A Simple Tutorial on Exploratory Data Analysis

What is Exploratory Data Analysis (EDA)?

- How to ensure you are ready to use machine learning algorithms in a project?
- How to choose the most suitable algorithms for your data set?
- How to define the feature variables that can potentially be used for machine learning?

Exploratory Data Analysis (EDA) helps to answer all these questions, ensuring the best outcomes for the project. It is an approach for summarizing, visualizing, and becoming intimately familiar with the important characteristics of a data set.

Value of Exploratory Data Analysis

Exploratory Data Analysis is valuable to data science projects since it allows to get closer to the certainty that the future results will be valid, correctly interpreted, and applicable to the desired business contexts. Such level of certainty can be achieved only after raw data is validated and checked for anomalies, ensuring that the data set was collected without errors. EDA also helps to find insights that were not evident or worth investigating to business stakeholders and data scientists but can be very informative about a particular business.

EDA is performed in order to define and refine the selection of feature variables that will be used for machine learning. Once data scientists become familiar with the data set, they often have to return to feature engineering step, since the initial features may turn out not to be serving their intended purpose. Once the EDA stage is complete, data scientists get a firm feature set they need for supervised and unsupervised machine learning.

Methods of Exploratory Data Analysis

It is always better to explore each data set using multiple exploratory techniques and compare the results. Once the data set is fully understood, it is quite possible that data scientist will have to go back to data collection and cleansing phases in order to transform the data set according to the desired business outcomes. The goal of this step is to become confident that the data set is ready to be used in a machine learning algorithm.

Exploratory Data Analysis is majorly performed using the following methods:

- Univariate visualization—provides summary statistics for each field in the raw data set
- Bivariate visualization—is performed to find the relationship between each variable in the dataset and the target variable of interest
- Multivariate visualization—is performed to understand interactions between different fields in the dataset
- Dimensionality reduction—helps to understand the fields in the data that account for the most variance between observations and allow for the processing of a reduced volume of data. Through these methods, the data scientist validates assumptions and identifies patterns that will allow for the understanding of the problem and model selection and validates that the data has been generated in the way it was expected to. So, value distribution of each field is checked, a number of missing values is defined, and the possible ways of replacing them are found.

Additional benefits Exploratory Data Analysis brings to projects Another side benefit of EDA is that it allows to specify or even define the questions you are trying to get the answer to from your data. Companies, that are only starting to leverage Data Science and AI technologies, often face the situation when they realize, that they have a lot of data and no ideas of what value that data can bring to their business decision making.

However, the questions always come first in data analysis. It doesn't matter how much data company has, how many tools they have available, whether the data is historical or real time unless business stakeholders have the questions they are trying to solve with their data. EDA can help such companies to start formalizing the right questions, since with wrong questions you get the wrong answers, and take the wrong decisions.

Why skipping Exploratory Data Analysis is a bad idea?

In a hurry to get to the machine learning stage or simply impress business stakeholders very fast, data scientists tend to either entirely skip the exploratory process or do a very shallow work. It is a very serious and, sadly, common mistake of amateur data science consulting "professionals".

Such inconsiderate behavior can lead to skewed data, with outliers and too many missing values and, therefore, some sad outcomes for the project:

- generating inaccurate models;
- generating accurate models on the wrong data;
- choosing the wrong variables for the model;
- inefficient use of the resources, including the rebuilding of the model.

Exploratory Data Analysis (EDA) is used on the one hand to answer questions, test business assumptions, generate hypotheses for further analysis. On the other hand, you can also use it to prepare the data for modeling.

The thing that these two probably have in common is a good knowledge of your data to either get the answers that you need or to develop an intuition for interpreting the results of future modeling.

There are a lot of ways to reach these goals as follows:

1. Import the data
2. Get a feel of the data ,describe the data,look at a sample of data like first and last rows
3. Take a deeper look into the data by querying or indexing the data

4. Identify features of interest
5. Recognise the challenges posed by data - missing values, outliers
6. Discover patterns in the data

One of the important things about EDA is Data profiling.

Data profiling is concerned with summarizing your dataset through descriptive statistics. You want to use a variety of measurements to better understand your dataset. The goal of data profiling is to have a solid understanding of your data so you can afterwards start querying and visualizing your data in various ways. However, this doesn't mean that you don't have to iterate: exactly because data profiling is concerned with summarizing your dataset, it is frequently used to assess the data quality. Depending on the result of the data profiling, you might decide to correct, discard or handle your data differently.

Key Concepts of Exploratory Data Analysis

- **2 types of Data Analysis**
  - Confirmatory Data Analysis
  - Exploratory Data Analysis
- **4 Objectives of EDA**
  - Discover Patterns
  - Spot Anomalies
  - Frame Hypothesis
  - Check Assumptions
- **2 methods for exploration**
  - Univariate Analysis
  - Bivariate Analysis
- **Stuff done during EDA**
  - Trends
  - Distribution
  - Mean
  - Median
  - Outlier
  - Spread measurement (SD)
  - Correlations
  - Hypothesis testing
  - Visual Exploration

Overview

This is an exploratory data analysis on the House Prices Kaggle Competition found at

https://www.kaggle.com/c/house-prices-advanced-regression-techniques (https://www.kaggle.com/c/house-prices-advanced-regression-techniques)

Description

Ask a home buyer to describe their dream house, and they probably won't begin with the height of the basement ceiling or the proximity to an east-west railroad. But this playground competition's dataset proves that much more influences price negotiations than the number of bedrooms or a white-picket fence.

With 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa, this competition challenges you to predict the final price of each home.

There are 1460 instances of training data and 1460 of test data. Total number of attributes equals 81, of which 36 are numerical, 43 are categorical + Id and SalePrice.

Numerical Features: 1stFlrSF, 2ndFlrSF, 3SsnPorch, BedroomAbvGr, BsmtFinSF1, BsmtFinSF2, BsmtFullBath, BsmtHalfBath, BsmtUnfSF, EnclosedPorch, Fireplaces, FullBath, GarageArea, GarageCars, GarageYrBlt, GrLivArea, HalfBath, KitchenAbvGr, LotArea, LotFrontage, LowQualFinSF, MSSubClass, MasVnrArea, MiscVal, MoSold, OpenPorchSF, OverallCond, OverallQual, PoolArea, ScreenPorch, TotRmsAbvGrd, TotalBsmtSF, WoodDeckSF, YearBuilt, YearRemodAdd, YrSold

Categorical Features: Alley, BldgType, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2, BsmtQual, CentralAir, Condition1, Condition2, Electrical, ExterCond, ExterQual, Exterior1st, Exterior2nd, Fence, FireplaceQu, Foundation, Functional, GarageCond, GarageFinish, GarageQual, GarageType, Heating, HeatingQC, HouseStyle, KitchenQual, LandContour, LandSlope, LotConfig, LotShape, MSZoning, MasVnrType, MiscFeature, Neighborhood, PavedDrive, PoolQC, RoofMatl, RoofStyle, SaleCondition, SaleType, Street, Utilitif

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import scipy.stats as st
from sklearn import ensemble, tree, linear_model
import missingno as msno
```

To start exploring your data, you'll need to start by actually loading in your data. You'll probably know this already, but thanks to the Pandas library, this becomes an easy task: you import the package as pd, following the convention, and you use the read_csv() function, to which you pass the URL in which the data can be found and a header argument. This last argument is one that you can use to make sure that your data is read in correctly: the first row of your data won't be interpreted as the column names of your DataFrame.

Alternatively, there are also other arguments that you can specify to ensure that your data is read in correctly: you can specify the delimiter to use with the sep or delimiter arguments, the column names to use with names or the column to use as the row labels for the resulting DataFrame with index_col.

In [2]:

```
train = pd.read_csv('../input/train.csv')
test = pd.read_csv('../input/test.csv')
```

One of the most elementary steps to do this is by getting a basic description of your data. A basic description of your data is indeed a very broad term: you can interpret it as a quick and dirty way to get some information on your data, as a way of getting some simple, easy-to-understand information on your data, to get a basic feel for your data. We can use the describe() function to get various summary statistics that exclude NaN values.

In [3]:

```
train.describe()
```

Out[3]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	146
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	197
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	3
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	187
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	195
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	197
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	200
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	201

Now that you have got a general idea about your data set, it's also a good idea to take a closer look at the data itself. With the help of the head() and tail() functions of the Pandas library, you can easily check out the first and last lines of your DataFrame, respectively.

Let us look at some sample data

In [4]:

```
train.head()
```

Out[4]:

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

In [5]:

```
train.tail()
```

Out[5]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandCo
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	

In [6]:

```
train.shape , test.shape
```

Out[6]:

```
((1460, 81), (1459, 80))
```

Let us examine numerical features in the train dataset

In [7]:

```
numeric_features = train.select_dtypes(include=[np.number])
numeric_features.columns
```

Out[7]:

```
Index(['Id', 'MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual',
       'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinS
F1',
       'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullB
ath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd',
       'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckS
F',
       'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolAr
ea',
       'MiscVal', 'MoSold', 'YrSold', 'SalePrice'],
      dtype='object')
```

In [8]:

```
numeric_features.head()
```

Out[8]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAc
0	1	60	65.0	8450	7	5	2003	200
1	2	20	80.0	9600	6	8	1976	197
2	3	60	68.0	11250	7	5	2001	200
3	4	70	60.0	9550	7	5	1915	197
4	5	60	84.0	14260	8	5	2000	200

Temporal Variables(Eg: Datetime Variables)

From the Dataset we have 4 year variables. We have extract information from the datetime variables like no of years or no of days.

In [9]:

```
# list of variables that contain year information
year_feature = [feature for feature in numeric_features if 'Yr' in feature or 'Year' in
feature]

year_feature
```

Out[9]:

```
['YearBuilt', 'YearRemodAdd', 'GarageYrBlt', 'YrSold']
```

In [10]:

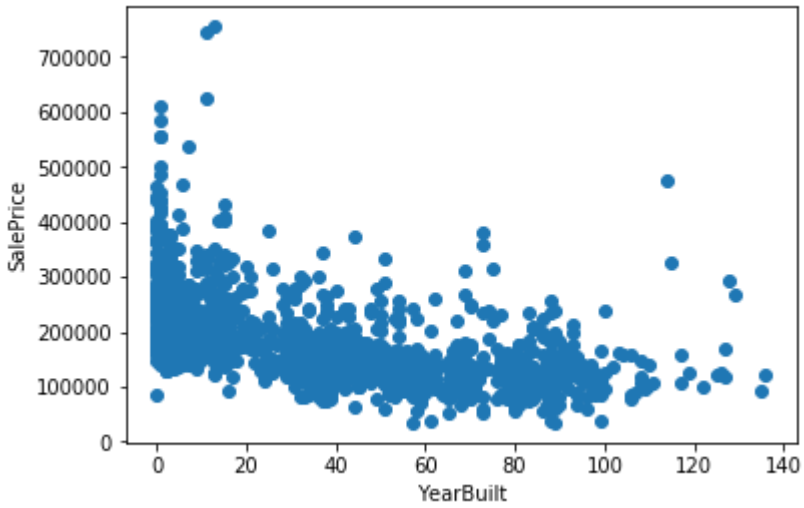
```
# Let us explore the contents of temporal  variables
for feature in year_feature:
    print(feature, train[feature].unique())
```

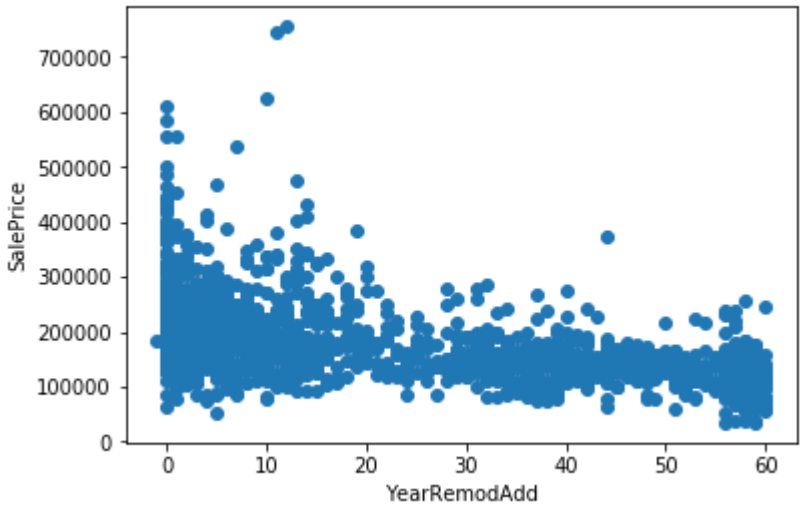
```
YearBuilt [2003 1976 2001 1915 2000 1993 2004 1973 1931 1939 1965 2005 196
2 2006
 1960 1929 1970 1967 1958 1930 2002 1968 2007 1951 1957 1927 1920 1966
 1959 1994 1954 1953 1955 1983 1975 1997 1934 1963 1981 1964 1999 1972
 1921 1945 1982 1998 1956 1948 1910 1995 1991 2009 1950 1961 1977 1985
 1979 1885 1919 1990 1969 1935 1988 1971 1952 1936 1923 1924 1984 1926
 1940 1941 1987 1986 2008 1908 1892 1916 1932 1918 1912 1947 1925 1900
 1980 1989 1992 1949 1880 1928 1978 1922 1996 2010 1946 1913 1937 1942
 1938 1974 1893 1914 1906 1890 1898 1904 1882 1875 1911 1917 1872 1905]
YearRemodAdd [2003 1976 2002 1970 2000 1995 2005 1973 1950 1965 2006 1962
2007 1960
 2001 1967 2004 2008 1997 1959 1990 1955 1983 1980 1966 1963 1987 1964
 1972 1996 1998 1989 1953 1956 1968 1981 1992 2009 1982 1961 1993 1999
 1985 1979 1977 1969 1958 1991 1971 1952 1975 2010 1984 1986 1994 1988
 1954 1957 1951 1978 1974]
GarageYrBlt [2003. 1976. 2001. 1998. 2000. 1993. 2004. 1973. 1931. 1939. 1
965. 2005.
 1962. 2006. 1960. 1991. 1970. 1967. 1958. 1930. 2002. 1968. 2007. 2008.
 1957. 1920. 1966. 1959. 1995. 1954. 1953.   nan 1983. 1977. 1997. 1985.
 1963. 1981. 1964. 1999. 1935. 1990. 1945. 1987. 1989. 1915. 1956. 1948.
 1974. 2009. 1950. 1961. 1921. 1900. 1979. 1951. 1969. 1936. 1975. 1971.
 1923. 1984. 1926. 1955. 1986. 1988. 1916. 1932. 1972. 1918. 1980. 1924.
 1996. 1940. 1949. 1994. 1910. 1978. 1982. 1992. 1925. 1941. 2010. 1927.
 1947. 1937. 1942. 1938. 1952. 1928. 1922. 1934. 1906. 1914. 1946. 1908.
 1929. 1933.]
YrSold [2008 2007 2006 2009 2010]
```

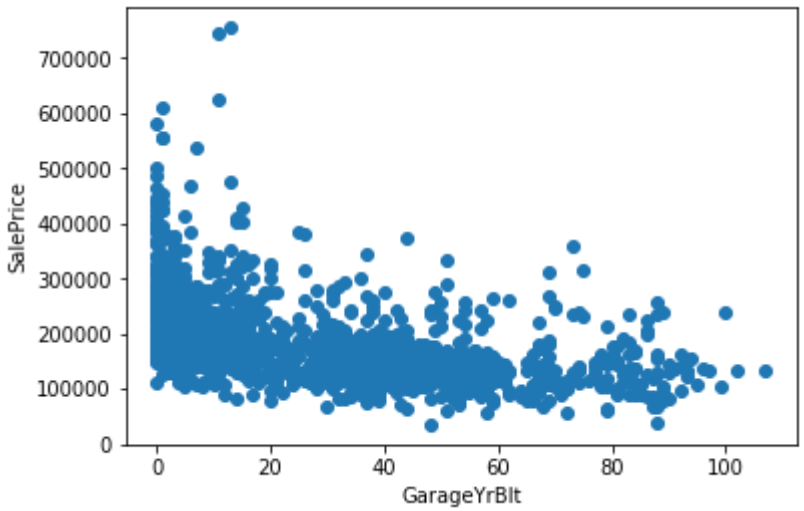
We will compare the difference between All **years** feature with **SalePrice**

In [11]:

```
for feature in year_feature:
    if feature!='YrSold':
        data=train.copy()
        ## We will capture the difference between year variable and year the house was
 sold for
        data[feature]=data['YrSold']-data[feature]
        plt.scatter(data[feature],data['SalePrice'])
        plt.xlabel(feature)
        plt.ylabel('SalePrice')
        plt.show()
```


SalePrice
YearBuilt
0
100000
200000
300000
400000
500000
600000
700000
0
20
40
60
80
100
120
140


SalePrice
YearRemodAdd
0
100000
200000
300000
400000
500000
600000
700000
0
10
20
30
40
50
60


SalePrice
GarageYrBlt
0
100000
200000
300000
400000
500000
600000
700000
0
20
40
60
80
100

Numerical variables Types

1. Discrete Variables

In [12]:

```
discrete_feature=[feature for feature in numeric_features if len(train[feature].unique
())<25 and feature not in year_feature+['Id']]
print("Discrete Variables Count: {}".format(len(discrete_feature)))
```

```
Discrete Variables Count: 17
```

In [13]:

