rocchio-assign1

July 3, 2023

Compute Resource Allocation

[1]:
```python
### I am going to spool an AWS instance as my local enviornment is broken at␣
↪the moment. I am checking to see if I am on the right instance below with␣
↪each run.
import subprocess
import psutil
values = psutil.virtual_memory()
total = values.total >> 30

cpuinfo=((subprocess.check_output("lscpu", shell=True).strip()).decode()).
↪splitlines()

for i in cpuinfo[0:8]:
    print(i)
print('Total Ram: {} Gb'.format(str(total).rjust(25)))
```

```
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Address sizes:                   48 bits physical, 48 bits virtual
Byte Order:                      Little Endian
CPU(s):                          64
On-line CPU(s) list:             0-63
Vendor ID:                       AuthenticAMD
Model name:                      AMD EPYC 7R13 Processor
Total Ram:                       123 Gb
```

1 Data Survey

A Data Survey: The Ames housing dataset represents residential property sales in Ames, Iowa from 2006 to 2010. The dataset includes a vast range of variables, totaling to 80 independent features that capture almost every aspect of residential homes. These features encompass a multitude of aspects including the physical properties of the dwelling (like area, number of rooms, type of utilities), the quality and condition of various elements of the property (exteriors, kitchen, heating, etc.), time-related information (year built, year remodeled), neighborhood details, and various other attributes (e.g., proximity to main road or railway).

With our objective of predicting the value of a property, the dataset seems quite appropriate given its comprehensive set of housing-related variables that are likely to influence property value. However, the extensive nature of this dataset may include features that may not have a strong influence on the target variable and hence, feature selection would be a crucial aspect of model development. Additionally, the presence of missing values in many of the columns necessitates thorough data cleaning and imputation.

Although the data provides a wide spectrum of information about the properties, some observations should be treated with caution. Properties with extreme characteristics or values, such as very high square footage or highly unusual features, may act as outliers and could potentially distort the model's predictions. It would be prudent to explore the dataset for such outliers and decide how to handle them, either through exclusion, transformation, or robust modeling techniques.

Given the dataset, we can address a variety of questions related to property valuation, such as identifying key drivers of property price, analyzing the influence of various property features on price, predicting property prices, and more. We can build a regression model with SalePrice as the target variable, with the understanding that we are predicting the price of residential properties in Ames, Iowa specifically between 2006 and 2010. However, it's crucial to keep in mind that the model's applicability might be limited to similar real estate markets and periods, given market dynamics and inflation effects.

Lastly, the dataset does not include information about the price per square foot, which is often a key metric in real estate valuation. If we were to generate this feature, it would allow us to assess property value in a more standardized way, accounting for differences in size. However, as it stands, the data allows us to estimate overall sale prices, assuming that we appropriately handle the complexities of the data, including missing data, outliers, and skewed distributions.

[2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

df = pd.read_csv("ames_housing_data.csv")
df['TotalFloorSF'] = df['FirstFlrSF'] + df['SecondFlrSF']
df['HouseAge'] = df['YrSold'] - df['YearBuilt']
df['QualityIndex'] = df['OverallQual'] * df['OverallCond']
df['logSalePrice'] = np.log(df['SalePrice'])
df['price_sqft'] = df['SalePrice'] / df['TotalFloorSF']
print(df.shape)
print(df.info())
print(df.head())
print(df.columns)
print(df['price_sqft'].describe())
```

```
(2930, 87)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2930 entries, 0 to 2929
Data columns (total 87 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
```

```
0   SID             2930 non-null   int64
1   PID             2930 non-null   int64
2   SubClass        2930 non-null   int64
3   Zoning          2930 non-null   object
4   LotFrontage     2440 non-null   float64
5   LotArea         2930 non-null   int64
6   Street          2930 non-null   object
7   Alley           198 non-null    object
8   LotShape        2930 non-null   object
9   LandContour     2930 non-null   object
10  Utilities       2930 non-null   object
11  LotConfig       2930 non-null   object
12  LandSlope       2930 non-null   object
13  Neighborhood    2930 non-null   object
14  Condition1      2930 non-null   object
15  Condition2      2930 non-null   object
16  BldgType        2930 non-null   object
17  HouseStyle      2930 non-null   object
18  OverallQual     2930 non-null   int64
19  OverallCond     2930 non-null   int64
20  YearBuilt       2930 non-null   int64
21  YearRemodel     2930 non-null   int64
22  RoofStyle       2930 non-null   object
23  RoofMat         2930 non-null   object
24  Exterior1       2930 non-null   object
25  Exterior2       2930 non-null   object
26  MasVnrType      1155 non-null   object
27  MasVnrArea      2907 non-null   float64
28  ExterQual       2930 non-null   object
29  ExterCond       2930 non-null   object
30  Foundation      2930 non-null   object
31  BsmtQual        2850 non-null   object
32  BsmtCond        2850 non-null   object
33  BsmtExposure    2847 non-null   object
34  BsmtFinType1    2850 non-null   object
35  BsmtFinSF1      2929 non-null   float64
36  BsmtFinType2    2849 non-null   object
37  BsmtFinSF2      2929 non-null   float64
38  BsmtUnfSF       2929 non-null   float64
39  TotalBsmtSF     2929 non-null   float64
40  Heating         2930 non-null   object
41  HeatingQC       2930 non-null   object
42  CentralAir      2930 non-null   object
43  Electrical      2929 non-null   object
44  FirstFlrSF      2930 non-null   int64
45  SecondFlrSF     2930 non-null   int64
46  LowQualFinSF    2930 non-null   int64
47  GrLivArea       2930 non-null   int64
```

```
 48  BsmtFullBath   2928 non-null   float64
 49  BsmtHalfBath   2928 non-null   float64
 50  FullBath       2930 non-null   int64
 51  HalfBath       2930 non-null   int64
 52  BedroomAbvGr   2930 non-null   int64
 53  KitchenAbvGr   2930 non-null   int64
 54  KitchenQual    2930 non-null   object
 55  TotRmsAbvGrd   2930 non-null   int64
 56  Functional     2930 non-null   object
 57  Fireplaces     2930 non-null   int64
 58  FireplaceQu    1508 non-null   object
 59  GarageType     2773 non-null   object
 60  GarageYrBlt    2771 non-null   float64
 61  GarageFinish   2771 non-null   object
 62  GarageCars     2929 non-null   float64
 63  GarageArea     2929 non-null   float64
 64  GarageQual     2771 non-null   object
 65  GarageCond     2771 non-null   object
 66  PavedDrive     2930 non-null   object
 67  WoodDeckSF     2930 non-null   int64
 68  OpenPorchSF    2930 non-null   int64
 69  EnclosedPorch  2930 non-null   int64
 70  ThreeSsnPorch  2930 non-null   int64
 71  ScreenPorch    2930 non-null   int64
 72  PoolArea       2930 non-null   int64
 73  PoolQC         13 non-null     object
 74  Fence          572 non-null    object
 75  MiscFeature    106 non-null    object
 76  MiscVal        2930 non-null   int64
 77  MoSold         2930 non-null   int64
 78  YrSold         2930 non-null   int64
 79  SaleType       2930 non-null   object
 80  SaleCondition  2930 non-null   object
 81  SalePrice      2930 non-null   int64
 82  TotalFloorSF   2930 non-null   int64
 83  HouseAge       2930 non-null   int64
 84  QualityIndex   2930 non-null   int64
 85  logSalePrice   2930 non-null   float64
 86  price_sqft     2930 non-null   float64
dtypes: float64(13), int64(31), object(43)
memory usage: 1.9+ MB
None
   SID        PID  SubClass Zoning  LotFrontage  LotArea Street Alley  \
0    1  526301100        20     RL        141.0    31770   Pave   NaN
1    2  526350040        20     RH         80.0    11622   Pave   NaN
2    3  526351010        20     RL         81.0    14267   Pave   NaN
3    4  526353030        20     RL         93.0    11160   Pave   NaN
4    5  527105010        60     RL         74.0    13830   Pave   NaN
```

```
  LotShape LandContour  … MoSold YrSold SaleType SaleCondition SalePrice  \
0      IR1         Lvl  …      5   2010       WD        Normal    215000
1      Reg         Lvl  …      6   2010       WD        Normal    105000
2      IR1         Lvl  …      6   2010       WD        Normal    172000
3      Reg         Lvl  …      4   2010       WD        Normal    244000
4      IR1         Lvl  …      3   2010       WD        Normal    189900

  TotalFloorSF HouseAge QualityIndex  logSalePrice  price_sqft
0         1656       50           30     12.278393  129.830918
1          896       49           30     11.561716  117.187500
2         1329       52           36     12.055250  129.420617
3         2110       42           35     12.404924  115.639810
4         1629       13           25     12.154253  116.574586

[5 rows x 87 columns]
Index(['SID', 'PID', 'SubClass', 'Zoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodel',
       'RoofStyle', 'RoofMat', 'Exterior1', 'Exterior2', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', 'FirstFlrSF', 'SecondFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', 'ThreeSsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice', 'TotalFloorSF', 'HouseAge',
       'QualityIndex', 'logSalePrice', 'price_sqft'],
      dtype='object')
count    2930.000000
mean      121.595174
std        31.893333
min        15.371394
25%       100.573697
50%       120.429633
75%       140.009301
max       276.250881
Name: price_sqft, dtype: float64
```

[3]:
```
df=df.replace('None', pd.NaT)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2930 entries, 0 to 2929
Data columns (total 87 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   SID             2930 non-null   int64
 1   PID             2930 non-null   int64
 2   SubClass        2930 non-null   int64
 3   Zoning          2930 non-null   object
 4   LotFrontage     2440 non-null   float64
 5   LotArea         2930 non-null   int64
 6   Street          2930 non-null   object
 7   Alley           198 non-null    object
 8   LotShape        2930 non-null   object
 9   LandContour     2930 non-null   object
 10  Utilities       2930 non-null   object
 11  LotConfig       2930 non-null   object
 12  LandSlope       2930 non-null   object
 13  Neighborhood    2930 non-null   object
 14  Condition1      2930 non-null   object
 15  Condition2      2930 non-null   object
 16  BldgType        2930 non-null   object
 17  HouseStyle      2930 non-null   object
 18  OverallQual     2930 non-null   int64
 19  OverallCond     2930 non-null   int64
 20  YearBuilt       2930 non-null   int64
 21  YearRemodel     2930 non-null   int64
 22  RoofStyle       2930 non-null   object
 23  RoofMat         2930 non-null   object
 24  Exterior1       2930 non-null   object
 25  Exterior2       2930 non-null   object
 26  MasVnrType      1155 non-null   object
 27  MasVnrArea      2907 non-null   float64
 28  ExterQual       2930 non-null   object
 29  ExterCond       2930 non-null   object
 30  Foundation      2930 non-null   object
 31  BsmtQual        2850 non-null   object
 32  BsmtCond        2850 non-null   object
 33  BsmtExposure    2847 non-null   object
 34  BsmtFinType1    2850 non-null   object
 35  BsmtFinSF1      2929 non-null   float64
 36  BsmtFinType2    2849 non-null   object
 37  BsmtFinSF2      2929 non-null   float64
 38  BsmtUnfSF       2929 non-null   float64
 39  TotalBsmtSF     2929 non-null   float64
 40  Heating         2930 non-null   object
 41  HeatingQC       2930 non-null   object
 42  CentralAir      2930 non-null   object
```

