Predict house sale prices in Ames, Iowa using machine learning techniques

August 27, 2023

1 SALE PRICE PREDICTION

1.1 Section 1 : Dataset description

The dataset has 82 columns that include 23 nominal, 23 ordinal, 14 discrete, and 20 continuous variables (and 2 additional case identifiers).

Data Soucre

[161]:
```
### important Librairies
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

[162]:
```
#Import dataset
df_description_clos = pd.read_csv("df_description_clos.csv")

df_description_clos.head()
```

[162]:
```
    Columns name                               Columns description
0          Order                          Observation number (Discrete)
1            PID  Parcel identification number can be used with …
2   MS SubClass   Identifies the type of dwelling involved in th…
3      MS Zoning  Identifies the general zoning classification o…
4  Lot Frontage  Linear feet of street connected to property (C…
```

[163]:
```
#Transform the column name into an index
df_description_clos = df_description_clos.set_index("Columns name")
df_description_clos.head()
```

[163]:
```
                                          Columns description
Columns name
Order                              Observation number (Discrete)
PID             Parcel identification number can be used with …
MS SubClass     Identifies the type of dwelling involved in th…
MS Zoning       Identifies the general zoning classification o…
```

```
Lot Frontage  Linear feet of street connected to property (C…
```

Reference indexing and selecting data

[164]:
```python
#Set a function to display the feature description
def info_func(col_name):
    return df_description_clos.loc[col_name]["Columns description"]
```

It is very important to have a function that displays the feature description, because when processing the data, we will need it to see the feature descriptions as we go.

[165]:
```python
# Display the description of the some features
info_func(["Order", "Lot Area", "Street", "Alley", "Pool Area", "Pool QC",␣
 ↪"Fence"])
```

[165]:
```
Columns name
Order                          Observation number (Discrete)
Lot Area               Lot size in square feet (Continuous)
Street             Type of road access to property (Nominal)
Alley             Type of alley access to property (Nominal)
Pool Area             Pool area in square feet (Continuous)
Pool QC                              Pool quality (Ordinal)
Fence                               Fence quality (Ordinal)
Name: Columns description, dtype: object
```

[166]:
```python
#Import dataset
df =pd.read_csv("AmesHousing.txt", delimiter="\t")
```

[167]:
```python
#Show the shape and types of the dataset
print(df.shape)
print(len(str(df.shape))*'-')
print(df.dtypes.value_counts())
```

```
(2930, 82)
----------
object     43
int64      28
float64    11
dtype: int64
```

[168]:
```python
#Display the first 5 rows of the dataset
df.head(3)
```

[168]:
```
   Order        PID  MS SubClass MS Zoning  Lot Frontage  Lot Area Street  \
0      1  526301100           20        RL         141.0     31770   Pave
1      2  526350040           20        RH          80.0     11622   Pave
2      3  526351010           20        RL          81.0     14267   Pave

  Alley Lot Shape Land Contour  … Pool Area Pool QC  Fence Misc Feature  \
```

```
0    NaN        IR1          Lvl  …          0     NaN    NaN            NaN
1    NaN        Reg          Lvl  …          0     NaN  MnPrv            NaN
2    NaN        IR1          Lvl  …          0     NaN    NaN           Gar2

  Misc Val Mo Sold Yr Sold Sale Type  Sale Condition  SalePrice
0        0       5    2010        WD          Normal     215000
1        0       6    2010        WD          Normal     105000
2    12500       6    2010        WD          Normal     172000

[3 rows x 82 columns]
```

We have a dataset with 2930 observations and 82 characteristics including 43 objects, 28 int64 and 11 float64

[169]:
```
#The last 5 rows of the datset
df.tail(3)
```

[169]:
```
      Order        PID  MS SubClass MS Zoning  Lot Frontage  Lot Area Street  \
2927   2928  923400125           85        RL          62.0     10441   Pave
2928   2929  924100070           20        RL          77.0     10010   Pave
2929   2930  924151050           60        RL          74.0      9627   Pave

     Alley Lot Shape Land Contour  … Pool Area Pool QC  Fence Misc Feature  \
2927   NaN       Reg          Lvl  …         0     NaN  MnPrv         Shed
2928   NaN       Reg          Lvl  …         0     NaN    NaN          NaN
2929   NaN       Reg          Lvl  …         0     NaN    NaN          NaN

     Misc Val Mo Sold Yr Sold Sale Type  Sale Condition  SalePrice
2927      700       7    2006        WD          Normal     132000
2928        0       4    2006        WD          Normal     170000
2929        0      11    2006        WD          Normal     188000

[3 rows x 82 columns]
```

1.2 Section 2 : Exploratory Data Analysis (EDA)

Exploratory data analysis (EDA) is a very important step in data analysis. It allows us with visualizations and statistical analysis (univariate, bivariate and multivariate) to understand the data with which we work and to better understand their relationships. So let's start exploring our target variable and how other features influence it.

1.2.1 *Duplicated*

In machine learning models, duplicates can lead to biases and inaccuracies. This is why it is very important to manage duplicates in the dataset.

Reference duplicated

Reference duplicated

[170]:
```
df[df.duplicated(keep = False)]
```

[170]:
```
Empty DataFrame
Columns: [Order, PID, MS SubClass, MS Zoning, Lot Frontage, Lot Area, Street,
Alley, Lot Shape, Land Contour, Utilities, Lot Config, Land Slope, Neighborhood,
Condition 1, Condition 2, Bldg Type, House Style, Overall Qual, Overall Cond,
Year Built, Year Remod/Add, Roof Style, Roof Matl, Exterior 1st, Exterior 2nd,
Mas Vnr Type, Mas Vnr Area, Exter Qual, Exter Cond, Foundation, Bsmt Qual, Bsmt
Cond, Bsmt Exposure, BsmtFin Type 1, BsmtFin SF 1, BsmtFin Type 2, BsmtFin SF 2,
Bsmt Unf SF, Total Bsmt SF, Heating, Heating QC, Central Air, Electrical, 1st
Flr SF, 2nd Flr SF, Low Qual Fin SF, Gr Liv Area, Bsmt Full Bath, Bsmt Half
Bath, Full Bath, Half Bath, Bedroom AbvGr, Kitchen AbvGr, Kitchen Qual, TotRms
AbvGrd, Functional, Fireplaces, Fireplace Qu, Garage Type, Garage Yr Blt, Garage
Finish, Garage Cars, Garage Area, Garage Qual, Garage Cond, Paved Drive, Wood
Deck SF, Open Porch SF, Enclosed Porch, 3Ssn Porch, Screen Porch, Pool Area,
Pool QC, Fence, Misc Feature, Misc Val, Mo Sold, Yr Sold, Sale Type, Sale
Condition, SalePrice]
Index: []

[0 rows x 82 columns]
```

There are no duplicate values

1.2.2 *Data info*

Reference data info

[171]:
```
#Show dataset information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2930 entries, 0 to 2929
Data columns (total 82 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Order            2930 non-null   int64
 1   PID              2930 non-null   int64
 2   MS SubClass      2930 non-null   int64
 3   MS Zoning        2930 non-null   object
 4   Lot Frontage     2440 non-null   float64
 5   Lot Area         2930 non-null   int64
 6   Street           2930 non-null   object
 7   Alley            198 non-null    object
 8   Lot Shape        2930 non-null   object
 9   Land Contour     2930 non-null   object
 10  Utilities        2930 non-null   object
 11  Lot Config       2930 non-null   object
 12  Land Slope       2930 non-null   object
 13  Neighborhood     2930 non-null   object
```

14	Condition 1	2930 non-null	object
15	Condition 2	2930 non-null	object
16	Bldg Type	2930 non-null	object
17	House Style	2930 non-null	object
18	Overall Qual	2930 non-null	int64
19	Overall Cond	2930 non-null	int64
20	Year Built	2930 non-null	int64
21	Year Remod/Add	2930 non-null	int64
22	Roof Style	2930 non-null	object
23	Roof Matl	2930 non-null	object
24	Exterior 1st	2930 non-null	object
25	Exterior 2nd	2930 non-null	object
26	Mas Vnr Type	2907 non-null	object
27	Mas Vnr Area	2907 non-null	float64
28	Exter Qual	2930 non-null	object
29	Exter Cond	2930 non-null	object
30	Foundation	2930 non-null	object
31	Bsmt Qual	2850 non-null	object
32	Bsmt Cond	2850 non-null	object
33	Bsmt Exposure	2847 non-null	object
34	BsmtFin Type 1	2850 non-null	object
35	BsmtFin SF 1	2929 non-null	float64
36	BsmtFin Type 2	2849 non-null	object
37	BsmtFin SF 2	2929 non-null	float64
38	Bsmt Unf SF	2929 non-null	float64
39	Total Bsmt SF	2929 non-null	float64
40	Heating	2930 non-null	object
41	Heating QC	2930 non-null	object
42	Central Air	2930 non-null	object
43	Electrical	2929 non-null	object
44	1st Flr SF	2930 non-null	int64
45	2nd Flr SF	2930 non-null	int64
46	Low Qual Fin SF	2930 non-null	int64
47	Gr Liv Area	2930 non-null	int64
48	Bsmt Full Bath	2928 non-null	float64
49	Bsmt Half Bath	2928 non-null	float64
50	Full Bath	2930 non-null	int64
51	Half Bath	2930 non-null	int64
52	Bedroom AbvGr	2930 non-null	int64
53	Kitchen AbvGr	2930 non-null	int64
54	Kitchen Qual	2930 non-null	object
55	TotRms AbvGrd	2930 non-null	int64
56	Functional	2930 non-null	object
57	Fireplaces	2930 non-null	int64
58	Fireplace Qu	1508 non-null	object
59	Garage Type	2773 non-null	object
60	Garage Yr Blt	2771 non-null	float64
61	Garage Finish	2771 non-null	object

