Notebook 2 Exploratory Data Analysis

June 14, 2022

[1]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Comment this if the data visualisations doesn't work on your side
%matplotlib inline

plt.style.use('bmh')
```

[2]:
```python
df = pd.read_csv('train.csv')
df.head()
```

[2]:
```
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0   1          60       RL         65.0     8450   Pave   NaN      Reg
1   2          20       RL         80.0     9600   Pave   NaN      Reg
2   3          60       RL         68.0    11250   Pave   NaN      IR1
3   4          70       RL         60.0     9550   Pave   NaN      IR1
4   5          60       RL         84.0    14260   Pave   NaN      IR1

  LandContour Utilities  … PoolArea PoolQC Fence MiscFeature MiscVal MoSold  \
0         Lvl    AllPub  …        0    NaN   NaN         NaN       0      2
1         Lvl    AllPub  …        0    NaN   NaN         NaN       0      5
2         Lvl    AllPub  …        0    NaN   NaN         NaN       0      9
3         Lvl    AllPub  …        0    NaN   NaN         NaN       0      2
4         Lvl    AllPub  …        0    NaN   NaN         NaN       0     12

  YrSold  SaleType  SaleCondition  SalePrice
0   2008        WD         Normal     208500
1   2007        WD         Normal     181500
2   2008        WD         Normal     223500
3   2006        WD        Abnorml     140000
4   2008        WD         Normal     250000

[5 rows x 81 columns]
```

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

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             1460 non-null   int64  
 1   MSSubClass     1460 non-null   int64  
 2   MSZoning       1460 non-null   object 
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64  
 5   Street         1460 non-null   object 
 6   Alley          91 non-null     object 
 7   LotShape       1460 non-null   object 
 8   LandContour    1460 non-null   object 
 9   Utilities      1460 non-null   object 
 10  LotConfig      1460 non-null   object 
 11  LandSlope      1460 non-null   object 
 12  Neighborhood   1460 non-null   object 
 13  Condition1     1460 non-null   object 
 14  Condition2     1460 non-null   object 
 15  BldgType       1460 non-null   object 
 16  HouseStyle     1460 non-null   object 
 17  OverallQual    1460 non-null   int64  
 18  OverallCond    1460 non-null   int64  
 19  YearBuilt      1460 non-null   int64  
 20  YearRemodAdd   1460 non-null   int64  
 21  RoofStyle      1460 non-null   object 
 22  RoofMatl       1460 non-null   object 
 23  Exterior1st    1460 non-null   object 
 24  Exterior2nd    1460 non-null   object 
 25  MasVnrType     1452 non-null   object 
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object 
 28  ExterCond      1460 non-null   object 
 29  Foundation     1460 non-null   object 
 30  BsmtQual       1423 non-null   object 
 31  BsmtCond       1423 non-null   object 
 32  BsmtExposure   1422 non-null   object 
 33  BsmtFinType1   1423 non-null   object 
 34  BsmtFinSF1     1460 non-null   int64  
 35  BsmtFinType2   1422 non-null   object 
 36  BsmtFinSF2     1460 non-null   int64  
 37  BsmtUnfSF      1460 non-null   int64  
 38  TotalBsmtSF    1460 non-null   int64  
 39  Heating        1460 non-null   object 
 40  HeatingQC      1460 non-null   object 
 41  CentralAir     1460 non-null   object 
 42  Electrical     1459 non-null   object 
 43  1stFlrSF       1460 non-null   int64  
```

```
 44  2ndFlrSF       1460 non-null   int64
 45  LowQualFinSF   1460 non-null   int64
 46  GrLivArea      1460 non-null   int64
 47  BsmtFullBath   1460 non-null   int64
 48  BsmtHalfBath   1460 non-null   int64
 49  FullBath       1460 non-null   int64
 50  HalfBath       1460 non-null   int64
 51  BedroomAbvGr   1460 non-null   int64
 52  KitchenAbvGr   1460 non-null   int64
 53  KitchenQual    1460 non-null   object
 54  TotRmsAbvGrd   1460 non-null   int64
 55  Functional     1460 non-null   object
 56  Fireplaces     1460 non-null   int64
 57  FireplaceQu    770 non-null    object
 58  GarageType     1379 non-null   object
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

From these informations we can already see that some features won't be relevant in our exploratory analysis as there are too much missing values (such as Alley and PoolQC). Plus there is so much features to analyse that it may be better to concentrate on the ones which can give us real insights. Let's just remove Id and the features with 30% or less NaN values.

[4]:
```
# df.count() does not include NaN values
df2 = df[[column for column in df if df[column].count() / len(df) >= 0.3]]
del df2['Id']
```

```python
print("List of dropped columns:", end=" ")
for c in df.columns:
    if c not in df2.columns:
        print(c, end=", ")
print('\n')
df = df2
```

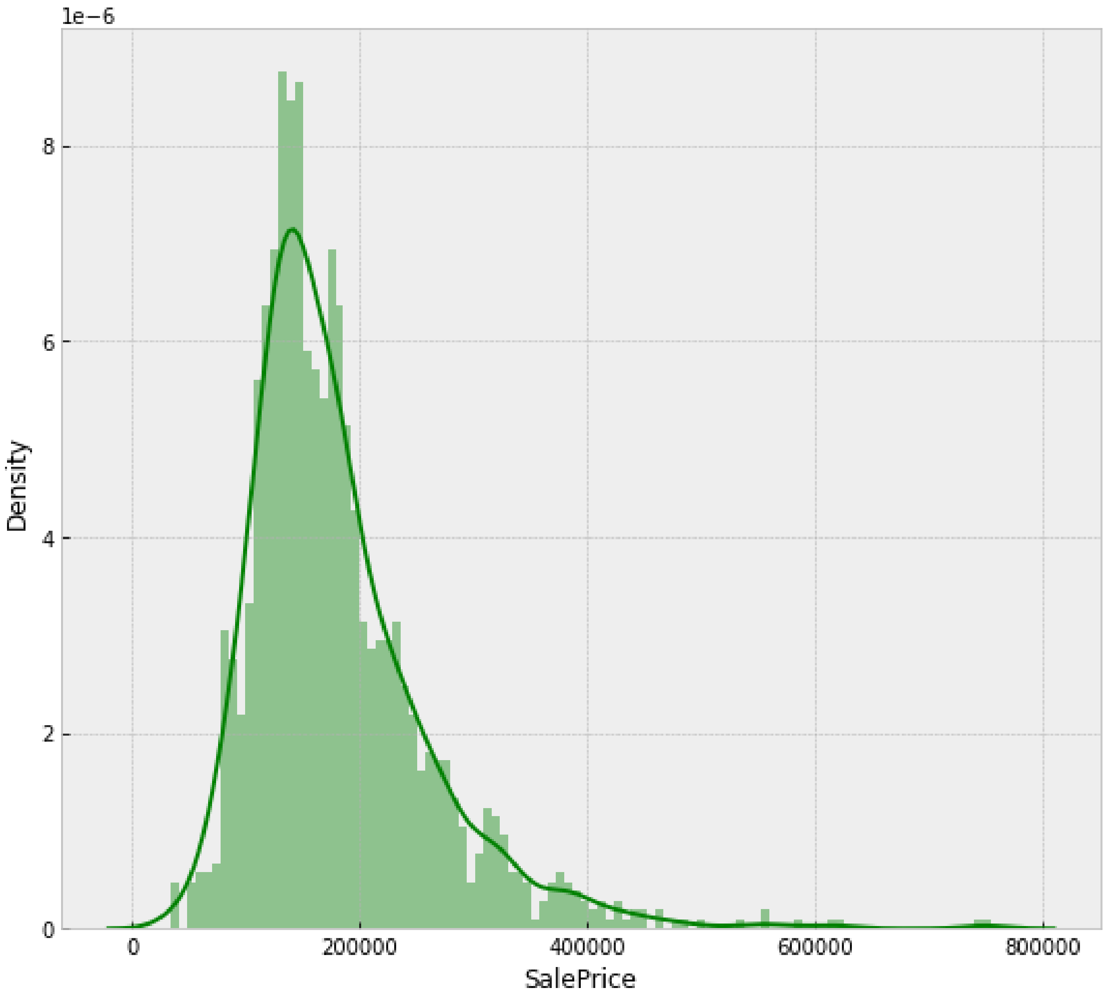
```
List of dropped columns: Id, Alley, PoolQC, Fence, MiscFeature,
```

[5]:

```python
# Now lets take a look at how the housing price is distributed
print(df['SalePrice'].describe())
plt.figure(figsize=(9, 8))
sns.distplot(df['SalePrice'], color='g', bins=100, hist_kws={'alpha': 0.4});
```

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

```
/home/sbweb/.local/lib/python3.10/site-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



With this information we can see that the prices are skewed right and some outliers lies above ~500,000. We will eventually want to get rid of the them to get a normal distribution of the independent variable (`SalePrice`) for machine learning.

1 Numerical data distribution

For this part lets look at the distribution of all of the features by ploting them

To do so lets first list all the types of our data from our dataset and take only the numerical ones:

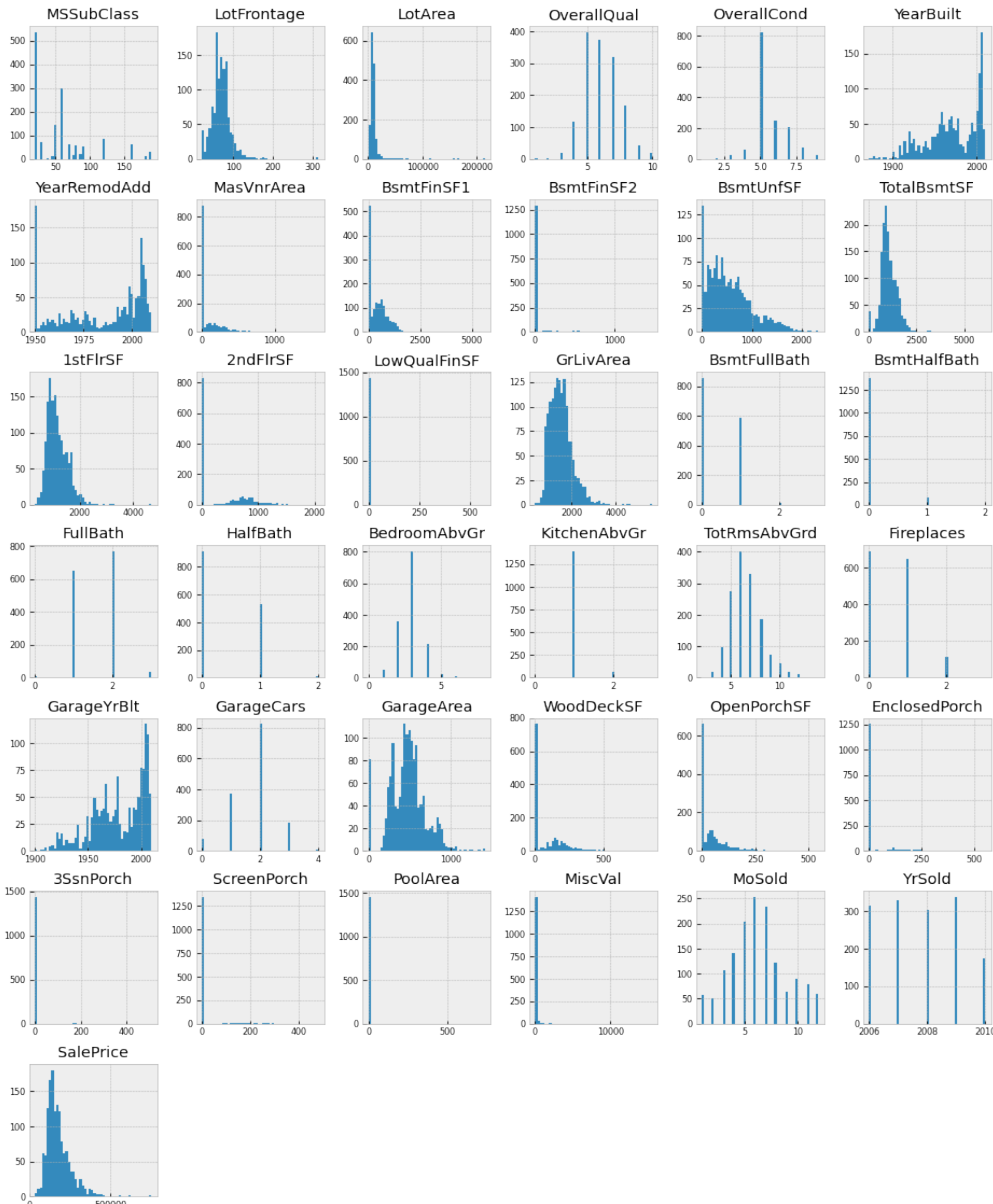
[6]:
```
list(set(df.dtypes.tolist()))
```

[6]:
```
[dtype('float64'), dtype('int64'), dtype('O')]
```

[7]:
```
df_num = df.select_dtypes(include = ['float64', 'int64'])
df_num.head()
```