```
 43  Electrical     2929 non-null   object
 44  FirstFlrSF     2930 non-null   int64
 45  SecondFlrSF    2930 non-null   int64
 46  LowQualFinSF   2930 non-null   int64
 47  GrLivArea      2930 non-null   int64
 48  BsmtFullBath   2928 non-null   float64
 49  BsmtHalfBath   2928 non-null   float64
 50  FullBath       2930 non-null   int64
 51  HalfBath       2930 non-null   int64
 52  BedroomAbvGr   2930 non-null   int64
 53  KitchenAbvGr   2930 non-null   int64
 54  KitchenQual    2930 non-null   object
 55  TotRmsAbvGrd   2930 non-null   int64
 56  Functional     2930 non-null   object
 57  Fireplaces     2930 non-null   int64
 58  FireplaceQu    1508 non-null   object
 59  GarageType     2773 non-null   object
 60  GarageYrBlt    2771 non-null   float64
 61  GarageFinish   2771 non-null   object
 62  GarageCars     2929 non-null   float64
 63  GarageArea     2929 non-null   float64
 64  GarageQual     2771 non-null   object
 65  GarageCond     2771 non-null   object
 66  PavedDrive     2930 non-null   object
 67  WoodDeckSF     2930 non-null   int64
 68  OpenPorchSF    2930 non-null   int64
 69  EnclosedPorch  2930 non-null   int64
 70  ThreeSsnPorch  2930 non-null   int64
 71  ScreenPorch    2930 non-null   int64
 72  PoolArea       2930 non-null   int64
 73  PoolQC         13 non-null     object
 74  Fence          572 non-null    object
 75  MiscFeature    106 non-null    object
 76  MiscVal        2930 non-null   int64
 77  MoSold         2930 non-null   int64
 78  YrSold         2930 non-null   int64
 79  SaleType       2930 non-null   object
 80  SaleCondition  2930 non-null   object
 81  SalePrice      2930 non-null   int64
 82  TotalFloorSF   2930 non-null   int64
 83  HouseAge       2930 non-null   int64
 84  QualityIndex   2930 non-null   int64
 85  logSalePrice   2930 non-null   float64
 86  price_sqft     2930 non-null   float64
dtypes: float64(13), int64(31), object(43)
memory usage: 1.9+ MB
```

2 Define the Sample Population:

When we set out to build a predictive model, it's crucial to define the population that the model is intended to serve. In this case, our aim is to estimate home values for "typical" homes in Ames, Iowa. Defining what constitutes "typical" may be subjective, but we can utilize the data to discern what might be considered "atypical". By excluding these atypical observations, we effectively define our population of interest.

In the context of our objective, it is necessary to clarify that not all properties are created equal. The value of a property can vary drastically based on its type. For example, a single-family residence, an apartment building, a warehouse, or a shopping center each carry different value propositions, driven by different factors. Including all these different types of properties in the same model would complicate the interpretation of our results and might lead to erroneous conclusions.

Therefore, for our purpose, it would be sensible to focus on single-family residences, as these constitute a significant portion of residential property transactions and represent a homogeneous category of properties, making our model more interpretable and robust.

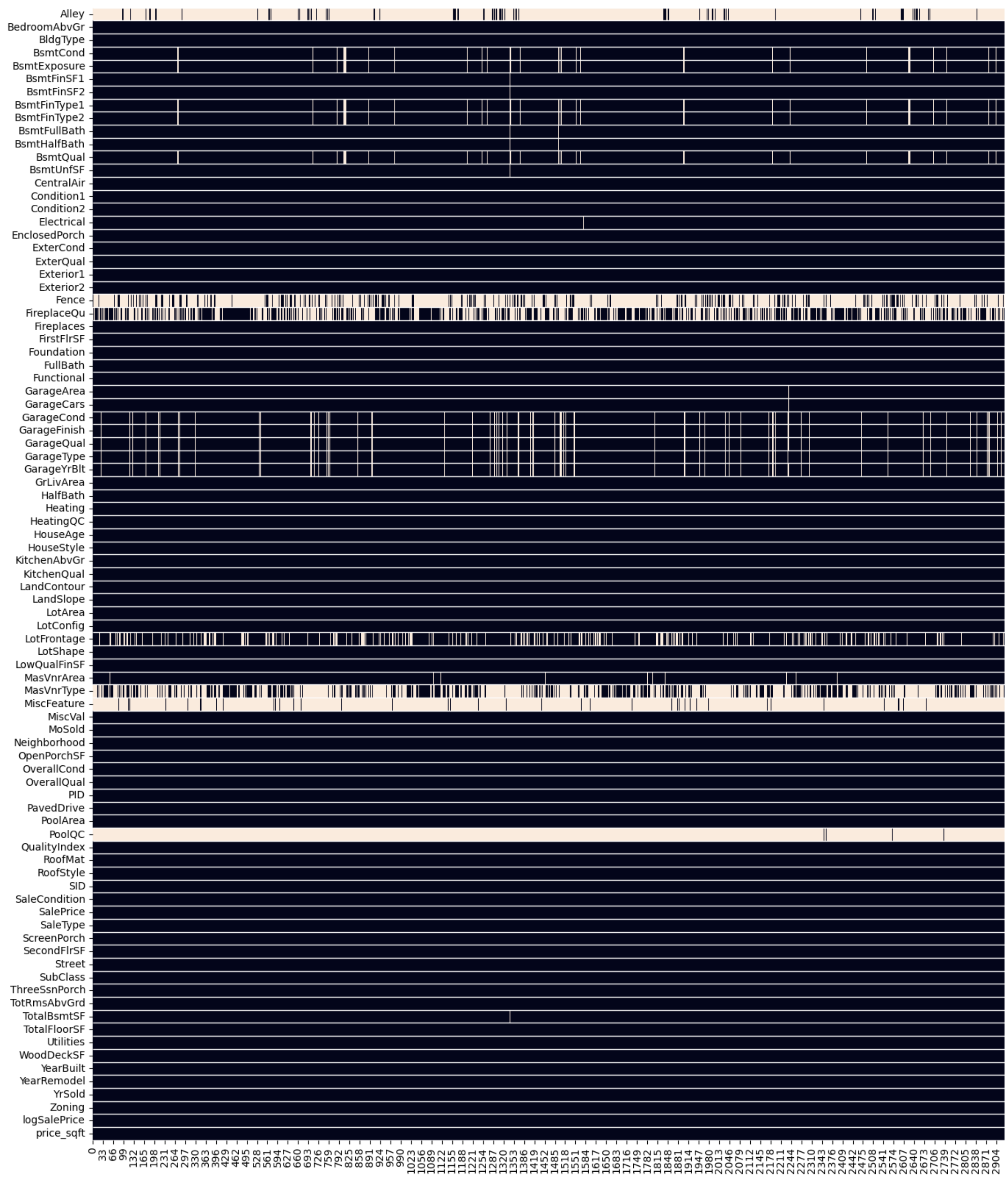
We can define our sample population using a series of 'drop conditions'. For instance, we can exclude observations where the 'BldgType' (Building Type) is not '1Fam' (Single-family Detached). Similarly, we may also want to exclude properties with extremely high or low values of 'GrLivArea' (Above grade (ground) living area square feet) as they could represent atypical properties.

The 'drop conditions' are listed below as follows:

2.1 Data Quality

2.1.1 Null Analysis and Cleanup (Nulls Represent Below in White)

[4]:
```
fig, ax = plt.subplots(figsize=(16, 20))
plt.rcParams["axes.grid"] = True

df_heat=df.sort_index(axis=1, ascending=False)
sns.heatmap(df_heat.T.isnull(), ax=ax, cbar=False).invert_yaxis()
ax.hlines(range(len(df_heat)), *ax.get_xlim(), color='white', linewidths=1)
ax.vlines([], [], [])
plt.yticks(rotation = 360)
plt.show()
```



[5]:
```python
Nulls=[]
for i in df.columns:
    if df[i].isnull().sum() > 0:
        Nulls.append(i)
        print("Column: {} Null Count: {}".format((i+"                    ")[0:
↪18], df[i].isnull().sum()))
```

```
Column: LotFrontage        Null Count: 490
```

```
Column: Alley                Null Count: 2732
Column: MasVnrType           Null Count: 1775
Column: MasVnrArea           Null Count: 23
Column: BsmtQual             Null Count: 80
Column: BsmtCond             Null Count: 80
Column: BsmtExposure         Null Count: 83
Column: BsmtFinType1         Null Count: 80
Column: BsmtFinSF1           Null Count: 1
Column: BsmtFinType2         Null Count: 81
Column: BsmtFinSF2           Null Count: 1
Column: BsmtUnfSF            Null Count: 1
Column: TotalBsmtSF          Null Count: 1
Column: Electrical           Null Count: 1
Column: BsmtFullBath         Null Count: 2
Column: BsmtHalfBath         Null Count: 2
Column: FireplaceQu          Null Count: 1422
Column: GarageType           Null Count: 157
Column: GarageYrBlt          Null Count: 159
Column: GarageFinish         Null Count: 159
Column: GarageCars           Null Count: 1
Column: GarageArea           Null Count: 1
Column: GarageQual           Null Count: 159
Column: GarageCond           Null Count: 159
Column: PoolQC               Null Count: 2917
Column: Fence                Null Count: 2358
Column: MiscFeature          Null Count: 2824
```

2.1.2 LotFrontage

[6]:
```python
df['LotFrontage']=df['LotFrontage'].fillna(df['LotFrontage'].median())
```

2.1.3 Alley

[7]:
```python
df['Alley']=df['Alley'].fillna('No alley')
```

2.1.4 MasVnrType & MasVnrArea: Filling with None and 0

[8]:
```python
df['MasVnrType']=df['MasVnrType'].fillna('None')
df['MasVnrArea']=df['MasVnrArea'].fillna(0)
```

2.1.5 BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2

[9]:
```python
for col in ['BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1',␣
↪'BsmtFinType2']:
    df[col].fillna('No basement')
```

2.1.6 Categorical garage-related columns (fill with 'No garage' or 0 because NaN likely means no garage). For the year we will fill it with the median.

[10]:
```
for col in ['GarageType', 'GarageFinish', 'GarageQual', 'GarageCond']:
    if df[col].dtype == 'object':
        df[col]=df[col].fillna('No garage')
    else:
        df[col]=df[col].fillna('None')