```
train[discrete_feature].head()
```

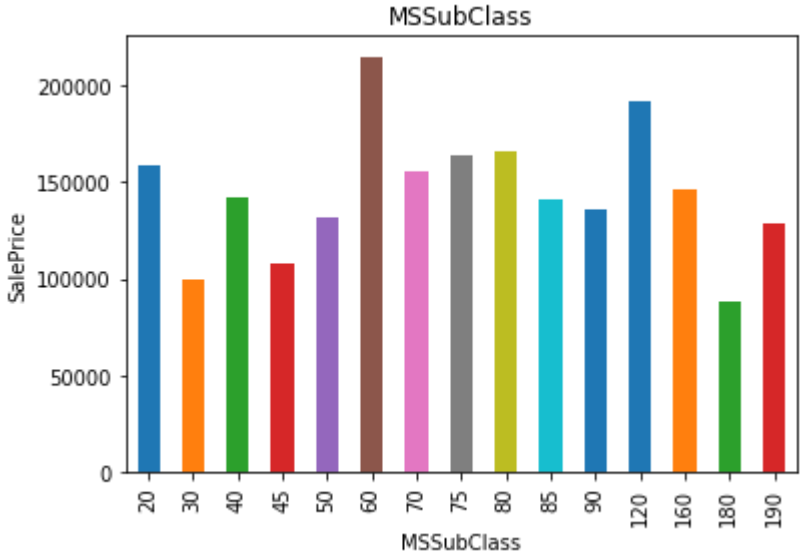
Out[13]:

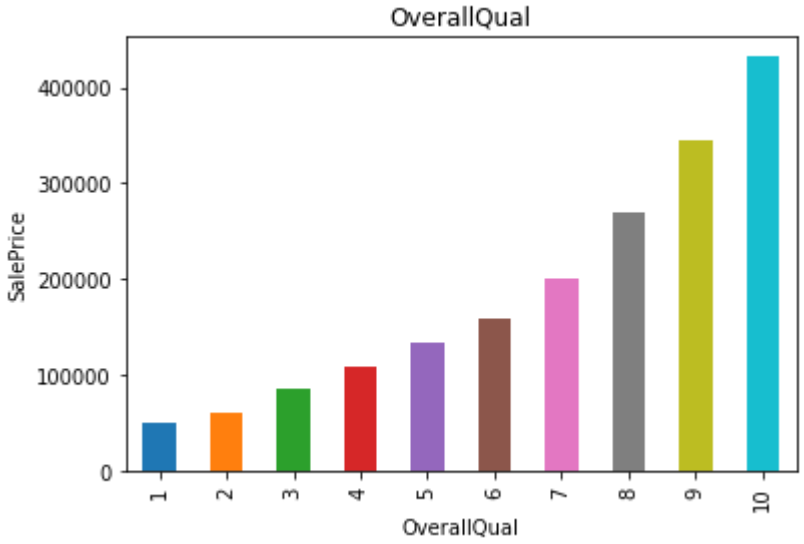
	MSSubClass	OverallQual	OverallCond	LowQualFinSF	BsmtFullBath	BsmtHalfBath	FullB
0	60	7	5	0	1	0	
1	20	6	8	0	0	1	
2	60	7	5	0	1	0	
3	70	7	5	0	1	0	
4	60	8	5	0	1	0	

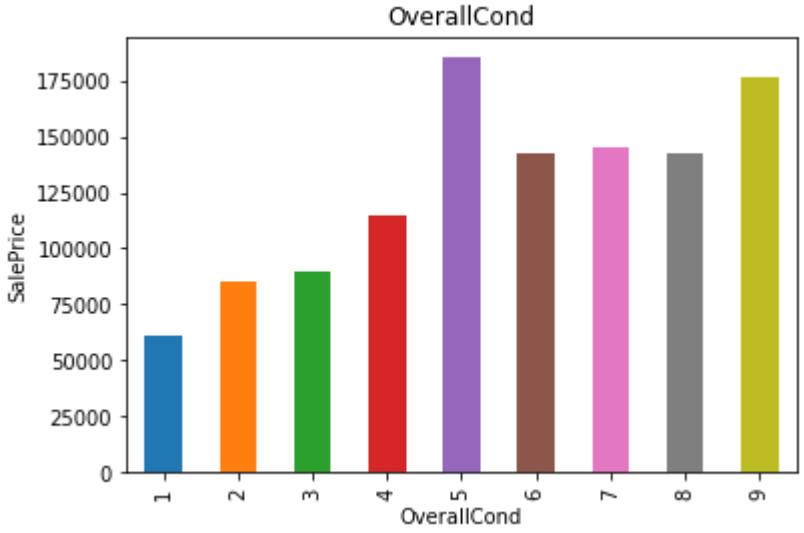
Now let us find the relationship between these discrete features and Sale Price

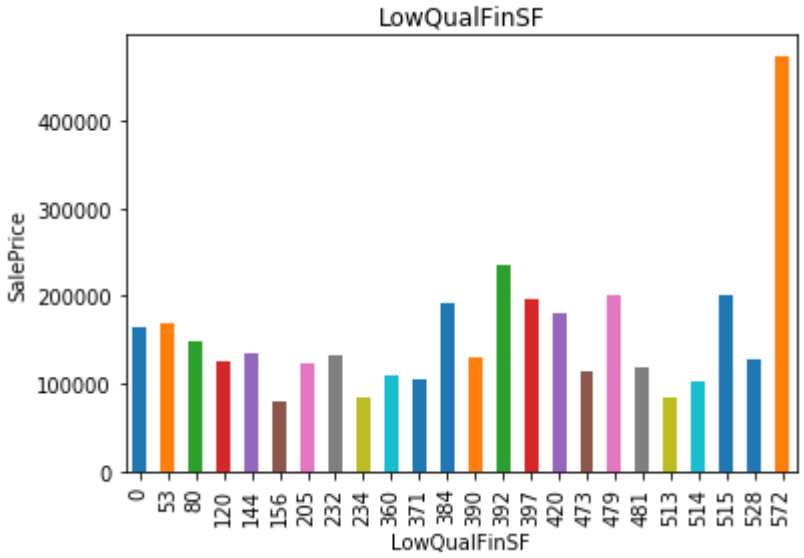
In [14]:

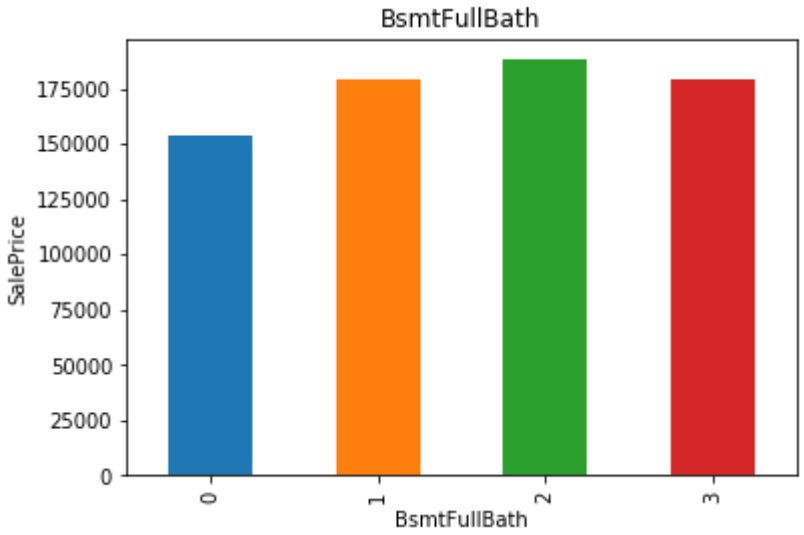
```
for feature in discrete_feature:
    data=train.copy()
    data.groupby(feature)['SalePrice'].median().plot.bar()
    plt.xlabel(feature)
    plt.ylabel('SalePrice')
    plt.title(feature)
    plt.show()
```

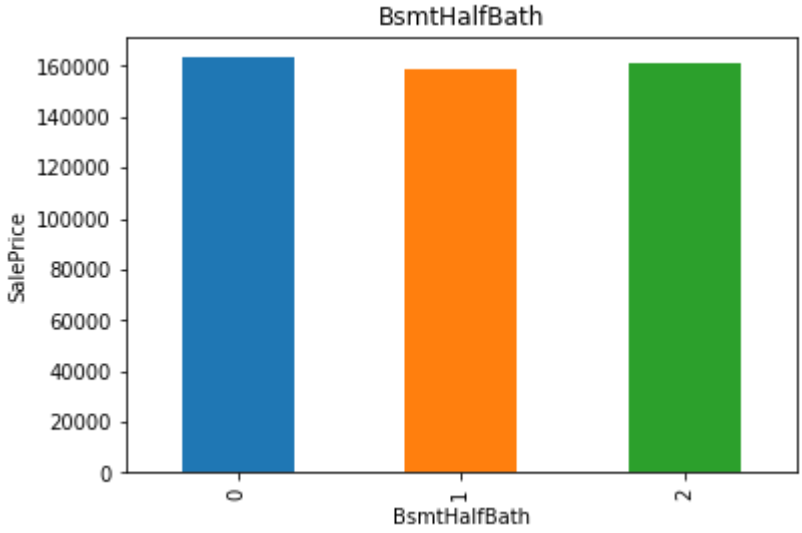

MSSubClass
SalePrice
0
50000
100000
150000
200000
20
30
40
45
50
60
70
75
80
85
90
120
160
180
190
MSSubClass

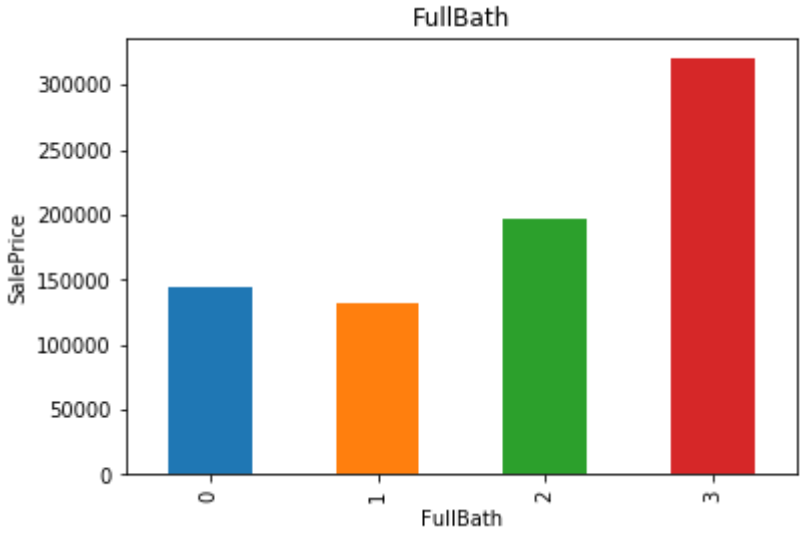

OverallQual
SalePrice
0
100000
200000
300000
400000
1
2
3
4
5
6
7
8
9
10
OverallQual

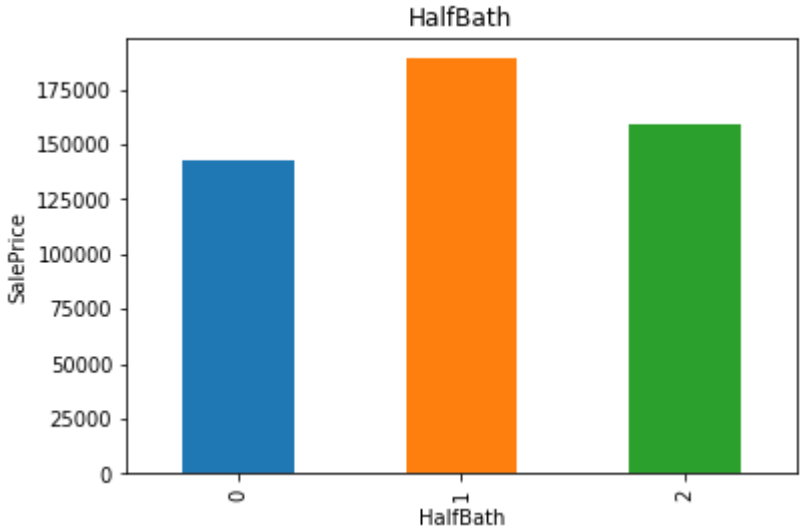

OverallCond
SalePrice
0
25000
50000
75000
100000
125000
150000
175000
1
2
3
4
5
6
7
8
9
OverallCond

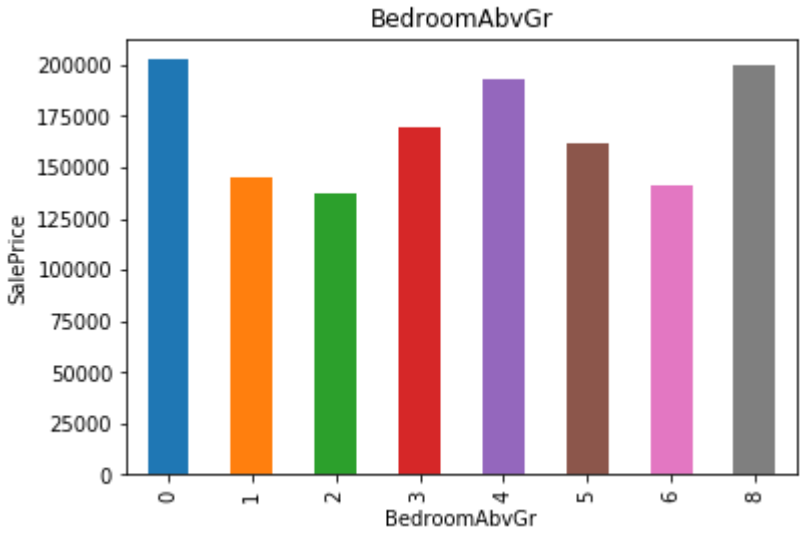

LowQualFinSF
SalePrice
0
100000
200000
300000
400000
0
53
80
120
144
156
205
232
234
360
371
384
390
392
397
420
473
479
481
513
514
515
528
572
LowQualFinSF

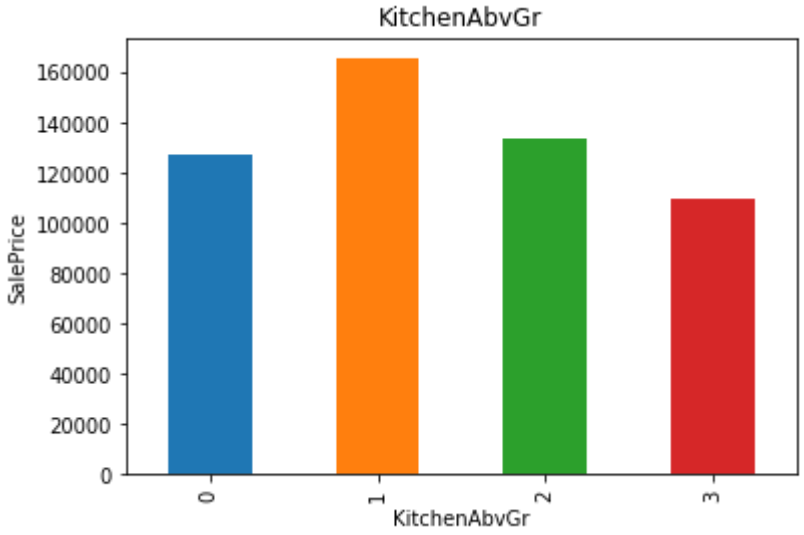

BsmtFullBath
SalePrice
0
25000
50000
75000
100000
125000
150000
175000
0
1
2
3
BsmtFullBath

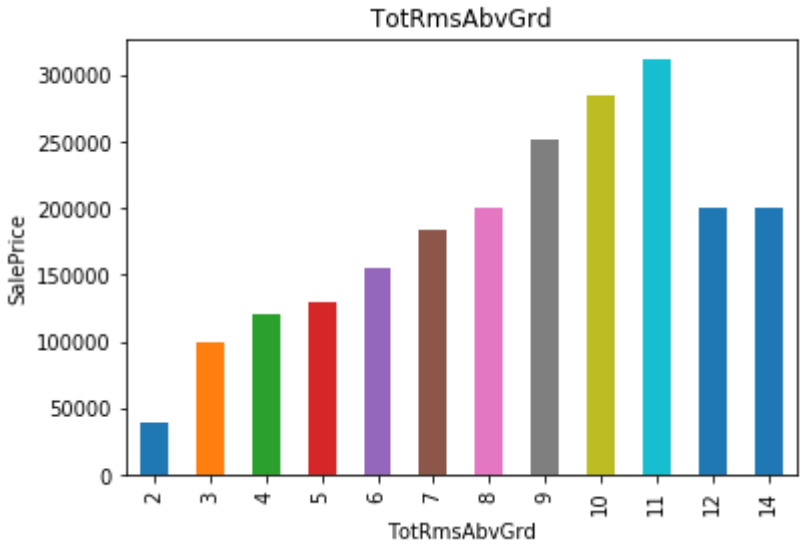

BsmtHalfBath
SalePrice
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
2
BsmtHalfBath

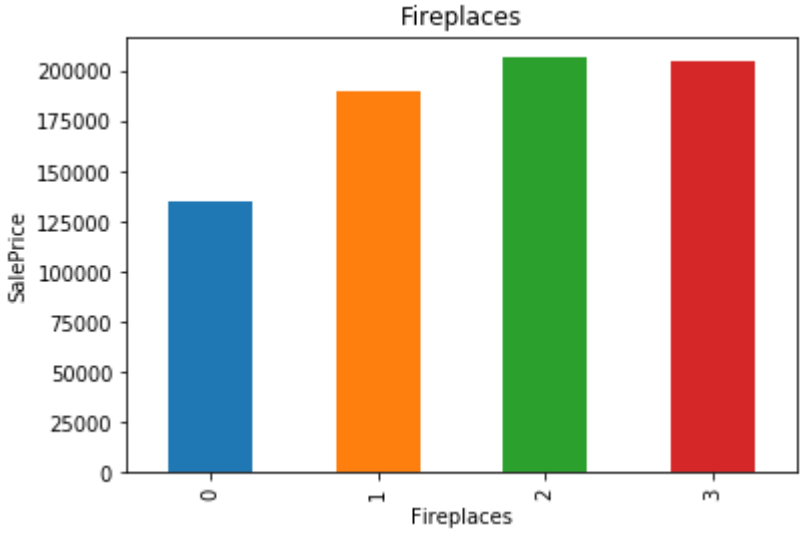

FullBath
SalePrice
300000
250000
200000
150000
100000
50000
0
0
1
2
3
FullBath

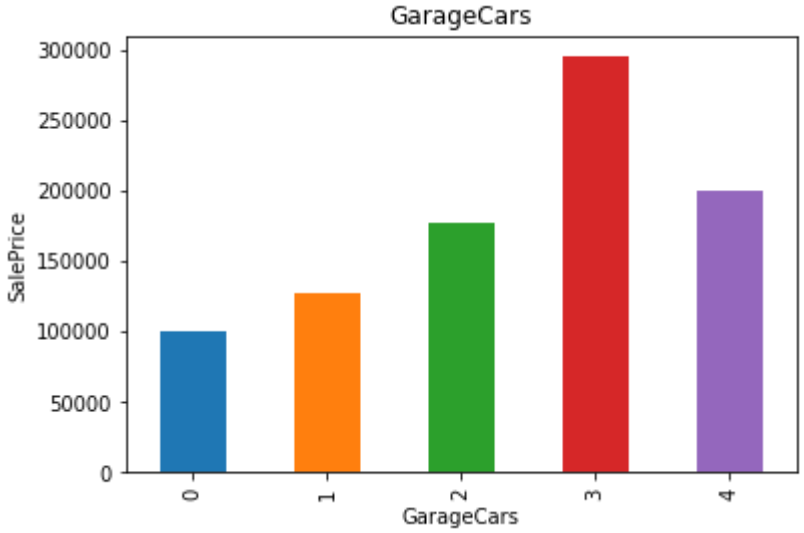

HalfBath
SalePrice
175000
150000
125000
100000
75000
50000
25000
0
0
1
2
HalfBath

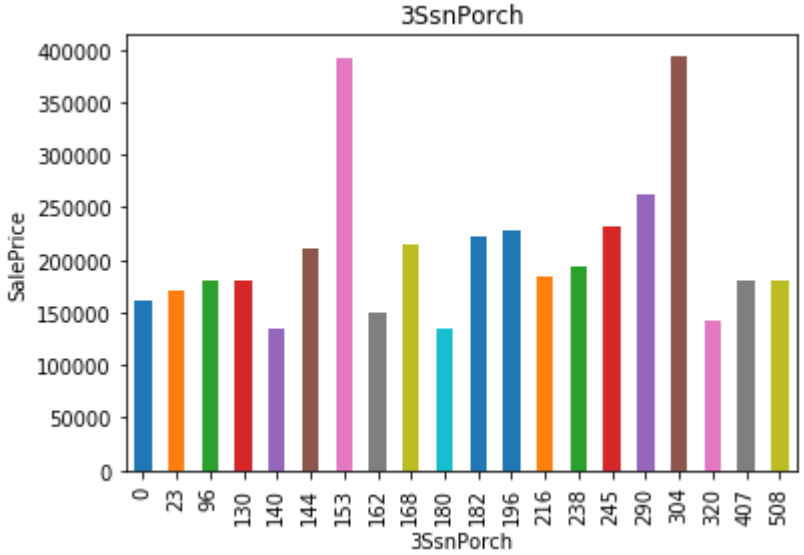

BedroomAbvGr
SalePrice
200000
175000
150000
125000
100000
75000
50000
25000
0
0
1
2
3
4
5
6
8
BedroomAbvGr

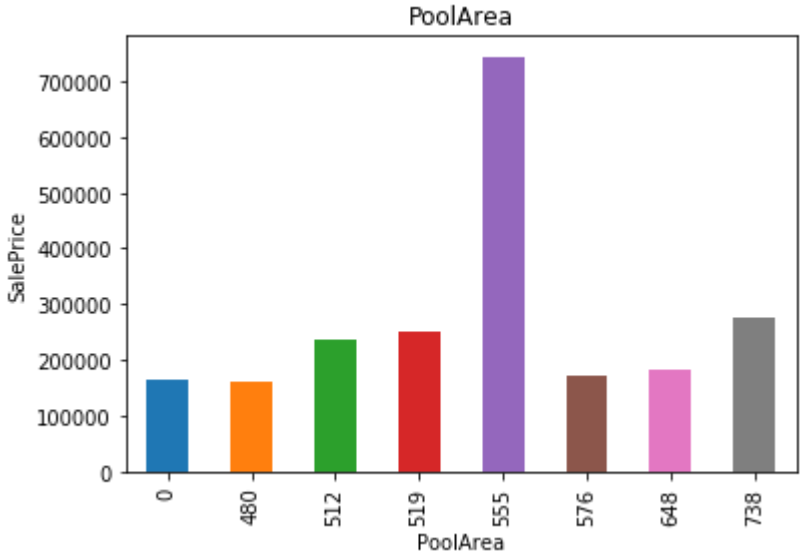

KitchenAbvGr
SalePrice
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
2
3
KitchenAbvGr

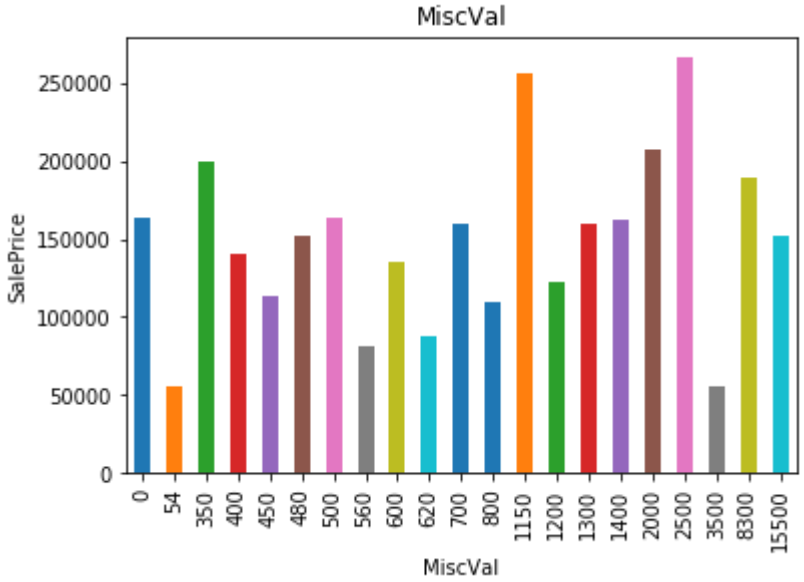

TotRmsAbvGrd
SalePrice
300000
250000
200000
150000
100000
50000
0
2
3
4
5
6
7
8
9
10
11
12
14
TotRmsAbvGrd

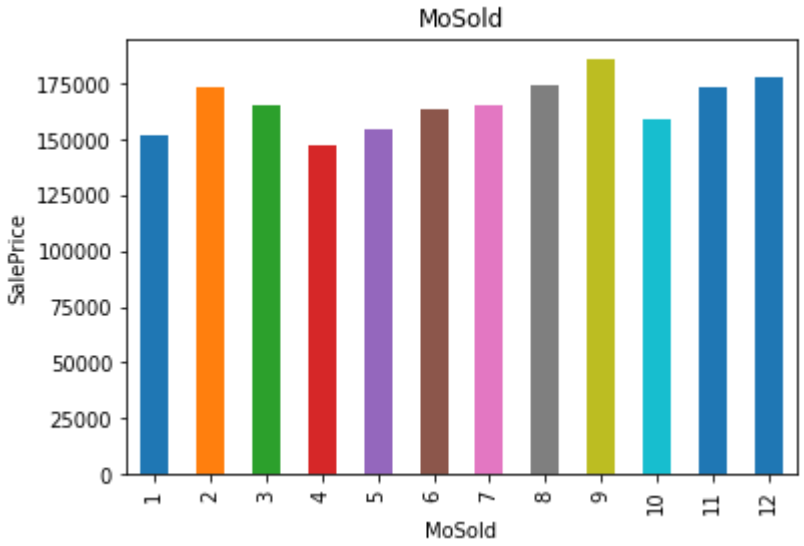

Fireplaces
SalePrice
200000
175000
150000
125000
100000
75000
50000
25000
0
0
1
2
3
Fireplaces


GarageCars
SalePrice
300000
250000
200000
150000
100000
50000
0
0
1
2
3
4
GarageCars


3SsnPorch
SalePrice
400000
350000
300000
250000
200000
150000
100000
50000
0
0
23
96
130
140
144
153
162
168
180
182
196
216
238
245
290
304
320
407
508
3SsnPorch


PoolArea