```
 62  Garage Cars      2929 non-null   float64
 63  Garage Area      2929 non-null   float64
 64  Garage Qual      2771 non-null   object
 65  Garage Cond      2771 non-null   object
 66  Paved Drive      2930 non-null   object
 67  Wood Deck SF     2930 non-null   int64
 68  Open Porch SF    2930 non-null   int64
 69  Enclosed Porch   2930 non-null   int64
 70  3Ssn Porch       2930 non-null   int64
 71  Screen Porch     2930 non-null   int64
 72  Pool Area        2930 non-null   int64
 73  Pool QC          13 non-null     object
 74  Fence            572 non-null    object
 75  Misc Feature     106 non-null    object
 76  Misc Val         2930 non-null   int64
 77  Mo Sold          2930 non-null   int64
 78  Yr Sold          2930 non-null   int64
 79  Sale Type        2930 non-null   object
 80  Sale Condition   2930 non-null   object
 81  SalePrice        2930 non-null   int64
dtypes: float64(11), int64(28), object(43)
memory usage: 1.8+ MB
```

We can see the type (int64, float64 and object), the shape (2930 entries and 82 columns) of the dataset and the missing values for each feature.

1.2.3 *Let's delete columns that are not useful for Maching learning*

These are not useful columns for machine learning. Indeed, these two variables will not have much effect on price prediction, because they are only property identifiers.

[172]:
```
#Let's look at the columns to delete
info_func(["Order", "PID"])
```

[172]:
```
Columns name
Order                             Observation number (Discrete)
PID      Parcel identification number can be used with …
Name: Columns description, dtype: object
```

[173]:
```
def fun_sup(df):
    df =df.drop(["Order", "PID"], axis=1)
    return df
```

[174]:
```
# Delete columns that are not useful for Machine Learning
data =fun_sup(df)
```

[175]:
```
# Display the shape of the dataset
data.shape
```

[175]: (2930, 80)

1.2.4 *Descriptive statistics*

Reference descriptive statistic

[176]:
```
#Descriptive statistics
data.describe().T
```

[176]:

	count	mean	std	min	25%	\
MS SubClass	2930.0	57.387372	42.638025	20.0	20.00	
Lot Frontage	2440.0	69.224590	23.365335	21.0	58.00	
Lot Area	2930.0	10147.921843	7880.017759	1300.0	7440.25	
Overall Qual	2930.0	6.094881	1.411026	1.0	5.00	
Overall Cond	2930.0	5.563140	1.111537	1.0	5.00	
Year Built	2930.0	1971.356314	30.245361	1872.0	1954.00	
Year Remod/Add	2930.0	1984.266553	20.860286	1950.0	1965.00	
Mas Vnr Area	2907.0	101.896801	179.112611	0.0	0.00	
BsmtFin SF 1	2929.0	442.629566	455.590839	0.0	0.00	
BsmtFin SF 2	2929.0	49.722431	169.168476	0.0	0.00	
Bsmt Unf SF	2929.0	559.262547	439.494153	0.0	219.00	
Total Bsmt SF	2929.0	1051.614544	440.615067	0.0	793.00	
1st Flr SF	2930.0	1159.557679	391.890885	334.0	876.25	
2nd Flr SF	2930.0	335.455973	428.395715	0.0	0.00	
Low Qual Fin SF	2930.0	4.676792	46.310510	0.0	0.00	
Gr Liv Area	2930.0	1499.690444	505.508887	334.0	1126.00	
Bsmt Full Bath	2928.0	0.431352	0.524820	0.0	0.00	
Bsmt Half Bath	2928.0	0.061134	0.245254	0.0	0.00	
Full Bath	2930.0	1.566553	0.552941	0.0	1.00	
Half Bath	2930.0	0.379522	0.502629	0.0	0.00	
Bedroom AbvGr	2930.0	2.854266	0.827731	0.0	2.00	
Kitchen AbvGr	2930.0	1.044369	0.214076	0.0	1.00	
TotRms AbvGrd	2930.0	6.443003	1.572964	2.0	5.00	
Fireplaces	2930.0	0.599317	0.647921	0.0	0.00	
Garage Yr Blt	2771.0	1978.132443	25.528411	1895.0	1960.00	
Garage Cars	2929.0	1.766815	0.760566	0.0	1.00	
Garage Area	2929.0	472.819734	215.046549	0.0	320.00	
Wood Deck SF	2930.0	93.751877	126.361562	0.0	0.00	
Open Porch SF	2930.0	47.533447	67.483400	0.0	0.00	
Enclosed Porch	2930.0	23.011604	64.139059	0.0	0.00	
3Ssn Porch	2930.0	2.592491	25.141331	0.0	0.00	
Screen Porch	2930.0	16.002048	56.087370	0.0	0.00	
Pool Area	2930.0	2.243345	35.597181	0.0	0.00	
Misc Val	2930.0	50.635154	566.344288	0.0	0.00	
Mo Sold	2930.0	6.216041	2.714492	1.0	4.00	
Yr Sold	2930.0	2007.790444	1.316613	2006.0	2007.00	
SalePrice	2930.0	180796.060068	79886.692357	12789.0	129500.00	

	50%	75%	max
MS SubClass	50.0	70.00	190.0
Lot Frontage	68.0	80.00	313.0
Lot Area	9436.5	11555.25	215245.0
Overall Qual	6.0	7.00	10.0
Overall Cond	5.0	6.00	9.0
Year Built	1973.0	2001.00	2010.0
Year Remod/Add	1993.0	2004.00	2010.0
Mas Vnr Area	0.0	164.00	1600.0
BsmtFin SF 1	370.0	734.00	5644.0
BsmtFin SF 2	0.0	0.00	1526.0
Bsmt Unf SF	466.0	802.00	2336.0
Total Bsmt SF	990.0	1302.00	6110.0
1st Flr SF	1084.0	1384.00	5095.0
2nd Flr SF	0.0	703.75	2065.0
Low Qual Fin SF	0.0	0.00	1064.0
Gr Liv Area	1442.0	1742.75	5642.0
Bsmt Full Bath	0.0	1.00	3.0
Bsmt Half Bath	0.0	0.00	2.0
Full Bath	2.0	2.00	4.0
Half Bath	0.0	1.00	2.0
Bedroom AbvGr	3.0	3.00	8.0
Kitchen AbvGr	1.0	1.00	3.0
TotRms AbvGrd	6.0	7.00	15.0
Fireplaces	1.0	1.00	4.0
Garage Yr Blt	1979.0	2002.00	2207.0
Garage Cars	2.0	2.00	5.0
Garage Area	480.0	576.00	1488.0
Wood Deck SF	0.0	168.00	1424.0
Open Porch SF	27.0	70.00	742.0
Enclosed Porch	0.0	0.00	1012.0
3Ssn Porch	0.0	0.00	508.0
Screen Porch	0.0	0.00	576.0
Pool Area	0.0	0.00	800.0
Misc Val	0.0	0.00	17000.0
Mo Sold	6.0	8.00	12.0
Yr Sold	2008.0	2009.00	2010.0
SalePrice	160000.0	213500.00	755000.0

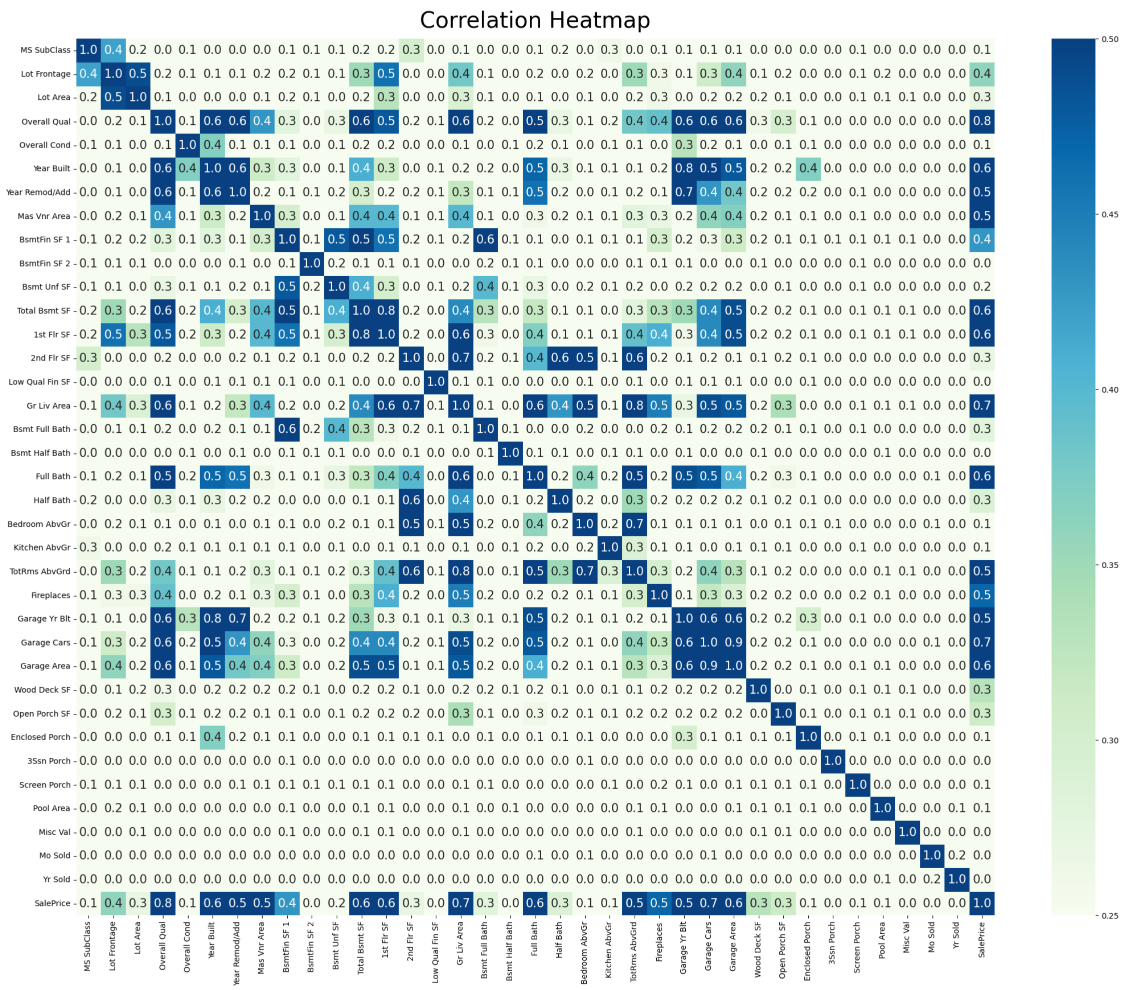
These descriptive statistics show a large variation between our data (for example by looking at the average, the min and the max of the variable of X, we notice this variation very quickly). This data will therefore have to be scaled before being used in a machine learning model. It should be noted that many models are subject to variations in the data. For example, linear regression and k-nearest neighbors algorithms are sensitive to variations in the data.

1.2.5 *Correlation*

Correlation explains how one or more variables relate to each other. These variables can be features of input data that were used to predict our target variable.

Reference correlation

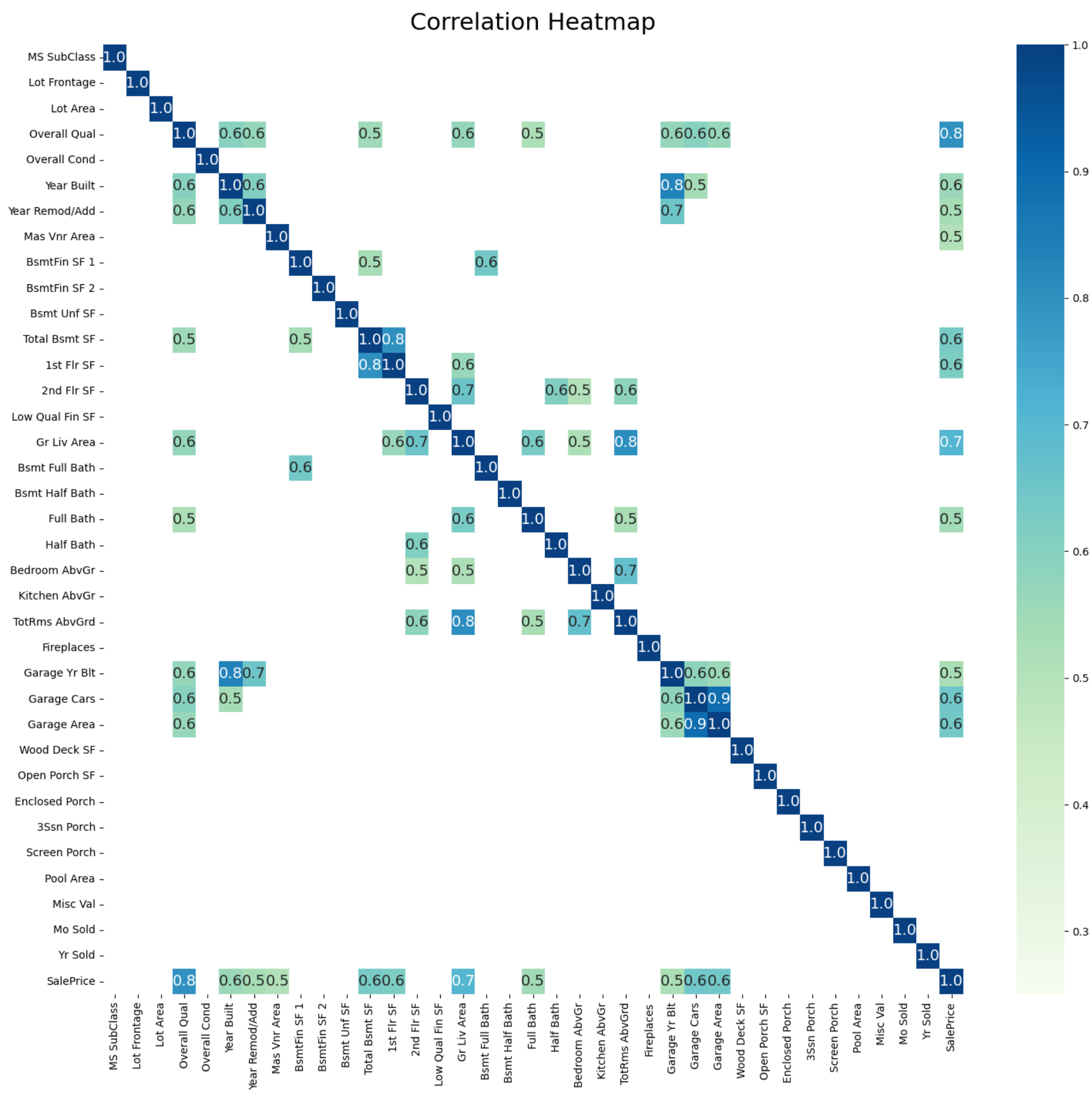
[177]:
```python
corr =round(data.select_dtypes(include =np.number).corr().abs(), 2)
plt.figure(figsize=(26, 20))

heatmap = sns.heatmap(corr, fmt='.1f', cmap='GnBu', vmin=0.25, vmax=0.5, annot
 ↪= True, annot_kws={"size": 15})

heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':28}, pad=14);
```



We see some high and low correlation between characteristics. We will later use a feature selection technique to select the most relevant features.

[429]:
```
# Create correlation matrix from train data excluding `SalePrice`
corr = data.select_dtypes(include =np.number).corr().abs()

# Select correlations greater than 0.5
high_corr = corr[abs(corr_mat) >= 0.5]

# Plot correlation heatmap
plt.figure(figsize=(18, 16))
heatmap = sns.heatmap(high_corr_mat, annot=True, fmt='.1f', cmap='GnBu', vmin=0.
 ↪25, vmax=1, annot_kws={"size": 14})
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':22}, pad=14)
```

[429]:
```
Text(0.5, 1.0, 'Correlation Heatmap')
```



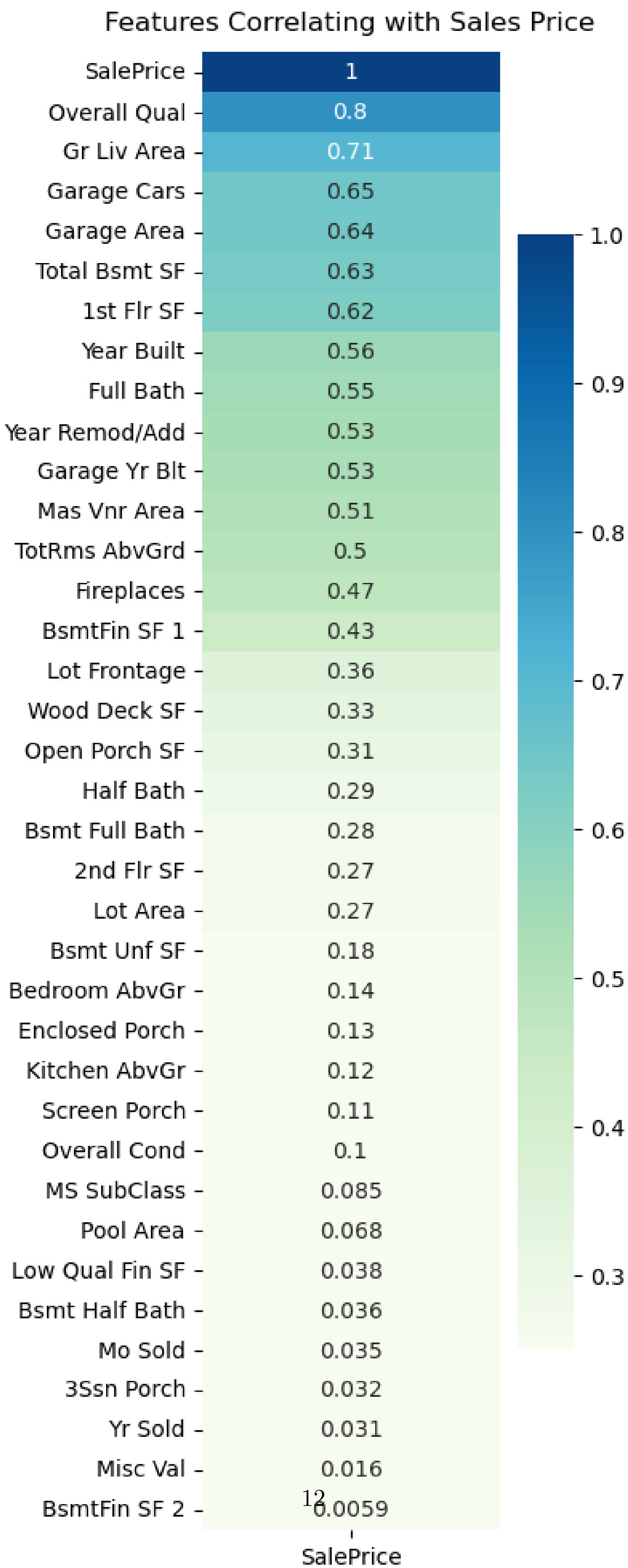
There is multicollinearity in our data. The features below are highly correlated :

- Garage Cars and Garage Area
- Garage Yr Blt and Year Built
- 1st Flr SF and Total Bsmt SF
- Gr Liv Area and TotRms AbvGrd

Multicolliniarity negatively impacts prediction models because it duplicates the same information and thus increases the standard errors of estimators. Therefore, it is useful to keep only one feature from each pair of highly correlated features. So, in each pair, we remove the feature that is weakly correlated with the sale price.

Correlation of all features with the sale price

[179]:
```
#Correlation level with features
plt.figure(figsize=(3, 12))
corr_SalePrice =data.select_dtypes(include =np.number).corr()[["SalePrice"]].
 ↪abs().sort_values(by = "SalePrice", ascending = False)