df['GarageYrBlt']=df['GarageYrBlt'].fillna(df['GarageYrBlt'].median())
```

2.1.7 Numeric garage-related columns: Fill with zero Assuming there is no garage.

[11]:
```
df['GarageCars']=df['GarageCars'].fillna(0)
df['GarageArea']=df['GarageArea'].fillna(0)
```

PoolQC: fill with 'No pool' because NaN likely means no pool

[12]:
```
df['PoolQC']=df['PoolQC'].fillna('No pool')
```

2.1.8 Fence: fill with 'No fence' because NaN likely means no fence

[13]:
```
df['Fence']=df['Fence'].fillna('No fence')
```

2.1.9 MiscFeature: fill with 'No feature' because NaN likely means there are no additional features

[14]:
```
df['MiscFeature']=df['MiscFeature'].fillna('No feature')
```

2.1.10 Electrical: As this is a categorical feature, we can replace missing values with the most common class

[15]:
```
df['Electrical']=df['Electrical'].fillna(df['Electrical'].mode()[0])
```

2.1.11 FireplaceQu: NaN probably means no fireplace

[16]:
```
df['FireplaceQu']=df['FireplaceQu'].fillna('No fireplace')
```

2.1.12 BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, and BsmtFinType2: Fill missing categorical basement columns with 'No basement'

[17]:
```
for col in ['BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1',␣
↪'BsmtFinType2']:
    df[col]=df[col].fillna('No basement')
```

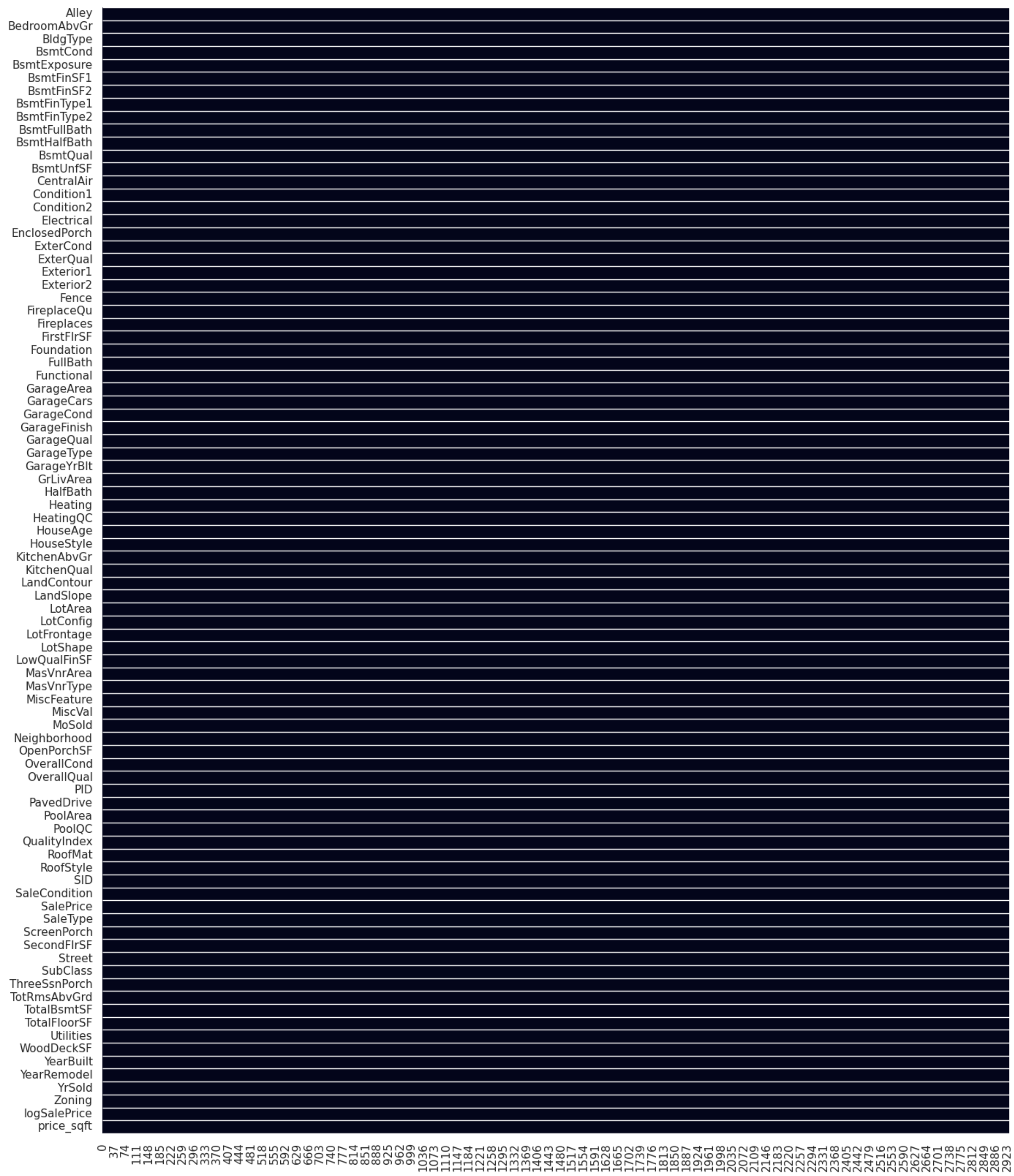
2.1.13 BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, BsmtFullBath, & BsmtHalfBath: Fill missing numerical basement columns with 0

[18]:
```
for col in ['BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF',␣
↪'BsmtFullBath', 'BsmtHalfBath']:
    df[col]=df[col].fillna(0)
```

2.2 Here is our cleaned dataset

2.2.1 No Rows were dropped.

[19]:
```
sns.set()
fig, ax1 = plt.subplots(figsize=(16, 20))

df_heat1=df.sort_index(axis=1, ascending=False)
sns.heatmap(df_heat1.T.isnull(), ax=ax1, cbar=False).invert_yaxis()
ax1.hlines(range(len(df_heat)), *ax1.get_xlim(), color='white', linewidths=1)
ax1.vlines([], [], [])
plt.yticks(rotation = 360)
plt.show()
```



3 A Data Quality Check

Even the most meticulously collected datasets can have issues with data quality, be it in the form of errors, outliers, or missing values. Beacuse of this, it is a fundamental step in any data analysis process to conduct a thorough data quality check.

If a data dictionary is available, it provides a valuable guide to what each field in the dataset should contain. Comparing the actual data against this dictionary can help identify inconsistencies and errors. However, even without a data dictionary, logical reasoning and exploratory analysis can assist in identifying and rectifying data quality issues.

For instance, in this project where we aim to model the sales price of houses, it's evident that a sales price of zero or negative doesn't make sense and should be considered an error.

Another example is handling outliers. If we have a small number of housing transactions with a sale price significantly higher than the majority, say over a million dollars, these could be valid but rare occurrences (outliers) or they could be errors. In either case, if our objective is to model 'typical' home prices, these high-value transactions may not be relevant and might skew our model.

Python libraries like pandas, numpy, and seaborn provide various functions that can be used to perform a data quality check. Let's select twenty variables from the dataset for this check. We'll use functions like describe(), value_counts(), isna().sum() for descriptive statistics, frequency counts, and null value check, respectively. Also, we'll use visualizations like histograms and box plots to check for outliers.

Based on the information below here is a comprehensive data quality check for each of our choosen variables:

SalePrice: This is our target variable. The mean sale price of houses is around 180,796 dollars. The minimum sale price is 12,789 dollars, while the maximum is 755,000 dollars. There are no zero or negative sale prices, which is consistent with our expectations.

OverallQual: This variable ranges from 1 to 10, with an average of around 6. There doesn't appear to be any data quality issues with this variable.

GrLivArea: The average ground living area is approximately 1500 square feet. There are some houses with large ground living area values that could be considered outliers. We may need to investigate these further or handle them appropriately during modeling.

GarageCars: This variable, indicating the size of the garage in car capacity, ranges from 0 to 5. The average garage size is around 1.76 cars.

GarageArea: The average garage area is about 472 square feet. Similar to GrLivArea, there are some large values that may be outliers.

GarageYrBlt: This variable seems to have missing data, denoted as "None". We'll need to decide how to handle these missing values, perhaps by filling in with an appropriate value or removing these records.

MasVnrArea: The average area of masonry veneer is around 101 square feet, but half of the houses don't have masonry veneer (median and 25th percentile is zero). There are a few large values that may be considered outliers.

Fireplaces: The majority of the houses have either one or no fireplaces. However, there are some houses with up to four fireplaces.

BsmtFinSF1: The average finished square feet of the basement area is around 442. However, many houses do not have a finished basement area (indicated by zero values).

LotFrontage: The average linear feet of street connected to the property is around 69 feet. There are some properties with exceptionally large frontage.

TotalBsmtSF: This variable represents the total square feet of the basement area. The average is about 1051 square feet. Similar to GrLivArea, it appears that there are some outliers present with an exceptionally large basement area.

FirstFlrSF: This variable represents the First Floor square feet. The average is around 1159 square feet. The data range from 334 to 5095 square feet, suggesting a wide range of house sizes.

FullBath: The average number of full bathrooms is approximately 1.57. The values range from 0 to 4. This variable does not seem to have any anomalies or outliers.

TotRmsAbvGrd: This variable represents the total rooms above grade (does not include bathrooms). The average number is around 6.44, and the range is from 2 to 15. There don't appear to be any data quality issues with this variable.

YearBuilt: This variable indicates the original construction date. The average year of construction is around 1971. It ranges from 1872 to 2010. There doesn't appear to be any data quality issues with this variable.

LotFrontage: This variable measures the Linear feet of street connected to the property. The average lot frontage is around 69 feet, but some houses have as much as 313 feet of street connected to the property.

WoodDeckSF: This variable measures the wood deck area in square feet. On average, houses have around 94 square feet of wood deck area. However, many houses don't have a wood deck at all (indicated by the median and 25th percentile being zero), while some have very large wood decks.

SecondFlrSF: This variable measures the second floor area in square feet. On average, houses have around 335 square feet of the second-floor area. However, many houses don't have a second floor at all (indicated by the median and 25th percentile being zero), while some have a very large second floor.

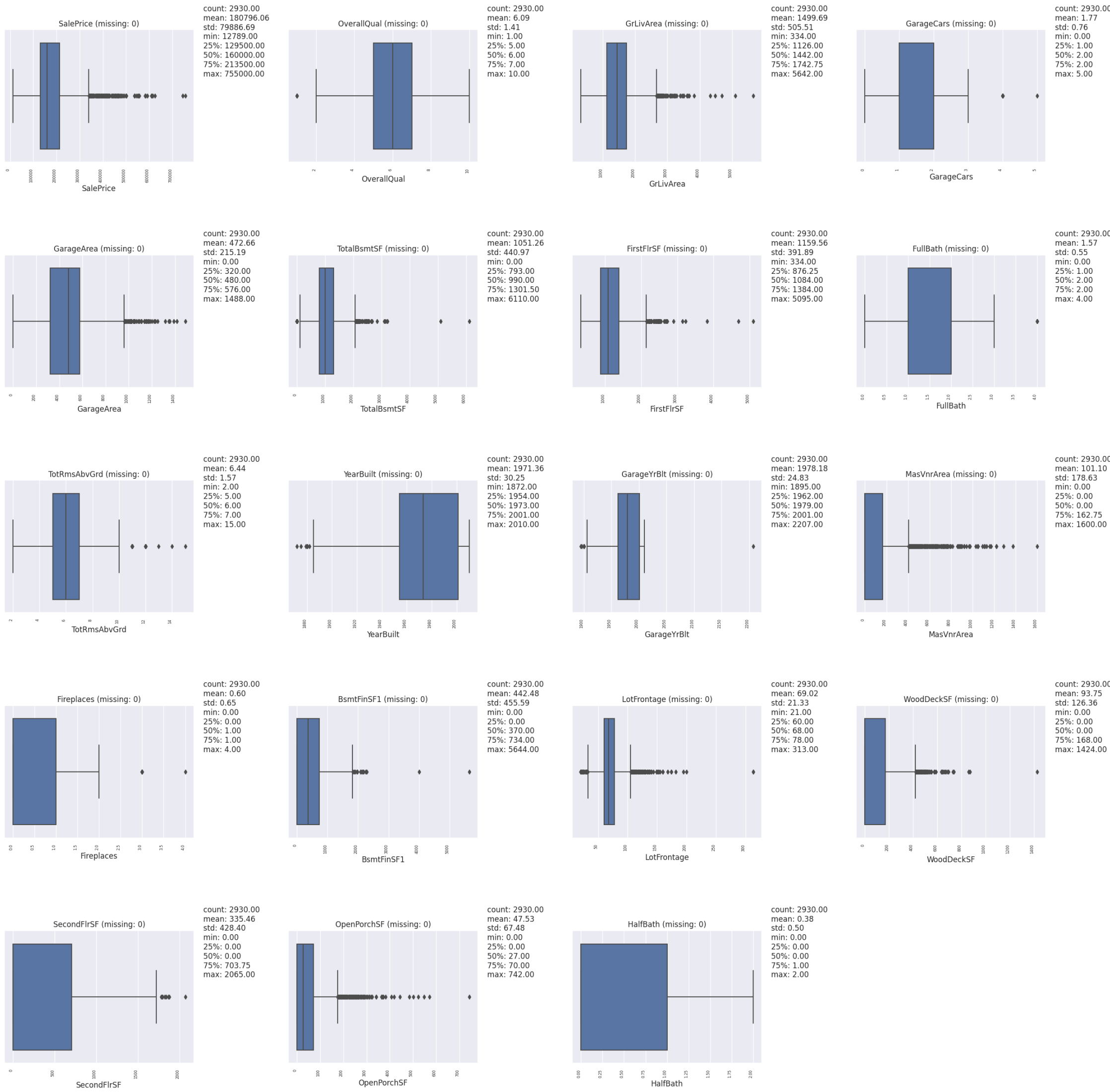
OpenPorchSF: This variable measures the open porch area in square feet. The average open porch area is around 48 square feet. However, many houses don't have an open porch (indicated by the median and 25th percentile being zero), while some have a very large open porch.

HalfBath: The average number of half bathrooms is around 0.38, indicating that many houses do not have a half bathroom. The variable ranges from 0 to 2, and there doesn't seem to be any anomalies or outliers.

[20]:
```
variables_to_check = ['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars',␣
 ↪'GarageArea',
                      'TotalBsmtSF', 'FirstFlrSF', 'FullBath', 'TotRmsAbvGrd',␣
 ↪'YearBuilt',
                      'GarageYrBlt', 'MasVnrArea', 'Fireplaces',
                      'BsmtFinSF1', 'LotFrontage', 'WoodDeckSF', 'SecondFlrSF',
                      'OpenPorchSF', 'HalfBath']
```

[21]:
```
fig, axs = plt.subplots(5, 4, figsize=(30, 30))
axs = axs.ravel()
df['GarageYrBlt']=round(df['GarageYrBlt'].astype(float),0)
for i, var in enumerate(variables_to_check):
    if df[var].dtype in ['int64', 'float64']:
```