[7]:
```
   MSSubClass  LotFrontage  LotArea  OverallQual  OverallCond  YearBuilt  \
0          60         65.0     8450            7            5       2003
1          20         80.0     9600            6            8       1976
2          60         68.0    11250            7            5       2001
3          70         60.0     9550            7            5       1915
4          60         84.0    14260            8            5       2000

   YearRemodAdd  MasVnrArea  BsmtFinSF1  BsmtFinSF2  …  WoodDeckSF  \
0          2003       196.0         706           0  …           0
1          1976         0.0         978           0  …         298
2          2002       162.0         486           0  …           0
3          1970         0.0         216           0  …           0
4          2000       350.0         655           0  …         192

   OpenPorchSF  EnclosedPorch  3SsnPorch  ScreenPorch  PoolArea  MiscVal  \
0           61              0          0            0         0        0
1            0              0          0            0         0        0
2           42              0          0            0         0        0
3           35            272          0            0         0        0
4           84              0          0            0         0        0

   MoSold  YrSold  SalePrice
0       2    2008     208500
1       5    2007     181500
2       9    2008     223500
3       2    2006     140000
4      12    2008     250000

[5 rows x 37 columns]
```

[8]:
```python
# plot them all
df_num.hist(figsize=(16, 20), bins=50, xlabelsize=8, ylabelsize=8); # ; avoid␣
↪having the matplotlib verbose informations
```



Features such as `1stFlrSF`, `TotalBsmtSF`, `LotFrontage`, `GrLiveArea`... seems to share a similar distribution to the one we have with `SalePrice`. Lets see if we can find new clues later.

2 Correlation

Now we'll try to find which features are strongly correlated with SalePrice. We'll store them in a var called golden_features_list. We'll reuse our df_num dataset to do so.

[9]:
```
df_num_corr = df_num.corr()['SalePrice'][:-1] # -1 because the latest row is SalePrice
golden_features_list = df_num_corr[abs(df_num_corr) > 0.5].sort_values(ascending=False)
print("There is {} strongly correlated values with SalePrice:\n{}".format(len(golden_features_list), golden_features_list))
```

```
There is 10 strongly correlated values with SalePrice:
OverallQual     0.790982
GrLivArea       0.708624
GarageCars      0.640409
GarageArea      0.623431
TotalBsmtSF     0.613581
1stFlrSF        0.605852
FullBath        0.560664
TotRmsAbvGrd    0.533723
YearBuilt       0.522897
YearRemodAdd    0.507101
Name: SalePrice, dtype: float64
```

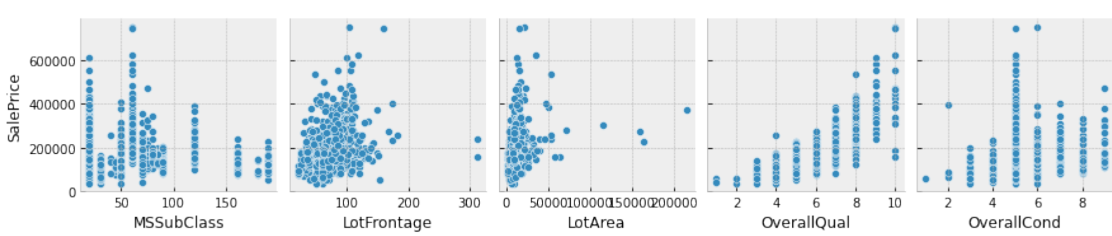
Perfect, we now have a list of strongly correlated values but this list is incomplete as we know that correlation is affected by outliers. So we could proceed as follow:

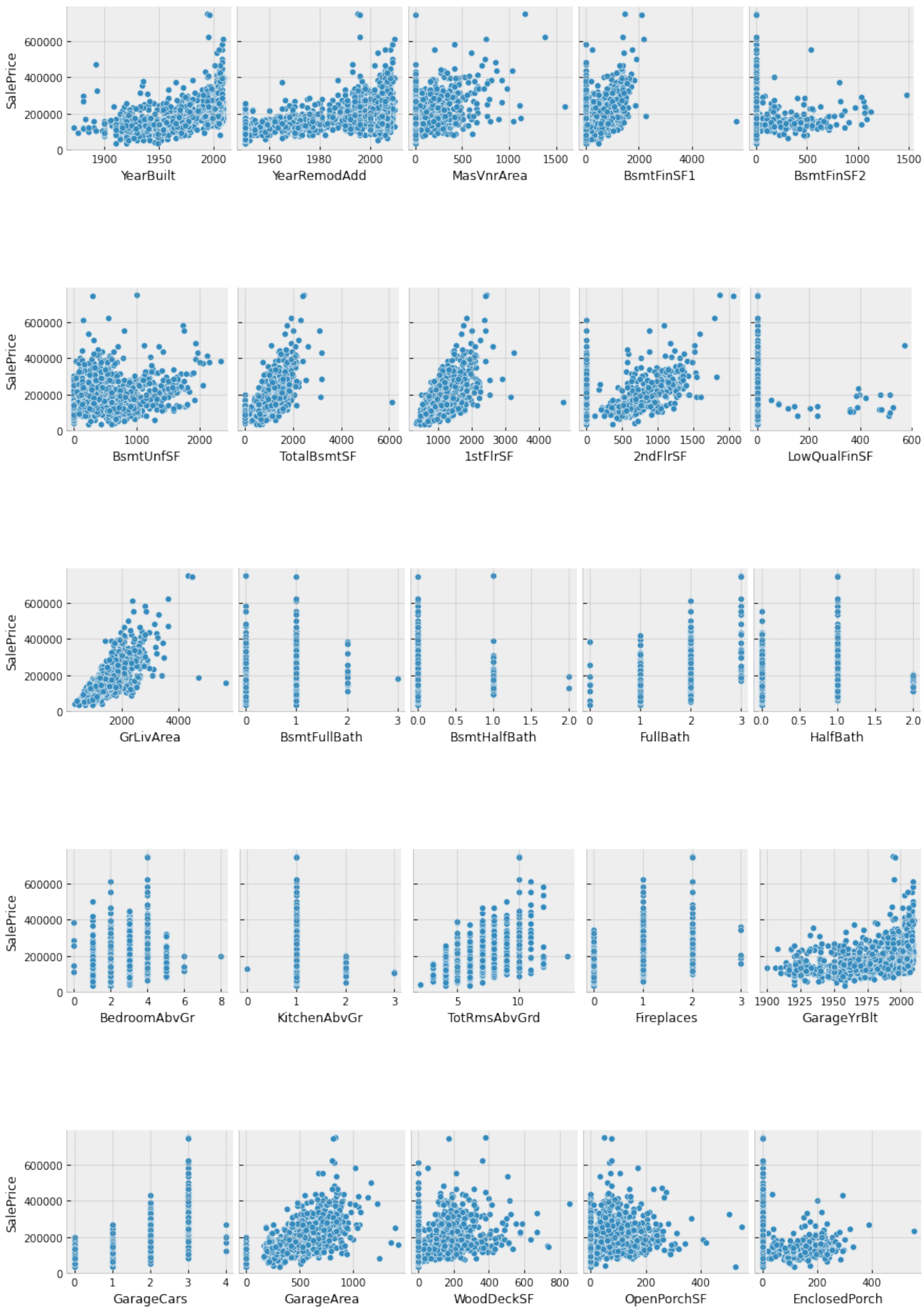
```
Plot the numerical features and see which ones have very few or explainable outliers
Remove the outliers from these features and see which one can have a good correlation without
```