heatmap =sns.heatmap(corr_SalePrice, cmap='GnBu', vmin=0.25, vmax=1, annot =␣
 ↪True)

heatmap.set_title('Features Correlating with Sales Price', fontdict={'fontsize':
 ↪12}, pad=10);
```


Features Correlating with Sales Price
SalePrice 1
Overall Qual 0.8
Gr Liv Area 0.71
Garage Cars 0.65
Garage Area 0.64
Total Bsmt SF 0.63
1st Flr SF 0.62
Year Built 0.56
Full Bath 0.55
Year Remod/Add 0.53
Garage Yr Blt 0.53
Mas Vnr Area 0.51
TotRms AbvGrd 0.5
Fireplaces 0.47
BsmtFin SF 1 0.43
Lot Frontage 0.36
Wood Deck SF 0.33
Open Porch SF 0.31
Half Bath 0.29
Bsmt Full Bath 0.28
2nd Flr SF 0.27
Lot Area 0.27
Bsmt Unf SF 0.18
Bedroom AbvGr 0.14
Enclosed Porch 0.13
Kitchen AbvGr 0.12
Screen Porch 0.11
Overall Cond 0.1
MS SubClass 0.085
Pool Area 0.068
Low Qual Fin SF 0.038
Bsmt Half Bath 0.036
Mo Sold 0.035
3Ssn Porch 0.032
Yr Sold 0.031
Misc Val 0.016
BsmtFin SF 2 0.0059
SalePrice
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3

After examining the correlation of the features with the sale price, we remove the characteristics weakly correlated with the selling price such as: Garage Cars, Garage Yr Built, 1st Flr SF and TotRms AbvGrd.

[180]:
```
info_func(["Garage Cars", "Garage Yr Blt", "1st Flr SF", "TotRms AbvGrd"])
```

[180]:
```
Columns name
Garage Cars              Size of garage in car capacity (Discrete)
Garage Yr Blt                     Year garage was built (Discrete)
1st Flr SF                    First Floor square feet (Continuous)
TotRms AbvGrd    Total rooms above grade (does not include bath…
Name: Columns description, dtype: object
```

[181]:
```
data[["Garage Cars", "Garage Yr Blt", "1st Flr SF", "TotRms AbvGrd"]]
```

[181]:
```
      Garage Cars  Garage Yr Blt  1st Flr SF  TotRms AbvGrd
0             2.0         1960.0        1656              7
1             1.0         1961.0         896              5
2             1.0         1958.0        1329              6
3             2.0         1968.0        2110              8
4             2.0         1997.0         928              6
…            …             …            …            …
2925          2.0         1984.0        1003              6
2926          2.0         1983.0         902              5
2927          0.0            NaN         970              6
2928          2.0         1975.0        1389              6
2929          3.0         1993.0         996              9

[2930 rows x 4 columns]
```

[182]:
```
for col in ["Garage Cars", "Garage Yr Blt", "1st Flr SF", "TotRms AbvGrd"]:
    data = data.drop(col, axis = 1)
```

[183]:
```
data.shape
```

[183]:
```
(2930, 76)
```

https://ecampusontario.pressbooks.pub/introstats/chapter/13-3-standard-error-of-the-estimate/

https://chriskhanhtran.github.io/minimal-portfolio/projects/ames-house-price.html

1.2.6 *Data visualization*

Let's draw the graphs of some features of the dataset

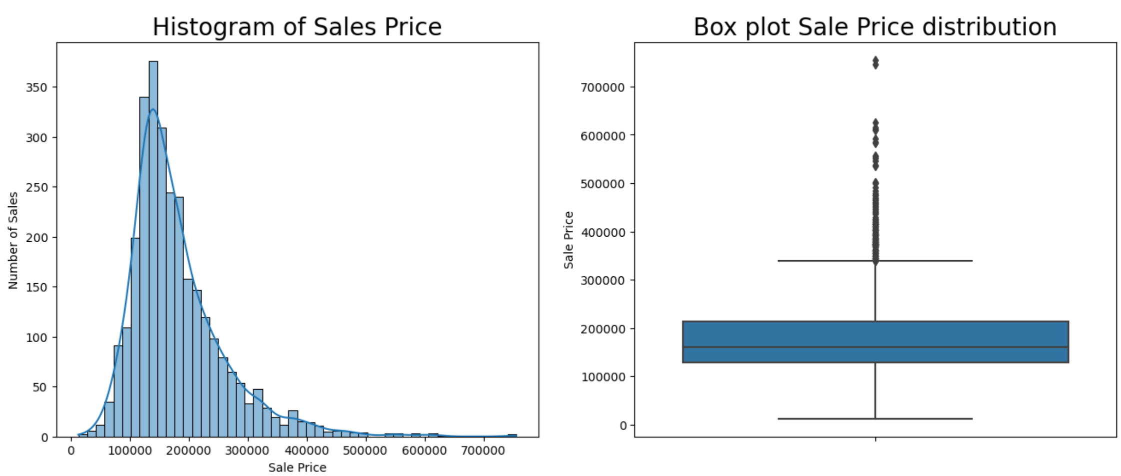
Sale Price

[184]:
```
info_func("SalePrice")
```

[184]: 'Sale price (Continuous)'

[185]:
```
# Price distribution
fig, ax = plt.subplots(figsize=(16,6), ncols=2)
sns.histplot(x = "SalePrice", bins=50, kde=True, ax=ax[0], data =data)
ax[0].set_title('Histogram of Sales Price', fontsize=20)
ax[0].set_xlabel('Sale Price')
ax[0].set_ylabel('Number of Sales')

#Price box plot
sns.boxplot(y = "SalePrice", ax=ax[1], data =data)
ax[1].set_title("Box plot Sale Price distribution", fontsize=20)
ax[1].set_ylabel('Sale Price')
```

[185]: Text(0, 0.5, 'Sale Price')



Note that most homes cost between 100,000 dollars and 300,000 dollars. On the other hand, the most expensive houses cost up to more than 700,000 dollars, few houses are sold as soon as the price exceeds a value of 400,000 dollars. This variable has many outliers as seen in the histogram and boxplot above.

There is a high correlation between price and the following variables : "Overall Qual", "Gr Liv Area", "Garage Cars", "Garage Area"... Let's plot a scatter plot of these features with the sale price.

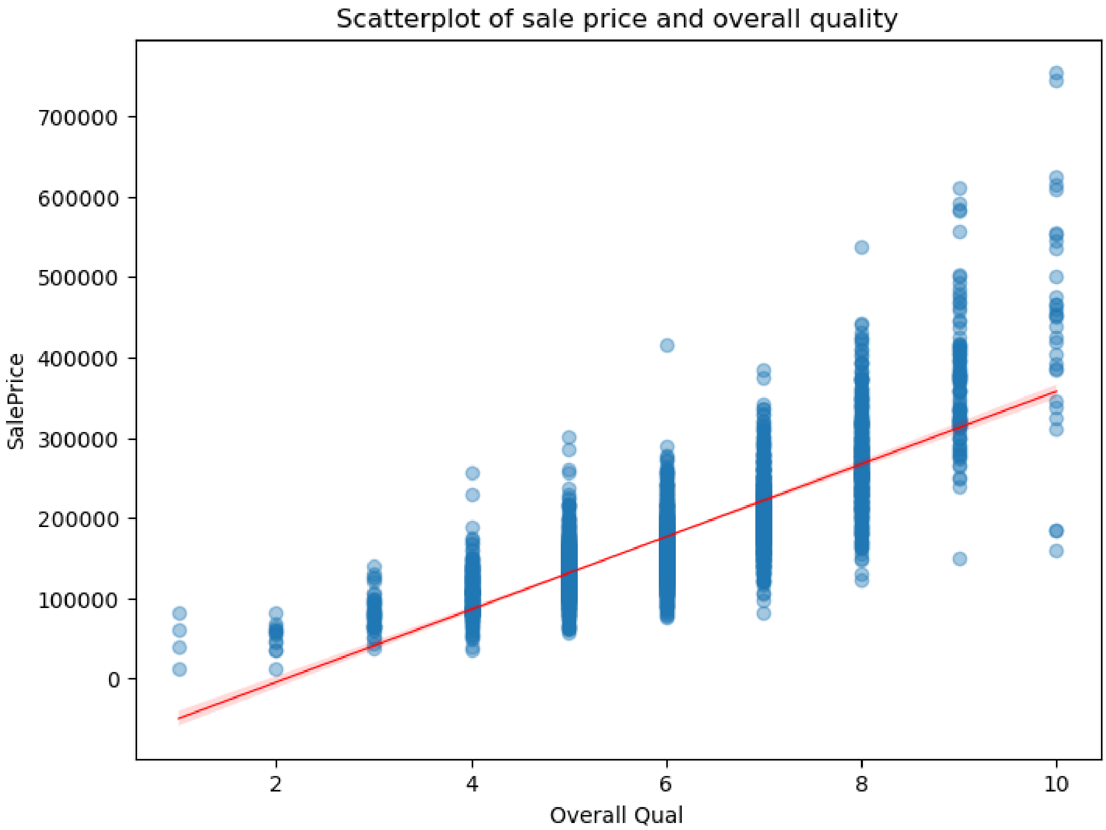
Feature scatterplot with high correlation to price

[186]:
```
info_func("Overall Qual")
```

[186]: 'Rates the overall material and finish of the house (Ordinal)'

[187]:
```
#Scatterplot of sale price and overall quality
plt.figure(figsize=(8, 6))
sns.regplot(x ="Overall Qual", y="SalePrice", scatter_kws={'alpha': 0.4},
                line_kws={'color': 'red','linewidth':0.8} ,data = data)
plt.title("Scatterplot of sale price and overall quality")
```

[187]: Text(0.5, 1.0, 'Scatterplot of sale price and overall quality')



[188]:
```
info_func("Gr Liv Area")
```

[188]: 'Above grade (ground) living area square feet (Continuous)'

[189]:
```
plt.figure(figsize=(8, 6))
sns.regplot(x ="Gr Liv Area", y="SalePrice", scatter_kws={'alpha': 0.4},
                line_kws={'color': 'red','linewidth':0.8}, data =data)
plt.title("Scatterplot of sale price and Gr Liv Area")
```

[189]: Text(0.5, 1.0, 'Scatterplot of sale price and Gr Liv Area')

[190]:
```
info_func("Garage Cars")
```

[190]: 'Size of garage in car capacity (Discrete)'

[430]:
```
plt.figure(figsize=(8, 6))
sns.regplot(x ="Total Bsmt SF", y="SalePrice", scatter_kws={'alpha': 0.4},
                line_kws={'color': 'red','linewidth':0.8} ,data = data)
plt.title("Scatterplot of sale price and Total Bsmt SF")
```

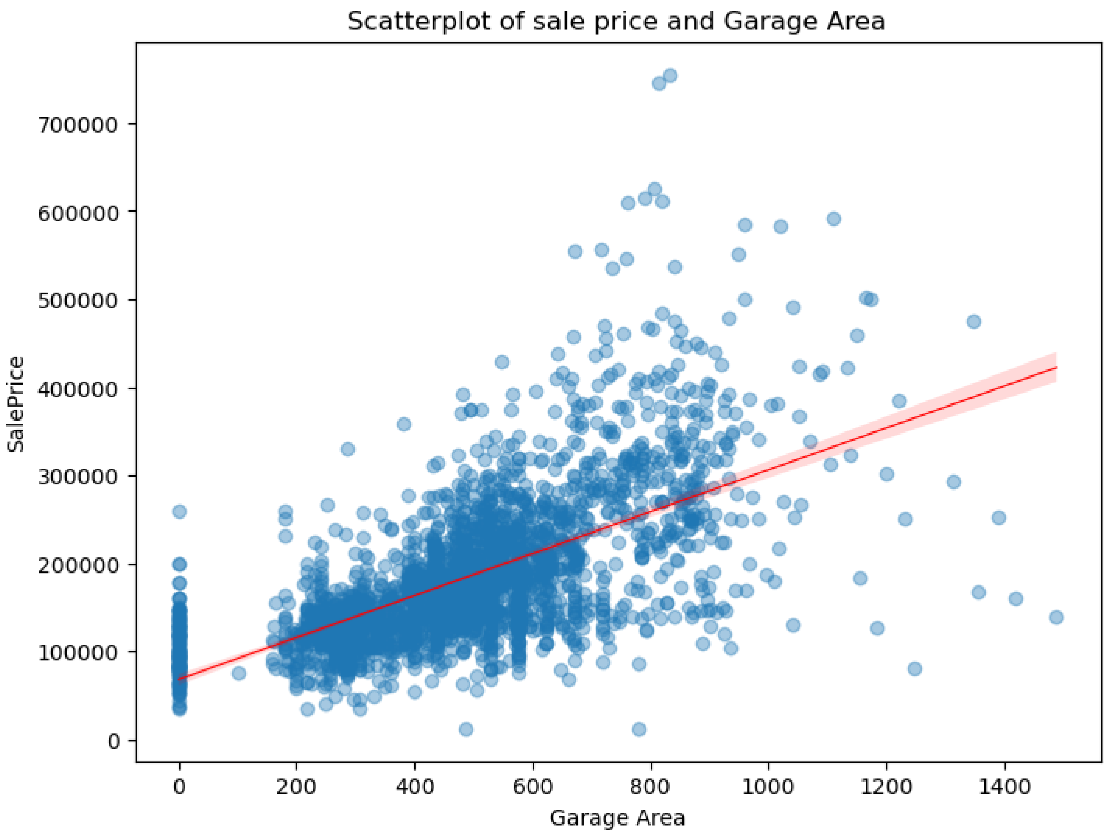
[430]: Text(0.5, 1.0, 'Scatterplot of sale price and Total Bsmt SF')

[192]:
```
info_func("Garage Area")
```

[192]: 'Size of garage in square feet (Continuous)'

[193]:
```
plt.figure(figsize=(8, 6))
sns.regplot(x ="Garage Area", y="SalePrice", scatter_kws={'alpha': 0.4},
                line_kws={'color': 'red','linewidth':0.8} ,data = data)
plt.title("Scatterplot of sale price and Garage Area")
```

[193]: Text(0.5, 1.0, 'Scatterplot of sale price and Garage Area')



All scatter plots show many outliers. Let's adjust the data and use the logarithm to transform the selling price to reduce the discrepancy between the data. We visualize the distribution of certain characteristics with histograms, bar charts and box plots

Zoning type

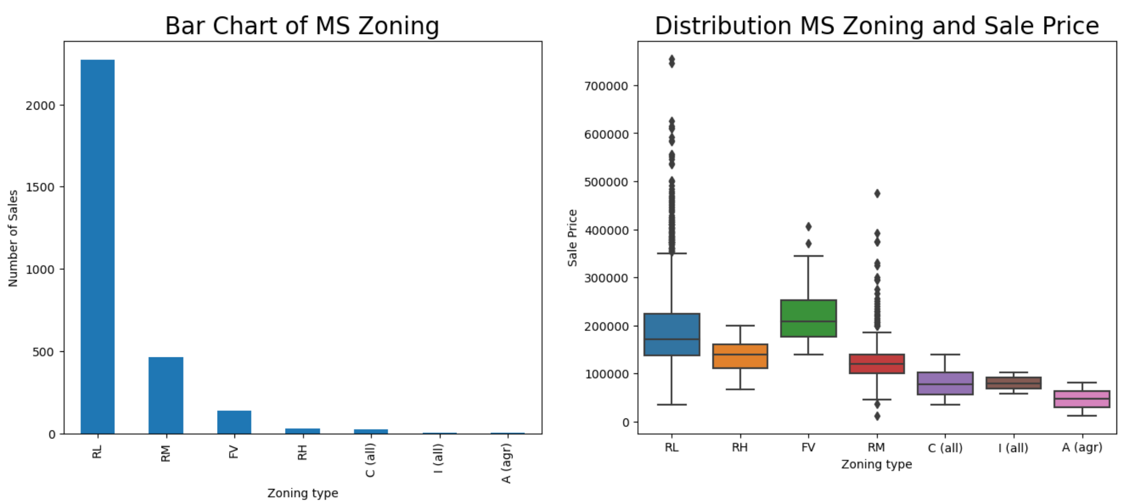
[194]:
```
print(info_func("MS Zoning"))

print( "\nUnique values :", data['MS Zoning'].unique())

print( "\nCount values :\n", data['MS Zoning'].value_counts(dropna = False))
```

```
Identifies the general zoning classification of the sale. (Nominal)