```
        summary_stats = df[var].describe()
        sns.boxplot(x=df[var], ax=axs[i])

        axs[i].set_title(f'{var} (missing: {df[var].isna().sum()})')
        stats_text = "\n".join([f'{stat}: {value:.2f}' for stat, value in␣
↪summary_stats.items()])
        axs[i].annotate(stats_text, xy=(1.05, .65), xycoords=axs[i].transAxes)

    else:
        sns.countplot(x=df[var].dropna(), ax=axs[i])
        axs[i].set_title(f'{var} (missing: {df[var].isna().sum()})')

    for label in axs[i].get_xticklabels():
        label.set_rotation(90)
        label.set_horizontalalignment('right')
        label.set_fontsize(6)

for i in range(len(variables_to_check), 20):
    fig.delaxes(axs[i])

plt.subplots_adjust(wspace=0.5, hspace=0.7)
plt.show()
```



[23]:
```python
### Everything looks right to me but the year built has one odd value. I am␣
↪going to fill it with the median.

df['GarageYrBlt']=df['GarageYrBlt'].replace(df['GarageYrBlt'].max(),␣
↪df['GarageYrBlt'].median())
df['GarageYrBlt'].describe()
```

[23]:
```
count    2930.000000
mean     1978.101706
std        24.463835
min      1895.000000
25%      1962.000000
50%      1979.000000
```

