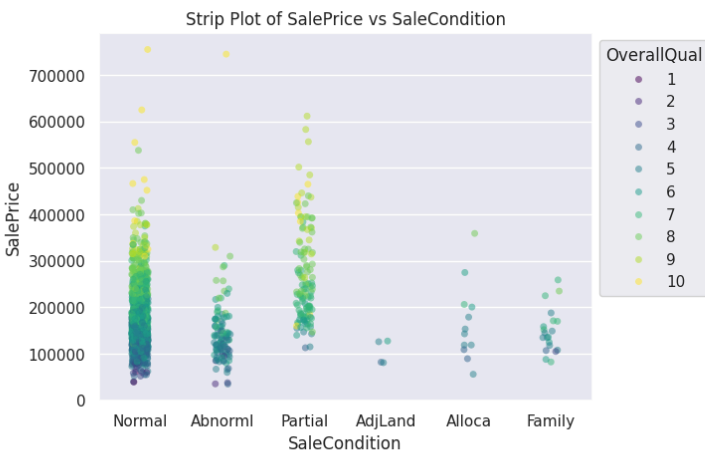
```
# Calculate mean and standard deviation of SalePrice for each year
year_stats = df_train.groupby('YrSold')['SalePrice'].agg(['mean', 'std'])

# Create line plot with error bars
plt.figure(figsize=(10, 6))
plt.errorbar(year_stats.index, year_stats['mean'], yerr=year_stats['std'], fmt='-o', capsize=5)
plt.title('Mean SalePrice with Standard Deviation by Year Sold')
plt.xlabel('Year Sold')
plt.xticks(year_stats.index)
plt.ylabel('SalePrice')
plt.show()
```



No significant change in SalePrice across years.

SaleCondition vs SalePrice

```
sns.stripplot(x='SaleCondition', y='SalePrice', data=df_train, jitter=True, alpha=0.5, hue='OverallQual', palette='viridis', legend = 'full')
plt.title('Strip Plot of SalePrice vs SaleCondition')
plt.xlabel('SaleCondition')
plt.ylabel('SalePrice')
sns.move_legend(plt.gca(), "upper left", bbox_to_anchor=(1, 1))
plt.show()
```



High quality homes tend to be Normal or Partial sales. Very few partial sales are low quality, most are average quality or higher.

Average room size

The dataset contains features such as TotRmsAbvGrd and other features that indicate number of rooms but does not provide information on the spaciousness of the house.

```
#AvgRmSize provides information on how spacious rooms above ground are.
df_train['AvgRmSize'] = df_train['GrLivArea']/df_train['TotRmsAbvGrd']


plt.figure(figsize=(10, 6))  # Adjust figure size as needed
sns.scatterplot(x='AvgRmSize', y='SalePrice', hue='OverallQual', data=df_train, palette='viridis', legend = 'full')
plt.title('Average Room Size vs. Sale Price (Color-coded by Overall Quality)')
plt.xlabel('Average Room Size (sqft)')
plt.ylabel('Sale Price')

# Add best fit lines for each OverallQual type
for quality in df_train['OverallQual'].unique():
    sns.regplot(x='AvgRmSize', y='SalePrice', data=df_train[df_train['OverallQual'] == quality],
                scatter=False, color=sns.color_palette('viridis', as_cmap=True)(quality / 10), ci = None)  # Match color to scatterplot

plt.show()
```

As average room size increases, the overall quality of the house and sale prices of the house increase.

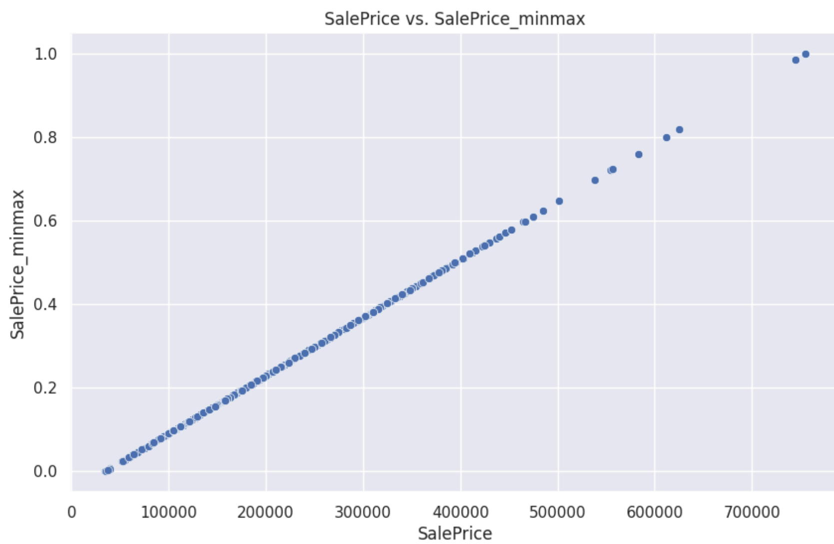
Scaling

Using the dependent variable, perform both min-max and standard scaling in Python.

Min-max scaling SalePrice

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Create a MinMaxScaler object
minmax_scaler = MinMaxScaler()

# Fit the scaler to the 'SalePrice' column and transform it
df_train['SalePrice_minmax'] = minmax_scaler.fit_transform(df_train[['SalePrice']])
```

```
plt.figure(figsize=(10, 6))  # Adjust figure size as needed
sns.scatterplot(x='SalePrice', y='SalePrice_minmax', data=df_train)
plt.title('SalePrice vs. SalePrice_minmax')
plt.xlabel('SalePrice')
plt.ylabel('SalePrice_minmax')
plt.show()
```



As expected after scaling SalePrice_minmax and SalePrice have a linear perfect correlation relationship.

Standard scaling SalePrice

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Create a StandardScaler object
standard_scaler = StandardScaler()

# Fit the scaler to the 'SalePrice' column and transform it
df_train['SalePrice_standard'] = standard_scaler.fit_transform(df_train[['SalePrice']])
```

```
plt.figure(figsize=(10, 6))  # Adjust figure size as needed
sns.scatterplot(x='SalePrice', y='SalePrice_standard', data=df_train)
plt.title('SalePrice vs. SalePrice_standard')
plt.xlabel('SalePrice')
plt.ylabel('SalePrice_standard')
plt.show()
```