Unique values : ['RL' 'RH' 'FV' 'RM' 'C (all)' 'I (all)' 'A (agr)']

Count values :
 RL          2273
RM           462
FV           139
RH            27
C (all)       25
```

```
I (all)       2
A (agr)       2
Name: MS Zoning, dtype: int64
```

[195]:
```python
fig, ax = plt.subplots(figsize=(16,6), ncols=2)
data['MS Zoning'].value_counts().plot(kind='bar', ax=ax[0])
ax[0].set_title("Bar Chart of MS Zoning", fontsize=20)
ax[0].set_xlabel('Zoning type')
ax[0].set_ylabel('Number of Sales')

sns.boxplot(x = "MS Zoning", y = "SalePrice", ax=ax[1], data = data)
ax[1].set_title("Distribution MS Zoning and Sale Price", fontsize=20)
ax[1].set_xlabel('Zoning type')
ax[1].set_ylabel('Sale Price')
```

[195]:
```
Text(0, 0.5, 'Sale Price')
```



MS Zoning (Nominal) : Identifies the general zoning classification of the sale.

```
A    Agriculture
C    Commercial
FV   Floating Village Residential
I    Industrial
RH   Residential High Density
RL   Residential Low Density
RP   Residential Low Density Park
RM   Residential Medium Density
```

Note that the low-density residential area includes the best-selling and most expensive homes with extreme prices of up to over $700,000. However, there are many outliers in this category.

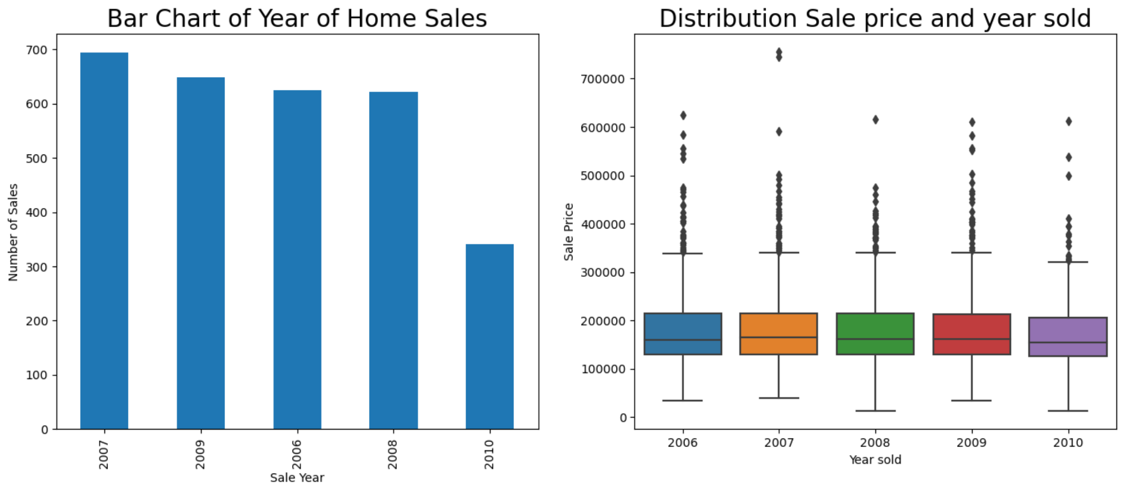
Year Sold

[196]:
```
print(info_func("Yr Sold"))

print( "\nUnique values :", data['Yr Sold'].unique())

print( "\nCount values :\n", data['Yr Sold'].value_counts(dropna = False))
```

```
Year Sold (YYYY) (Discrete)

Unique values : [2010 2009 2008 2007 2006]

Count values :
 2007    694
2009    648
2006    625
2008    622
2010    341
Name: Yr Sold, dtype: int64
```

[197]:
```
fig, ax = plt.subplots(figsize=(16,6), ncols=2)
data['Yr Sold'].value_counts().plot(kind='bar', ax=ax[0])
ax[0].set_title("Bar Chart of Year of Home Sales", fontsize=20)
ax[0].set_xlabel('Sale Year')
ax[0].set_ylabel('Number of Sales')

sns.boxplot(x = 'Yr Sold', y = "SalePrice", data = data)
ax[1].set_title("Distribution Sale price and year sold" , fontsize=20)
ax[1].set_xlabel('Year sold')
ax[1].set_ylabel('Sale Price')
```

[197]:
```
Text(0, 0.5, 'Sale Price')
```



We see that the years from 2006 to 2009 have almost the same number of sales, this corresponds to the real estate boom before the financial crisis of 2008. In 2010, sales fell by almost half.

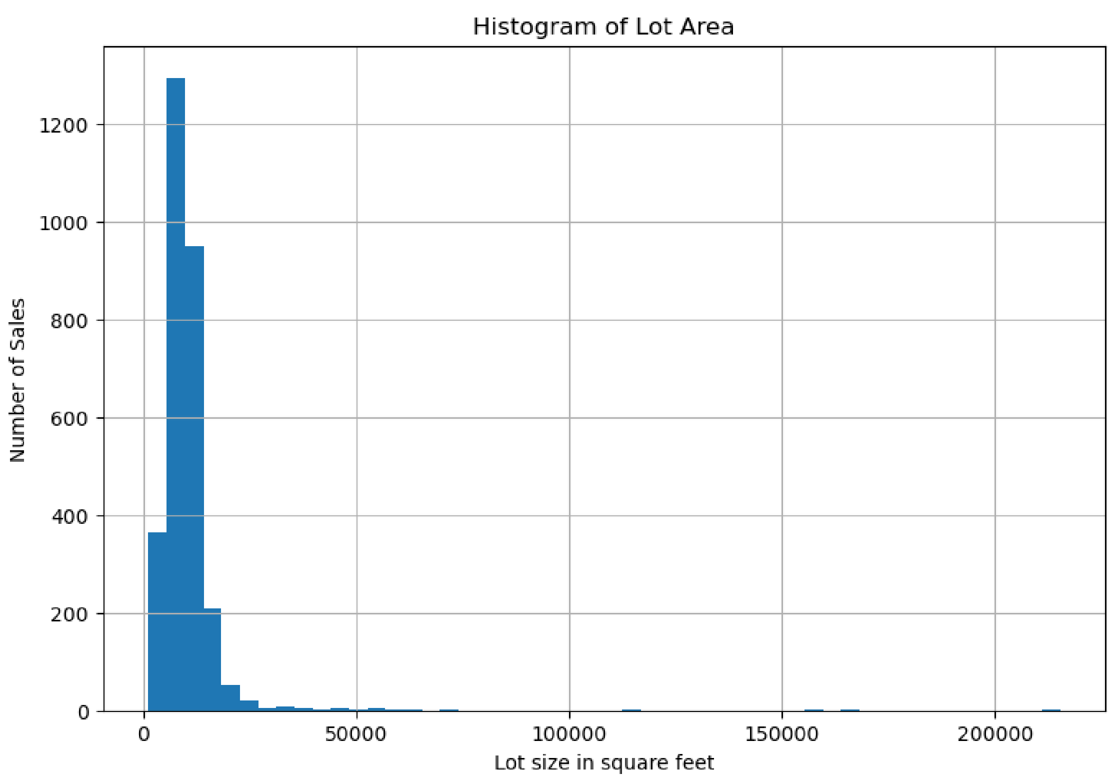
Lot Area

[198]:
```
info_func("Lot Area")
```

[198]: 'Lot size in square feet (Continuous)'

[199]:
```
plt.figure(figsize=(9,6))
data['Lot Area'].hist(bins=50)
plt.title("Histogram of Lot Area")
plt.xlabel('Lot size in square feet')
plt.ylabel('Number of Sales')
```

[199]: Text(0, 0.5, 'Number of Sales')

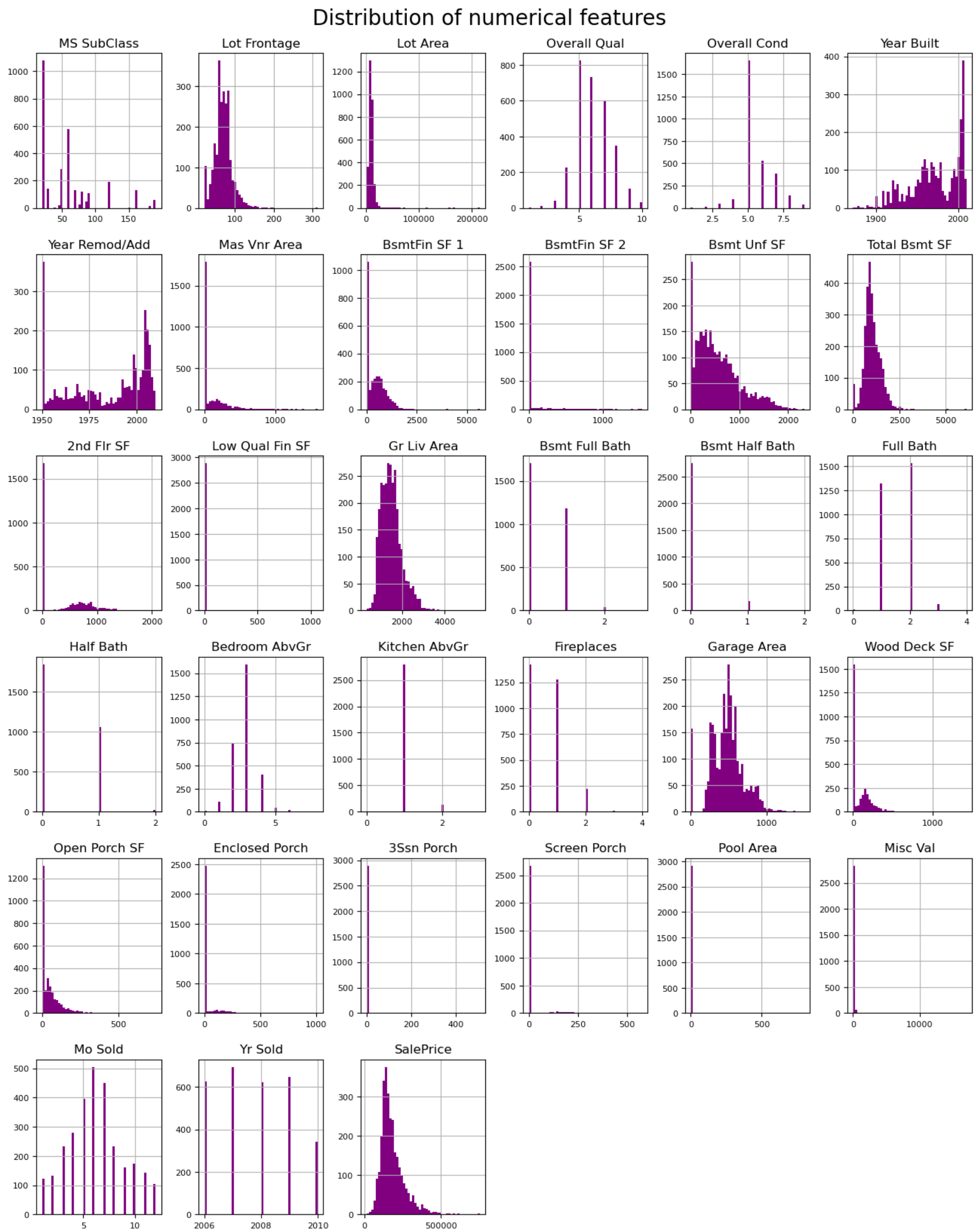


Most of the homes sold are about less than 20,000 square feet. We also note that the values are very asymmetrical on the right.

Histograms of all numerical features

[200]:
```
data.hist(figsize=(16, 20), color="purple", bins=50, xlabelsize=8,␣
    ↪ylabelsize=8);
plt.suptitle("Distribution of numerical features", x = 0.5, y = 0.91, fontsize␣
    ↪= 20)
```

[200]:
```
Text(0.5, 0.91, 'Distribution of numerical features')
```



Just looking at the table of descriptive statistics and graphs of some variables above, we find that most of our numerical variables are asymmetries and that numerical and categorical variables have many outliers. In this case, as we said, let's need to scale data before implementing some machine learning models, especially linear models that require data normality.

1.3 Section 3 : *Preprocessing Data*

1.3.1 *Missing values*

In data science, whenever we have to deal with missing values in a data set, we have to ask ourselves, "Do we know what the absence of these values means?" If the answer is no, we should ask our source. It is very important to know where the missing values come from, because after processing them will be faster and much better.

Here we deal with missing data taking into account the documentation you will find here

For more information, see the references

[201]:
```
#Missing values
def missing_values(df):
    missing_values = df.isnull().sum().sort_values()
    missing_values = missing_values[missing_values!=0]
    pourcentages_missing_values = round(missing_values[(missing_values>0)]/
 ↪len(df)*100, 3)
    data_missing_values = pd.DataFrame({"number of missing values" :␣
 ↪missing_values,"pourcentage of missing values":pourcentages_missing_values})
    return data_missing_values
```

[202]:
```
# Number and percentage of missing values
data_missing_values = missing_values(data)
data_missing_values
```

[202]:

	number of missing values	pourcentage of missing values
Total Bsmt SF	1	0.034
Electrical	1	0.034
BsmtFin SF 1	1	0.034
BsmtFin SF 2	1	0.034
Bsmt Unf SF	1	0.034
Garage Area	1	0.034
Bsmt Full Bath	2	0.068
Bsmt Half Bath	2	0.068
Mas Vnr Area	23	0.785
Mas Vnr Type	23	0.785
Bsmt Qual	80	2.730
Bsmt Cond	80	2.730
BsmtFin Type 1	80	2.730
BsmtFin Type 2	81	2.765