```
75%       2001.000000
max       2010.000000
Name: GarageYrBlt, dtype: float64
```

4 An Initial Exploratory Data Analysis

I will be choosing the following variables:

Continuous Variables: LotFrontage: The scatterplot of 'SalePrice' and 'LotFrontage' shows a positive correlation. Properties with a larger street connected to their lot seem to fetch higher prices. The boxplot shows a few outliers, with some properties having unusually large frontages.

LotArea: This variable also indicates a positive correlation with 'SalePrice', indicating larger properties tend to have higher prices. The boxplot for 'LotArea' also reveals the presence of several outliers.

FirstFlrSF: First floor square feet, again, shows a positive correlation with 'SalePrice'. Houses with larger first floors are more expensive. The boxplot shows a few properties with unusually large first-floor areas.

GrLivArea: Above grade (ground) living area square feet shows a strong positive correlation with 'SalePrice'. The larger the living area, the higher the property's price. Outliers are present but fewer compared to other variables.

GarageArea: This variable too exhibits a positive correlation with 'SalePrice'. Houses with larger garage areas demand higher prices. There are a few outliers as seen in the boxplot. Discrete or Categorical Variables:

YearBuilt: The year in which the property was built can have a significant impact on its sale price. Generally, newer houses tend to command higher prices. The boxplot of 'SalePrice' for each year reveals that some years have a higher median sale price compared to others, indicating the influence of the year of construction on the property's value.

OverallQual: The overall quality shows a strong correlation with 'SalePrice'. Properties with better overall quality score higher prices.

OverallCond: While one might expect a significant impact of overall condition on 'SalePrice', it seems to have a more nuanced relationship with price.

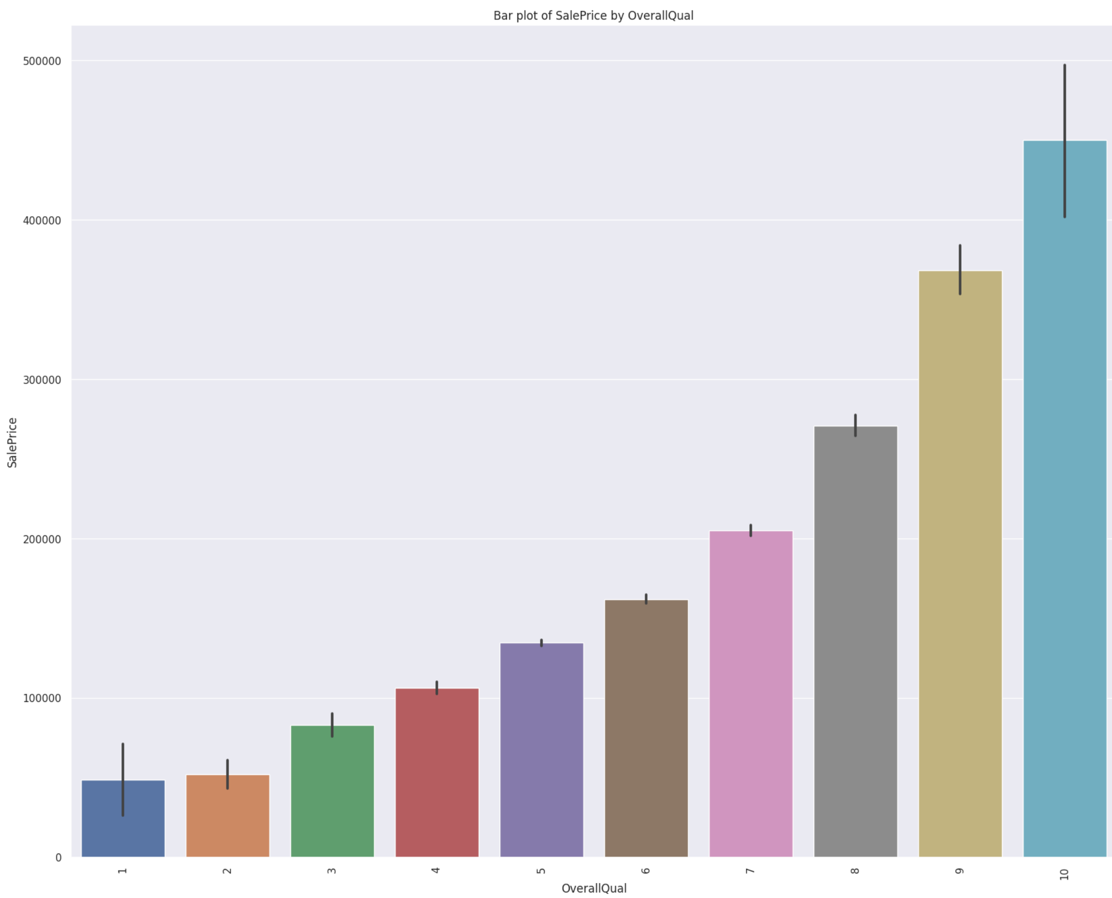
BsmtFullBath: Properties with more full bathrooms in the basement are likely to fetch higher prices. However, there's significant variation within each category.

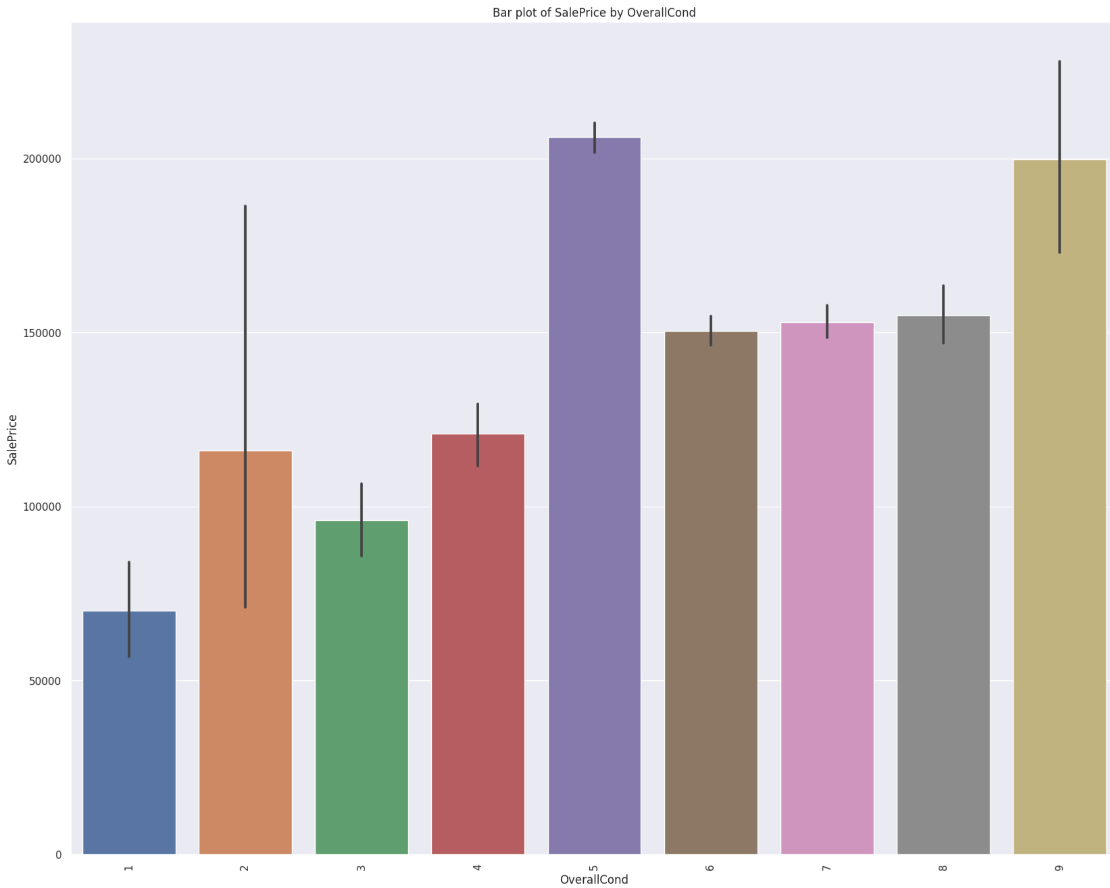
GarageCars: The capacity of the garage (in car capacity) significantly influences the 'SalePrice'. Properties with larger garages (in terms of car capacity) have higher prices.

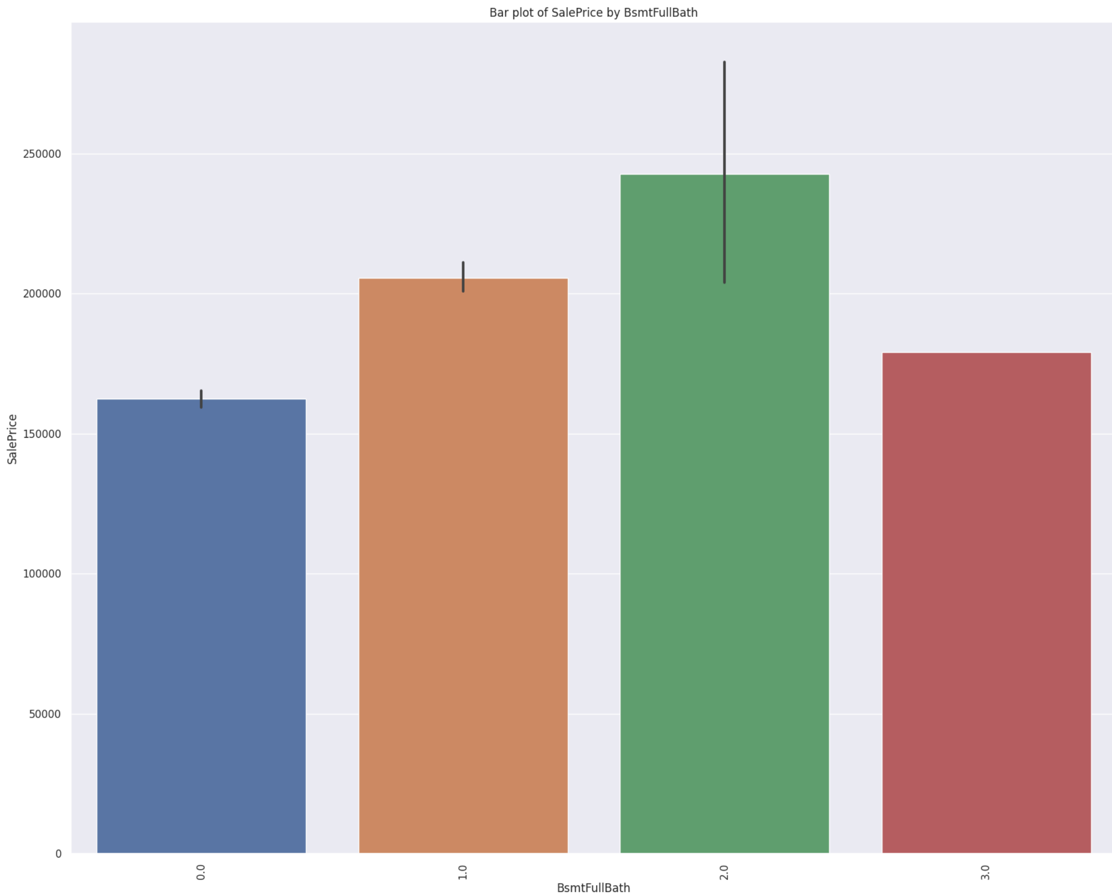
For all boxplots of categorical variables, we observe a trend where certain categories have a higher median sale price, indicative of their impact on the house pricing.

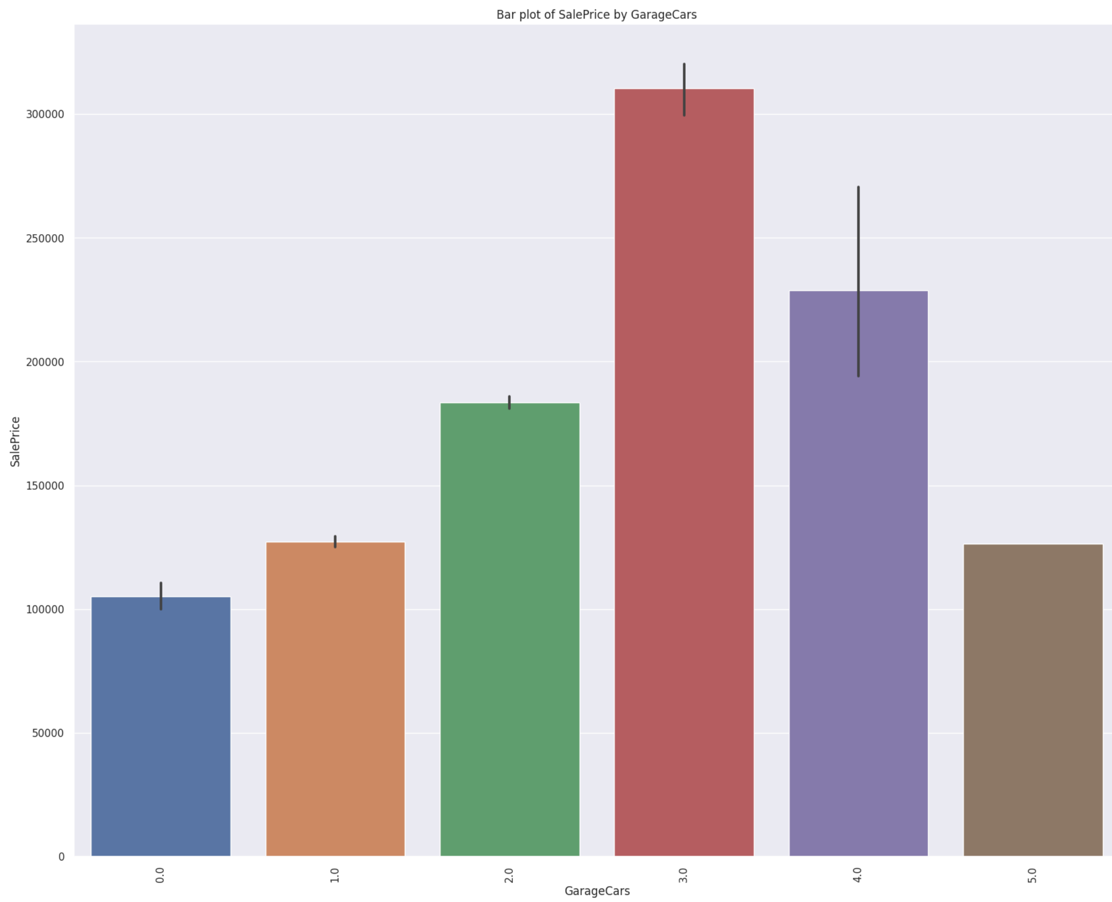
[33]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
categorical_vars = ['OverallQual', 'OverallCond', 'BsmtFullBath', 'GarageCars',
 'YearBuilt']

for var in categorical_vars:
    plt.figure(figsize=(20, 16))
    sns.barplot(x=var, y='SalePrice', data=df)
    plt.title(f'Bar plot of SalePrice by {var}')
    plt.ylabel('SalePrice')
    plt.xlabel(var)
    plt.xticks(rotation=90)
    plt.show()
```

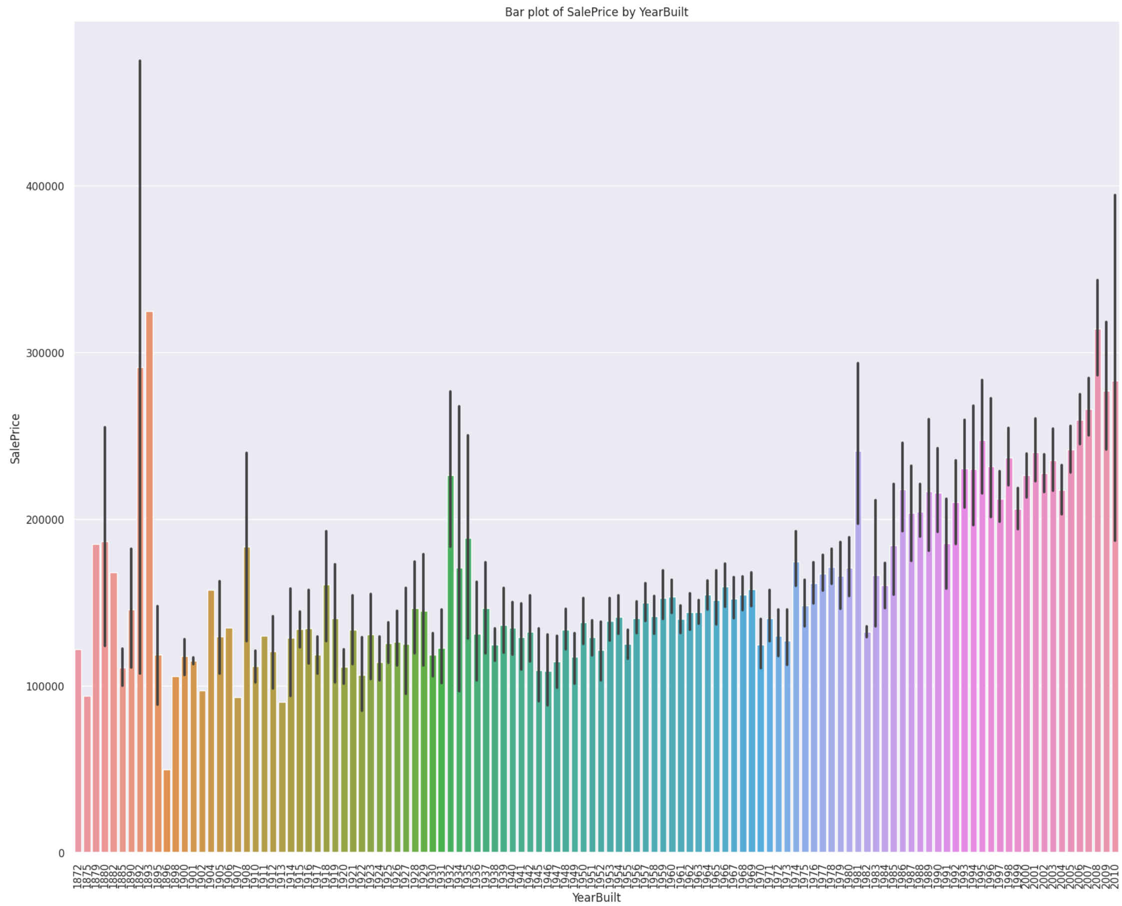



Bar plot of SalePrice by OverallCond
SalePrice
OverallCond
0
50000
100000
150000
200000
1
2
3
4
5
6
7
8
9