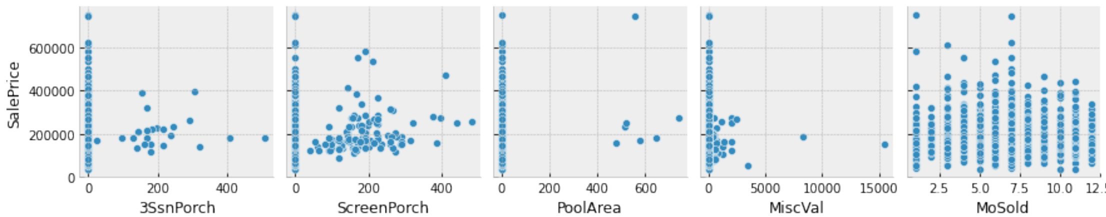
Btw, correlation by itself does not always explain the relationship between data so ploting them could even lead us to new insights and in the same manner, check that our correlated values have a linear relationship to the SalePrice.

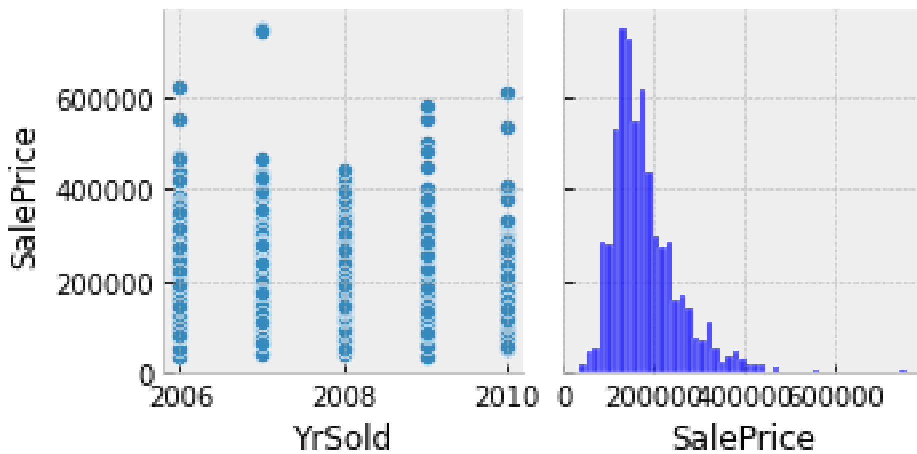
For example, relationships such as curvilinear relationship cannot be guessed just by looking at the correlation value so lets take the features we excluded from our correlation table and plot them to see if they show some kind of pattern.

[10]:
```
for i in range(0, len(df_num.columns), 5):
    sns.pairplot(data=df_num,
                x_vars=df_num.columns[i:i+5],
                y_vars=['SalePrice'])
```




SalePrice
YearBuilt
YearRemodAdd
MasVnrArea
BsmtFinSF1
BsmtFinSF2
BsmtUnfSF
TotalBsmtSF
1stFlrSF
2ndFlrSF
LowQualFinSF
GrLivArea
BsmtFullBath
BsmtHalfBath
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageYrBlt
GarageCars
GarageArea
WoodDeckSF
OpenPorchSF
EnclosedPorch





We can clearly identify some relationships. Most of them seems to have a linear relationship with the SalePrice and if we look closely at the data we can see that a lot of data points are located on x = 0 which may indicate the absence of such feature in the house.

Take OpenPorchSF, I doubt that all houses have a porch (mine doesn't for instance but I don't lose hope that one day… yeah one day…).

So now lets remove these 0 values and repeat the process of finding correlated values:

[11]:
```
import operator

individual_features_df = []
for i in range(0, len(df_num.columns) - 1): # -1 because the last column is␣
 ↪SalePrice
    tmpDf = df_num[[df_num.columns[i], 'SalePrice']]
    tmpDf = tmpDf[tmpDf[df_num.columns[i]] != 0]
    individual_features_df.append(tmpDf)

all_correlations = {feature.columns[0]: feature.corr()['SalePrice'][0] for␣
 ↪feature in individual_features_df}
```

```
all_correlations = sorted(all_correlations.items(), key=operator.itemgetter(1))
for (key, value) in all_correlations:
    print("{:>15}: {:>15}".format(key, value))
```

```
   KitchenAbvGr: -0.1392006921778576
       HalfBath: -0.08439171127179902
     MSSubClass: -0.08428413512659509
    OverallCond: -0.07785589404867797
         YrSold: -0.028922585168736813
   BsmtHalfBath: -0.02883456718548182
       PoolArea: -0.014091521506356765
   BsmtFullBath: 0.011439163340408606
         MoSold: 0.046432245223819446
      3SsnPorch: 0.06393243256889088
    OpenPorchSF: 0.08645298857147718
        MiscVal: 0.08896338917298921
     Fireplaces: 0.1216605842136391
      BsmtUnfSF: 0.16926100049514173
   BedroomAbvGr: 0.18093669310848806
     WoodDeckSF: 0.1937060123752066
     BsmtFinSF2: 0.19895609430836594
  EnclosedPorch: 0.24127883630117497
    ScreenPorch: 0.2554300795487841
        LotArea: 0.2638433538714051
   LowQualFinSF: 0.30007501655501323
    LotFrontage: 0.35179909657067737
     MasVnrArea: 0.43409021975689227
     BsmtFinSF1: 0.47169042652357296
    GarageYrBlt: 0.4863616774878596
   YearRemodAdd: 0.5071009671113866
      YearBuilt: 0.5228973328794967
   TotRmsAbvGrd: 0.5337231555820284
       FullBath: 0.5745626737760822
       1stFlrSF: 0.6058521846919153
     GarageArea: 0.6084052829168346
    TotalBsmtSF: 0.6096808188074374
     GarageCars: 0.6370954062078923
       2ndFlrSF: 0.6733048324568376
      GrLivArea: 0.7086244776126515
    OverallQual: 0.7909816005838053
```

Very interesting! We found another strongly correlated value by cleaning up the data a bit. Now our golden_features_list var looks like this:

[12]:
```
golden_features_list = [key for key, value in all_correlations if abs(value) >=␣
↪0.5]
print("There is {} strongly correlated values with SalePrice:\n{}".
↪format(len(golden_features_list), golden_features_list))
```

```
There is 11 strongly correlated values with SalePrice:
['YearRemodAdd', 'YearBuilt', 'TotRmsAbvGrd', 'FullBath', '1stFlrSF',
'GarageArea', 'TotalBsmtSF', 'GarageCars', '2ndFlrSF', 'GrLivArea',
'OverallQual']
```

We found strongly correlated predictors with `SalePrice`. Later with feature engineering we may add dummy values where value of a given feature > 0 would be 1 (precense of such feature) and 0 would be 0. For `2ndFlrSF` for example, we could create a dummy value for its precense or non-precense and finally sum it up to `1stFlrSF`.