```
Bsmt Exposure                     83              2.833
Garage Type                      157              5.358
Garage Cond                      159              5.427
Garage Finish                    159              5.427
Garage Qual                      159              5.427
Lot Frontage                     490             16.724
Fireplace Qu                    1422             48.532
Fence                           2358             80.478
Alley                           2732             93.242
Misc Feature                    2824             96.382
Pool QC                         2917             99.556
```

1.3.2 *Imputation of missing values*

Imputation is the process of replacing a missing value with a substituted value, the "best estimate". Imputation should be one of the first feature engineering steps we undertake as it will affect any downstream pre-processing.

Let's first convert the MS SubClass feature to object type feature and create two new features.

[203]:
```
info_func('MS SubClass')
```

[203]:
```
'Identifies the type of dwelling involved in the sale. (Nominal)'
```

[204]:
```
# According to the documentation, the X function is nominal, so we convert it␣
↪into an object type
new_data = data.copy()

print(new_data['MS SubClass'].dtypes, "\n")
print(new_data['MS SubClass'].value_counts())
```

```
int64

20     1079
60      575
50      287
120     192
30      139
160     129
70      128
80      118
90      109
190      61
85       48
75       23
45       18
180      17
40        6
```

```
150        1
Name: MS SubClass, dtype: int64
```

[205]:
```
# https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.astype.html
new_data['MS SubClass'] = new_data['MS SubClass'].astype("str")

for col in ['MS SubClass'] :
    print(new_data[col].dtypes)
```

```
object
```

Create two new features

We create two new features, a binary feature that takes the value 1 if the property is renewed and 0 if not, and the second feature is the age of the property, from the year of construction to the year of sale.

[206]:
```
new_data["YearRemodDemy"]= new_data.apply(lambda x: 1 if x["Year Built"]==␣
 ↪x["Year Remod/Add"] else 0, axis=1)

new_data["AgeBuilt"]= new_data.apply(lambda x: x["Yr Sold"] - x["Year Built"],␣
 ↪axis =1)
```

[207]:
```
new_data[new_data["AgeBuilt"]<0]
```

[207]:
```
     MS SubClass MS Zoning  Lot Frontage  Lot Area Street Alley Lot Shape  \
2180          20        RL         128.0     39290   Pave   NaN       IR1

     Land Contour Utilities Lot Config  … Fence Misc Feature Misc Val  \
2180          Bnk    AllPub     Inside  …   NaN         Elev    17000

     Mo Sold Yr Sold Sale Type  Sale Condition  SalePrice  YearRemodDemy  \
2180      10    2007       New         Partial     183850              0

      AgeBuilt
2180        -1

[1 rows x 78 columns]
```

Remove negative value in AgeBuilt feature and unnecessary features

[208]:
```
new_data = new_data.drop(2180, axis = 0)
new_data = new_data.drop(["Year Remod/Add", 'Mo Sold',"Year Built", "Yr Sold"]␣
 ↪, axis =1)
```

[209]:
```
new_data.shape
```

[209]:
```
(2929, 74)
```

Handling missing values

reference

reference

[210]:
```
# Describe features with missing values
info_func(data_missing_values.index)
```

[210]:
```
Total Bsmt SF       Total square feet of basement area (Continuous)
Electrical                              Electrical system (Ordinal)
BsmtFin SF 1                  Type 1 finished square feet (continuous)
BsmtFin SF 2                  Type 2 finished square feet (Continuous)
Bsmt Unf SF       Unfinished square feet of basement area (Conti…
Garage Area                 Size of garage in square feet (Continuous)
Bsmt Full Bath                    Basement full bathrooms (Discrete)
Bsmt Half Bath                    Basement half bathrooms (Discrete)
Mas Vnr Area        Masonry veneer area in square feet (Continuous)
Mas Vnr Type                           Masonry veneer type (Nominal)
Bsmt Qual             Evaluates the height of the basement (Ordinal)
Bsmt Cond         Evaluates the general condition of the basemen…
BsmtFin Type 1             Rating of basement finished area (Ordinal)
BsmtFin Type 2    Rating of basement finished area (if multiple …
Bsmt Exposure     Refers to walkout or garden level walls (Ordinal)
Garage Type                                Garage location (Nominal)
Garage Cond                               Garage condition (Ordinal)
Garage Finish                   Interior finish of the garage (Ordinal)
Garage Qual                                 Garage quality (Ordinal)
Lot Frontage      Linear feet of street connected to property (C…
Fireplace Qu                              Fireplace quality (Ordinal)
Fence                                          Fence quality (Ordinal)
Alley                          Type of alley access to property (Nominal)
Misc Feature      Miscellaneous feature not covered in other cat…
Pool QC                                         Pool quality (Ordinal)
Name: Columns description, dtype: object
```

Fill missing values in categorical features considering documentation

[211]:
```
for col in ['Mas Vnr Type', 'Misc Feature']:
    new_data[col]=new_data['Misc Feature'].fillna("None")

for col in ['BsmtFin Type 1', 'Bsmt Cond', 'Bsmt Qual', 'BsmtFin Type 2', 'Bsmt␣
↪Exposure']:
    new_data[col]=new_data[col].fillna('NoBasement')

for col in ['Garage Type', 'Garage Finish', 'Garage Qual','Garage Cond']:
    new_data[col]=new_data[col].fillna("NoGarage")

new_data['Fireplace Qu'] = new_data['Fireplace Qu'].fillna("NoFireplace")
new_data['Electrical'] = new_data['Electrical'].fillna("SBrkr")
```

```
new_data['Fence'] = new_data['Fence'].fillna("NoFence")
new_data['Alley'] = new_data['Alley'].fillna("Noalleyaccess")
new_data['Pool QC'] = new_data['Pool QC'].fillna("NoPool")


# Check if there are still categorical features with NaNs
nan_object =new_data.select_dtypes(include =["object"])
nan_object = nan_object.isnull().sum()
nan_object= nan_object[nan_object !=0]
nan_object
```

[211]:
```
Series([], dtype: int64)
```

Checking the numerical features with missing values

[212]:
```
#Checking the numerical features with missing values
data_na = new_data.select_dtypes(include =np.number)
data_na = data_na.isnull().sum()
data_na = data_na[data_na !=0]
data_na
```

[212]:
```
Lot Frontage      490
Mas Vnr Area       23
BsmtFin SF 1        1
BsmtFin SF 2        1
Bsmt Unf SF         1
Total Bsmt SF       1
Bsmt Full Bath      2
Bsmt Half Bath      2
Garage Area         1
dtype: int64
```

[213]:
```
# Let's calculate the most frequent value for each column.
replacement_values_dict = new_data[data_na.index].mode().to_dict(
    orient='records')[0]
replacement_values_dict
```

[213]:
```
{'Lot Frontage': 60.0,
 'Mas Vnr Area': 0.0,
 'BsmtFin SF 1': 0.0,
 'BsmtFin SF 2': 0.0,
 'Bsmt Unf SF': 0.0,
 'Total Bsmt SF': 0.0,
 'Bsmt Full Bath': 0.0,
 'Bsmt Half Bath': 0.0,
 'Garage Area': 0.0}
```

[214]:
```
# Replace missing values with the most frequent value of the corresponding␣
↪column.
new_data= new_data.fillna(replacement_values_dict)

## Check that all columns have 0 missing values
new_data.isnull().sum().value_counts()
```

[214]:
```
0    74
dtype: int64
```

1.3.3 *Convert categorical features type to "category" type*

Let's check the categorical features before conversion

Reference

[215]:
```
new_data.select_dtypes(include = "object").head()
```

[215]:
```
  MS SubClass MS Zoning Street          Alley Lot Shape Land Contour  \
0          20        RL   Pave  Noalleyaccess       IR1          Lvl
1          20        RH   Pave  Noalleyaccess       Reg          Lvl
2          20        RL   Pave  Noalleyaccess       IR1          Lvl
3          20        RL   Pave  Noalleyaccess       Reg          Lvl
4          60        RL   Pave  Noalleyaccess       IR1          Lvl

  Utilities Lot Config Land Slope Neighborhood  … Garage Type Garage Finish  \
0    AllPub     Corner        Gtl        NAmes  …      Attchd           Fin
1    AllPub     Inside        Gtl        NAmes  …      Attchd           Unf
2    AllPub     Corner        Gtl        NAmes  …      Attchd           Unf
3    AllPub     Corner        Gtl        NAmes  …      Attchd           Fin
4    AllPub     Inside        Gtl      Gilbert  …      Attchd           Fin

  Garage Qual Garage Cond Paved Drive Pool QC    Fence Misc Feature Sale Type  \
0          TA          TA           P  NoPool  NoFence         None        WD
1          TA          TA           Y  NoPool    MnPrv         None        WD
2          TA          TA           Y  NoPool  NoFence         Gar2        WD
3          TA          TA           Y  NoPool  NoFence         None        WD
4          TA          TA           Y  NoPool    MnPrv         None        WD

  Sale Condition
0         Normal
1         Normal
2         Normal
3         Normal
4         Normal

[5 rows x 44 columns]
```

[216]:
```
new_data.select_dtypes(include = "object").columns
```

[216]:
```
Index(['MS SubClass', 'MS Zoning', 'Street', 'Alley', 'Lot Shape',
       'Land Contour', 'Utilities', 'Lot Config', 'Land Slope', 'Neighborhood',
       'Condition 1', 'Condition 2', 'Bldg Type', 'House Style', 'Roof Style',
       'Roof Matl', 'Exterior 1st', 'Exterior 2nd', 'Mas Vnr Type',
       'Exter Qual', 'Exter Cond', 'Foundation', 'Bsmt Qual', 'Bsmt Cond',
       'Bsmt Exposure', 'BsmtFin Type 1', 'BsmtFin Type 2', 'Heating',
       'Heating QC', 'Central Air', 'Electrical', 'Kitchen Qual', 'Functional',
       'Fireplace Qu', 'Garage Type', 'Garage Finish', 'Garage Qual',
       'Garage Cond', 'Paved Drive', 'Pool QC', 'Fence', 'Misc Feature',
       'Sale Type', 'Sale Condition'],
      dtype='object')
```

[217]:
```
print("Shape data with the type object :\n", new_data.select_dtypes(include =␣
↪"object").shape)
```

```
Shape data with the type object :
 (2929, 44)
```

[218]:
```
#Description of categorical features
info_func(new_data.select_dtypes(include = "object").columns)
```

[218]:
```
MS SubClass       Identifies the type of dwelling involved in th…
MS Zoning         Identifies the general zoning classification o…
Street                    Type of road access to property (Nominal)
Alley                    Type of alley access to property (Nominal)
Lot Shape                       General shape of property (Ordinal)
Land Contour                     Flatness of the property (Nominal)
Utilities                      Type of utilities available (Ordinal)
Lot Config                              Lot configuration (Nominal)
Land Slope                              Slope of property (Ordinal)
Neighborhood      Physical locations within Ames city limits (ma…
Condition 1               Proximity to various conditions (Nominal)
Condition 2       Proximity to various conditions (if more than …
Bldg Type                                Type of dwelling (Nominal)
House Style                             Style of dwelling (Nominal)
Roof Style                                   Type of roof (Nominal)
Roof Matl                                   Roof material (Nominal)
Exterior 1st                   Exterior covering on house (Nominal)
Exterior 2nd      Exterior covering on house (if more than one m…
Mas Vnr Type                          Masonry veneer type (Nominal)
Exter Qual        Evaluates the quality of the material on the e…
Exter Cond        Evaluates the present condition of the materia…
Foundation                             Type of foundation (Nominal)
Bsmt Qual            Evaluates the height of the basement (Ordinal)
Bsmt Cond         Evaluates the general condition of the basemen…
Bsmt Exposure     Refers to walkout or garden level walls (Ordinal)
```

```
BsmtFin Type 1           Rating of basement finished area (Ordinal)
BsmtFin Type 2    Rating of basement finished area (if multiple …
Heating                                   Type of heating (Nominal)
Heating QC                     Heating quality and condition (Ordinal)
Central Air                         Central air conditioning (Nominal)
Electrical                                 Electrical system (Ordinal)
Kitchen Qual                                 Kitchen quality (Ordinal)
Functional        Home functionality (Assume typical unless dedu…
Fireplace Qu                               Fireplace quality (Ordinal)
Garage Type                                  Garage location (Nominal)
Garage Finish                  Interior finish of the garage (Ordinal)
Garage Qual                                   Garage quality (Ordinal)
Garage Cond                                 Garage condition (Ordinal)
Paved Drive                                   Paved driveway (Ordinal)
Pool QC                                         Pool quality (Ordinal)
Fence                                          Fence quality (Ordinal)
Misc Feature      Miscellaneous feature not covered in other cat…
Sale Type                                       Type of sale (Nominal)
Sale Condition                             Condition of sale (Nominal)
Name: Columns description, dtype: object
```

[219]:
```
# How many unique values in each categorical column?
unique_values = new_data.select_dtypes(include = "object").apply(lambda cols:␣
↪len(cols.value_counts())).sort_values()
unique_values
```

[219]:
```
Street             2
Central Air        2
Alley              3
Utilities          3
Land Slope         3
Paved Drive        3
Lot Shape          4
Land Contour       4
Exter Qual         4
Garage Finish      4
Bldg Type          5
Mas Vnr Type       5
Kitchen Qual       5
Electrical         5
Pool QC            5
Bsmt Exposure      5
Fence              5
Lot Config         5
Misc Feature       5
Heating QC         5
Exter Cond         5
```