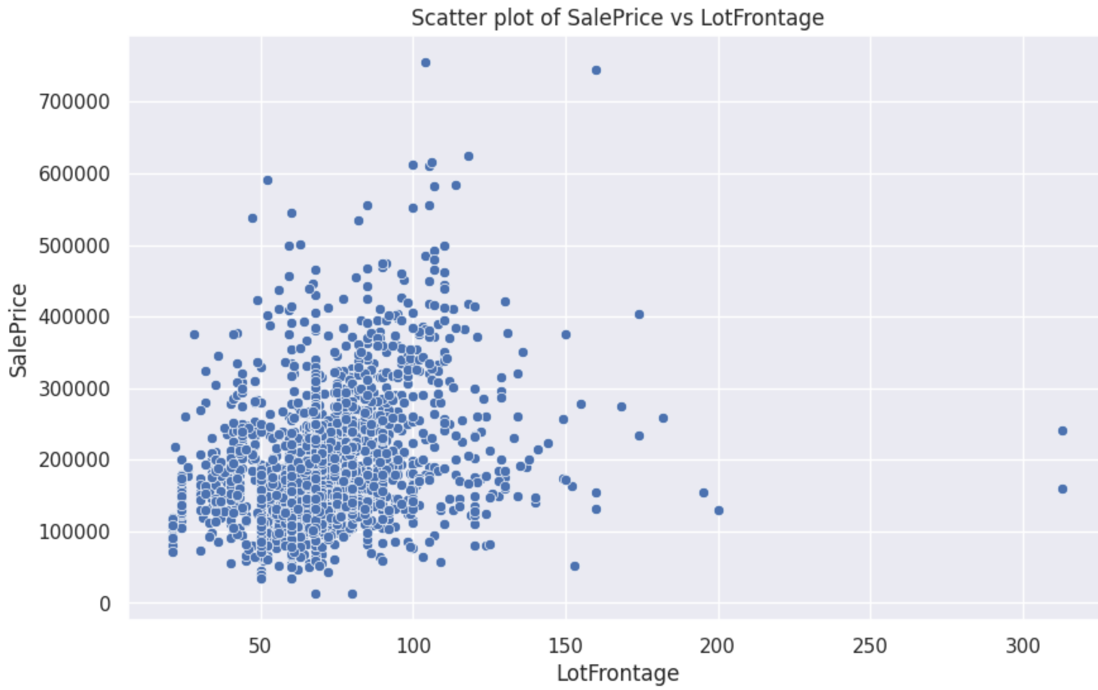

Bar plot of SalePrice by BsmtFullBath
250000
200000
150000
100000
50000
0
SalePrice
0.0
1.0
2.0
3.0
BsmtFullBath

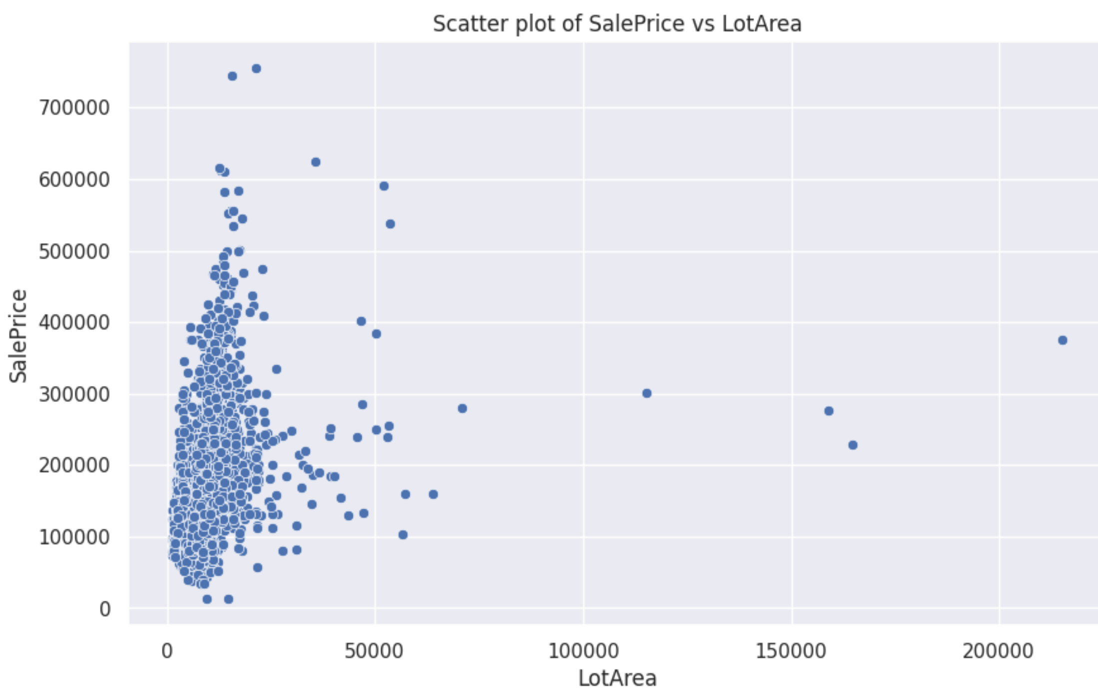


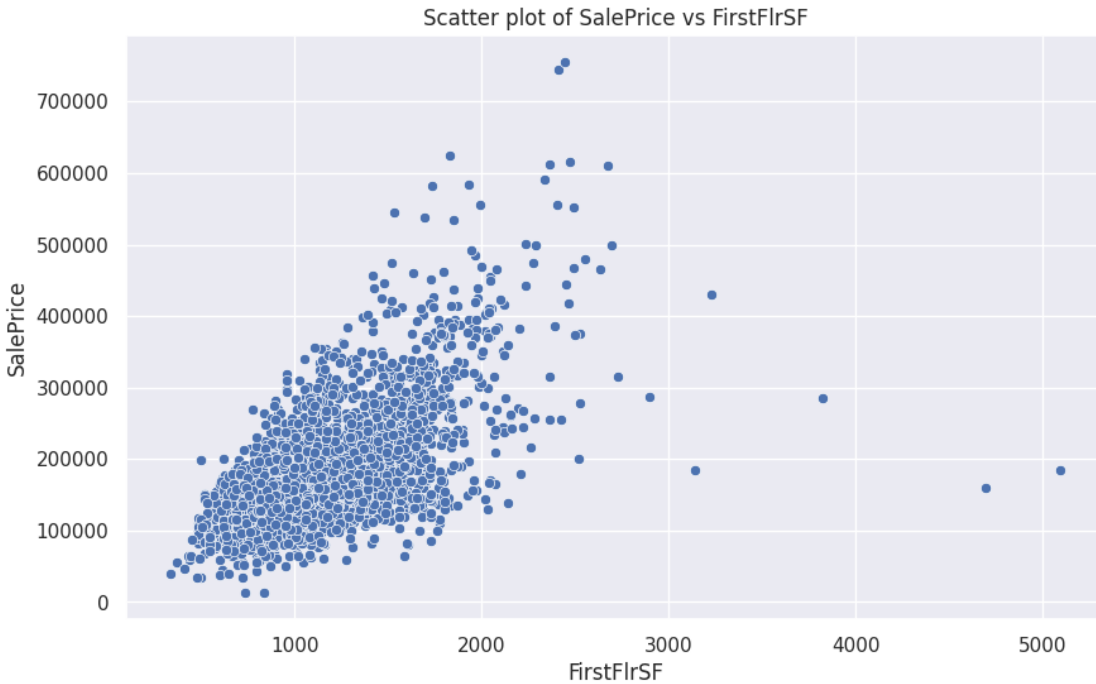
Bar plot of SalePrice by GarageCars
SalePrice
300000
250000
200000
150000
100000
50000
0
0.0
1.0
2.0
3.0
4.0
5.0
GarageCars