2.0.1 Conclusion

By looking at correlation between numerical values we discovered 11 features which have a strong relationship to a house price. Besides correlation we didn't find any notable pattern on the datas which are not correlated.

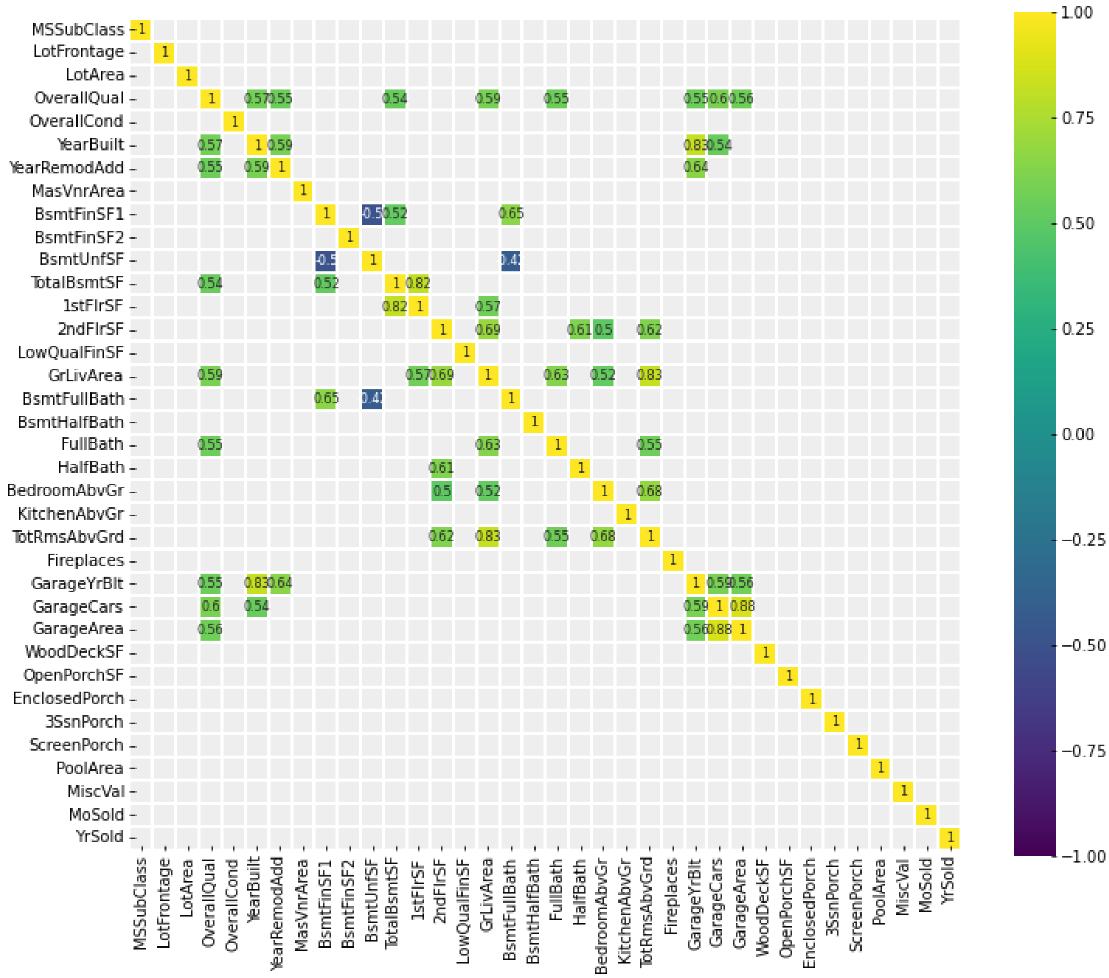
Notes:

```
There may be some patterns I wasn't able to identify due to my lack of expertise
Some values such as GarageCars -> SalePrice or Fireplaces -> SalePrice shows a particular patte
```

3 Feature to feature relationship

Trying to plot all the numerical features in a seaborn pairplot will take us too much time and will be hard to interpret. We can try to see if some variables are linked between each other and then explain their relation with common sense.

[13]:
```
corr = df_num.drop('SalePrice', axis=1).corr() # We already examined SalePrice␣
↪correlations
plt.figure(figsize=(12, 10))

sns.heatmap(corr[(corr >= 0.5) | (corr <= -0.4)],
            cmap='viridis', vmax=1.0, vmin=-1.0, linewidths=0.1,
            annot=True, annot_kws={"size": 8}, square=True);
```



A lot of features seems to be correlated between each other but some of them such as YearBuild/GarageYrBlt may just indicate a price inflation over the years. As for 1stFlrSF/TotalBsmtSF, it is normal that the more the 1st floor is large (considering many houses have only 1 floor), the more the total basement will be large.

Now for the ones which are less obvious we can see that:

`There is a strong negative correlation between BsmtUnfSF (Unfinished square feet of basement a`
`HalfBath/2ndFlrSF is interesting and may indicate that people gives an importance of not having`

There is of course a lot more to discover but I can't really explain the rest of the features except the most obvious ones.

We can conclude that, by essence, some of those features may be combined between each other in order to reduce the number of features (`1stFlrSF/TotalBsmtSF`, `GarageCars/GarageArea`) and others indicates that people expect multiples features to be packaged together.