```
Foundation          6
Garage Qual         6
Garage Cond         6
Fireplace Qu        6
Heating             6
Sale Condition      6
Bsmt Cond           6
Bsmt Qual           6
Roof Style          6
BsmtFin Type 1      7
MS Zoning           7
Garage Type         7
BsmtFin Type 2      7
Condition 2         8
House Style         8
Roof Matl           8
Functional          8
Condition 1         9
Sale Type          10
Exterior 1st       16
MS SubClass        16
Exterior 2nd       17
Neighborhood       28
dtype: int64
```

The X function contains up to 28 different categories, that is, we will have 28 functions once this function is converted to a numeric variable. For this reason, we limit the different categories to 10. Of course, we could have kept all the categories, but we are just experimenting with the possibilities.

[220]:
```
# Arbitrary limit of 10 unique values
drop_cate_cols = unique_values[unique_values > 10].index
print(drop_cate_cols, "\n")
```

```
Index(['Exterior 1st', 'MS SubClass', 'Exterior 2nd', 'Neighborhood'],
dtype='object')
```

[221]:
```
#Delete this features : 'Exterior 1st', 'Exterior 2nd', 'Neighborhood'
data_transform = new_data.drop(drop_cate_cols, axis=1)
print(data_transform.shape)
data_transform.head()
```

```
(2929, 70)
```

[221]:
```
  MS Zoning  Lot Frontage  Lot Area Street          Alley Lot Shape  \
0        RL         141.0     31770   Pave  Noalleyaccess       IR1
1        RH          80.0     11622   Pave  Noalleyaccess       Reg
2        RL          81.0     14267   Pave  Noalleyaccess       IR1
3        RL          93.0     11160   Pave  Noalleyaccess       Reg
```

```
4        RL          74.0     13830    Pave   Noalleyaccess       IR1

  Land Contour Utilities Lot Config Land Slope  … Pool Area Pool QC  \
0          Lvl    AllPub     Corner        Gtl  …         0  NoPool
1          Lvl    AllPub     Inside        Gtl  …         0  NoPool
2          Lvl    AllPub     Corner        Gtl  …         0  NoPool
3          Lvl    AllPub     Corner        Gtl  …         0  NoPool
4          Lvl    AllPub     Inside        Gtl  …         0  NoPool

     Fence Misc Feature  Misc Val  Sale Type Sale Condition SalePrice  \
0  NoFence         None         0         WD         Normal    215000
1    MnPrv         None         0         WD         Normal    105000
2  NoFence         Gar2     12500         WD         Normal    172000
3  NoFence         None         0         WD         Normal    244000
4    MnPrv         None         0         WD         Normal    189900

  YearRemodDemy  AgeBuilt
0             1        50
1             1        49
2             1        52
3             1        42
4             0        13

[5 rows x 70 columns]
```

[222]:
```
#shape of categorical columns
data_transform.select_dtypes(include=['object']).shape
```

[222]:
```
(2929, 40)
```

[223]:
```
# Select only the remaining text columns and convert them to categories
objetc_cols = data_transform.select_dtypes(include=['object'])
for col in objetc_cols:
    data_transform[col] = data_transform[col].astype('category')
```

[224]:
```
data_transform.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2929 entries, 0 to 2929
Data columns (total 70 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   MS Zoning        2929 non-null   category
 1   Lot Frontage     2929 non-null   float64
 2   Lot Area         2929 non-null   int64
 3   Street           2929 non-null   category
 4   Alley            2929 non-null   category
 5   Lot Shape        2929 non-null   category
```

```
6   Land Contour     2929 non-null   category
7   Utilities        2929 non-null   category
8   Lot Config       2929 non-null   category
9   Land Slope       2929 non-null   category
10  Condition 1      2929 non-null   category
11  Condition 2      2929 non-null   category
12  Bldg Type        2929 non-null   category
13  House Style      2929 non-null   category
14  Overall Qual     2929 non-null   int64
15  Overall Cond     2929 non-null   int64
16  Roof Style       2929 non-null   category
17  Roof Matl        2929 non-null   category
18  Mas Vnr Type     2929 non-null   category
19  Mas Vnr Area     2929 non-null   float64
20  Exter Qual       2929 non-null   category
21  Exter Cond       2929 non-null   category
22  Foundation       2929 non-null   category
23  Bsmt Qual        2929 non-null   category
24  Bsmt Cond        2929 non-null   category
25  Bsmt Exposure    2929 non-null   category
26  BsmtFin Type 1   2929 non-null   category
27  BsmtFin SF 1     2929 non-null   float64
28  BsmtFin Type 2   2929 non-null   category
29  BsmtFin SF 2     2929 non-null   float64
30  Bsmt Unf SF      2929 non-null   float64
31  Total Bsmt SF    2929 non-null   float64
32  Heating          2929 non-null   category
33  Heating QC       2929 non-null   category
34  Central Air      2929 non-null   category
35  Electrical       2929 non-null   category
36  2nd Flr SF       2929 non-null   int64
37  Low Qual Fin SF  2929 non-null   int64
38  Gr Liv Area      2929 non-null   int64
39  Bsmt Full Bath   2929 non-null   float64
40  Bsmt Half Bath   2929 non-null   float64
41  Full Bath        2929 non-null   int64
42  Half Bath        2929 non-null   int64
43  Bedroom AbvGr    2929 non-null   int64
44  Kitchen AbvGr    2929 non-null   int64
45  Kitchen Qual     2929 non-null   category
46  Functional       2929 non-null   category
47  Fireplaces       2929 non-null   int64
48  Fireplace Qu     2929 non-null   category
49  Garage Type      2929 non-null   category
50  Garage Finish    2929 non-null   category
51  Garage Area      2929 non-null   float64
52  Garage Qual      2929 non-null   category
53  Garage Cond      2929 non-null   category
```

```
 54  Paved Drive      2929 non-null   category
 55  Wood Deck SF     2929 non-null   int64
 56  Open Porch SF    2929 non-null   int64
 57  Enclosed Porch   2929 non-null   int64
 58  3Ssn Porch       2929 non-null   int64
 59  Screen Porch     2929 non-null   int64
 60  Pool Area        2929 non-null   int64
 61  Pool QC          2929 non-null   category
 62  Fence            2929 non-null   category
 63  Misc Feature     2929 non-null   category
 64  Misc Val         2929 non-null   int64
 65  Sale Type        2929 non-null   category
 66  Sale Condition   2929 non-null   category
 67  SalePrice        2929 non-null   int64
 68  YearRemodDemy    2929 non-null   int64
 69  AgeBuilt         2929 non-null   int64
dtypes: category(40), float64(9), int64(21)
memory usage: 833.1 KB
```

[225]:
```
# Create dummy columns and add them to the DataFrame
data_transform = pd.concat([
    data_transform,
    pd.get_dummies(data_transform.select_dtypes(include=['category']),␣
 ↪drop_first=True)
], axis=1)
```

[226]:
```
data_transform.shape
```

[226]:
```
(2929, 250)
```

[227]:
```
data_transform.head()
```

[227]:
```
  MS Zoning  Lot Frontage  Lot Area Street          Alley Lot Shape  \
0        RL         141.0     31770   Pave  Noalleyaccess       IR1
1        RH          80.0     11622   Pave  Noalleyaccess       Reg
2        RL          81.0     14267   Pave  Noalleyaccess       IR1
3        RL          93.0     11160   Pave  Noalleyaccess       Reg
4        RL          74.0     13830   Pave  Noalleyaccess       IR1

  Land Contour Utilities Lot Config Land Slope  … Sale Type_ConLw  \
0          Lvl    AllPub     Corner        Gtl  …               0
1          Lvl    AllPub     Inside        Gtl  …               0
2          Lvl    AllPub     Corner        Gtl  …               0
3          Lvl    AllPub     Corner        Gtl  …               0
4          Lvl    AllPub     Inside        Gtl  …               0

  Sale Type_New Sale Type_Oth Sale Type_VWD  Sale Type_WD   \
0             0             0             0              1
```

```
1              0              0              0                 1
2              0              0              0                 1
3              0              0              0                 1
4              0              0              0                 1

   Sale Condition_AdjLand Sale Condition_Alloca Sale Condition_Family  \
0                      0                     0                     0
1                      0                     0                     0
2                      0                     0                     0
3                      0                     0                     0
4                      0                     0                     0

  Sale Condition_Normal  Sale Condition_Partial
0                     1                       0
1                     1                       0
2                     1                       0
3                     1                       0
4                     1                       0

[5 rows x 250 columns]
```

[228]:
```
#Delete categorical columns
categorical_columns = data_transform.select_dtypes(include=['category']).columns

data_transform = data_transform.drop(categorical_columns, axis = 1)
```

[229]:
```
data_transform.head()
```

[229]:
```
   Lot Frontage  Lot Area  Overall Qual  Overall Cond  Mas Vnr Area  \
0         141.0     31770             6             5         112.0
1          80.0     11622             5             6           0.0
2          81.0     14267             6             6         108.0
3          93.0     11160             7             5           0.0
4          74.0     13830             5             5           0.0

   BsmtFin SF 1  BsmtFin SF 2  Bsmt Unf SF  Total Bsmt SF  2nd Flr SF  …  \
0         639.0           0.0        441.0         1080.0           0  …
1         468.0         144.0        270.0          882.0           0  …
2         923.0           0.0        406.0         1329.0           0  …
3        1065.0           0.0       1045.0         2110.0           0  …
4         791.0           0.0        137.0          928.0         701  …

   Sale Type_ConLw  Sale Type_New  Sale Type_Oth  Sale Type_VWD  \
0                0              0              0              0
1                0              0              0              0
2                0              0              0              0
3                0              0              0              0
```

```
4                0              0              0              0

   Sale Type_WD   Sale Condition_AdjLand  Sale Condition_Alloca  \
0             1                       0                      0
1             1                       0                      0
2             1                       0                      0
3             1                       0                      0
4             1                       0                      0

   Sale Condition_Family  Sale Condition_Normal  Sale Condition_Partial
0                      0                      1                       0
1                      0                      1                       0
2                      0                      1                       0
3                      0                      1                       0
4                      0                      1                       0

[5 rows x 210 columns]
```

[230]:
```python
data_transform.shape
```

[230]:
```
(2929, 210)
```

1.3.4 *Training / Test split*

[231]:
```python
from sklearn.model_selection import train_test_split
```

Let's set the x and y variables to the feature and label values

[232]:
```python
X =data_transform.drop("SalePrice", axis = 1)
y =data_transform["SalePrice"]
```

[233]:
```python
X.shape
```

[233]:
```
(2929, 209)
```

[234]:
```python
y.shape
```

[234]:
```
(2929,)
```

Train / test split with test_size=0.3 and a random_state of 42

[235]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
                        X, y, test_size=0.30, random_state=42)
```

[236]:
```python
X_train.shape
```

[236]:
```
(2050, 209)
```

[237]:
```
X_test.shape
```

[237]: (879, 209)

1.3.5 *Normalization*

Normalization reduces outliers by compressing values between an accurate scale.

Reference

Reference

[238]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[239]:
```
X_train.shape
```

[239]: (2050, 209)

[240]:
```
X_test.shape
```

[240]: (879, 209)

1.4 Section 4 : Modelization

[241]:
```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
```

1.4.1 *Lineaire Regression*

Reference

[246]:
```
# Entrainement
np.random.seed(0)
model = LinearRegression()
model.fit(X_train, y_train.values)

# Prédiction
predictions = model.predict(X_test)
mse = mean_squared_error(y_test.values, predictions)
rmse = np.sqrt(mse)
print(rmse)
```

2905945045232419.0

We have a very very high RMSE, which proves the inefficiency of our model which is in a situation of under-learning. We will apply a feature selection technique to see if there will be an improvement.

1.4.2 *Features selection*

The feature selection allows according to certain techniques to remove the least information features and which aims to improve the performance of machine learning models.

Reference

Reference

[247]:
```
#We use a threshold of 0.01 to remove all features with zero and near-zero␣
↪variance.

from sklearn.feature_selection import VarianceThreshold
var_thr = VarianceThreshold(threshold = 0.01)
var_thr.fit(X)

var_thr.get_support()
```

[247]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True, False,  True, False, False,  True,  True, False,
        True,  True,  True, False,  True,  True,  True,  True, False,
       False,  True,  True, False,  True,  True, False,  True,  True,
       False,  True, False,  True, False, False, False,  True, False,
       False, False, False, False,  True,  True,  True,  True, False,
        True, False, False,  True,  True,  True,  True, False,  True,
       False, False,  True, False, False, False, False, False, False,
        True, False,  True, False,  True,  True,  True,  True,  True,
       False,  True,  True,  True,  True, False, False,  True,  True,
        True, False,  True,  True,  True,  True, False,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True, False, False,
       False, False,  True,  True, False,  True,  True,  True, False,
       False,  True,  True,  True, False,  True, False,  True,  True,
        True, False, False,  True,  True,  True,  True,  True,  True,
        True,  True,  True, False,  True,  True,  True,  True,  True,
        True, False,  True, False,  True,  True, False,  True, False,
        True,  True,  True, False, False, False, False,  True,  True,
       False,  True,  True, False,  True, False, False, False, False,
       False, False,  True, False, False,  True, False, False,  True,
        True,  True])
```

- True : High Variance
- False : Low Variance

[248]:
```python
# The low Variance features
concol = [column for column in X.columns
          if column not in X.columns[var_thr.get_support()]]

for features in concol:
    print(features)
```

```
MS Zoning_C (all)
MS Zoning_I (all)
MS Zoning_RH
Street_Pave
Lot Shape_IR3
Utilities_NoSeWa
Utilities_NoSewr
Lot Config_FR3
Land Slope_Sev
Condition 1_PosA
Condition 1_RRAe
Condition 1_RRNe
Condition 1_RRNn
Condition 2_Feedr
Condition 2_PosA
Condition 2_PosN
Condition 2_RRAe
Condition 2_RRAn
Condition 2_RRNn
House Style_1.5Unf
House Style_2.5Fin
House Style_2.5Unf
Roof Style_Gambrel
Roof Style_Mansard
Roof Style_Shed
Roof Matl_Membran
Roof Matl_Metal
Roof Matl_Roll
Roof Matl_Tar&Grv
Roof Matl_WdShake
Roof Matl_WdShngl
Mas Vnr Type_Othr
Mas Vnr Type_TenC
Exter Cond_Po
Foundation_Stone
Foundation_Wood
Bsmt Qual_Po
Bsmt Cond_Po
Heating_GasW
Heating_Grav
Heating_OthW
```