SalePrice
700000
600000
500000
400000
300000
200000
100000
0
0
480
512
519
555
576
648
738
PoolArea


MiscVal
SalePrice
250000
200000
150000
100000
50000
0
0
54
350
400
450
480
500
560
600
620
700
800
1150
1200
1300
1400
2000
2500
3500
8300
15500
MiscVal



2. Continuous Variables:

In [15]:

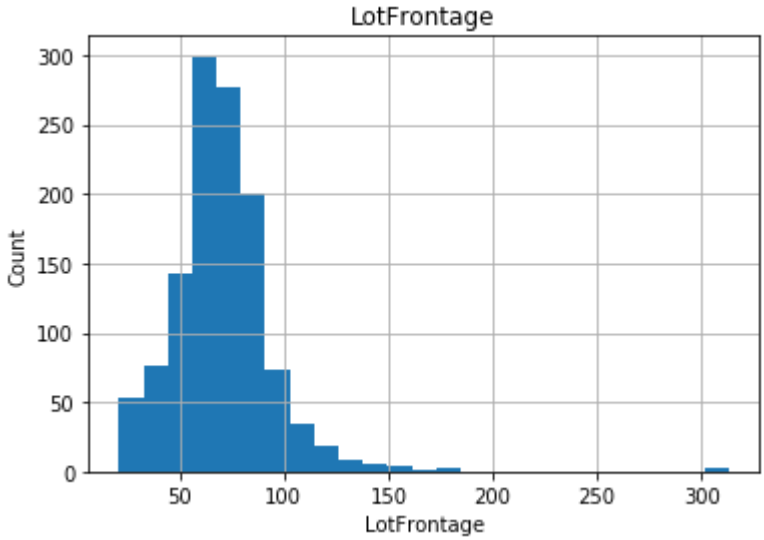
```
continuous_feature=[feature for feature in numeric_features if feature not in discrete_
feature+year_feature+['Id']]
print("Continuous Feature Count {}".format(len(continuous_feature)))
```

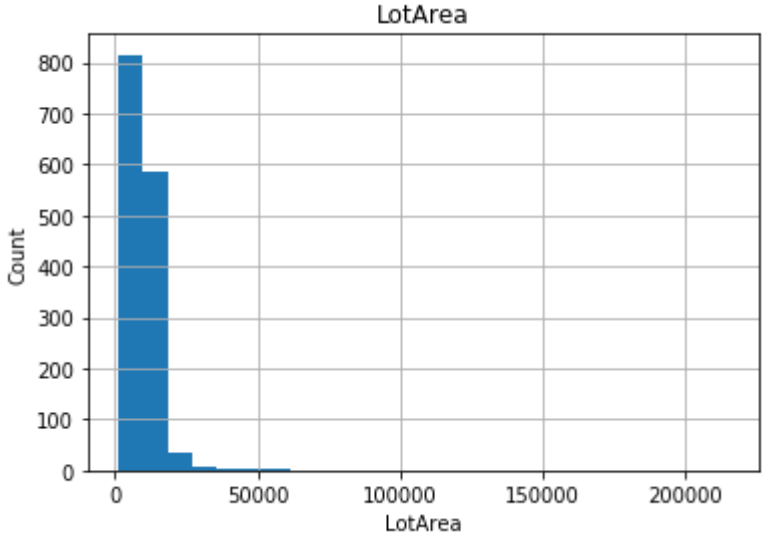
```
Continuous Feature Count 16
```

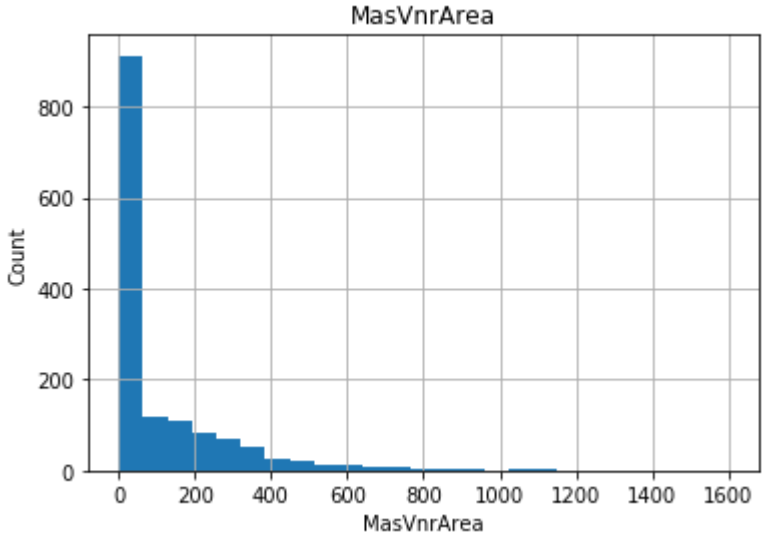
Let us analyse the continuous values with data visualisation to understand the data distribution

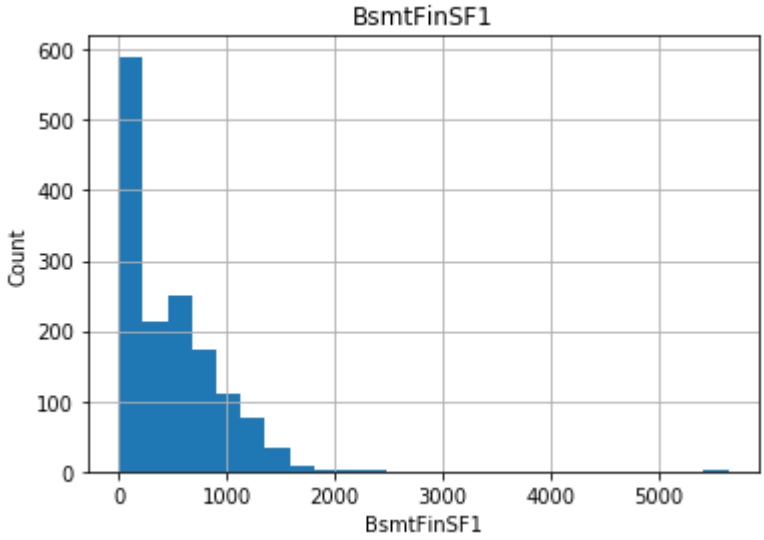
In [16]:

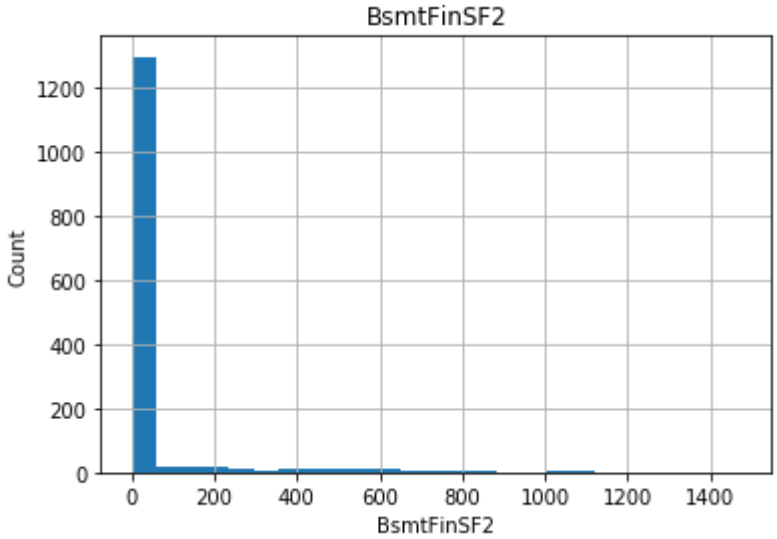
```
for feature in continuous_feature:
    data=train.copy()
    data[feature].hist(bins=25)
    plt.xlabel(feature)
    plt.ylabel("Count")
    plt.title(feature)
    plt.show()
```

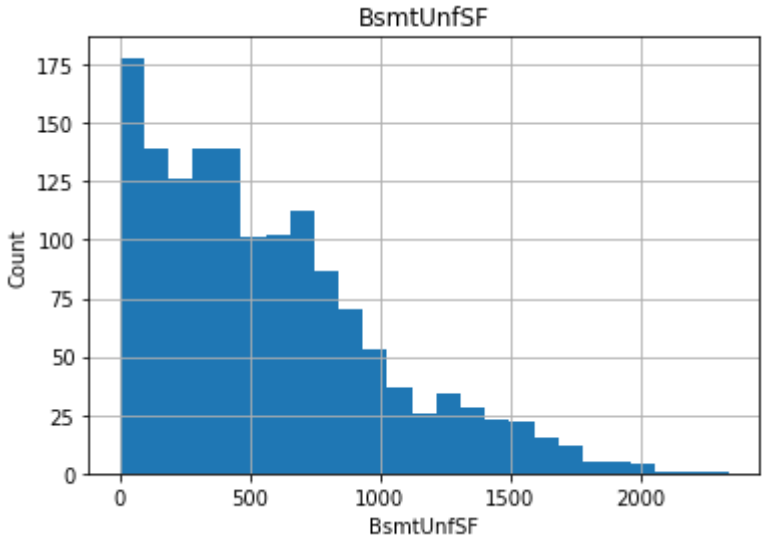

LotFrontage
Count
LotFrontage
0
50
100
150
200
250
300

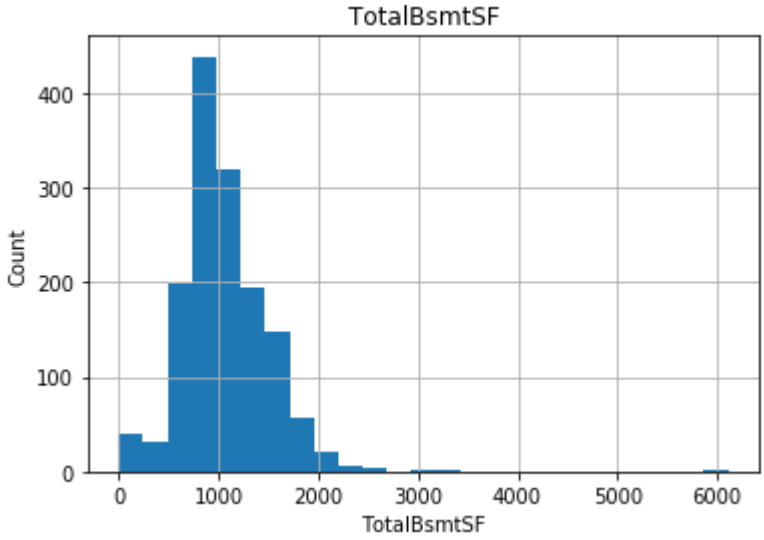

LotArea
Count
LotArea
0
100
200
300
400
500
600
700
800
0
50000
100000
150000
200000

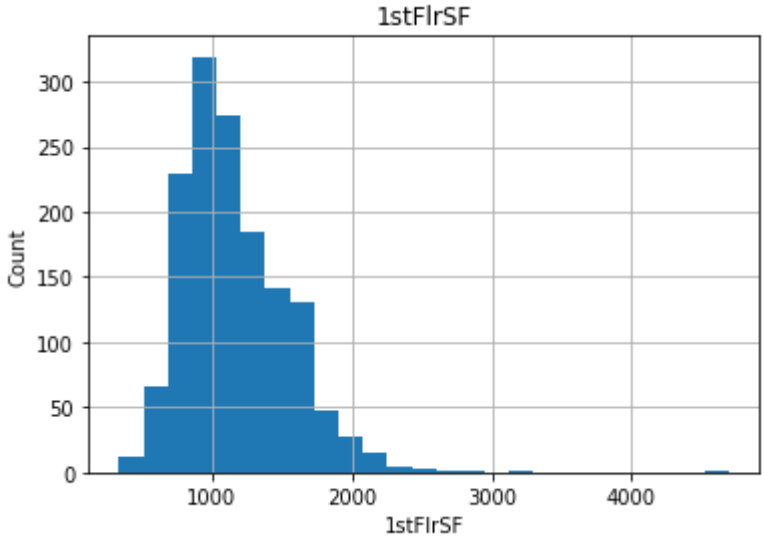

MasVnrArea
Count
MasVnrArea
0
200
400
600
800
0
200
400
600
800
1000
1200
1400
1600

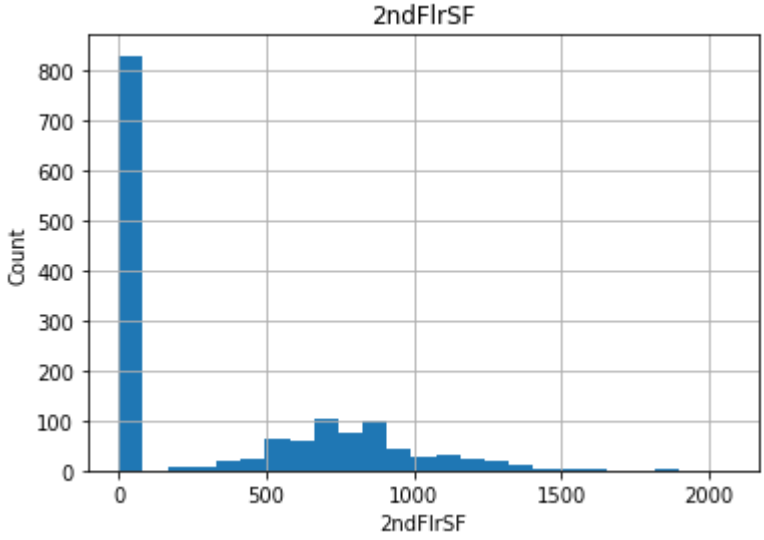

BsmtFinSF1
Count
BsmtFinSF1
0
100
200
300
400
500
600
0
1000
2000
3000
4000
5000

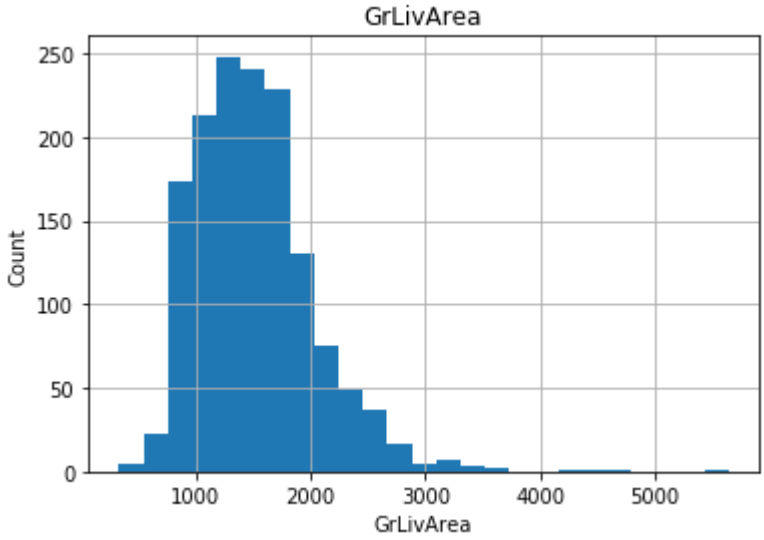

BsmtFinSF2
Count
BsmtFinSF2
0
200
400
600
800
1000
1200
0
200
400
600
800
1000
1200
1400

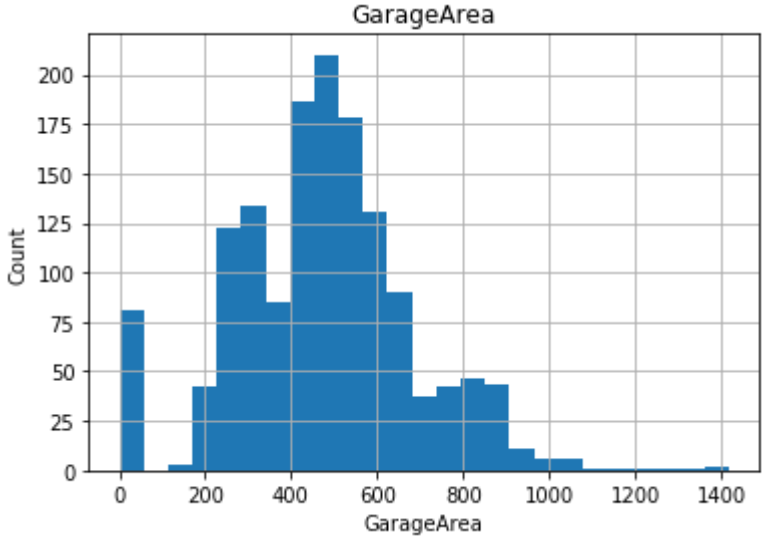

BsmtUnfSF
Count
BsmtUnfSF
0
25
50
75
100
125
150
175
0
500
1000
1500
2000

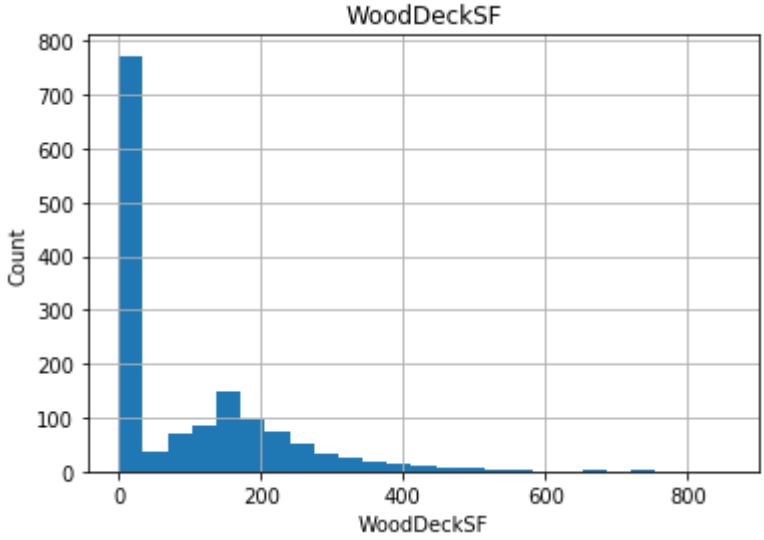

TotalBsmtSF
Count
TotalBsmtSF
0
100
200
300
400
0
1000
2000
3000
4000
5000
6000

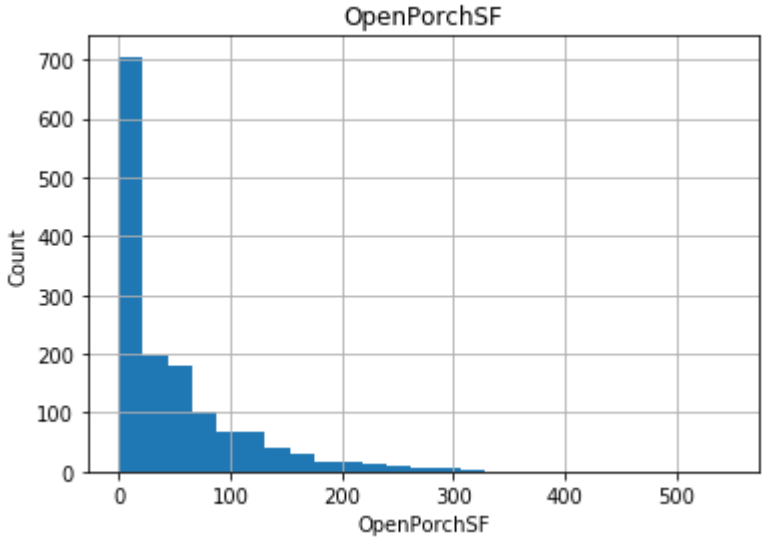

1stFlrSF
Count
1stFlrSF
0
50
100
150
200
250
300
1000
2000
3000
4000

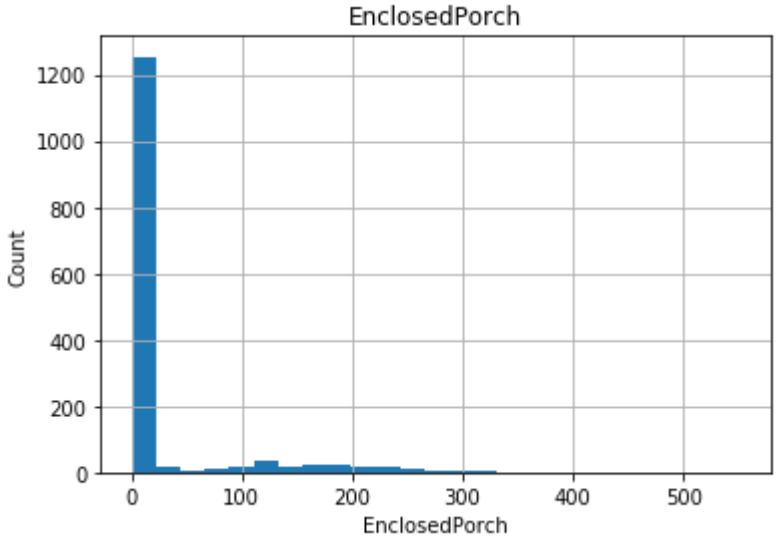

2ndFlrSF
Count
2ndFlrSF
0
100
200
300
400
500
600
700
800
0
500
1000
1500
2000

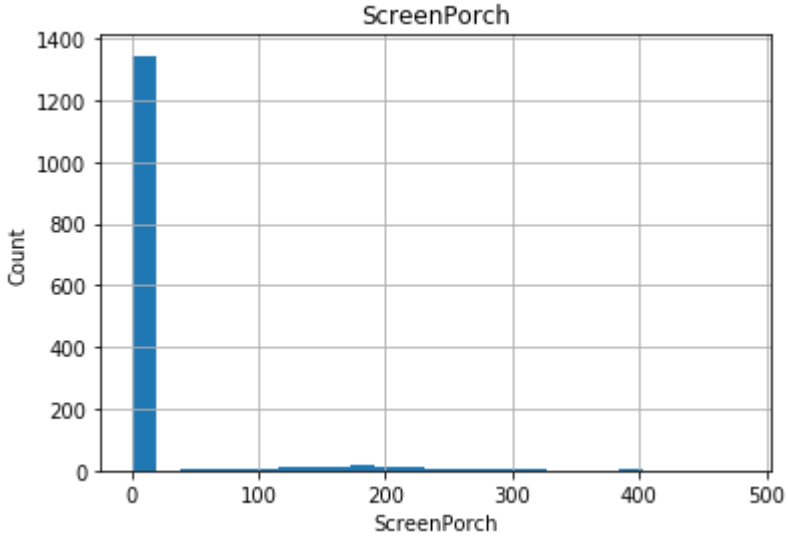

GrLivArea
Count
GrLivArea
0
50
100
150
200
250
1000
2000
3000
4000
5000

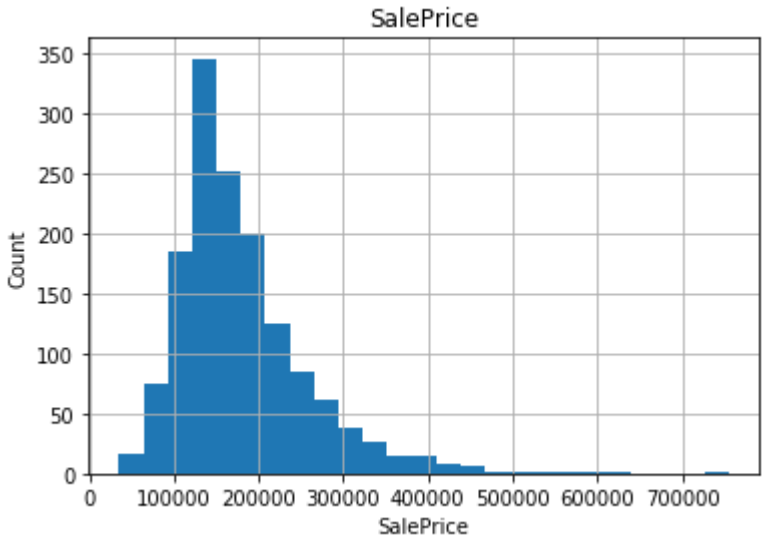

GarageArea
Count
GarageArea
0
25
50
75
100
125
150
175
200
0
200
400
600
800
1000
1200
1400


WoodDeckSF
Count
WoodDeckSF
0
100
200
300
400
500
600
700
800
0
200
400
600
800


OpenPorchSF
Count
OpenPorchSF
0
100
200
300
400
500
600
700


EnclosedPorch
Count
EnclosedPorch
0
200
400
600
800
1000
1200


ScreenPorch
Count
ScreenPorch
0
200
400
600
800
1000
1200
1400



Let us examine categorical features in the train dataset

In [17]:

```
categorical_features = train.select_dtypes(include=[np.object])
categorical_features.columns
```

Out[17]:

```
Index(['MSZoning', 'Street', 'Alley', 'LotShape', 'LandContour', 'Utilitie
s',
       'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition
2',
       'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st',
       'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond', 'Foundatio
n',
       'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinTyp
e2',
       'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual',
       'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQ
ual',
       'GarageCond', 'PavedDrive', 'PoolQC', 'Fence', 'MiscFeature',
       'SaleType', 'SaleCondition'],
      dtype='object')
```

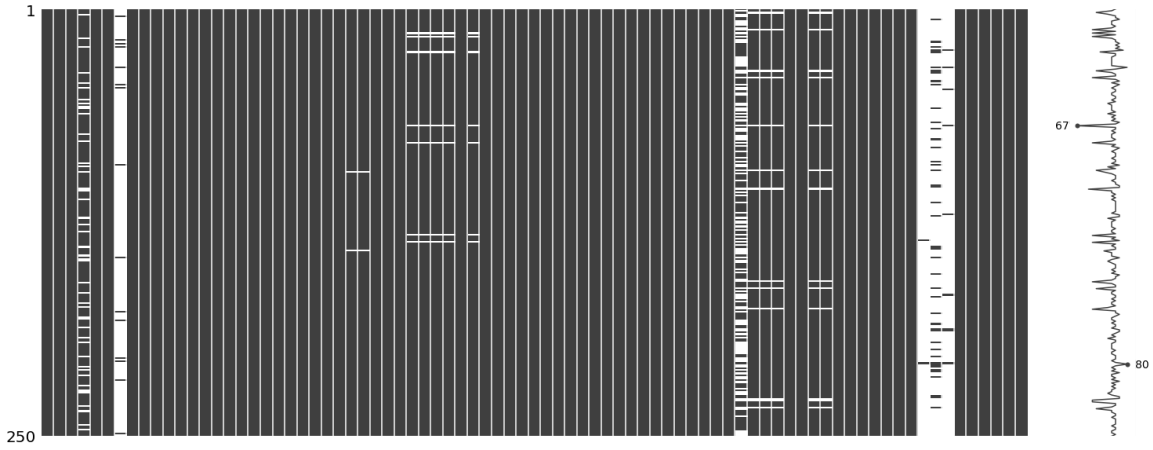
Visualising missing values for a sample of 250

In [18]:

```
msno.matrix(train.sample(250))
```

Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f00c9885400>
```



Heatmap

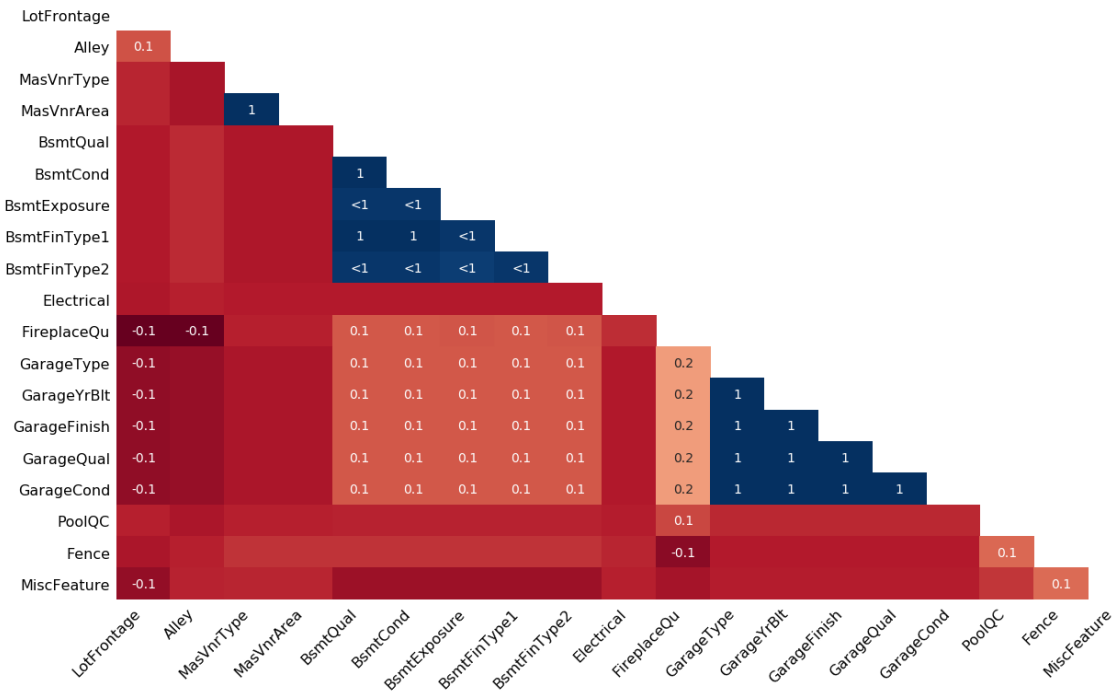
The **missingno** correlation heatmap measures nullity correlation: how strongly the presence or absence of one variable affects the presence of another:

In [19]:

```
msno.heatmap(train)
```

Out[19]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f00c9e0a9e8>
```

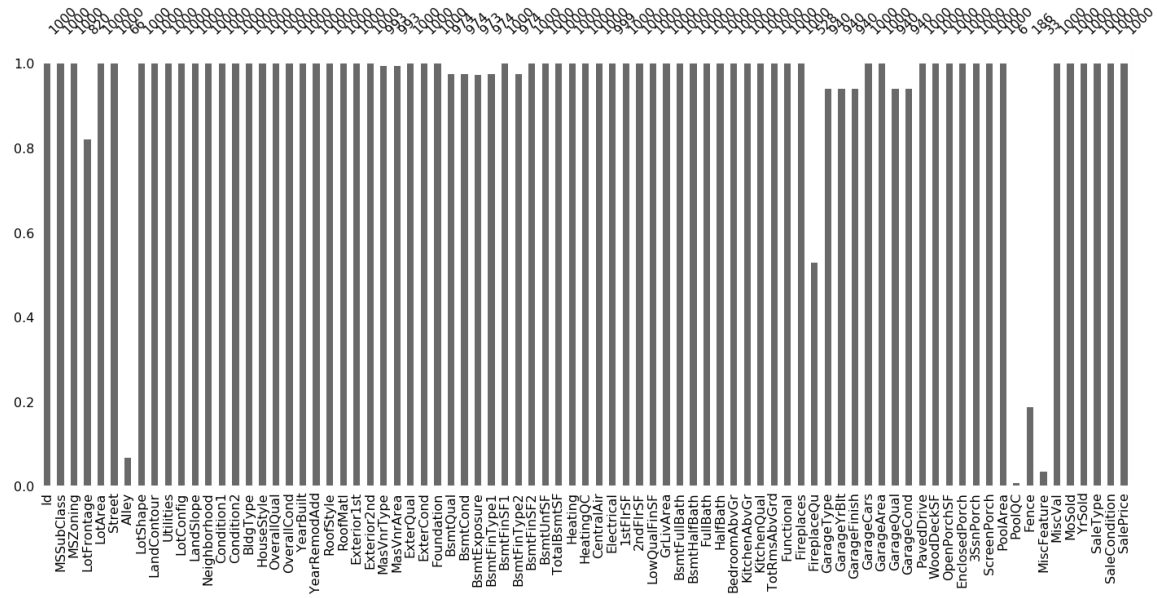


In [20]:

```
msno.bar(train.sample(1000))
```

Out[20]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f00c9e0a128>
```



Dendrogram

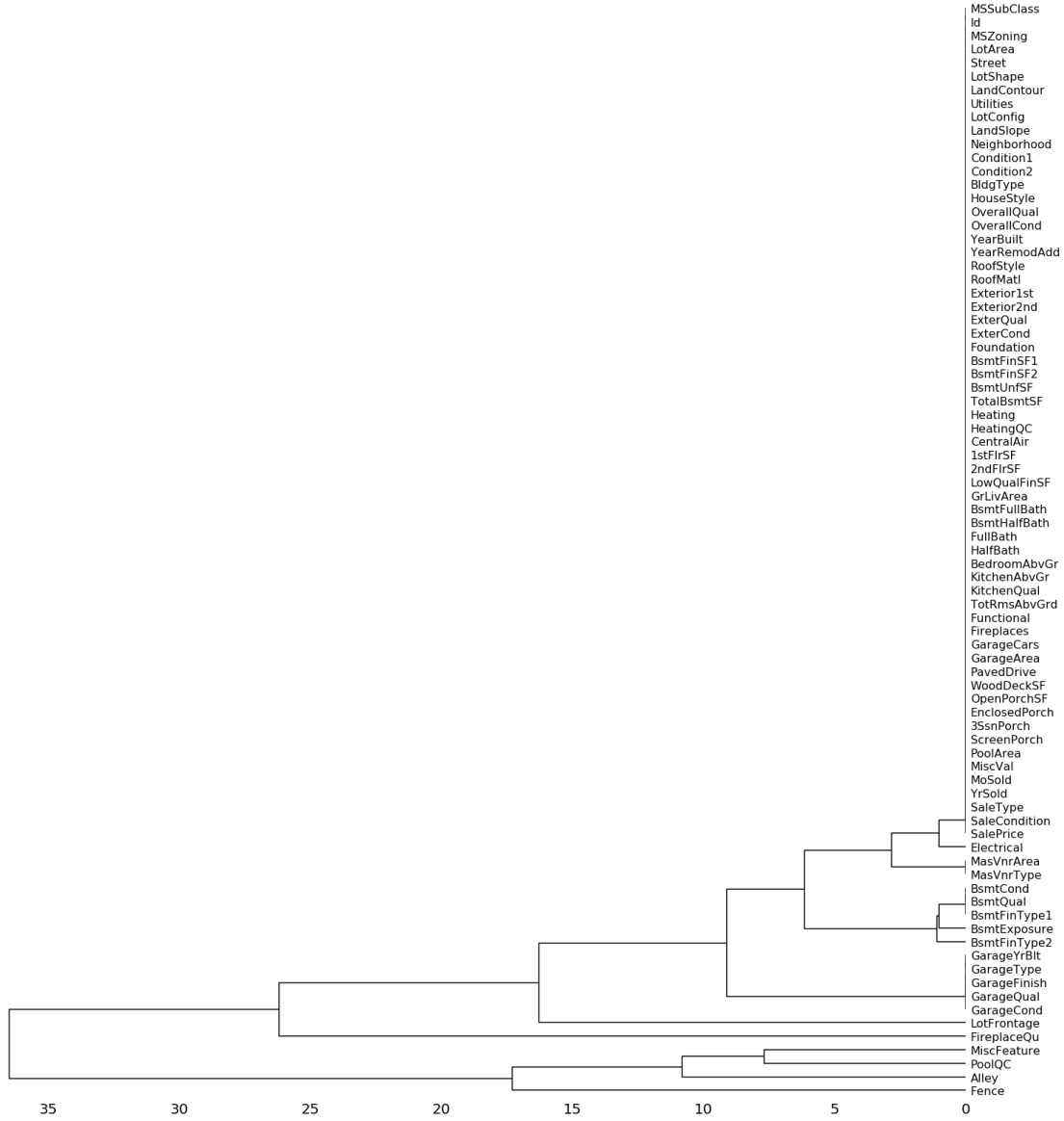
The dendrogram allows you to more fully correlate variable completion, revealing trends deeper than the pairwise ones visible in the correlation heatmap:

In [21]:

```
msno.dendrogram(train)
```

Out[21]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f00c9bc00f0>
```



The dendrogram uses a hierarchical clustering algorithm (courtesy of scipy) to bin variables against one another by their nullity correlation (measured in terms of binary distance). At each step of the tree the variables are split up based on which combination minimizes the distance of the remaining clusters. The more monotone the set of variables, the closer their total distance is to zero, and the closer their average distance (the y-axis) is to zero.

To interpret this graph, read it from a top-down perspective. Cluster leaves which linked together at a distance of zero fully predict one another's presence—one variable might always be empty when another is filled, or they might always both be filled or both empty, and so on. In this specific example the dendrogram glues together the variables which are required and therefore present in every record.

Cluster leaves which split close to zero, but not at it, predict one another very well, but still imperfectly. If your own interpretation of the dataset is that these columns actually are or ought to be match each other in nullity , then the height of the cluster leaf tells you, in absolute terms, how often the records are "mismatched" or incorrectly filed—that is, how many values you would have to fill in or drop, if you are so inclined.

As with matrix, only up to 50 labeled columns will comfortably display in this configuration. However the dendrogram more elegantly handles extremely large datasets by simply flipping to a horizontal configuration.

The Challenges of Your Data

Now that we have gathered some basic information on your data, it’s a good idea to just go a little bit deeper into the challenges that the data might pose.

There are two factors mostly observed in EDA exercise which are **missing values** and **outliers** For understanding in detail on how to handle missing values in detail please visit https://www.kaggle.com/pavansanagapati/simple-tutorial-on-how-to-handle-missing-data (https://www.kaggle.com/pavansanagapati/simple-tutorial-on-how-to-handle-missing-data) For determining the outliers boxplot is used in the later part of this kernel

Estimate Skewness and Kurtosis

In [22]:

```
train.skew(), train.kurt()
```

Out[22]:

```
(Id               0.000000
 MSSubClass       1.407657
 LotFrontage      2.163569
 LotArea         12.207688
 OverallQual      0.216944
 OverallCond      0.693067
 YearBuilt       -0.613461
 YearRemodAdd    -0.503562
 MasVnrArea       2.669084
 BsmtFinSF1       1.685503
 BsmtFinSF2       4.255261
 BsmtUnfSF        0.920268
 TotalBsmtSF      1.524255
 1stFlrSF         1.376757
 2ndFlrSF         0.813030
 LowQualFinSF     9.011341
 GrLivArea        1.366560
 BsmtFullBath     0.596067
 BsmtHalfBath     4.103403
 FullBath         0.036562
 HalfBath         0.675897
 BedroomAbvGr     0.211790
 KitchenAbvGr     4.488397
 TotRmsAbvGrd     0.676341
 Fireplaces       0.649565
 GarageYrBlt     -0.649415
 GarageCars      -0.342549
 GarageArea       0.179981
 WoodDeckSF       1.541376
 OpenPorchSF      2.364342
 EnclosedPorch    3.089872
 3SsnPorch       10.304342
 ScreenPorch      4.122214
 PoolArea        14.828374
 MiscVal         24.476794
 MoSold           0.212053
 YrSold           0.096269
 SalePrice        1.882876
 dtype: float64, Id               -1.200000
 MSSubClass        1.580188
 LotFrontage      17.452867
 LotArea         203.243271
 OverallQual       0.096293
 OverallCond       1.106413
 YearBuilt        -0.439552
 YearRemodAdd     -1.272245
 MasVnrArea       10.082417
 BsmtFinSF1       11.118236
 BsmtFinSF2       20.113338
 BsmtUnfSF         0.474994
 TotalBsmtSF      13.250483
 1stFlrSF          5.745841
 2ndFlrSF         -0.553464
 LowQualFinSF     83.234817
 GrLivArea         4.895121
 BsmtFullBath     -0.839098
 BsmtHalfBath     16.396642
 FullBath         -0.857043
 HalfBath         -1.076927
```

```
BedroomAbvGr       2.230875
KitchenAbvGr      21.532404
TotRmsAbvGrd       0.880762
Fireplaces        -0.217237
GarageYrBlt       -0.418341
GarageCars         0.220998
GarageArea         0.917067
WoodDeckSF         2.992951
OpenPorchSF        8.490336
EnclosedPorch     10.430766
3SsnPorch        123.662379
ScreenPorch       18.439068
PoolArea         223.268499
MiscVal          701.003342
MoSold            -0.404109
YrSold            -1.190601
SalePrice          6.536282
dtype: float64)
```

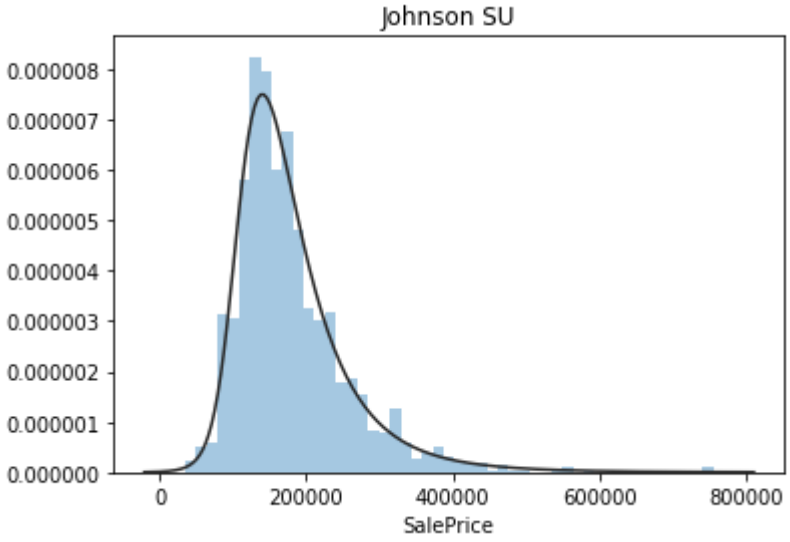
In [23]:

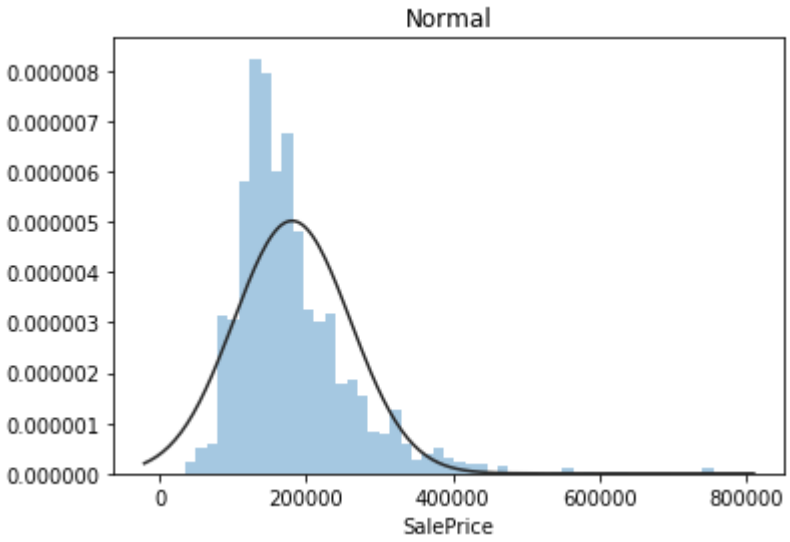
```
y = train['SalePrice']
plt.figure(1); plt.title('Johnson SU')
sns.distplot(y, kde=False, fit=st.johnsonsu)
plt.figure(2); plt.title('Normal')
sns.distplot(y, kde=False, fit=st.norm)
plt.figure(3); plt.title('Log Normal')
sns.distplot(y, kde=False, fit=st.lognorm)
```

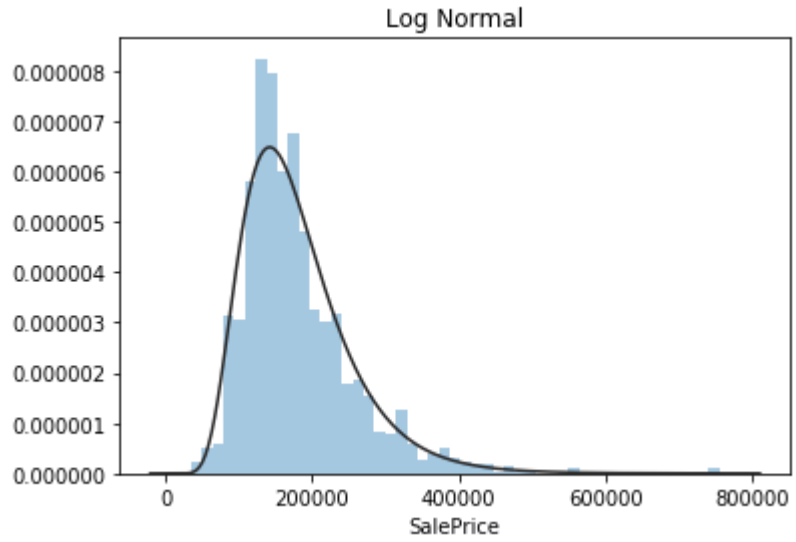
```
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWa
rning: Using a non-tuple sequence for multidimensional indexing is depreca
ted; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will
be interpreted as an array index, `arr[np.array(seq)]`, which will result
either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[23]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f00c9cf5828>
```







It is apparent that SalePrice doesn't follow normal distribution, so before performing regression it has to be transformed. While log transformation does pretty good job, best fit is unbounded Johnson distribution.

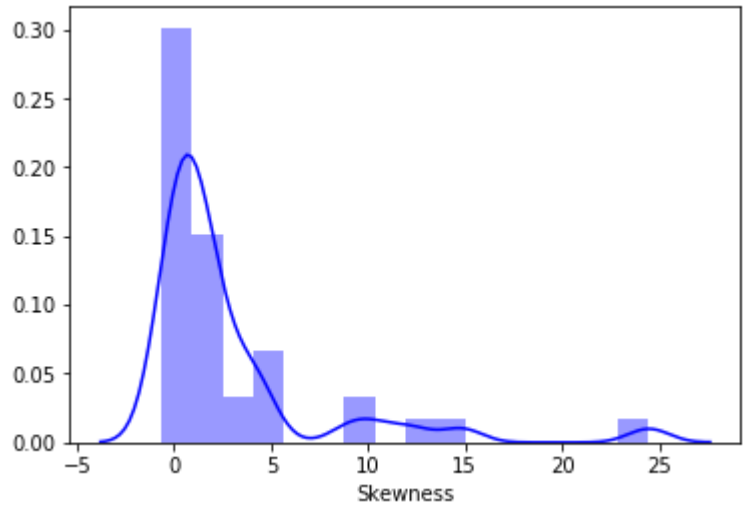
In [24]:

```
sns.distplot(train.skew(),color='blue',axlabel ='Skewness')
```

```
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWa
rning: Using a non-tuple sequence for multidimensional indexing is depreca
ted; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will
be interpreted as an array index, `arr[np.array(seq)]`, which will result
either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[24]:

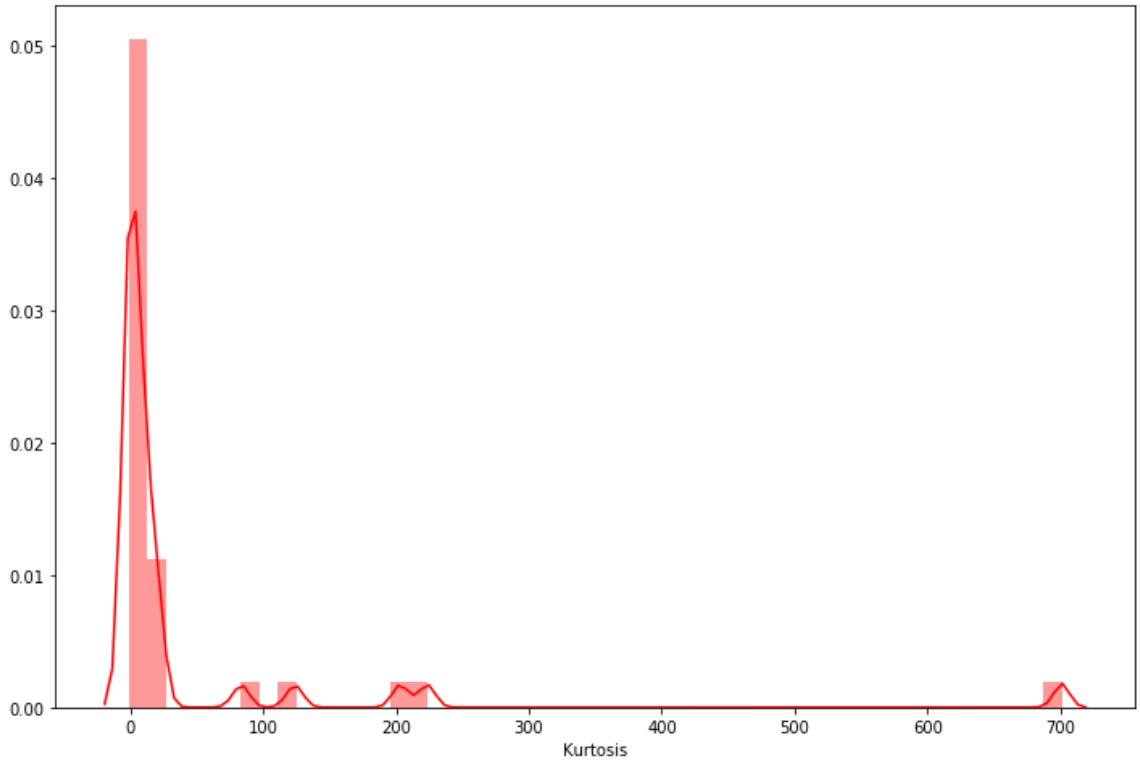
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f00c9f86ef0>
```



In [25]:

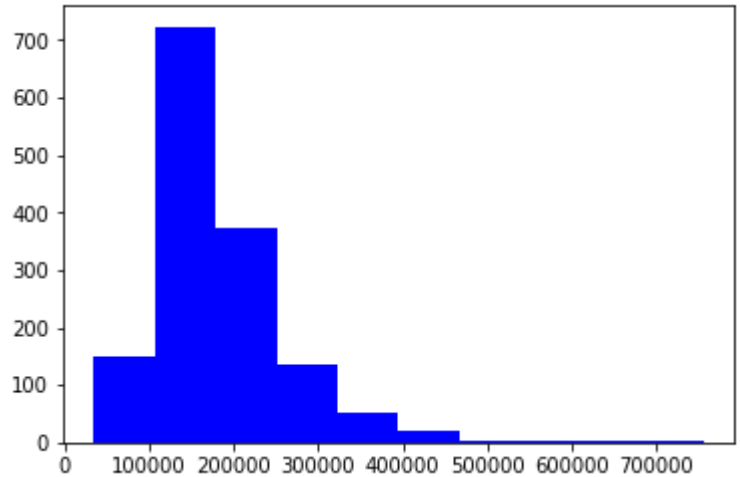
```
plt.figure(figsize = (12,8))
sns.distplot(train.kurt(),color='r',axlabel ='Kurtosis',norm_hist= False, kde = True,rug = False)
#plt.hist(train.kurt(),orientation = 'vertical',histtype = 'bar',label ='Kurtosis', color ='blue')
plt.show()
```

```
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



In [26]:

```
plt.hist(train['SalePrice'],orientation = 'vertical',histtype = 'bar', color ='blue')
plt.show()
```

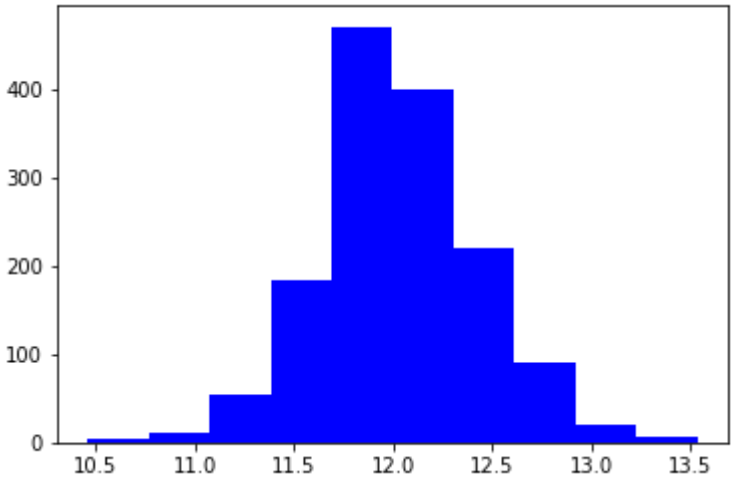


In [27]:

```
target = np.log(train['SalePrice'])
target.skew()
plt.hist(target,color='blue')
```

Out[27]:

```
(array([  5.,  12.,  54., 184., 470., 400., 220.,  90.,  19.,   6.]),
 array([10.46024211, 10.7676652 , 11.07508829, 11.38251138, 11.68993448,
        11.99735757, 12.30478066, 12.61220375, 12.91962684, 13.22704994,
        13.53447303]),
 <a list of 10 Patch objects>)
```



Finding Correlation coefficients between numeric features and SalePrice

In [28]:

```
correlation = numeric_features.corr()
print(correlation['SalePrice'].sort_values(ascending = False),'\n')
```

```
SalePrice        1.000000
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
GarageYrBlt      0.486362
MasVnrArea       0.477493
Fireplaces       0.466929
BsmtFinSF1       0.386420
LotFrontage      0.351799
WoodDeckSF       0.324413
2ndFlrSF         0.319334
OpenPorchSF      0.315856
HalfBath         0.284108
LotArea          0.263843
BsmtFullBath     0.227122
BsmtUnfSF        0.214479
BedroomAbvGr     0.168213
ScreenPorch      0.111447
PoolArea         0.092404
MoSold           0.046432
3SsnPorch        0.044584
BsmtFinSF2      -0.011378
BsmtHalfBath    -0.016844
MiscVal         -0.021190
Id              -0.021917
LowQualFinSF    -0.025606
YrSold          -0.028923
OverallCond     -0.077856
MSSubClass      -0.084284
EnclosedPorch   -0.128578
KitchenAbvGr    -0.135907
Name: SalePrice, dtype: float64
```

To explore further we will start with the following visualisation methods to analyze the data better:

- Correlation Heat Map
- Zoomed Heat Map
- Pair Plot
- Scatter Plot

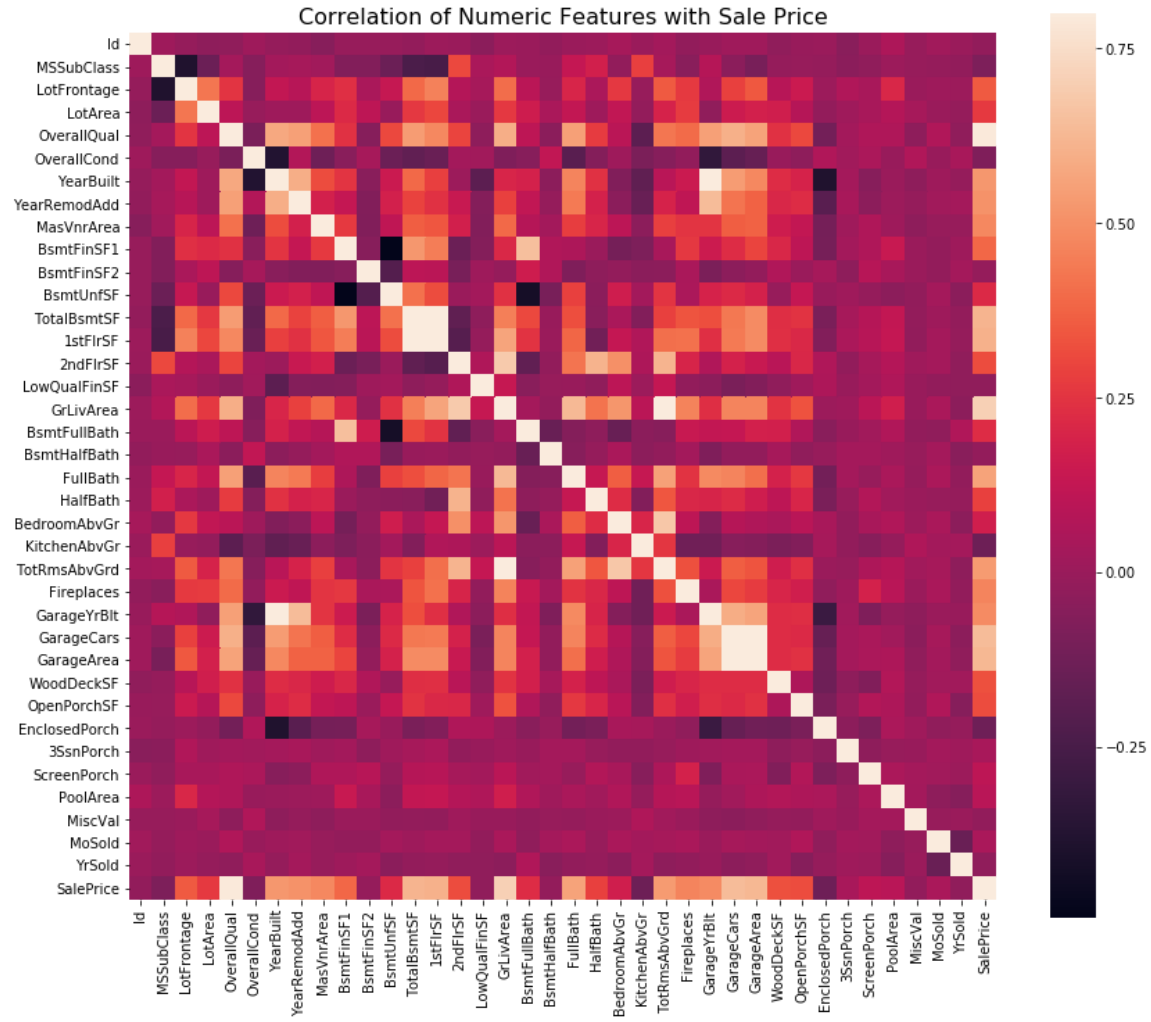
Correlation Heat Map

In [29]:

```
f , ax = plt.subplots(figsize = (14,12))
plt.title('Correlation of Numeric Features with Sale Price',y=1,size=16)
sns.heatmap(correlation,square = True,  vmax=0.8)
```

Out[29]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f00c84e25f8>
```



The heatmap is the best way to get a quick overview of correlated features thanks to seaborn!

At initial glance it is observed that there are two red colored squares that get my attention.

1. The first one refers to the 'TotalBsmtSF' and '1stFlrSF' variables.
2. Second one refers to the 'GarageX' variables. Both cases show how significant the correlation is between these variables. Actually, this correlation is so strong that it can indicate a situation of multicollinearity. If we think about these variables, we can conclude that they give almost the same information so multicollinearity really occurs.

Heatmaps are great to detect this kind of multicollinearity situations and in problems related to feature selection like this project, it comes as an excellent exploratory tool.

Another aspect I observed here is the 'SalePrice' correlations.As it is observed that 'GrLivArea', 'TotalBsmtSF', and 'OverallQual' saying a big 'Hello !' to SalePrice, however we cannot exclude the fact that rest of the features have some level of correlation to the SalePrice. To observe this correlation closer let us see it in Zoomed Heat Map

Zoomed HeatMap

SalePrice Correlation matrix

In [30]:

```
k= 11
cols = correlation.nlargest(k,'SalePrice')['SalePrice'].index
print(cols)
cm = np.corrcoef(train[cols].values.T)
f , ax = plt.subplots(figsize = (14,12))
sns.heatmap(cm, vmax=.8, linewidths=0.01,square=True,annot=True,cmap='viridis',
            linecolor="white",xticklabels = cols.values ,annot_kws = {'size':12},ytickl
abels = cols.values)
```

```
Index(['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'GarageAre
a',
       'TotalBsmtSF', '1stFlrSF', 'FullBath', 'TotRmsAbvGrd', 'YearBuilt',
       'YearRemodAdd'],
      dtype='object')
```

Out[30]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f00c83aa668>
```

From above zoomed heatmap it is observed that GarageCars & GarageArea are closely correlated . Similarly TotalBsmtSF and 1stFlrSF are also closely correlated.

My observations :

- 'OverallQual', 'GrLivArea' and 'TotalBsmtSF' are strongly correlated with 'SalePrice'.
- 'GarageCars' and 'GarageArea' are strongly correlated variables. It is because the number of cars that fit into the garage is a consequence of the garage area. 'GarageCars' and 'GarageArea' are like twin brothers. So it is hard to distinguish between the two. Therefore, we just need one of these variables in our analysis (we can keep 'GarageCars' since its correlation with 'SalePrice' is higher).
- 'TotalBsmtSF' and '1stFloor' also seem to be twins. In this case let us keep 'TotalBsmtSF'
- 'TotRmsAbvGrd' and 'GrLivArea', twins
- 'YearBuilt' it appears like is slightly correlated with 'SalePrice'. This required more analysis to arrive at a conclusion may be do some time series analysis.

Pair Plot

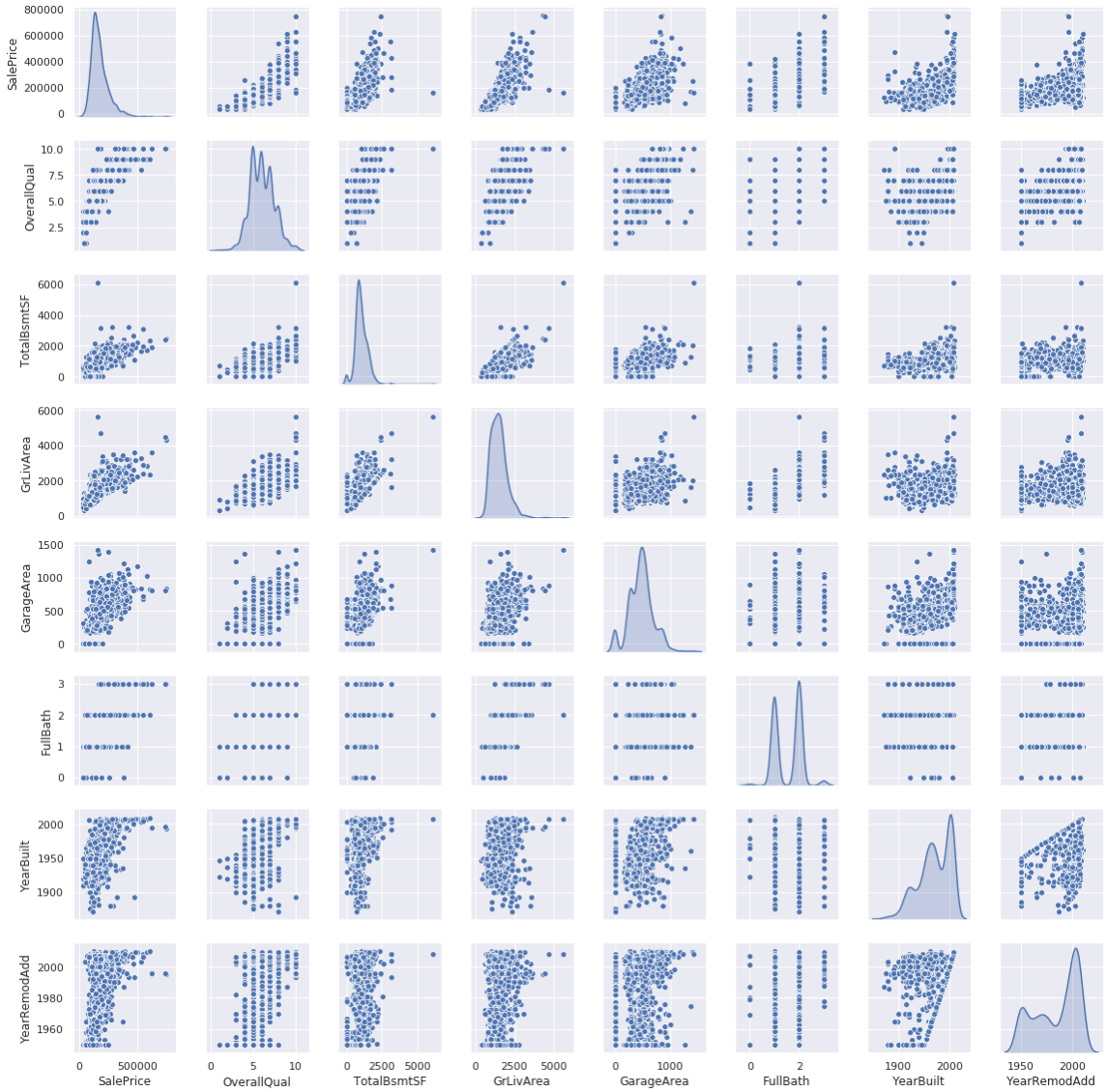
Pair Plot between 'SalePrice' and correlated variables

Visualisation of 'OverallQual','TotalBsmtSF','GrLivArea','GarageArea','FullBath','YearBuilt','YearRemodAdd' features with respect to SalePrice in the form of pair plot & scatter pair plot for better understanding.