4 Q -> Q (Quantitative to Quantitative relationship)

Let's now examine the quantitative features of our dataframe and how they relate to the SalePrice which is also quantitative (hence the relation Q -> Q). I will conduct this analysis with the help of the Q -> Q chapter of the Standford MOOC

Some of the features of our dataset are categorical. To separate the categorical from quantitative features lets refer ourselves to the data_description.txt file. According to this file we end up with the folowing columns:

[14]:
```
quantitative_features_list = ['LotFrontage', 'LotArea', 'MasVnrArea',␣
 ↪'BsmtFinSF1', 'BsmtFinSF2', 'TotalBsmtSF', '1stFlrSF',
    '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath',␣
 ↪'FullBath', 'HalfBath',
    'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageCars',␣
 ↪'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
    'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',␣
 ↪'SalePrice']
df_quantitative_values = df[quantitative_features_list]
df_quantitative_values.head()
```

[14]:
```
   LotFrontage  LotArea  MasVnrArea  BsmtFinSF1  BsmtFinSF2  TotalBsmtSF  \
0         65.0     8450       196.0         706           0          856
1         80.0     9600         0.0         978           0         1262
2         68.0    11250       162.0         486           0          920
3         60.0     9550         0.0         216           0          756
4         84.0    14260       350.0         655           0         1145

   1stFlrSF  2ndFlrSF  LowQualFinSF  GrLivArea  …  GarageCars  GarageArea  \
0       856       854             0       1710  …           2         548
1      1262         0             0       1262  …           2         460
2       920       866             0       1786  …           2         608
3       961       756             0       1717  …           3         642
4      1145      1053             0       2198  …           3         836

   WoodDeckSF  OpenPorchSF  EnclosedPorch  3SsnPorch  ScreenPorch  PoolArea  \
0           0           61              0          0            0         0
1         298            0              0          0            0         0
2           0           42              0          0            0         0
3           0           35            272          0            0         0
4         192           84              0          0            0         0

   MiscVal  SalePrice
0        0     208500
1        0     181500
2        0     223500
3        0     140000
4        0     250000
```

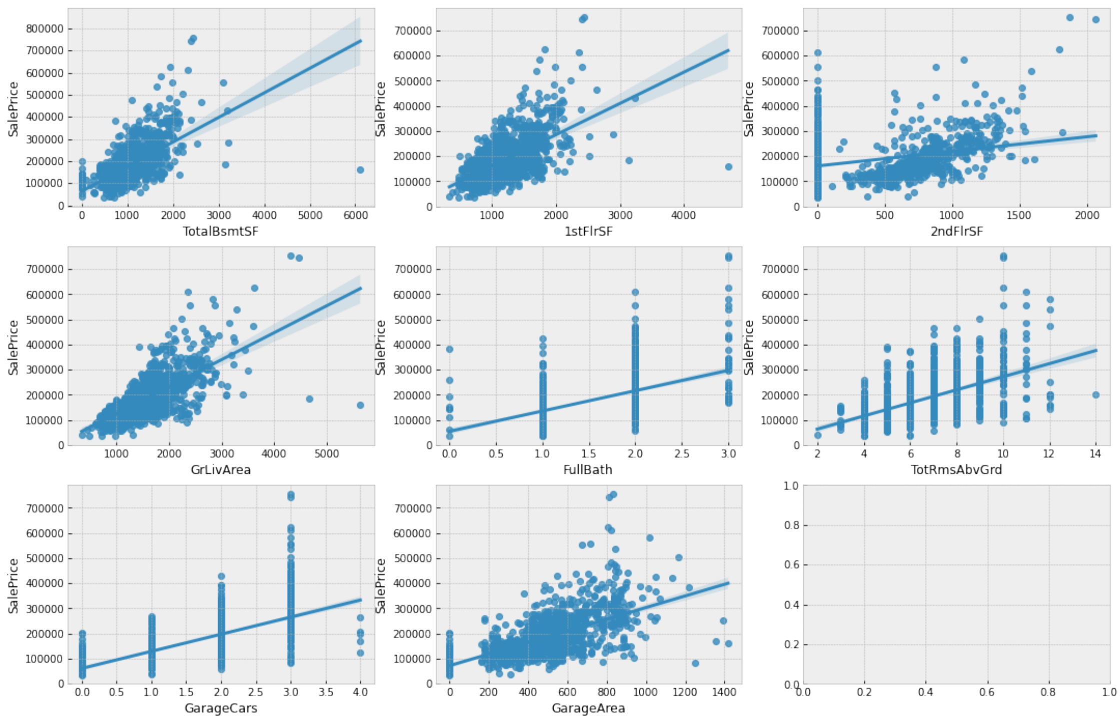
```
[5 rows x 28 columns]
```

Still, we have a lot of features to analyse here so let's take the strongly correlated quantitative features from this dataset and analyse them one by one

[15]:
```
features_to_analyse = [x for x in quantitative_features_list if x in␣
    ↪golden_features_list]
features_to_analyse.append('SalePrice')
features_to_analyse
```

[15]:
```
['TotalBsmtSF',
 '1stFlrSF',
 '2ndFlrSF',
 'GrLivArea',
 'FullBath',
 'TotRmsAbvGrd',
 'GarageCars',
 'GarageArea',
 'SalePrice']
```

[16]:
```
# Lets look at their disctibution

fig, ax = plt.subplots(round(len(features_to_analyse) / 3), 3, figsize = (18,␣
    ↪12))

for i, ax in enumerate(fig.axes):
    if i < len(features_to_analyse) - 1:
        sns.regplot(x=features_to_analyse[i],y='SalePrice',␣
    ↪data=df[features_to_analyse], ax=ax)
```



e can see that features such as TotalBsmtSF, 1stFlrSF, GrLivArea have a big spread but I cannot tell what insights this information gives us

5 C -> Q (Categorical to Quantitative relationship)

We will base this part of the exploration on the C -> Q chapter of the Standford MOOC

Lets get all the categorical features of our dataset and see if we can find some insight in them. Instead of opening back our data_description.txt file and checking which data are categorical, lets just remove quantitative_features_list from our entire dataframe.

[17]:
```
# quantitative_features_list[:-1] as the last column is SalePrice and we want␣
↪to keep it
categorical_features = [a for a in quantitative_features_list[:-1] + df.columns.
↪tolist() if (a not in quantitative_features_list[:-1]) or (a not in df.
↪columns.tolist())]
df_categ = df[categorical_features]
df_categ.head()
```