```
Heating_Wall
Heating QC_Po
Electrical_FuseP
Electrical_Mix
Kitchen Qual_Po
Functional_Maj2
Functional_Sal
Functional_Sev
Garage Type_CarPort
Garage Qual_Gd
Garage Qual_Po
Garage Cond_Gd
Garage Cond_Po
Pool QC_Fa
Pool QC_Gd
Pool QC_NoPool
Pool QC_TA
Fence_MnWw
Misc Feature_Othr
Misc Feature_TenC
Sale Type_CWD
Sale Type_Con
Sale Type_ConLD
Sale Type_ConLI
Sale Type_ConLw
Sale Type_Oth
Sale Type_VWD
Sale Condition_AdjLand
Sale Condition_Alloca
```

[249]:
```python
# Delete the low Variance features :
data_transform_select = data_transform.drop(concol,axis=1)
data_transform_select.shape
```

[249]: (2929, 140)

[250]:
```python
X =data_transform_select.drop("SalePrice", axis = 1)
y =data_transform_select["SalePrice"]
```

[252]:
```python
y.shape
```

[252]: (2929,)

1.4.3 *Lineaire Regression with the high Variance features*

[254]:
```
X_train, X_test, y_train, y_test = train_test_split(
                        X, y, test_size=0.30, random_state=42)
```

[255]:
```
X_train.shape
```

[255]: (2050, 139)

[257]:
```
X_test.shape
```

[257]: (879, 139)

[258]:
```
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

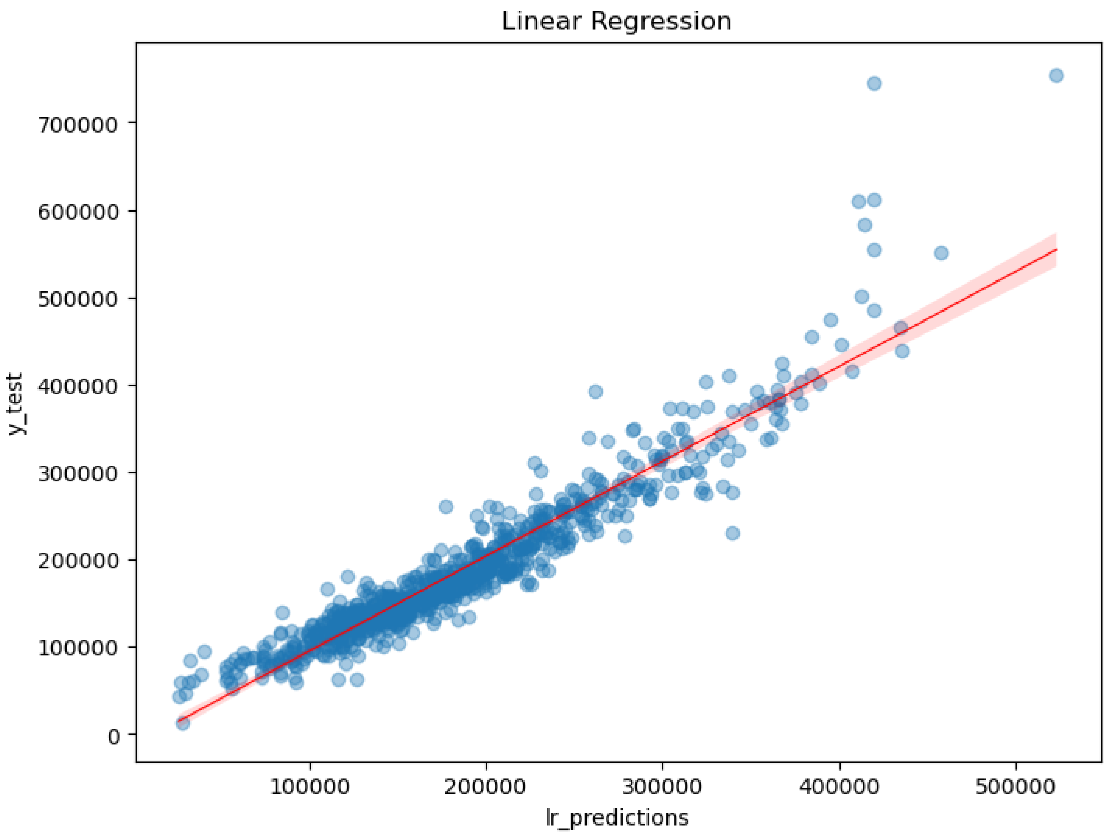
[261]:
```
# Entrainement
np.random.seed(0)
lr_model = LinearRegression()
lr_model.fit(X_train, y_train.values)

# Prédiction
lr_predictions = lr_model.predict(X_test)
mse = mean_squared_error(y_test.values, lr_predictions)
lr_rmse = np.sqrt(mse)
print(lr_rmse)
```

28416.90320185764

[262]:
```
data_pred =pd.DataFrame({"y_test" : y_test, "lr_predictions" : lr_predictions})
```

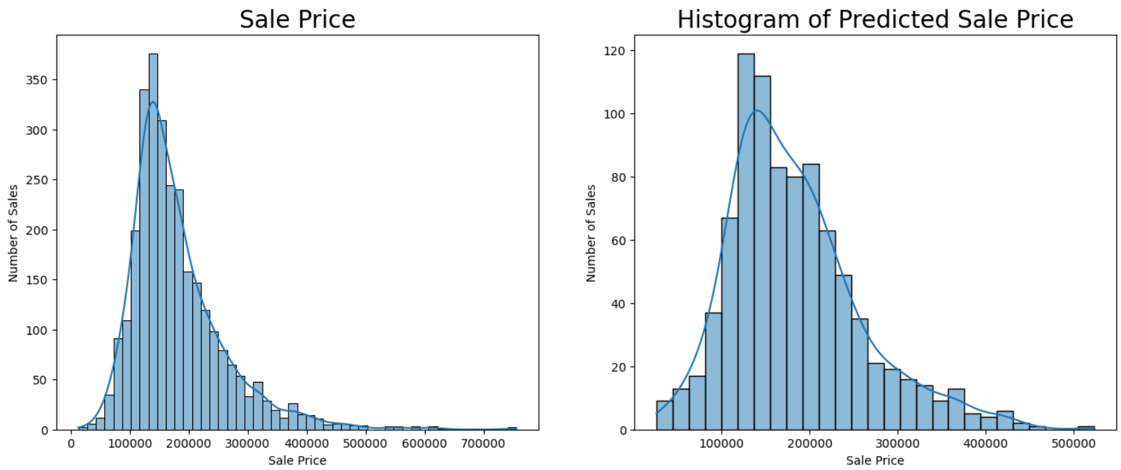
[263]:
```
plt.figure(figsize=(8, 6))
sns.regplot(x = "lr_predictions", y ="y_test",  scatter_kws={'alpha': 0.4},
                line_kws={'color': 'red','linewidth':0.8} ,data = data_pred)
plt.title("Linear Regression")
```

[263]: Text(0.5, 1.0, 'Linear Regression')



We have significantly reduced the rmse, but we are left with another problem: outliers related to the sale price. We will use the logarithmic technique to scale the selling price.

[264]:
```
# SalePrice after transformation
fig, ax = plt.subplots(figsize=(16,6), ncols=2)
sns.histplot(x = "SalePrice", bins=50, kde=True, ax=ax[0], data =data)
ax[0].set_title('Sale Price', fontsize=20)
ax[0].set_xlabel('Sale Price')
ax[0].set_ylabel('Number of Sales')

sns.histplot(x="lr_predictions", kde=True, ax=ax[1], data=data_pred)
ax[1].set_title("Histogram of Predicted Sale Price", fontsize=20)
plt.xlabel('Sale Price')
plt.ylabel('Number of Sales');
plt.savefig('pred_hist.png', bbox_inches="tight")
```



We notice that the price prediction histogram is better adjusted (the values are well distributed).

1.4.4 *Random Forest Regressor*

Reference

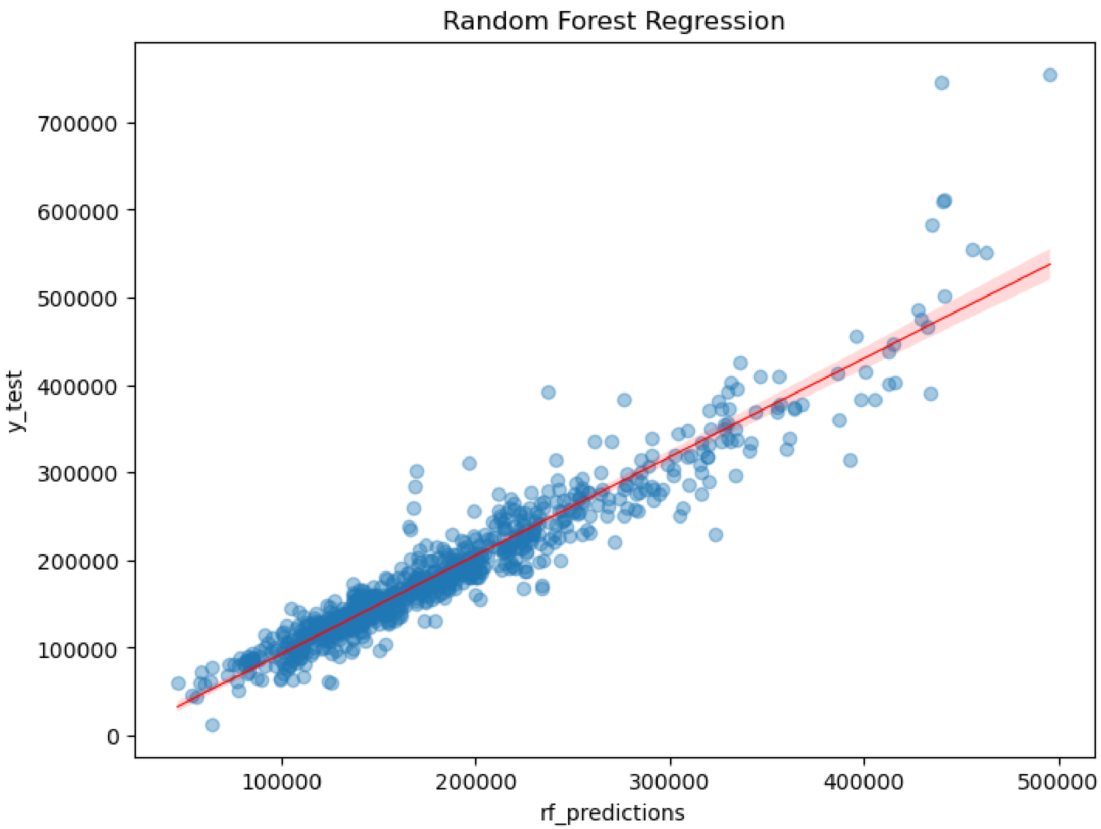
[265]:
```
rf_model = RandomForestRegressor(random_state=0)
rf_model.fit(X_train, y_train.values)

rf_predictions = rf_model.predict(X_test)

mse = mean_squared_error(y_test.values, rf_predictions)

rf_rmse = np.sqrt(mse)

print(rf_rmse)
```

```
28269.55558437841
```

[266]:
```
data_pred =pd.DataFrame({"y_test" : y_test, "rf_predictions" : rf_predictions})
```

[267]:
```
plt.figure(figsize=(8, 6))
sns.regplot(x = "rf_predictions", y ="y_test",  scatter_kws={'alpha': 0.4},
                line_kws={'color': 'red','linewidth':0.8} ,data = data_pred)
plt.title("Random Forest Regression")
```

[267]:
```
Text(0.5, 1.0, 'Random Forest Regression')
```



1.4.5 *Gradient Boosting Regressor*

Reference

[268]:
```
gbr_model = GradientBoostingRegressor(random_state=0)
gbr_model.fit(X_train, y_train.values)

gbr_predictions = gbr_model.predict(X_test)

mse = mean_squared_error(y_test.values, gbr_predictions)
gbr_rmse = np.sqrt(mse)
print(gbr_rmse)
```

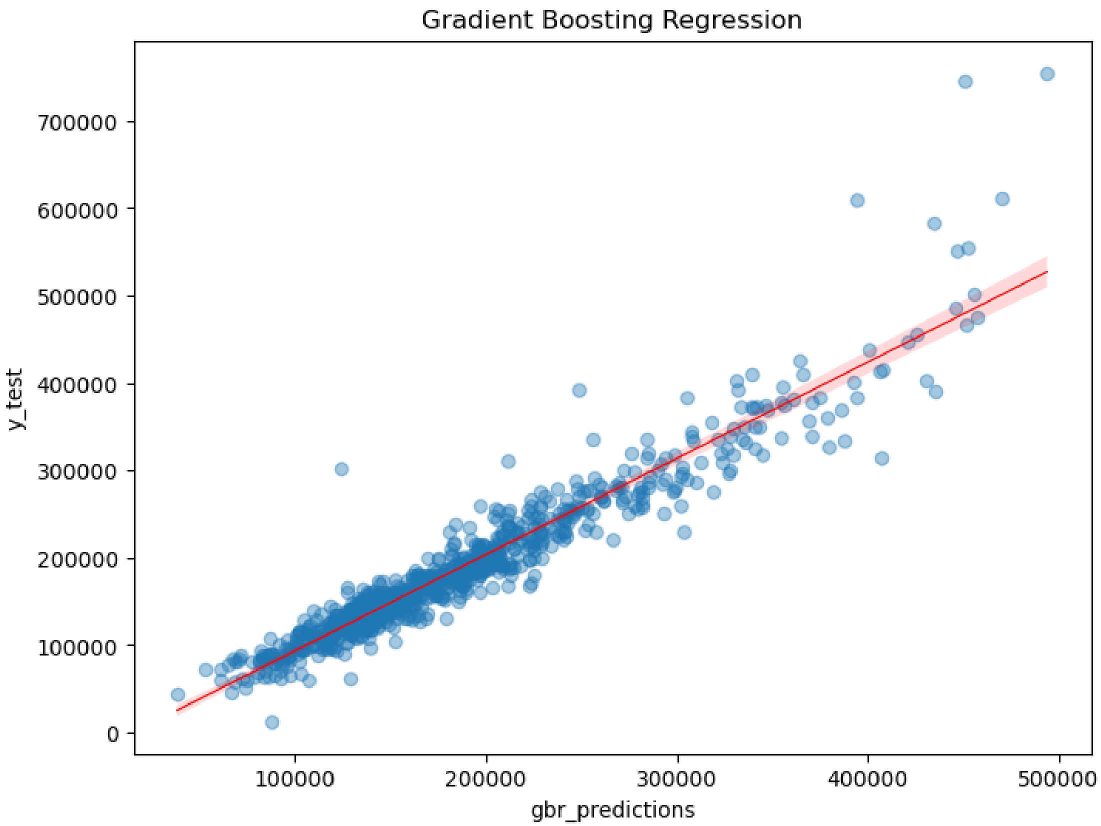
26813.232674058992

[269]:
```
data_pred =pd.DataFrame({"y_test" : y_test, "gbr_predictions" :␣
 ↪gbr_predictions})
```