In [31]:

```
sns.set()
columns = ['SalePrice','OverallQual','TotalBsmtSF','GrLivArea','GarageArea','FullBath',
'YearBuilt','YearRemodAdd']
sns.pairplot(train[columns],size = 2 ,kind ='scatter',diag_kind='kde')
plt.show()
```

```
/opt/conda/lib/python3.6/site-packages/seaborn/axisgrid.py:2065: UserWarni
ng: The `size` parameter has been renamed to `height`; pleaes update your
code.
  warnings.warn(msg, UserWarning)
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWa
rning: Using a non-tuple sequence for multidimensional indexing is depreca
ted; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will
be interpreted as an array index, `arr[np.array(seq)]`, which will result
either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



Although we already know some of the main figures, this pair plot gives us a reasonable overview insight about the correlated features .Here are some of my analysis.

- One interesting observation is between 'TotalBsmtSF' and 'GrLiveArea'. In this figure we can see the dots drawing a linear line, which almost acts like a border. It totally makes sense that the majority of the dots stay below that line. Basement areas can be equal to the above ground living area, but it is not expected a basement area bigger than the above ground living area.
- One more interesting observation is between 'SalePrice' and 'YearBuilt'. In the bottom of the 'dots cloud', we see what almost appears to be a exponential function.We can also see this same tendency in the upper limit of the 'dots cloud'
- Last observation is that prices are increasing faster now with respect to previous years.

Scatter Plot

Scatter plots between the most correlated variables

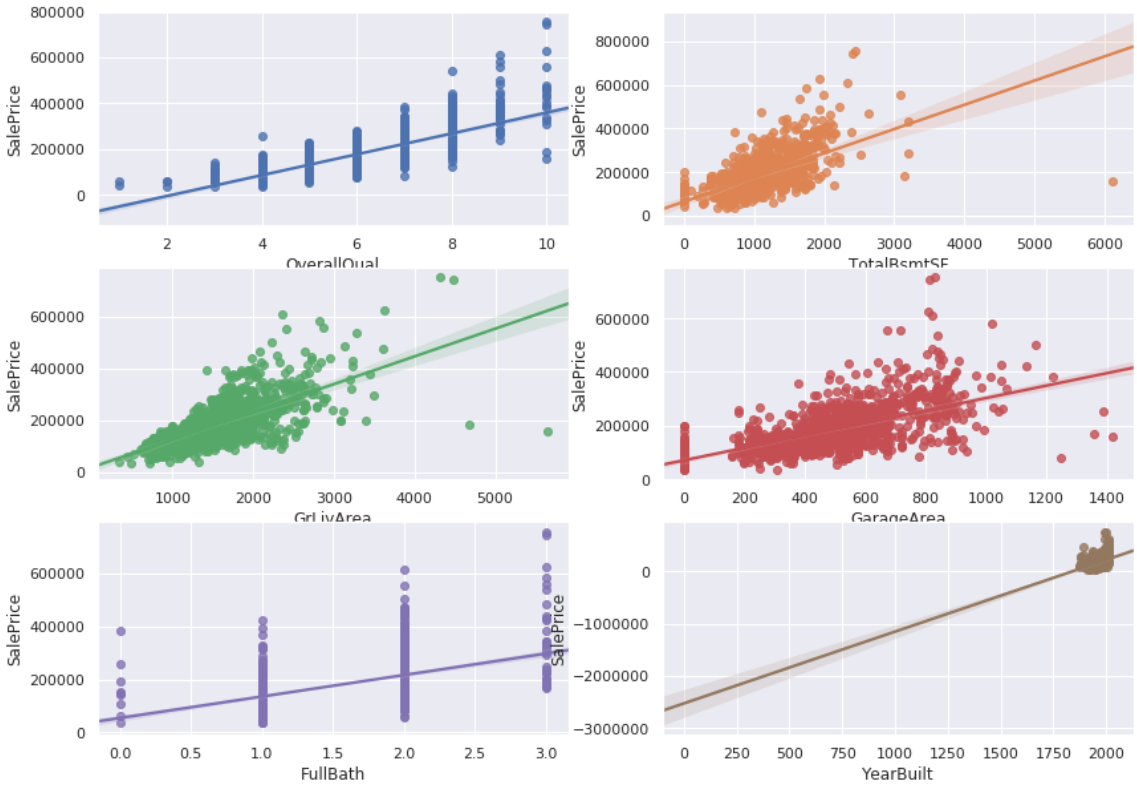
In [32]:

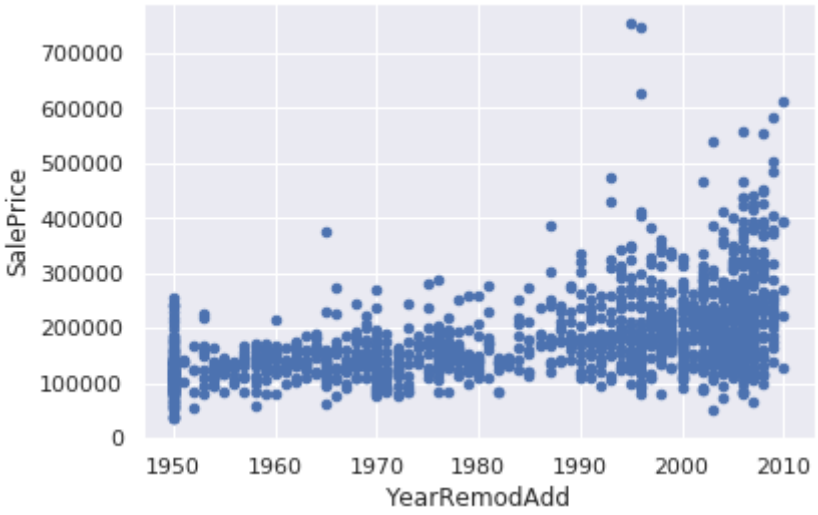
```
fig, ((ax1, ax2), (ax3, ax4),(ax5,ax6)) = plt.subplots(nrows=3, ncols=2, figsize=(14,10
))
OverallQual_scatter_plot = pd.concat([train['SalePrice'],train['OverallQual']],axis = 1
)
sns.regplot(x='OverallQual',y = 'SalePrice',data = OverallQual_scatter_plot,scatter= Tr
ue, fit_reg=True, ax=ax1)
TotalBsmtSF_scatter_plot = pd.concat([train['SalePrice'],train['TotalBsmtSF']],axis = 1
)
sns.regplot(x='TotalBsmtSF',y = 'SalePrice',data = TotalBsmtSF_scatter_plot,scatter= Tr
ue, fit_reg=True, ax=ax2)
GrLivArea_scatter_plot = pd.concat([train['SalePrice'],train['GrLivArea']],axis = 1)
sns.regplot(x='GrLivArea',y = 'SalePrice',data = GrLivArea_scatter_plot,scatter= True,
fit_reg=True, ax=ax3)
GarageArea_scatter_plot = pd.concat([train['SalePrice'],train['GarageArea']],axis = 1)
sns.regplot(x='GarageArea',y = 'SalePrice',data = GarageArea_scatter_plot,scatter= True
, fit_reg=True, ax=ax4)
FullBath_scatter_plot = pd.concat([train['SalePrice'],train['FullBath']],axis = 1)
sns.regplot(x='FullBath',y = 'SalePrice',data = FullBath_scatter_plot,scatter= True, fi
t_reg=True, ax=ax5)
YearBuilt_scatter_plot = pd.concat([train['SalePrice'],train['YearBuilt']],axis = 1)
sns.regplot(x='YearBuilt',y = 'SalePrice',data = YearBuilt_scatter_plot,scatter= True,
fit_reg=True, ax=ax6)
YearRemodAdd_scatter_plot = pd.concat([train['SalePrice'],train['YearRemodAdd']],axis =
1)
YearRemodAdd_scatter_plot.plot.scatter('YearRemodAdd','SalePrice')
```

```
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWa
rning: Using a non-tuple sequence for multidimensional indexing is depreca
ted; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will
be interpreted as an array index, `arr[np.array(seq)]`, which will result
either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[32]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f00befdfe10>
```





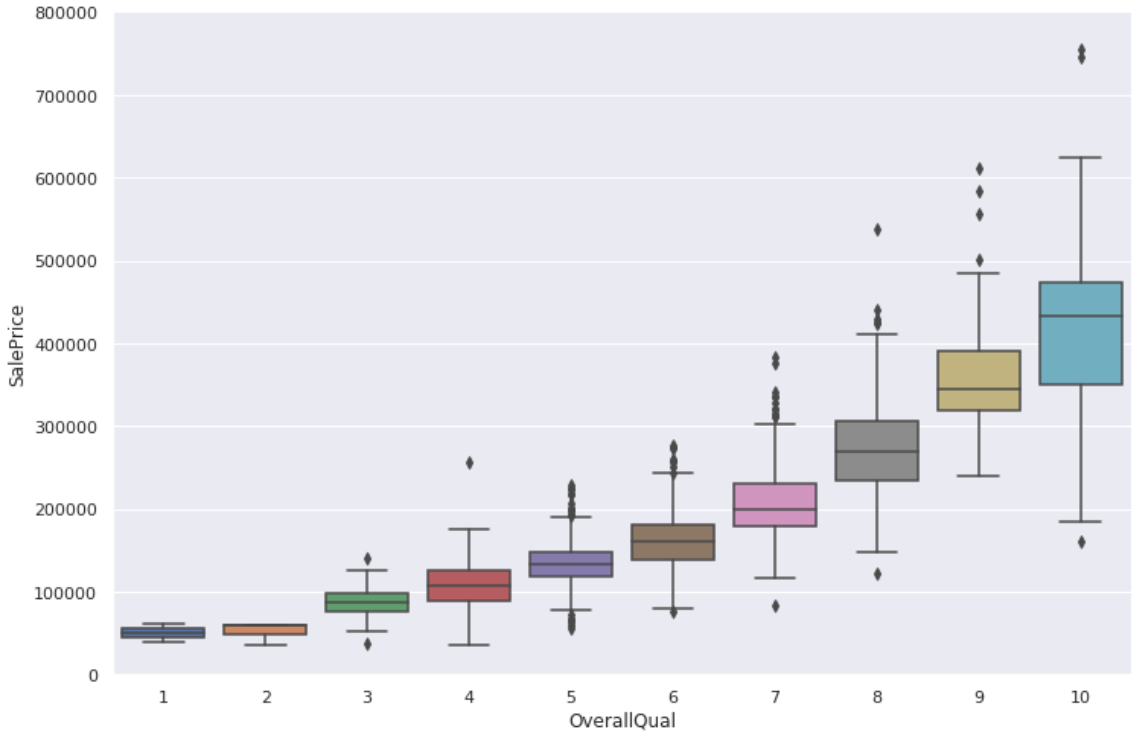
In [33]:

```
saleprice_overall_quality= train.pivot_table(index ='OverallQual',values = 'SalePrice',
aggfunc = np.median)
saleprice_overall_quality.plot(kind = 'bar',color = 'blue')
plt.xlabel('Overall Quality')
plt.ylabel('Median Sale Price')
plt.show()
```

Box plot - OverallQual

In [34]:

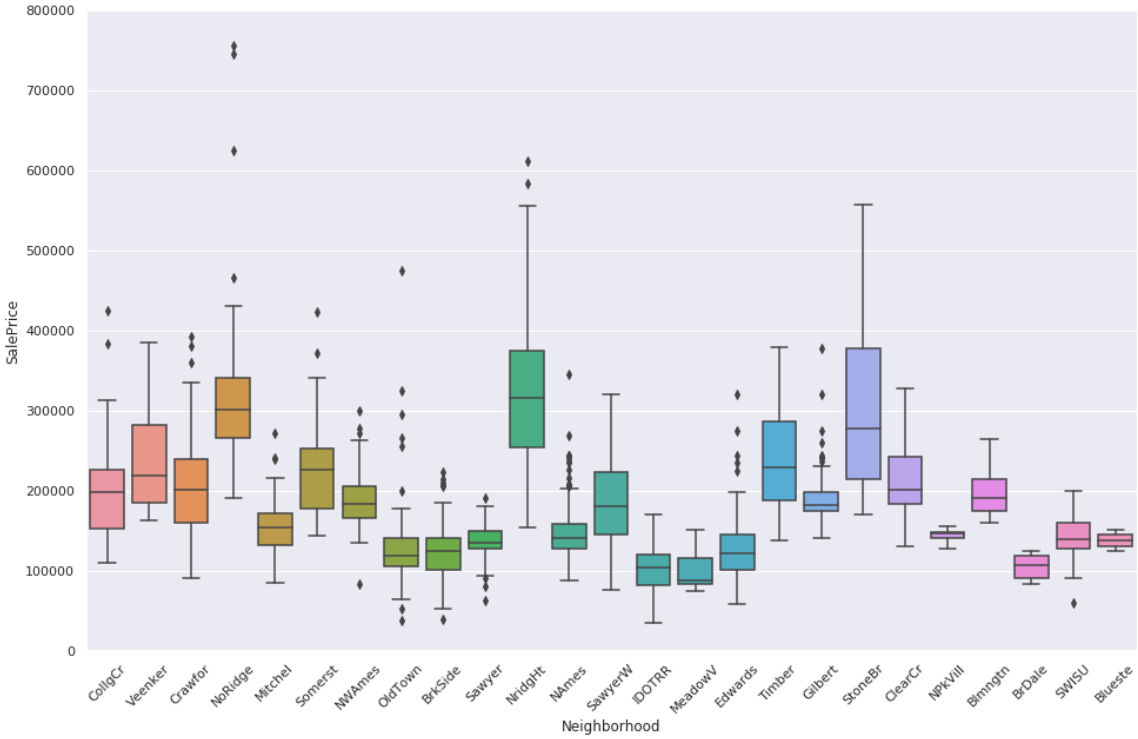
```
var = 'OverallQual'
data = pd.concat([train['SalePrice'], train[var]], axis=1)
f, ax = plt.subplots(figsize=(12, 8))
fig = sns.boxplot(x=var, y="SalePrice", data=data)
fig.axis(ymin=0, ymax=800000);
```



Box plot - Neighborhood

In [35]:

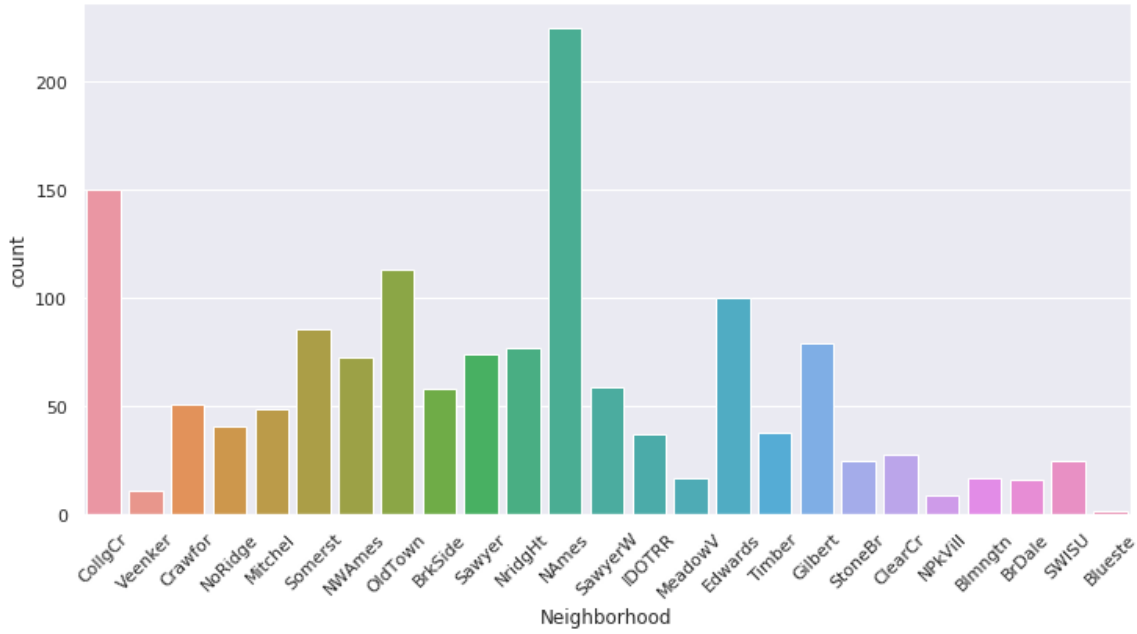
```
var = 'Neighborhood'
data = pd.concat([train['SalePrice'], train[var]], axis=1)
f, ax = plt.subplots(figsize=(16, 10))
fig = sns.boxplot(x=var, y="SalePrice", data=data)
fig.axis(ymin=0, ymax=800000);
xt = plt.xticks(rotation=45)
```



Count Plot - Neighborhood

In [36]:

```
plt.figure(figsize = (12, 6))
sns.countplot(x = 'Neighborhood', data = data)
xt = plt.xticks(rotation=45)
```



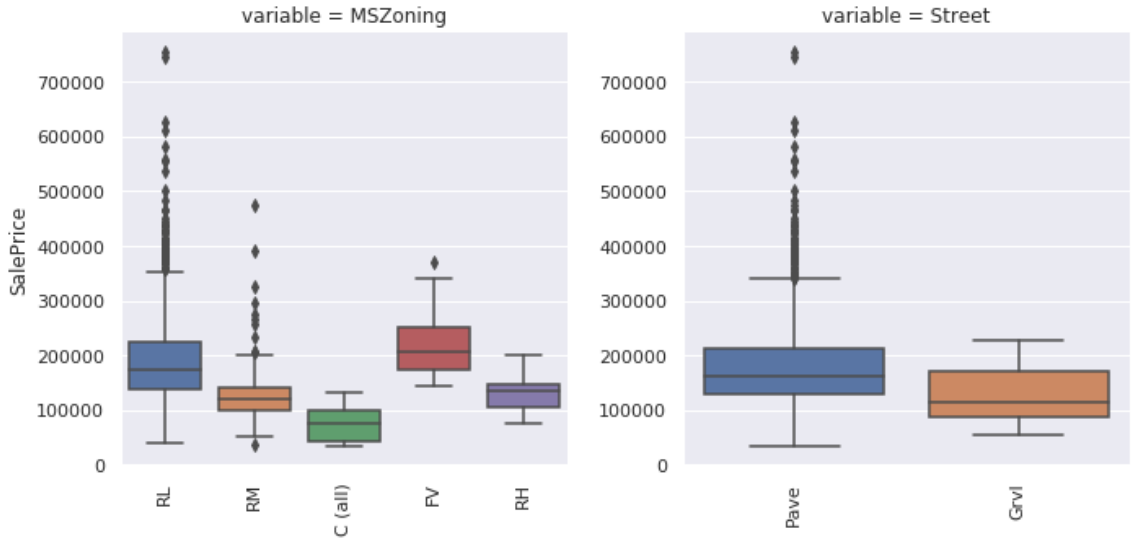
Based on the above observation can group those Neighborhoods with similar housing price into a same bucket for dimension-reduction.Let us see this in the preprocessing stage

With qualitative variables we can check distribution of SalePrice with respect to variable values and enumerate them.

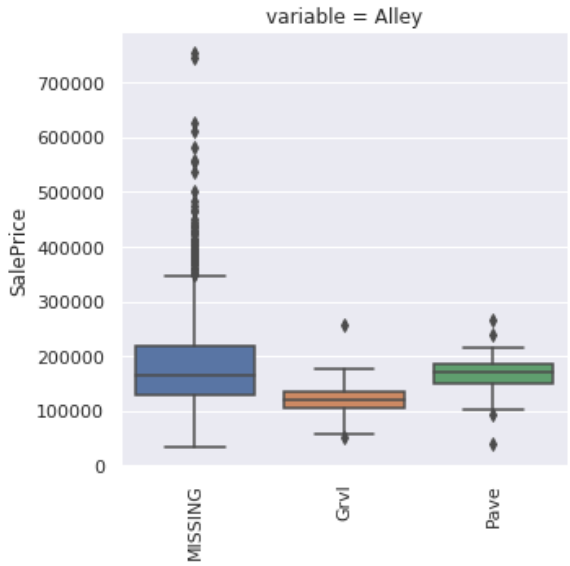
In [37]:

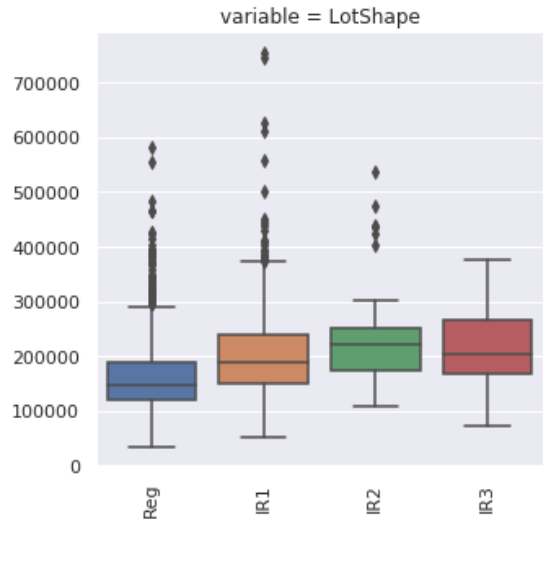
```
for c in categorical_features:
    train[c] = train[c].astype('category')
    if train[c].isnull().any():
        train[c] = train[c].cat.add_categories(['MISSING'])
        train[c] = train[c].fillna('MISSING')