[17]:
```
   MSSubClass MSZoning Street LotShape LandContour Utilities LotConfig  \
0          60       RL   Pave      Reg         Lvl    AllPub    Inside
1          20       RL   Pave      Reg         Lvl    AllPub       FR2
2          60       RL   Pave      IR1         Lvl    AllPub    Inside
3          70       RL   Pave      IR1         Lvl    AllPub    Corner
```

```
4          60       RL    Pave       IR1         Lvl     AllPub       FR2

  LandSlope Neighborhood Condition1  … GarageYrBlt GarageFinish GarageQual  \
0       Gtl      CollgCr       Norm  …      2003.0          RFn         TA
1       Gtl      Veenker      Feedr  …      1976.0          RFn         TA
2       Gtl      CollgCr       Norm  …      2001.0          RFn         TA
3       Gtl      Crawfor       Norm  …      1998.0          Unf         TA
4       Gtl      NoRidge       Norm  …      2000.0          RFn         TA

   GarageCond  PavedDrive  MoSold  YrSold SaleType SaleCondition SalePrice
0          TA           Y       2    2008       WD        Normal    208500
1          TA           Y       5    2007       WD        Normal    181500
2          TA           Y       9    2008       WD        Normal    223500
3          TA           Y       2    2006       WD       Abnorml    140000
4          TA           Y      12    2008       WD        Normal    250000

[5 rows x 49 columns]
```

[18]:
```
# And don't forget the non-numerical features
df_not_num = df_categ.select_dtypes(include = ['O'])
print('There is {} non numerical features including:\n{}'.format(len(df_not_num.
    columns), df_not_num.columns.tolist()))
```

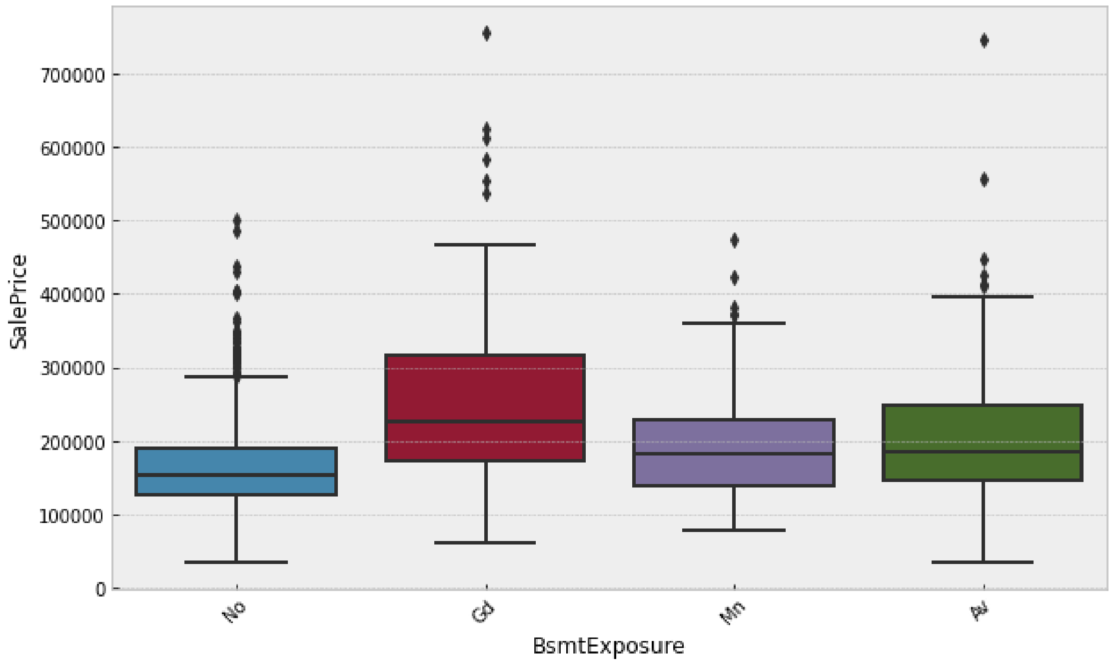
```
There is 39 non numerical features including:
['MSZoning', 'Street', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd',
'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond',
'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Heating', 'HeatingQC',
'CentralAir', 'Electrical', 'KitchenQual', 'Functional', 'FireplaceQu',
'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond', 'PavedDrive',
'SaleType', 'SaleCondition']
```

Looking at these features we can see that a lot of them are of the type `Object(O)`. In our data transformation notebook we could use Pandas categorical functions (equivalent to R's factor) to shape our data in a way that would be interpretable for our machine learning algorithm. `ExterQual` for instace could be transformed to an ordered categorical object.

Now lets plot some of them

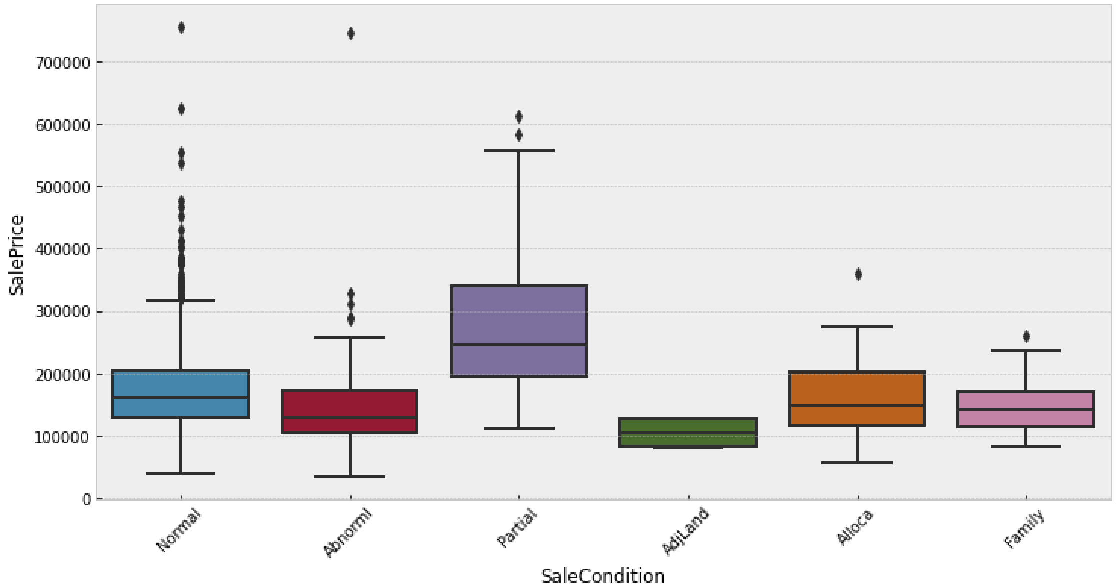
[19]:
```
plt.figure(figsize = (10, 6))
ax = sns.boxplot(x='BsmtExposure', y='SalePrice', data=df_categ)
plt.setp(ax.artists, alpha=.5, linewidth=2, edgecolor="k")
plt.xticks(rotation=45)
```

[19]:
```
(array([0, 1, 2, 3]),
 [Text(0, 0, 'No'), Text(1, 0, 'Gd'), Text(2, 0, 'Mn'), Text(3, 0, 'Av')])
```