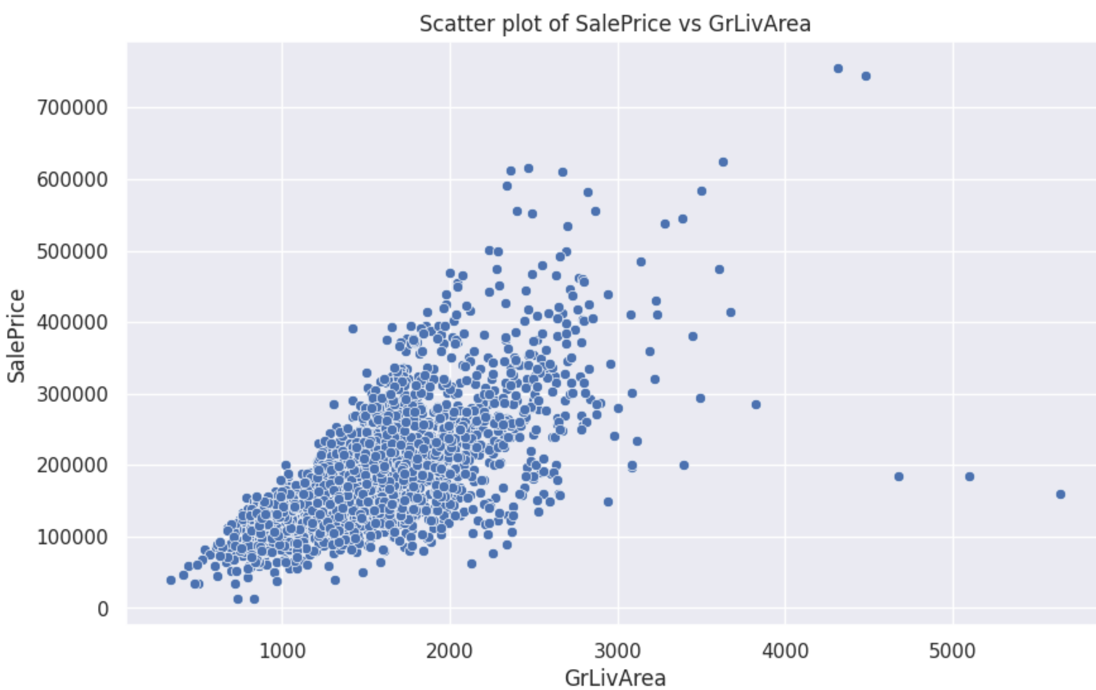


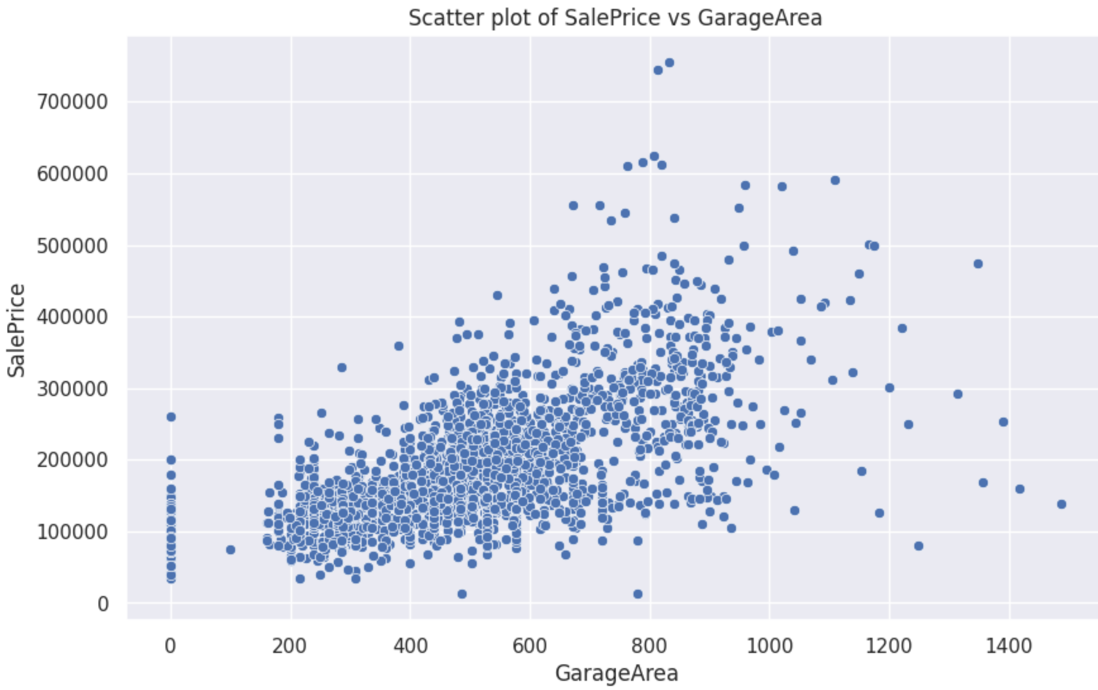
[35]:
```
continuous_vars = ['LotFrontage', 'LotArea', 'FirstFlrSF', 'GrLivArea',␣
 ↪'GarageArea']

for var in continuous_vars:
    plt.figure(figsize=(10, 6))
    sns.scatterplot(x=var, y='SalePrice', data=df)
    plt.title(f'Scatter plot of SalePrice vs {var}')
    plt.xlabel(var)
    plt.ylabel('SalePrice')
    plt.show()
```


Scatter plot of SalePrice vs LotFrontage
SalePrice
0
100000
200000
300000
400000
500000
600000
700000
LotFrontage
50
100
150
200
250
300


Scatter plot of SalePrice vs LotArea
SalePrice
0
100000
200000
300000
400000
500000
600000
700000
LotArea
0
50000
100000
150000
200000


Scatter plot of SalePrice vs FirstFlrSF
SalePrice
700000
600000
500000
400000
300000
200000
100000
0
1000
2000
3000
4000
5000
FirstFlrSF


Scatter plot of SalePrice vs GrLivArea
SalePrice
700000
600000
500000
400000
300000
200000
100000
0
1000
2000
3000
4000
5000
GrLivArea



5 An Initial Exporlatory Data Analysis for Modeling

The response variable in this problem is the SalePrice, which represents the sale price of houses. It is a continuous variable that we aim to predict using regression models.

In addition to the raw SalePrice variable, it is worth considering a transformation of the response variable. One common transformation is taking the logarithm of the SalePrice (log(SalePrice)). This transformation can help address issues of non-linearity and heteroscedasticity in the relationship between the predictor variables and the SalePrice. By taking the logarithm, we can potentially achieve a more linear and normally distributed relationship, which is desirable for regression modeling.

From the initial exploratory data analysis, three variables that showed a notable relationship with SalePrice and log(SalePrice) are OverallQual, GrLivArea, and GarageCars. These variables exhibited clear patterns and correlations with the SalePrice, indicating their potential importance in predicting house prices.

The EDA suggests some potential difficulties or concerns for the model building process. Firstly, there are some outliers present in the relationship between the predictor variables and the SalePrice. These outliers may impact the model's performance and could require further investigation and potential removal. Additionally, there are instances where the relationship between the predictor variables and the SalePrice is non-linear, indicating the need to consider nonlinear modeling techniques or transformations.

Furthermore, the EDA suggests that there may be a need to consider transformations in the predictor variables during the model building process. For example, variables such as GrLivArea and

GarageCars exhibited skewed distributions, which might benefit from transformation to improve linearity and distributional assumptions.

Overall, the EDA provides valuable insights into the relationships between the selected variables and the SalePrice. It highlights potential challenges, such as outliers and non-linear relationships, which should be addressed during the model building process. Additionally, it suggests considering transformations for both the response variable and the predictor variables to improve the model's performance and meet the assumptions of linear regression.

6 Summary/ Conclusions

The exploratory data analysis (EDA) conducted on the Ames housing dataset has provided valuable insights into the variables and their relationships, offering guidance for the model building process. The EDA highlighted potential difficulties and concerns that should be addressed during the model building process. Outliers were observed in some variables, indicating the need for robust modeling techniques or potential removal of extreme values. Additionally, non-linear relationships were identified between certain predictor variables and the SalePrice, suggesting the need to consider non-linear modeling approaches or transformations.

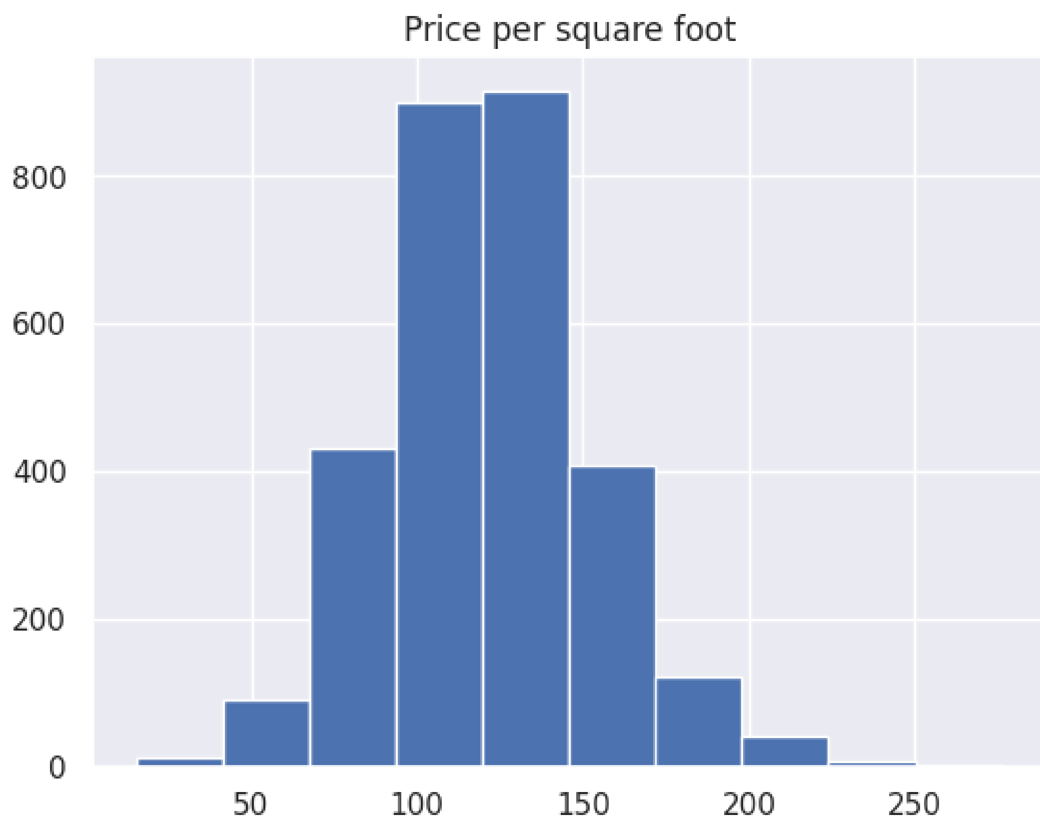
The EDA also suggested the consideration of transformations in the predictor variables. Skewed distributions and non-linear patterns were observed in some variables, indicating the potential benefits of transformations to improve linearity and meet the assumptions of linear regression.

Overall, the EDA has provided a solid foundation for the model building process. It has shed light on potential challenges, such as outliers and non-linear relationships, that should be carefully addressed. The findings suggest that the incorporation of transformations in predictor variables and the use of appropriate modeling techniques will be crucial in building accurate and reliable regression models for predicting house prices in Ames, Iowa.

7 Extra: Converted EDA Starter Code

(Was used to provide direction during assignment.)

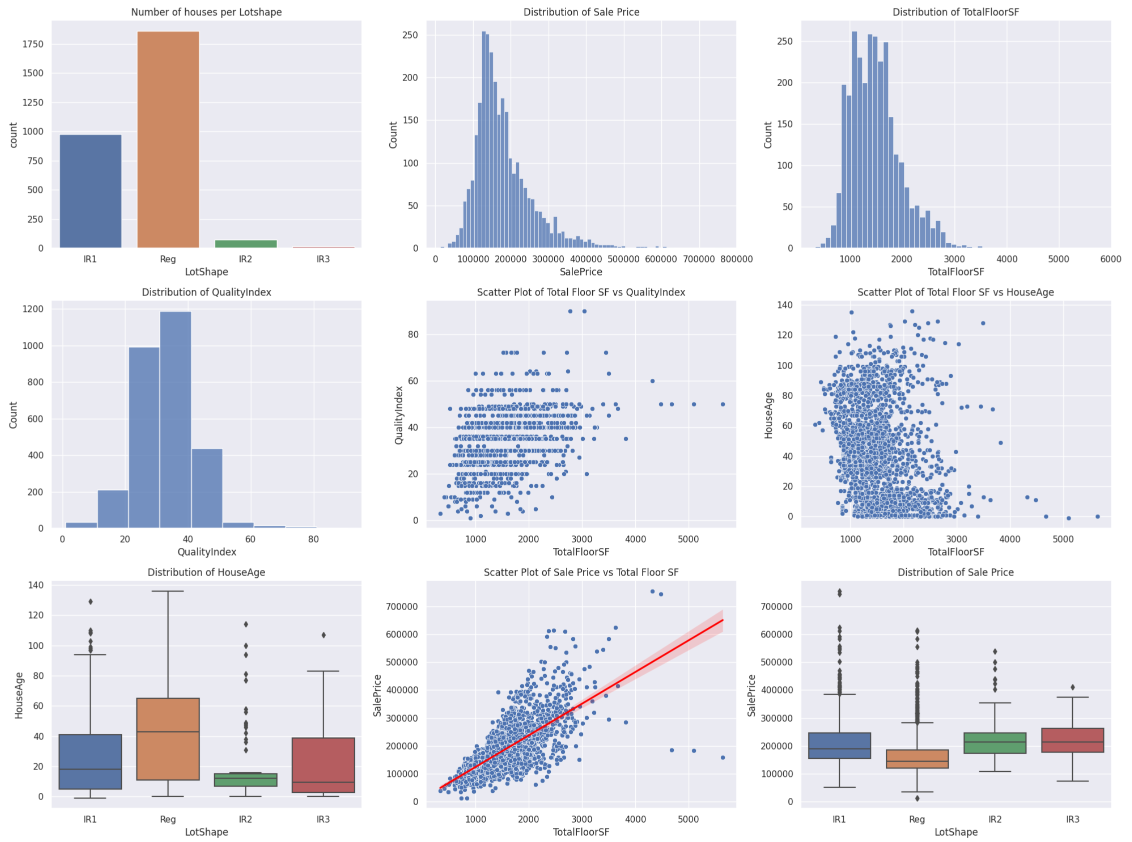
[22]:
```
plt.hist(df['price_sqft'])
plt.title('Price per square foot')
plt.show()
```