[270]:
```
plt.figure(figsize=(8, 6))
sns.regplot(x = "gbr_predictions", y ="y_test",  scatter_kws={'alpha': 0.4},
                line_kws={'color': 'red','linewidth':0.8} ,data = data_pred)
```

```
plt.title("Gradient Boosting Regression")
```

[270]: Text(0.5, 1.0, 'Gradient Boosting Regression')



1.4.6 *Delete outliers values of sale price*

Reference

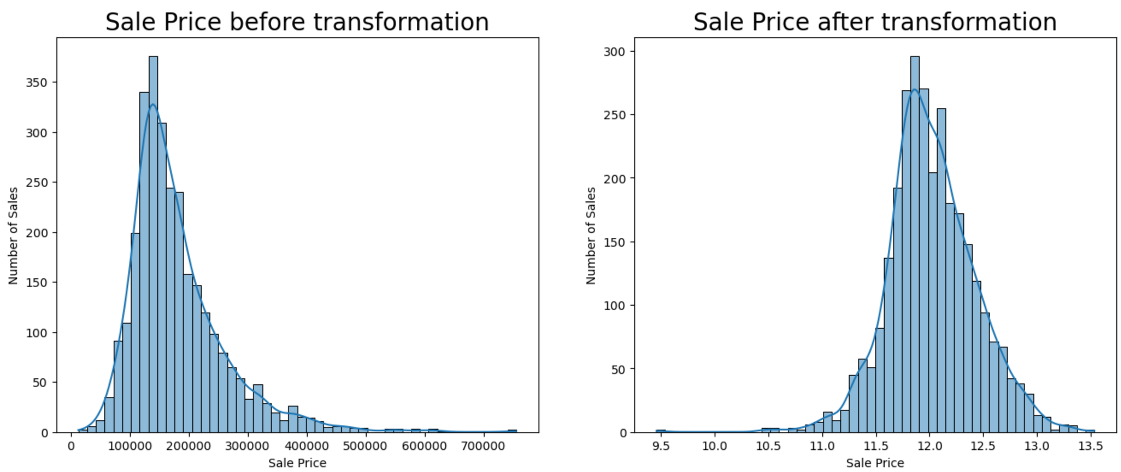
[271]:
```
y_log = y.copy()
target_log = np.log1p(y)

data_target_log =pd.Series({"target_log":target_log})
data_target_log
```

[271]:
```
target_log    0        12.278398
1        11.561725
2        12…
dtype: object
```

[272]:
```
# SalePrice after transformation
fig, ax = plt.subplots(figsize=(16,6), ncols=2)
```

```
sns.histplot(x = "SalePrice", bins=50, kde=True, ax=ax[0], data =data)
ax[0].set_title('Sale Price before transformation', fontsize=20)
ax[0].set_xlabel('Sale Price')
ax[0].set_ylabel('Number of Sales')

sns.histplot(x = "target_log", bins=50, kde=True, ax=ax[1], data␣
 ↪=data_target_log)
ax[1].set_title('Sale Price after transformation', fontsize=20)
ax[1].set_xlabel('Sale Price')
ax[1].set_ylabel('Number of Sales')
```

[272]: Text(0, 0.5, 'Number of Sales')



1.4.7 *Linear Regression after logarithmic transformation of the sale price*

[273]:
```
X_train, X_test, y_train, y_test_log = train_test_split(
                        X, target_log, test_size=0.30, random_state=42)
```

[274]:
```
# Entrainement
lr_model_log = LinearRegression()
lr_model_log.fit(X_train, y_train.values)

# Prédiction
lr_predictions_log = lr_model_log.predict(X_test)
mse = mean_squared_error(y_test_log.values, lr_predictions_log )
lr_rmse_log = np.sqrt(mse)
print(round(lr_rmse_log, 3)*100)
```

12.5

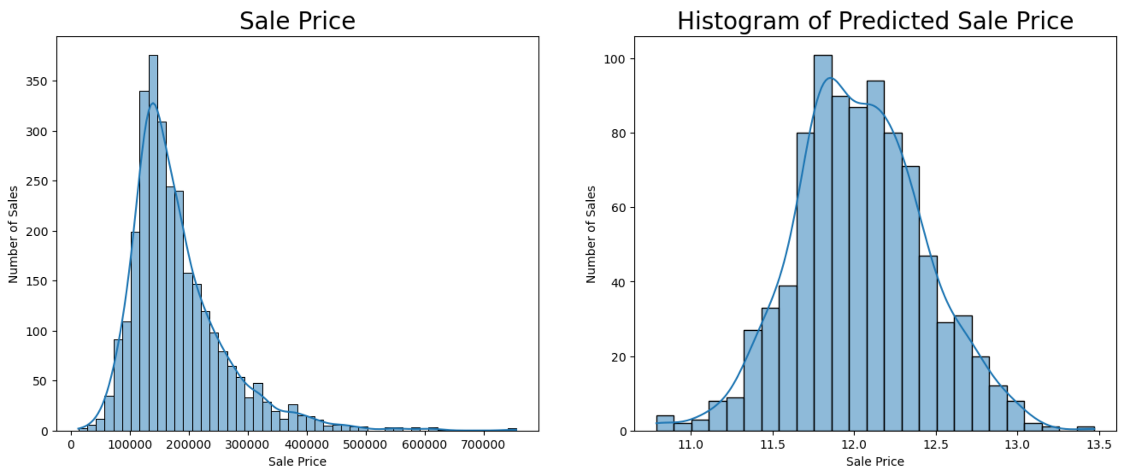
[275]:
```
data_log =pd.DataFrame({"y_test_log" : y_test_log, "lr_predictions_log" :␣
↪lr_predictions_log})
```

[276]:
```
plt.figure(figsize=(8, 6))
sns.regplot(x = "lr_predictions_log", y ="y_test_log",  scatter_kws={'alpha': 0.
↪4},
                line_kws={'color': 'red','linewidth':0.8} ,data = data_log)
plt.title("Linear Regression after log transformation of sale price")
```

[276]:
```
Text(0.5, 1.0, 'Linear Regression after log transformation of sale price')
```

[277]:
```
# SalePrice after transformation
fig, ax = plt.subplots(figsize=(16,6), ncols=2)
sns.histplot(x = "SalePrice", bins=50, kde=True, ax=ax[0], data =data)
ax[0].set_title('Sale Price', fontsize=20)
ax[0].set_xlabel('Sale Price')
ax[0].set_ylabel('Number of Sales')

sns.histplot(x="lr_predictions_log", kde=True, ax=ax[1], data=data_log)
ax[1].set_title("Histogram of Predicted Sale Price", fontsize=20)
```

```
plt.xlabel('Sale Price')
plt.ylabel('Number of Sales');
plt.savefig('pred_hist.png', bbox_inches="tight")
```



We notice that the price prediction histogram is better and better adjusted (the values are well distributed) after the logarithmic transformation.

1.4.8 *Random Forest Regressor after logarithmic transformation of the sale price*

[278]:
```
rf_model_log= RandomForestRegressor(random_state=0)

rf_model_log.fit(X_train, y_train.values)

rf_predictions_log = rf_model_log.predict(X_test)

mse = mean_squared_error(y_test_log.values, rf_predictions_log)
rf_rmse_log = np.sqrt(mse)
print(round(rf_rmse_log, 3)*100)
```

```
14.2
```

[279]:
```
data_log =pd.DataFrame({"y_test_log" : y_test_log, "rf_predictions_log" :␣
↪rf_predictions_log})
```

[280]:
```
plt.figure(figsize=(8, 6))

sns.regplot(x = "rf_predictions_log", y ="y_test_log",  scatter_kws={'alpha': 0.
↪4},

                line_kws={'color': 'red','linewidth':0.8} ,data = data_log)
plt.title("Random Forest Regression after log transformation of sale price")
```

[280]: Text(0.5, 1.0, 'Random Forest Regression after log transformation of sale price')

1.4.9 *Gradient Boosting Regressor after logarithmic transformation of the sale price*

[281]:
```
gbr_model_log= GradientBoostingRegressor(random_state=0)

gbr_model_log.fit(X_train, y_train.values)

gbr_predictions_log = gbr_model_log.predict(X_test)

mse = mean_squared_error(y_test_log.values, gbr_predictions_log)

gbr_rmse_log  = np.sqrt(mse)

print(round(gbr_rmse_log , 3)*100)
```

13.0

[282]:
```
data_pred_log =pd.DataFrame({"y_test_log" : y_test_log, "gbr_predictions_log" :␣
↪gbr_predictions_log})
```

[283]:
```
plt.figure(figsize=(8, 6))

sns.regplot(x = "gbr_predictions_log", y ="y_test_log",  scatter_kws={'alpha':␣
↪0.4},

                line_kws={'color': 'red','linewidth':0.8} ,data = data_pred_log)
plt.title("Gradient Boosting Regression after log transformation of sale price")
```

[283]:
```
Text(0.5, 1.0, 'Gradient Boosting Regression after log transformation of sale
price')
```

As you can see, scaling the sale price reduced the rate significantly to 12.5%.

[284]:
```
data_models = pd.DataFrame({"Linear Regression" : [lr_rmse, lr_rmse_log],␣
↪"Random Forest Regressor": [rf_rmse, rf_rmse_log],
                          "Gradient Boosting Regressor" : [gbr_rmse,␣
↪gbr_rmse_log]}, index = ["rmse", "rmse_log"])
```

```
data_models
```

[284]:
```
            Linear Regression  Random Forest Regressor  \
rmse             28416.903202             28269.555584
rmse_log             0.125346                 0.142076

            Gradient Boosting Regressor
rmse                       26813.232674
rmse_log                       0.130468
```

Looking at the RMSEs of the different models, we notice that the ensemble models are less affected by outliers related to the selling price, but as soon as we use the natural logarithmic transformation to transform the selling price, the linear regression becomes better than all models d'ensemble.

[285]:
```
data_preds = pd.DataFrame({"y_test" : y_test, "lr_predictions": lr_predictions␣
 ↪, "rf_predictions": rf_predictions,
                           "gbr_predictions " : gbr_predictions , "y_test_log" :
 ↪ y_test_log, "lr_predictions_log" : lr_predictions_log,
                           "rf_predictions_log" : rf_predictions_log,␣
 ↪"gbr_predictions_log" : gbr_predictions_log})
data_preds.reset_index(drop = True).head()
```

[285]:
```
   y_test  lr_predictions  rf_predictions  gbr_predictions   y_test_log  \
0  214000   223337.318933       224416.64    224377.947721    12.273736
1  157000   180205.537683       139989.74    157659.419722    11.964007
2  260000   256604.240808       235108.81    247338.083767    12.468441
3  183000   187039.303308       188387.74    185877.361957    12.117247
4  120750   113737.428308       123824.66    113697.100820    11.701486

   lr_predictions_log  rf_predictions_log  gbr_predictions_log
0           12.291416           12.360415            12.369354
1           11.995184           11.879825            11.943125
2           12.446257           12.425970            12.408737
3           12.146563           12.139312            12.186276
4           11.694305           11.710226            11.667202
```

The predictions after the logarithmic transformation of the selling price approximate true selling price values (y_test_log).

1.4.10 *Predict the price of a house taken randomly from the database data_transform_select*

Let's take a random house from the data_transform_select database and predict its price and compare it to its current selling price.

[402]:
```
import random
random.seed(101)
random_ind = random.randint(0,len(data_transform_select))
```

```
single_house = data_transform_select.drop('SalePrice',axis=1).iloc[random_ind]
single_house
```

[402]:
```
Lot Frontage                 110.0
Lot Area                   16163.0
Overall Qual                   8.0
Overall Cond                   5.0
Mas Vnr Area                 232.0
                            …
Sale Type_New                  0.0
Sale Type_WD                   1.0
Sale Condition_Family          0.0
Sale Condition_Normal          1.0
Sale Condition_Partial         0.0
Name: 2381, Length: 139, dtype: float64
```

[403]:
```
single_house = scaler.transform(single_house.values.reshape(-1, 139))
```

```
C:\Users\HP\anaconda3\envs\DeepLearning\lib\site-packages\sklearn\base.py:439:
UserWarning: X does not have valid feature names, but MinMaxScaler was fitted
with feature names
  warnings.warn(
```

[411]:
```
gbr_model.predict(single_house)
```

[411]:
```
array([253913.30813697])
```

[405]:
```
data_transform_select.iloc[random_ind]["SalePrice"]
```

[405]:
```
252000.0
```

We were predicting a selling price of 253913 dollars for this house while its actual selling price is 252000 dollars. We are not far from the actual price. If we continue to choose another house at random, we will necessarily find a price almost equal to the actual sale price.

1.5 Conclusion

To push this work, we can adjust the hyperparameters of different models, increase the variance threshold for entity selection, adjust other regression models or artificial neural networks to see if we are further improving the performance of our models.

1.6 References

1. https://github.com/Quartz/bad-data-guide
2. https://bradleyboehmke.github.io/HOML/engineering.html
3. https://medium.com/analytics-vidhya/data-preprocessing-and-exploratory-data-analysis-for-machine-learning-75b8a6468b72

4. https://towardsdatascience.com/data-preprocessing-and-eda-for-data-science-50ba6ea65c0a
5. https://www.analyticsvidhya.com/blog/2022/07/step-by-step-exploratory-data-analysis-eda-using-python/
6. https://online.stat.psu.edu/stat857/node/224/
7. https://github.com/Murdocc007/Statistical-Modelling/tree/master
8. https://www.ormandata.com/projects/project-two-jhz68
9. https://www.kaggle.com/code/marto24/beginners-prediction-top3
10. https://medium.com/analytics-vidhya/how-to-remove-outliers-for-machine-learning-24620c4657e8
11. https://medium.com/nerd-for-tech/removing-constant-variables-feature-selection-463e2d6a30d9
12. https://github.com/maym5/ames-housing/blob/main/kaggle_housing.ipynb
13. https://python-graph-gallery.com/92-control-color-in-seaborn-heatmaps/
14. https://medium.com/@szabo.bibor/how-to-create-a-seaborn-correlation-heatmap-in-python-834c0686b88e
15. https://www.kaggle.com/code/gusthema/house-prices-prediction-using-tfdf
16. https://towardsdatascience.com/imputing-missing-data-with-simple-and-advanced-techniques-f5c7b157fb87
17. https://towardsdatascience.com/6-pandas-display-options-you-should-memories-84adf8887bc3
18. https://chriskhanhtran.github.io/minimal-portfolio/projects/ames-house-price.html