def boxplot(x, y, **kwargs):
    sns.boxplot(x=x, y=y)
    x=plt.xticks(rotation=90)
f = pd.melt(train, id_vars=['SalePrice'], value_vars=categorical_features)
g = sns.FacetGrid(f, col="variable",  col_wrap=2, sharex=False, sharey=False, size=5)
g = g.map(boxplot, "value", "SalePrice")
```

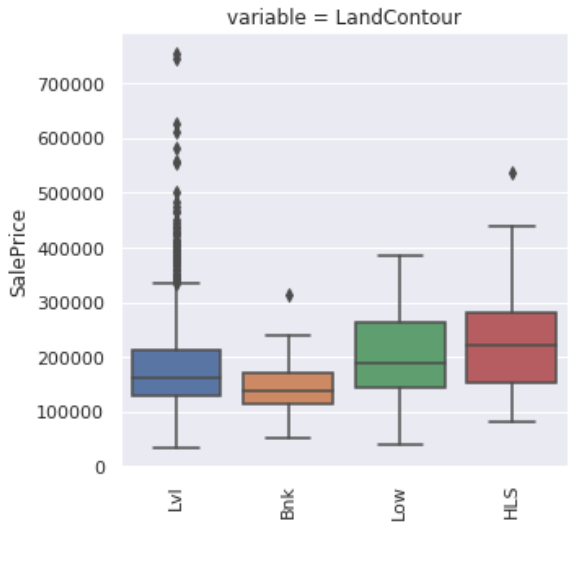
```
/opt/conda/lib/python3.6/site-packages/seaborn/axisgrid.py:230: UserWarnin
g: The `size` paramter has been renamed to `height`; please update your co
de.
  warnings.warn(msg, UserWarning)
```

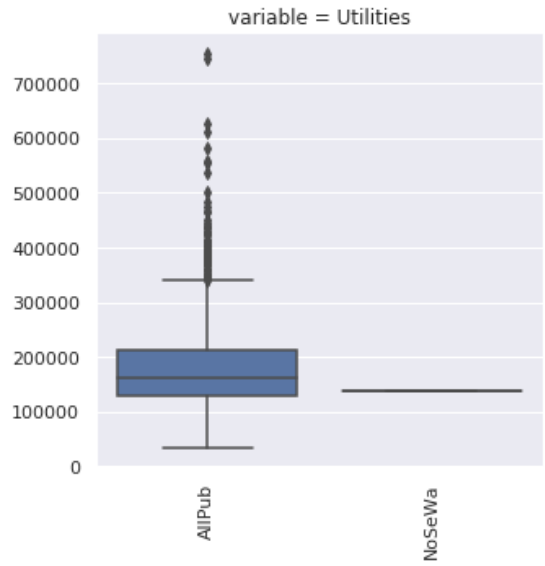

variable = MSZoning
SalePrice
0
100000
200000
300000
400000
500000
600000
700000
RL
RM
C (all)
FV
RH

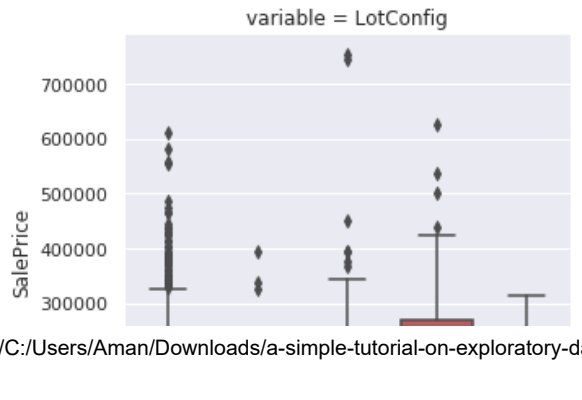
variable = Street
0
100000
200000
300000
400000
500000
600000
700000
Pave
Grvl

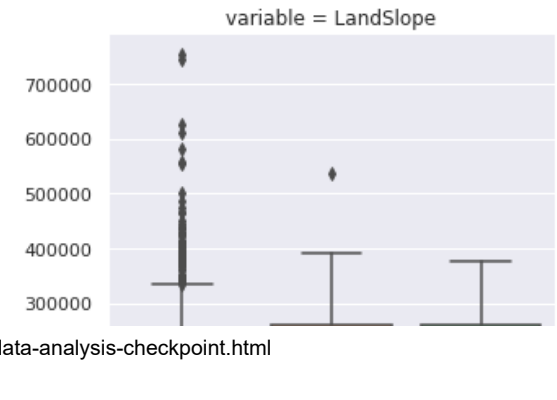

variable = Alley
SalePrice
0
100000
200000
300000
400000
500000
600000
700000
MISSING
Grvl
Pave

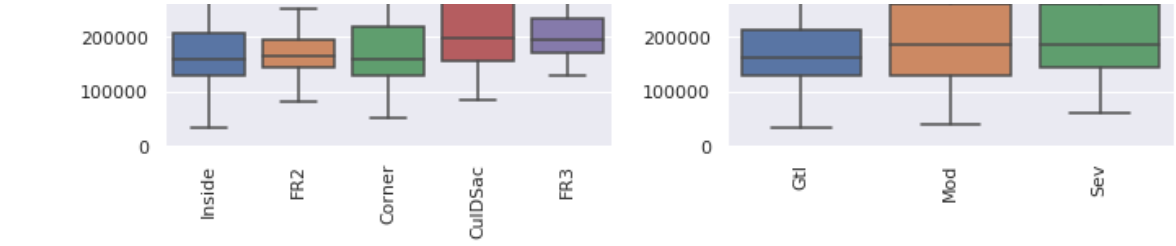

variable = LotShape
0
100000
200000
300000
400000
500000
600000
700000
Reg
IR1
IR2
IR3


variable = LandContour
SalePrice
0
100000
200000
300000
400000
500000
600000
700000
Lvl
Bnk
Low
HLS

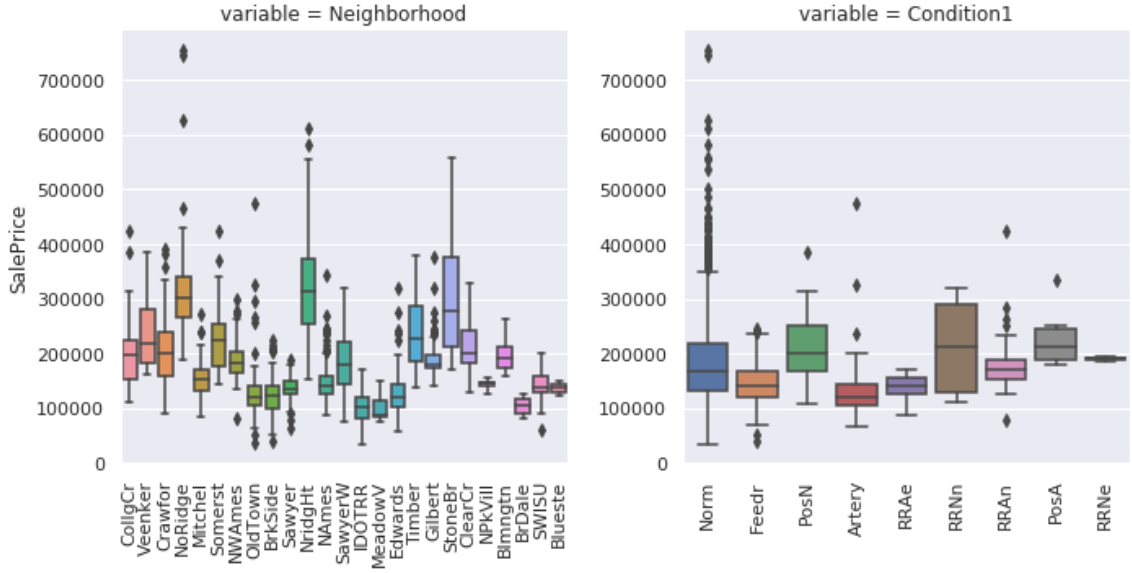

variable = Utilities
0
100000
200000
300000
400000
500000
600000
700000
AllPub
NoSeWa


variable = LotConfig
SalePrice
300000
400000
500000
600000
700000

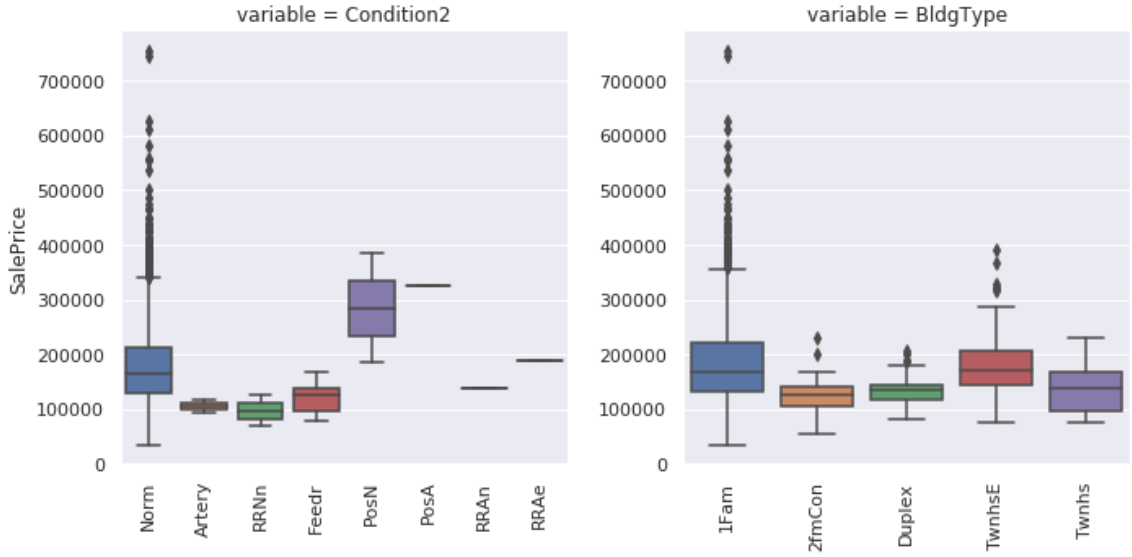

variable = LandSlope
300000
400000
500000
600000
700000


200000
100000
0
Inside
FR2
Corner
CulDSac
FR3

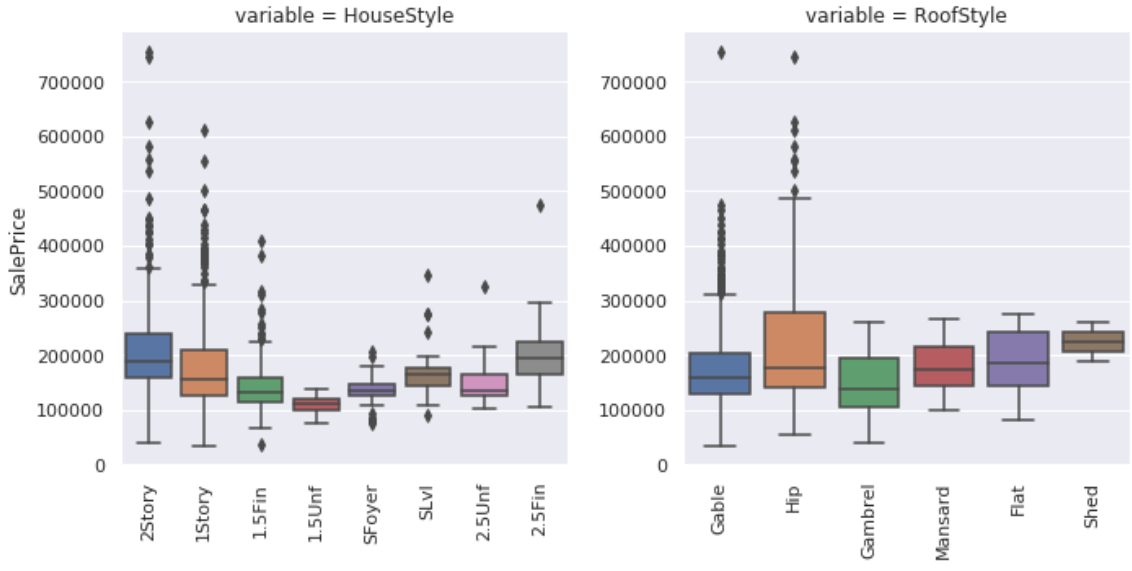
200000
100000
0
Gtl
Mod
Sev


variable = Neighborhood
SalePrice
700000
600000
500000
400000
300000
200000
100000
0
CollgCr
Veenker
Crawfor
NoRidge
Mitchel
Somerst
NWAmes
OldTown
BrkSide
Sawyer
NridgHt
NAmes
SawyerW
IDOTRR
MeadowV
Edwards
Timber
Gilbert
StoneBr
ClearCr
NPkVill
Blmngtn
BrDale
SWISU
Blueste

variable = Condition1
700000
600000
500000
400000
300000
200000
100000
0
Norm
Feedr
PosN
Artery
RRAe
RRNn
RRAn
PosA
RRNe


variable = Condition2
SalePrice
700000
600000
500000
400000
300000
200000
100000
0
Norm
Artery
RRNn
Feedr
PosN
PosA
RRAn
RRAe

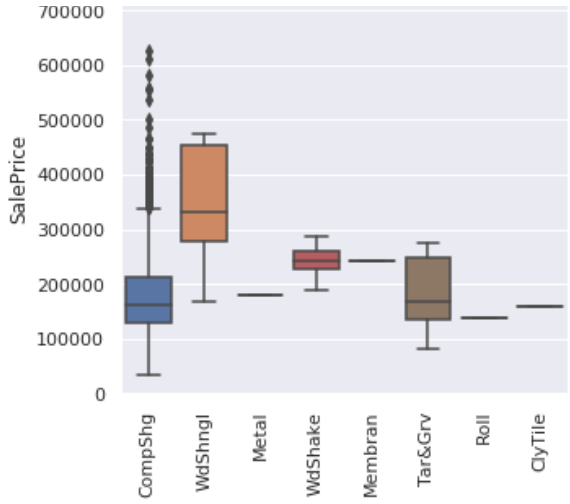
variable = BldgType
700000
600000
500000
400000
300000
200000
100000
0
1Fam
2fmCon
Duplex
TwnhsE
Twnhs

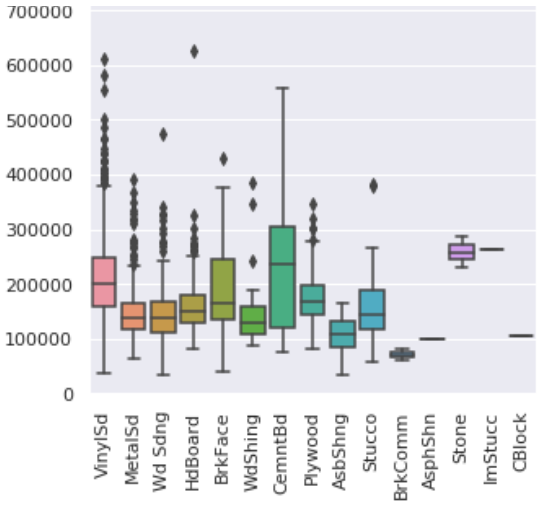

variable = HouseStyle
SalePrice
700000
600000
500000
400000
300000
200000
100000
0
2Story
1Story
1.5Fin
1.5Unf
SFoyer
SLvl
2.5Unf
2.5Fin

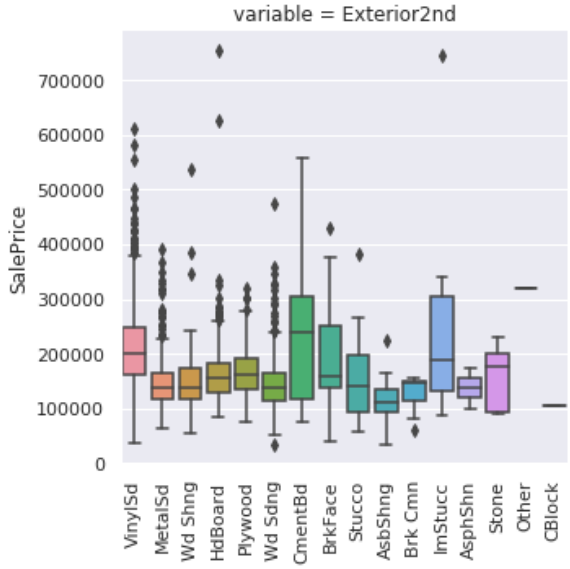
variable = RoofStyle
700000
600000
500000
400000
300000
200000
100000
0
Gable
Hip
Gambrel
Mansard
Flat
Shed

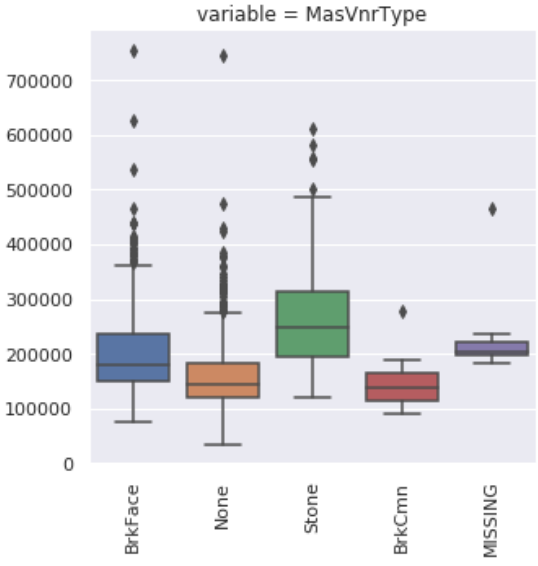

variable = RoofMatl

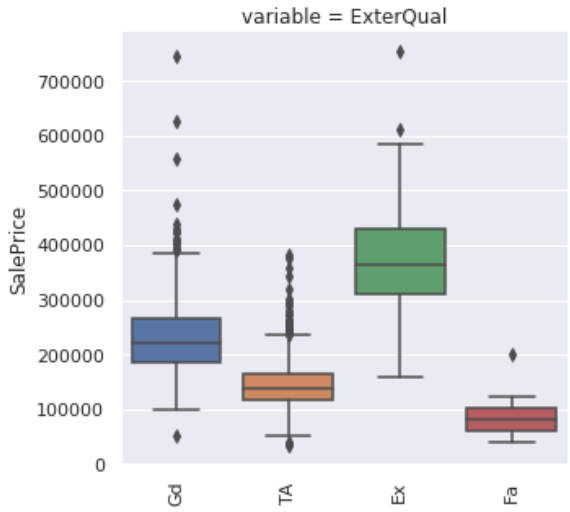
variable = Exterior1st

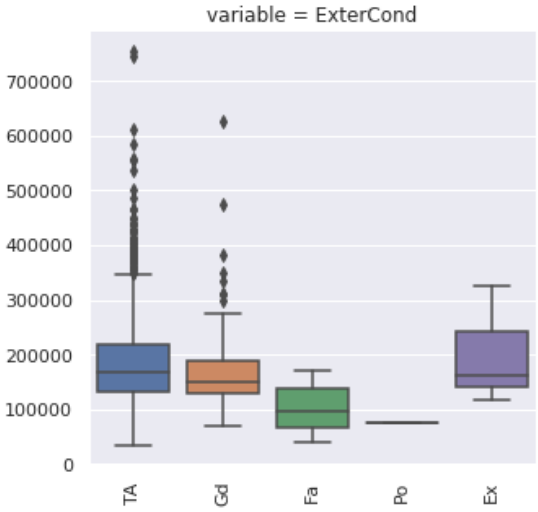

SalePrice
CompShg
WdShngl
Metal
WdShake
Membran
Tar&Grv
Roll
ClyTile

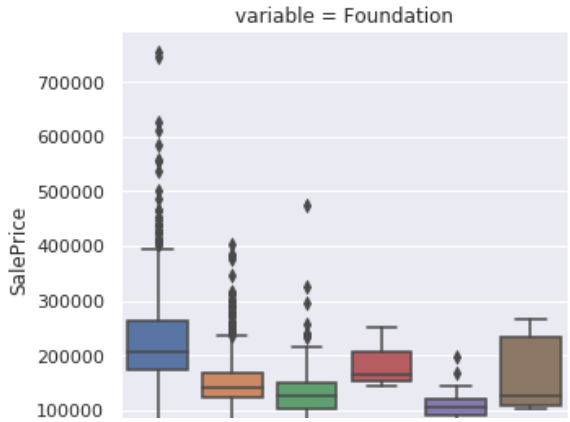

VinylSd
MetalSd
Wd Sdng
HdBoard
BrkFace
WdShing
CemntBd
Plywood
AsbShng
Stucco
BrkComm
AsphShn
Stone
ImStucc
CBlock

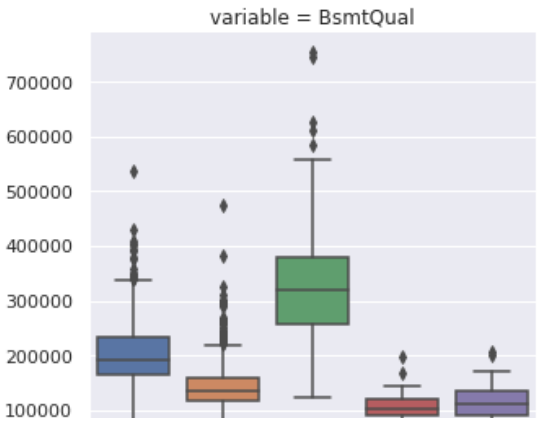

variable = Exterior2nd
SalePrice
VinylSd
MetalSd
Wd Shng
HdBoard
Plywood
Wd Sdng
CmentBd
BrkFace
Stucco
AsbShng
Brk Cmn
ImStucc
AsphShn
Stone