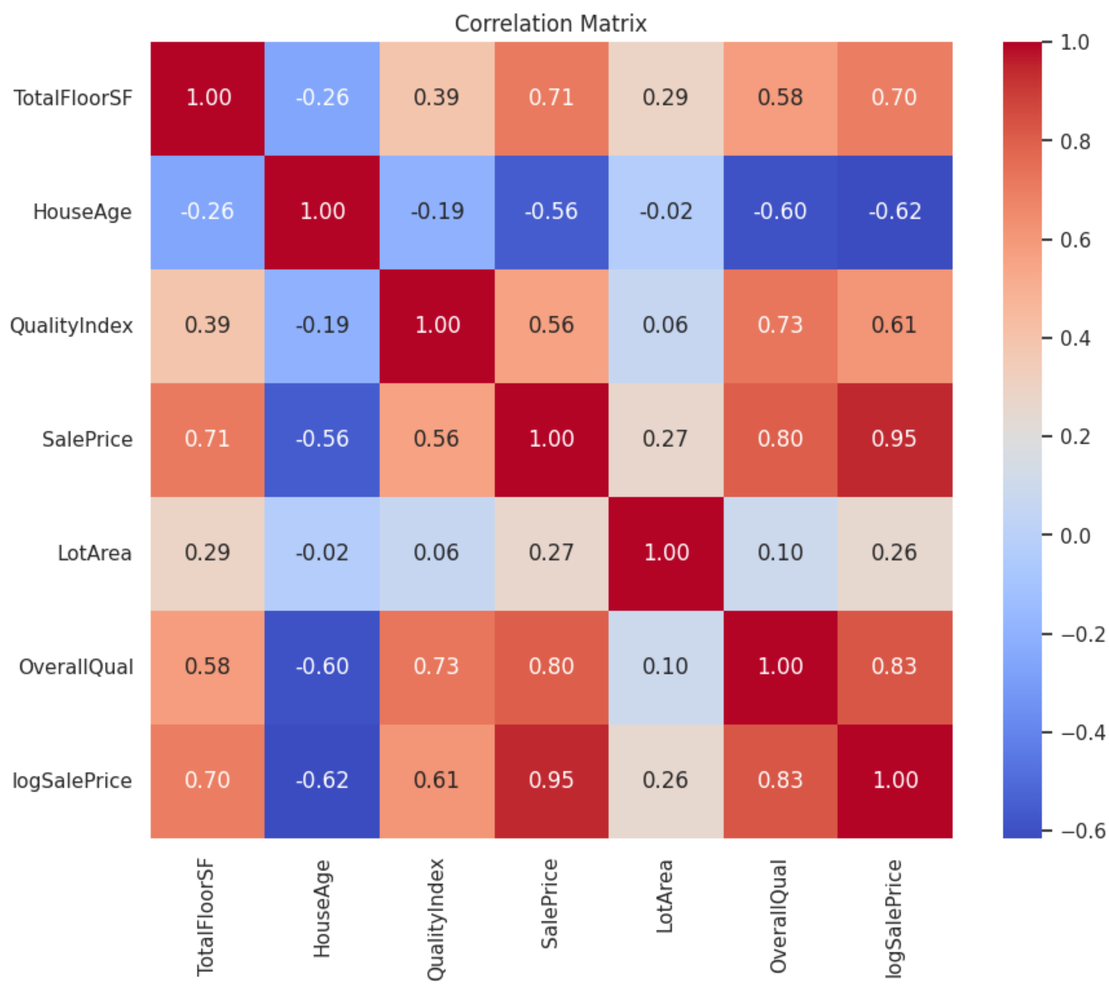
[23]:
```
subdat = df[["TotalFloorSF", "HouseAge", "QualityIndex", "price_sqft",
                 "SalePrice", "LotArea", "BsmtFinSF1", "Neighborhood",
                 "HouseStyle", "LotShape", "OverallQual", "logSalePrice",
                 "TotalBsmtSF", "HouseStyle"]]
subdatnum = df[["TotalFloorSF", "HouseAge", "QualityIndex",
                    "SalePrice", "LotArea", "OverallQual", "logSalePrice"]]
```

[24]:
```
import matplotlib.pyplot as plt
import seaborn as sns
f, axes = plt.subplots(3, 3, figsize=(20, 15))

# Univariate EDA
sns.countplot(data=subdat, x='LotShape', ax=axes[0, 0]).set_title('Number of␣
 ↪houses per Lotshape')

sns.histplot(data=subdat, x='SalePrice', binwidth=10000, ax=axes[0, 1]).
 ↪set_title('Distribution of Sale Price')

sns.histplot(data=subdat, x='TotalFloorSF', binwidth=100, ax=axes[0, 2]).
 ↪set_title('Distribution of TotalFloorSF')
```

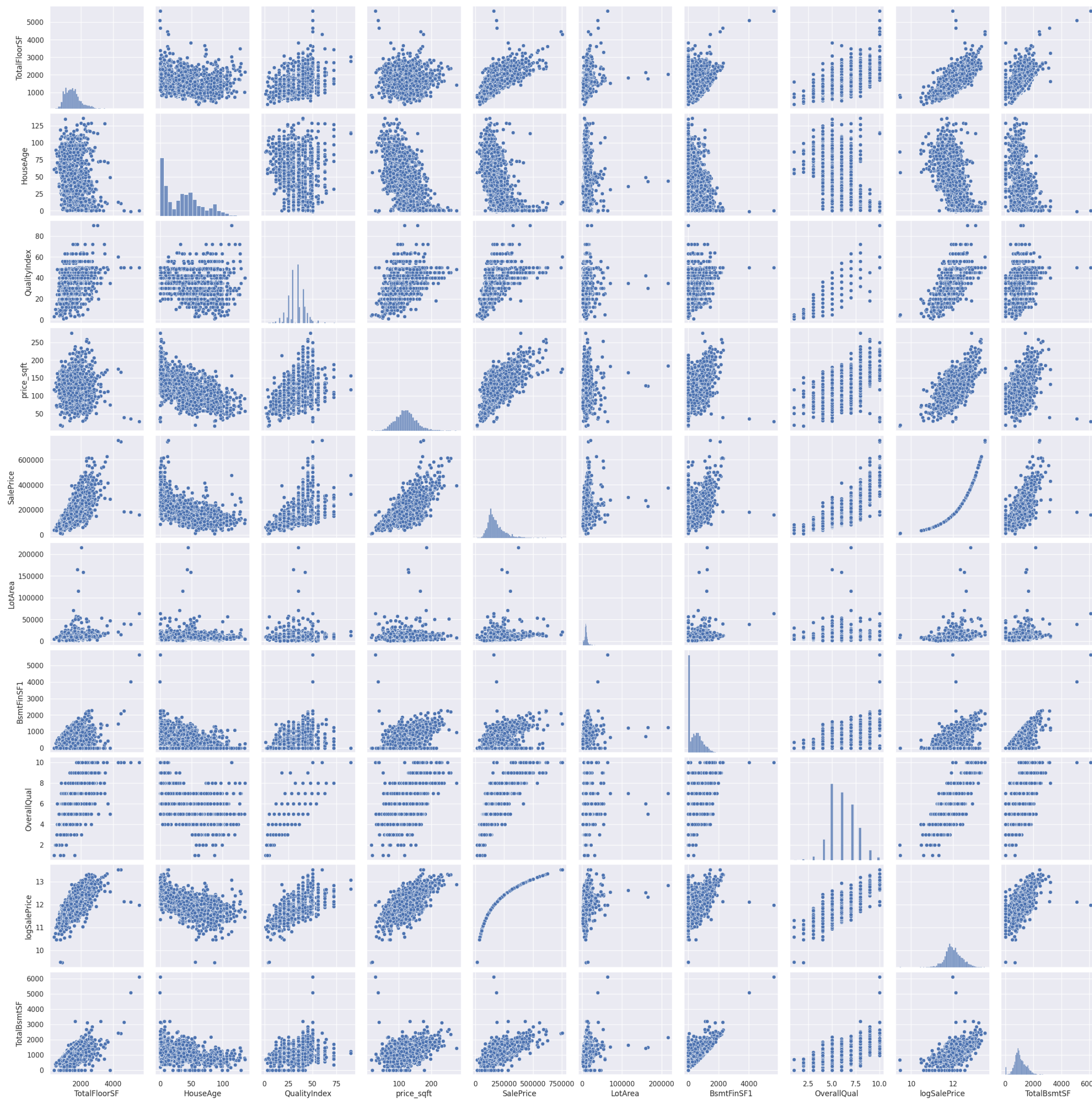
```
sns.histplot(data=subdat, x='QualityIndex', binwidth=10, ax=axes[1, 0]).
 ↪set_title('Distribution of QualityIndex')

# Bivariate EDA
sns.scatterplot(data=subdat, x='TotalFloorSF', y='QualityIndex', ax=axes[1, 1]).
 ↪set_title('Scatter Plot of Total Floor SF vs QualityIndex')

sns.scatterplot(data=subdat, x='TotalFloorSF', y='HouseAge', ax=axes[1, 2]).
 ↪set_title('Scatter Plot of Total Floor SF vs HouseAge')

sns.boxplot(data=subdat, x='LotShape', y='HouseAge', ax=axes[2, 0]).
 ↪set_title('Distribution of HouseAge')

# Model focussed EDA
sns.scatterplot(data=subdat, x='TotalFloorSF', y='SalePrice', ax=axes[2, 1]).
 ↪set_title('Scatter Plot of Sale Price vs Total Floor SF')
sns.regplot(data=subdat, x='TotalFloorSF', y='SalePrice', scatter=False,␣
 ↪color='red', ax=axes[2, 1])

sns.boxplot(data=subdat, x='LotShape', y='SalePrice', ax=axes[2, 2]).
 ↪set_title('Distribution of Sale Price')

plt.tight_layout()
plt.show()
```



[25]:
```
# Correlation plot
corr = subdatnum.corr()
plt.figure(figsize=(10,8))
sns.heatmap(corr, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



[26]:
```
subdat_numeric = subdat.select_dtypes(include=np.number)
sns.pairplot(subdat_numeric)
plt.show()
```


TotalFloorSF
HouseAge
QualityIndex
price_sqft
SalePrice
LotArea
BsmtFinSF1
OverallQual
logSalePrice
TotalBsmtSF