[20]:
```
plt.figure(figsize = (12, 6))
ax = sns.boxplot(x='SaleCondition', y='SalePrice', data=df_categ)
plt.setp(ax.artists, alpha=.5, linewidth=2, edgecolor="k")
plt.xticks(rotation=45)
```

[20]:
```
(array([0, 1, 2, 3, 4, 5]),
 [Text(0, 0, 'Normal'),
  Text(1, 0, 'Abnorml'),
  Text(2, 0, 'Partial'),
  Text(3, 0, 'AdjLand'),
  Text(4, 0, 'Alloca'),
  Text(5, 0, 'Family')])
```



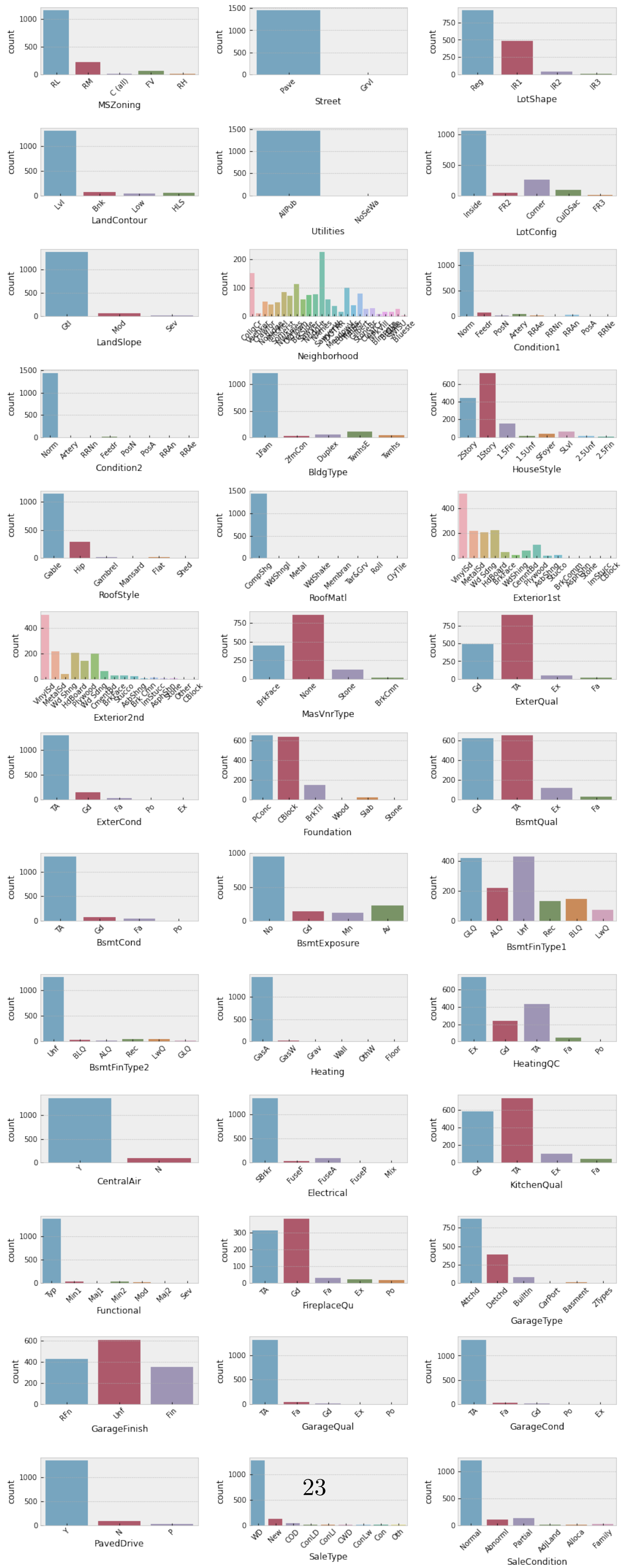
[21]:
```
# and finally look at their distribution



fig, axes = plt.subplots(round(len(df_not_num.columns) / 3), 3, figsize=(12,
 30))

for i, ax in enumerate(fig.axes):
    if i < len(df_not_num.columns):
        ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
        sns.countplot(x=df_not_num.columns[i], alpha=0.7, data=df_not_num,
 ax=ax)

fig.tight_layout()
```

```
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
```

```
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
```

```
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
```

```
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
/tmp/ipykernel_33718/3744514058.py:9: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
```


count
MSZoning: RL RM C (all) FV RH
Street: Pave Grvl
LotShape: Reg IR1 IR2 IR3
LandContour: Lvl Bnk Low HLS
Utilities: AllPub NoSeWa
LotConfig: Inside FR2 Corner CulDSac FR3
LandSlope: Gtl Mod Sev
Neighborhood
Condition1: Norm Feedr PosN Artery RRAe RRNn RRAn PosA RRNe
Condition2: Norm Artery RRNn Feedr PosN PosA RRAn RRAe
BldgType: 1Fam 2fmCon Duplex TwnhsE Twnhs
HouseStyle: 2Story 1Story 1.5Fin 1.5Unf SFoyer SLvl 2.5Unf 2.5Fin
RoofStyle: Gable Hip Gambrel Mansard Flat Shed
RoofMatl: CompShg WdShngl Metal WdShake Membran Tar&Grv Roll ClyTile
Exterior1st
Exterior2nd
MasVnrType: BrkFace None Stone BrkCmn
ExterQual: Gd TA Ex Fa
ExterCond: TA Gd Fa Po Ex
Foundation: PConc CBlock BrkTil Wood Slab Stone
BsmtQual: Gd TA Ex Fa
BsmtCond: TA Gd Fa Po
BsmtExposure: No Gd Mn Av
BsmtFinType1: GLQ ALQ Unf Rec BLQ LwQ
BsmtFinType2: Unf BLQ ALQ Rec LwQ GLQ
Heating: GasA GasW Grav Wall OthW Floor
HeatingQC: Ex Gd TA Fa Po
CentralAir: Y N
Electrical: SBrkr FuseF FuseA FuseP Mix
KitchenQual: Gd TA Ex Fa
Functional: Typ Min1 Maj1 Min2 Mod Maj2 Sev
FireplaceQu: TA Gd Fa Ex Po
GarageType: Attchd Detchd BuiltIn CarPort Basment 2Types
GarageFinish: RFn Unf Fin
GarageQual: TA Fa Gd Ex Po
GarageCond: TA Fa Gd Po Ex
PavedDrive: Y N P
SaleType: WD New COD ConLD ConLI CWD ConLw Con Oth
SaleCondition: Normal Abnorml Partial AdjLand Alloca Family

We can see that some categories are predominant for some features such as `Utilities`, `Heating`, `GarageCond`, `Functional`... These features may not be relevant for our predictive model

[ ]:

```

```