Other
CBlock

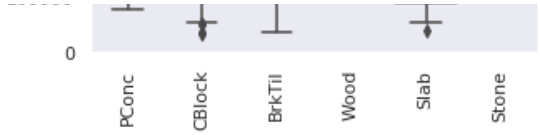

variable = MasVnrType
BrkFace
None
Stone
BrkCmn
MISSING


variable = ExterQual
SalePrice
Gd
TA
Ex
Fa

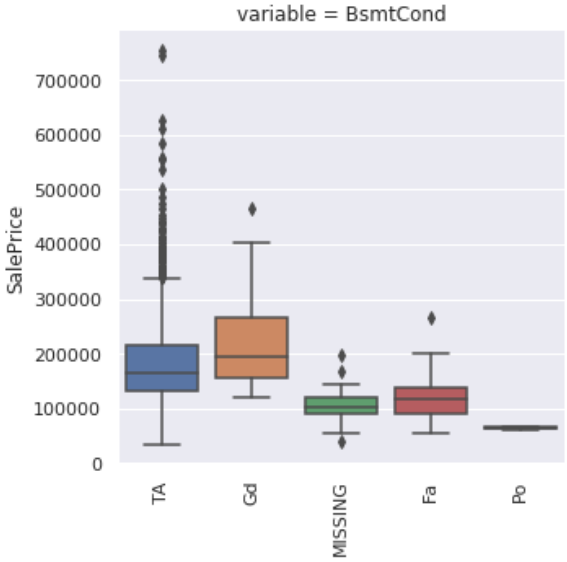

variable = ExterCond
TA
Gd
Fa
Po
Ex

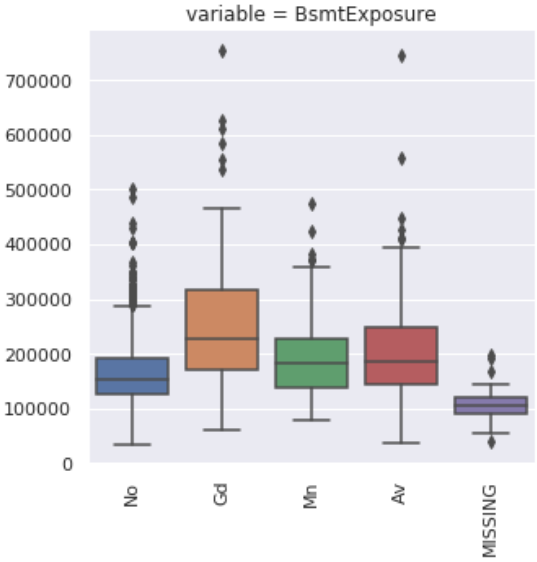

variable = Foundation
SalePrice

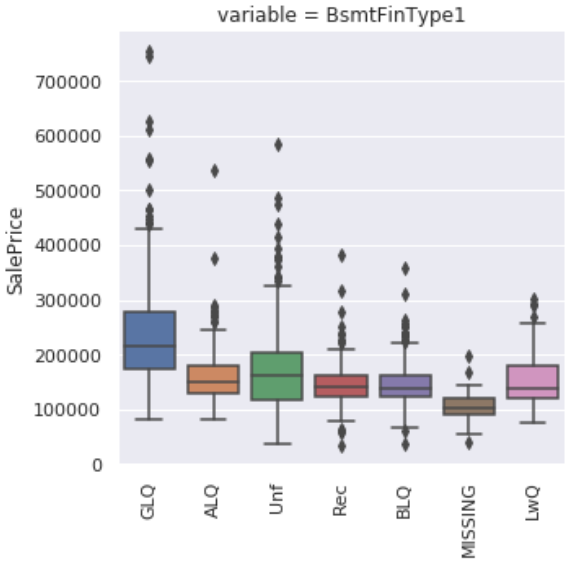

variable = BsmtQual

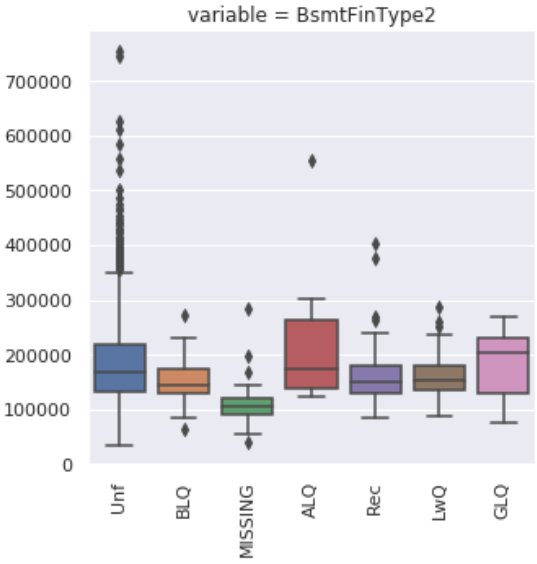

0
PConc
CBlock
BrkTil
Wood
Slab
Stone

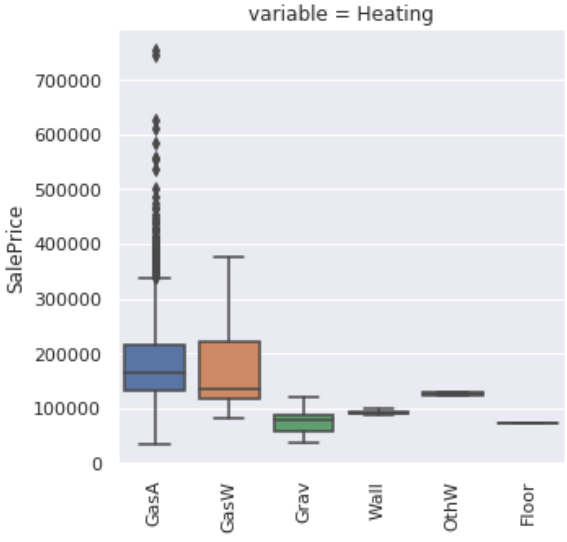

0
Gd
TA
Ex
MISSING
Fa

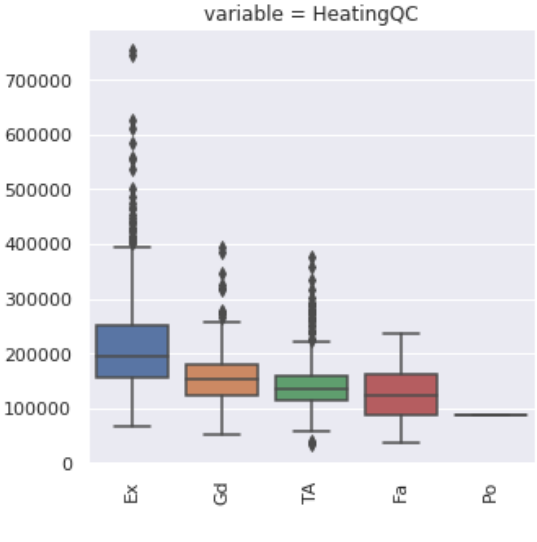

variable = BsmtCond
SalePrice
700000
600000
500000
400000
300000
200000
100000
0
TA
Gd
MISSING
Fa
Po

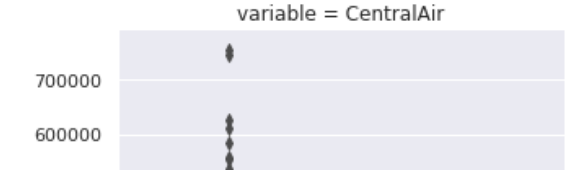

variable = BsmtExposure
700000
600000
500000
400000
300000
200000
100000
0
No
Gd
Mn
Av
MISSING

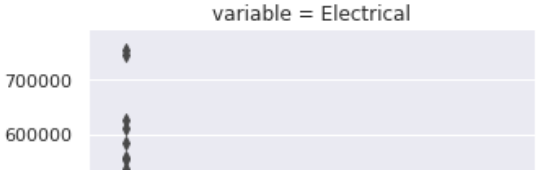

variable = BsmtFinType1
SalePrice
700000
600000
500000
400000
300000
200000
100000
0
GLQ
ALQ
Unf
Rec
BLQ
MISSING
LwQ

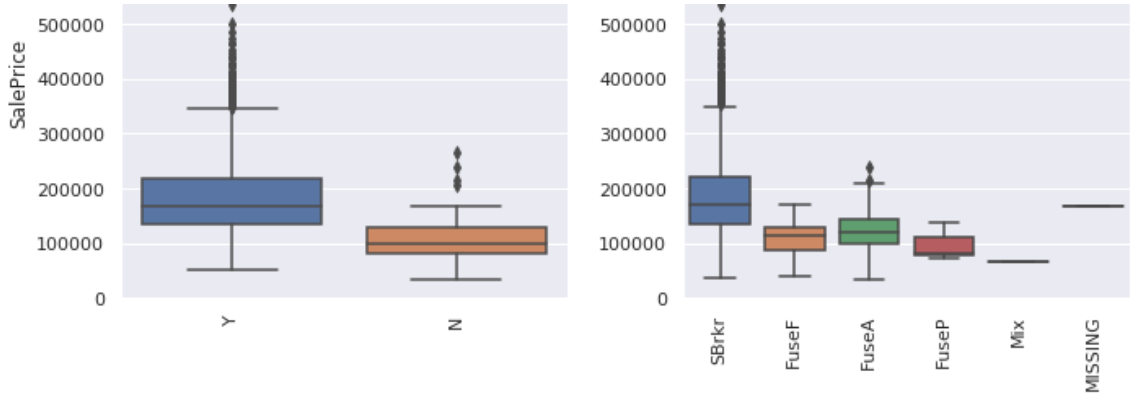

variable = BsmtFinType2
700000
600000
500000
400000
300000
200000
100000
0
Unf
BLQ
MISSING
ALQ
Rec
LwQ
GLQ

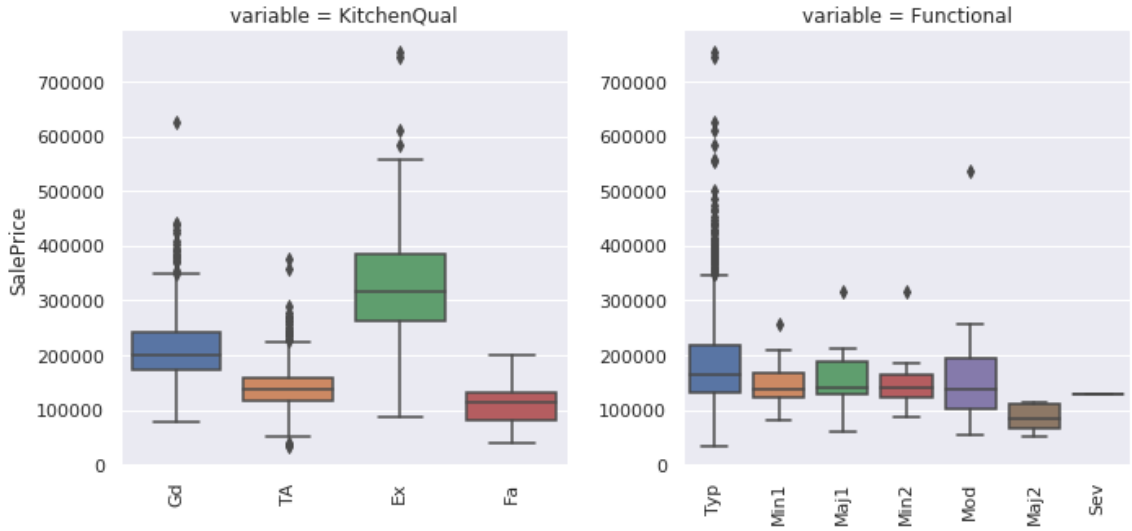

variable = Heating
SalePrice
700000
600000
500000
400000
300000
200000
100000
0
GasA
GasW
Grav
Wall
OthW
Floor

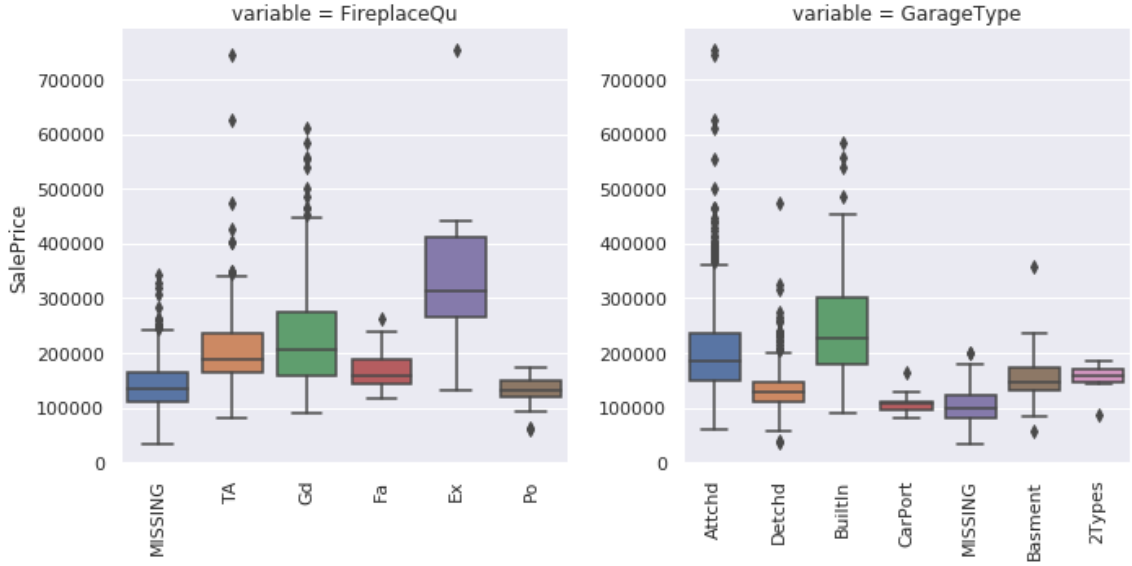

variable = HeatingQC
700000
600000
500000
400000
300000
200000
100000
0
Ex
Gd
TA
Fa
Po

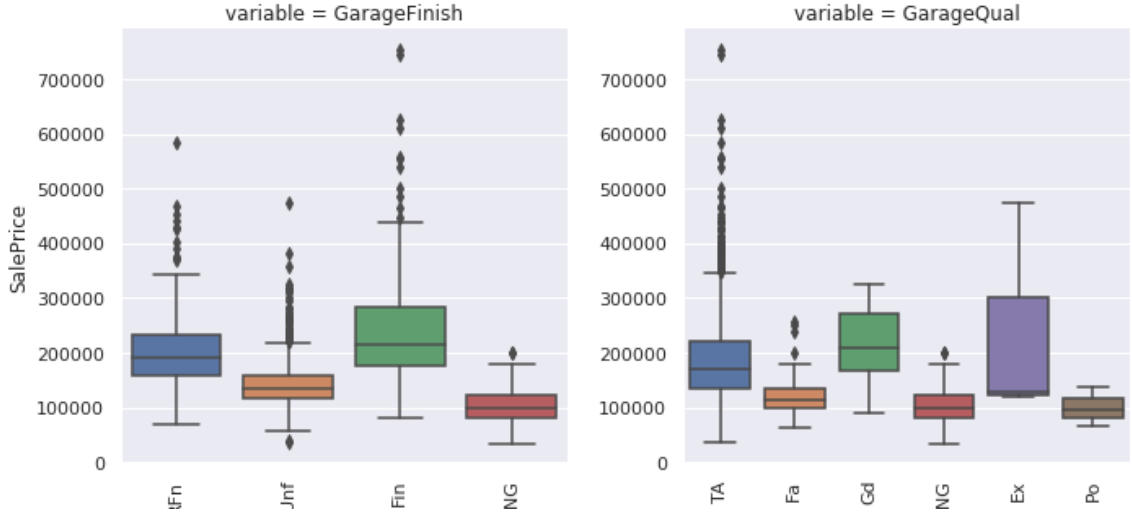

variable = CentralAir
700000
600000

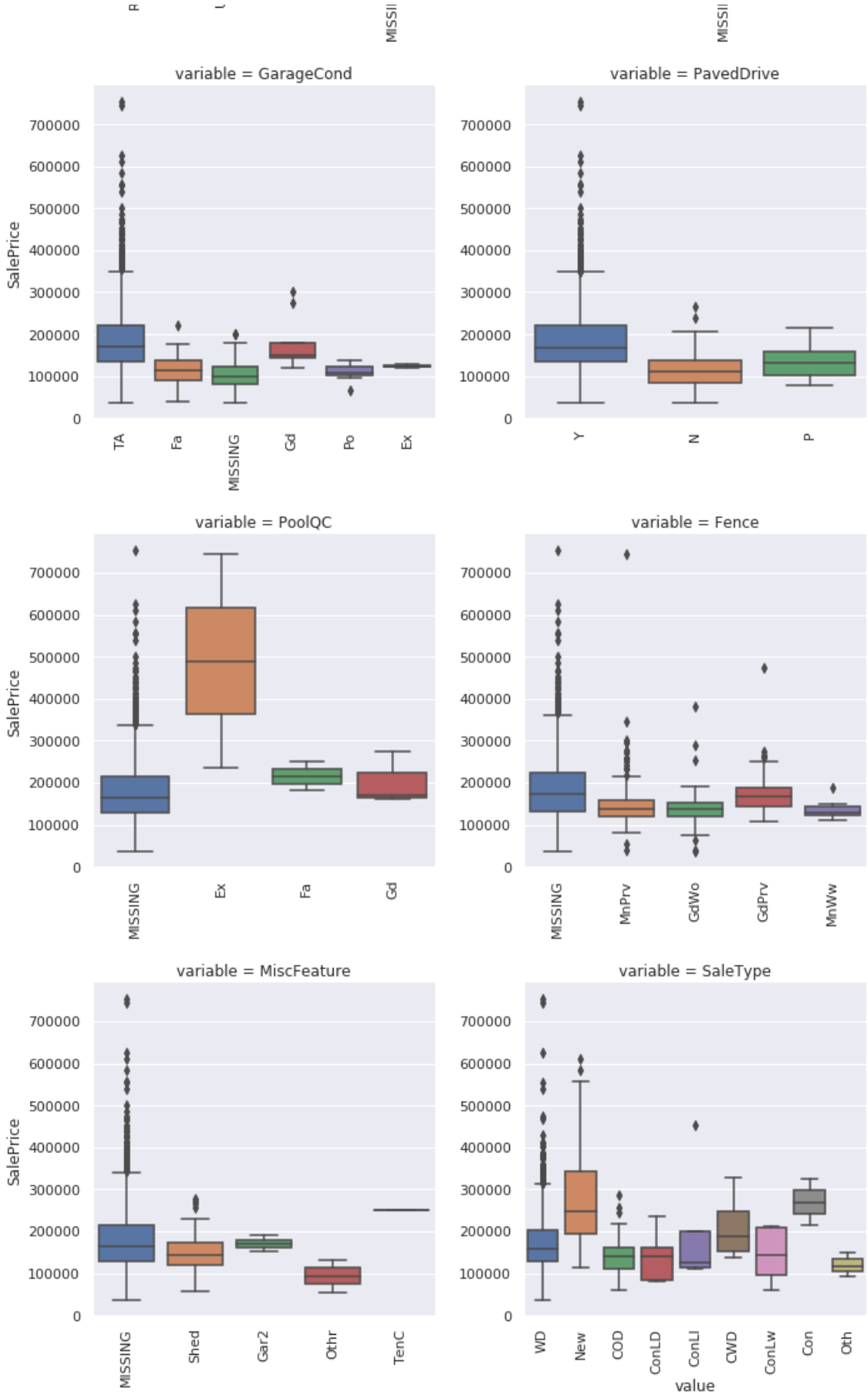

variable = Electrical
700000
600000

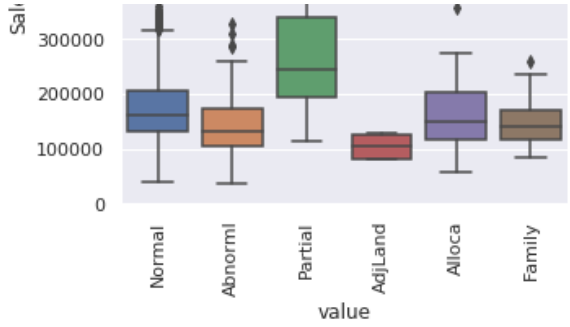

SalePrice
0
100000
200000
300000
400000
500000
Y
N
SBrkr
FuseF
FuseA
FuseP
Mix
MISSING


variable = KitchenQual
variable = Functional
SalePrice
0
100000
200000
300000
400000
500000
600000
700000
Gd
TA
Ex
Fa
Typ
Min1
Maj1
Min2
Mod
Maj2
Sev


variable = FireplaceQu
variable = GarageType
SalePrice
0
100000
200000
300000
400000
500000
600000
700000
MISSING
TA
Gd
Fa
Ex
Po
Attchd
Detchd
BuiltIn
CarPort
MISSING
Basment
2Types


variable = GarageFinish
variable = GarageQual
SalePrice
0
100000
200000
300000
400000
500000
600000
700000
Fin
Unf
Fin
NG
TA
Fa
Gd
NG
Ex
Po


variable = GarageCond
SalePrice
TA
Fa
MISSING
Gd
Po
Ex
variable = PavedDrive
Y
N
P
variable = PoolQC
MISSING
Ex
Fa
Gd
variable = Fence
MISSING
MnPrv
GdWo
GdPrv
MnWw
variable = MiscFeature
MISSING
Shed
Gar2
Othr
TenC
variable = SaleType
WD
New
COD
ConLD
ConLI
CWD
ConLw
Con
Oth
value
variable = SaleCondition

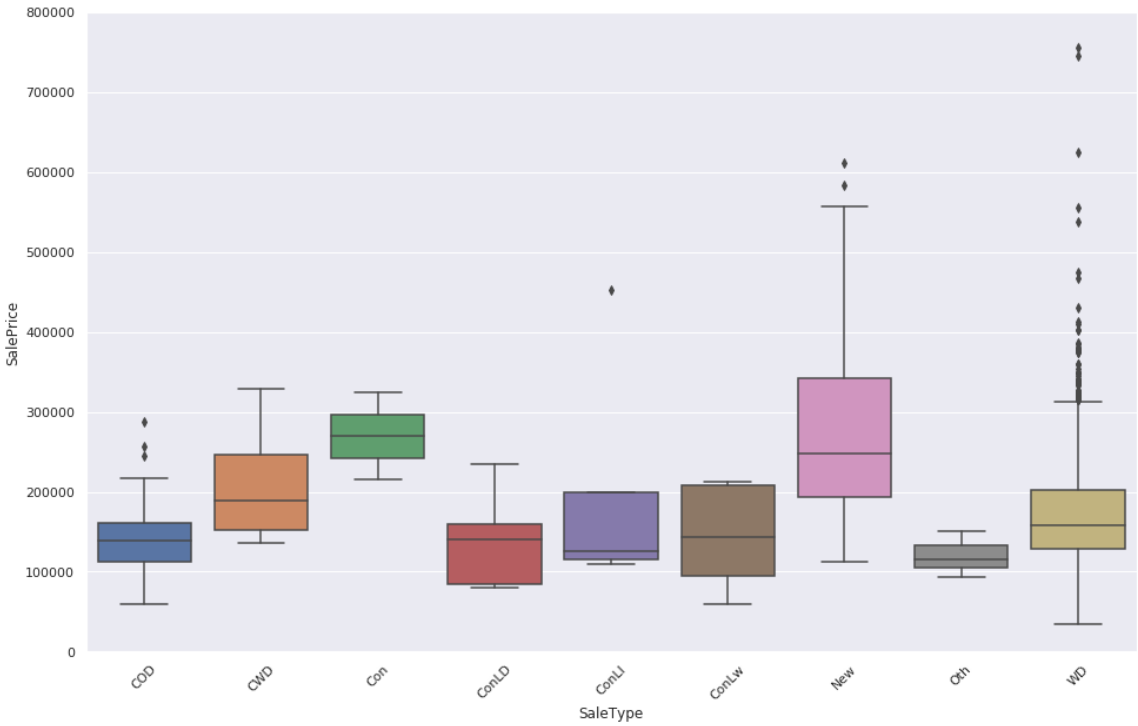


Housing Price vs Sales

- Sale Type & Condition
- Sales Seasonality

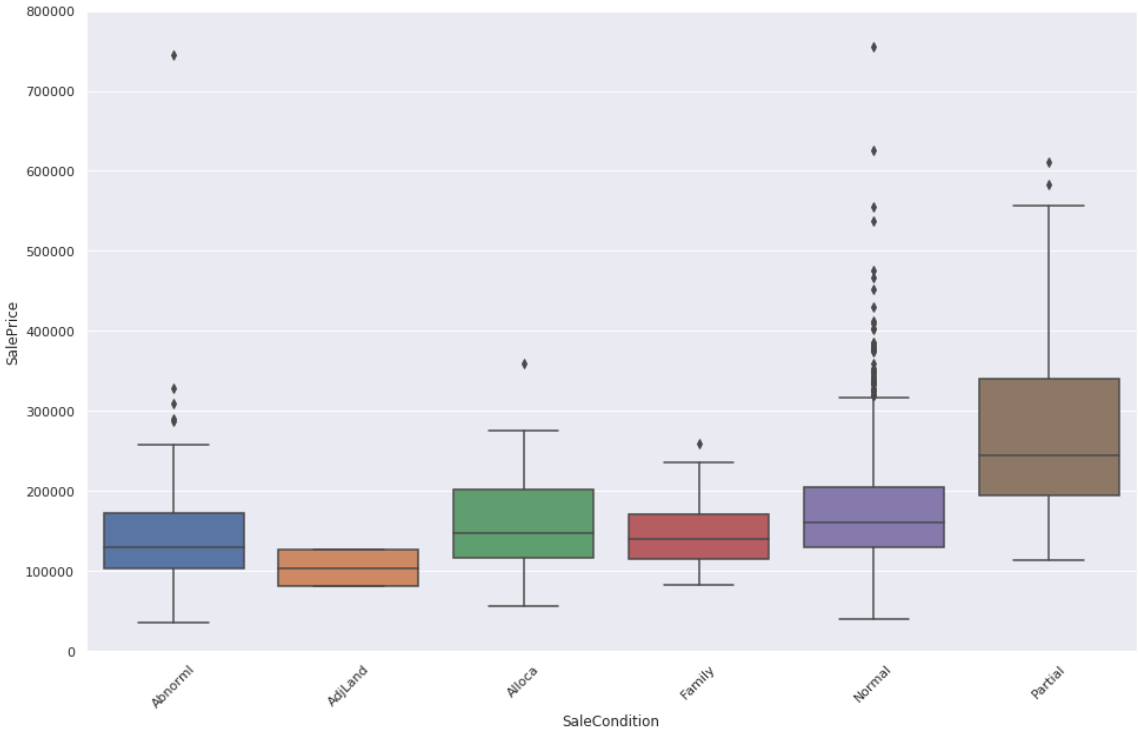
In [38]:

```
var = 'SaleType'
data = pd.concat([train['SalePrice'], train[var]], axis=1)
f, ax = plt.subplots(figsize=(16, 10))
fig = sns.boxplot(x=var, y="SalePrice", data=data)
fig.axis(ymin=0, ymax=800000);
xt = plt.xticks(rotation=45)
```



In [39]:

```
var = 'SaleCondition'
data = pd.concat([train['SalePrice'], train[var]], axis=1)
f, ax = plt.subplots(figsize=(16, 10))
fig = sns.boxplot(x=var, y="SalePrice", data=data)
fig.axis(ymin=0, ymax=800000);
xt = plt.xticks(rotation=45)
```



ViolinPlot - Functional vs.SalePrice

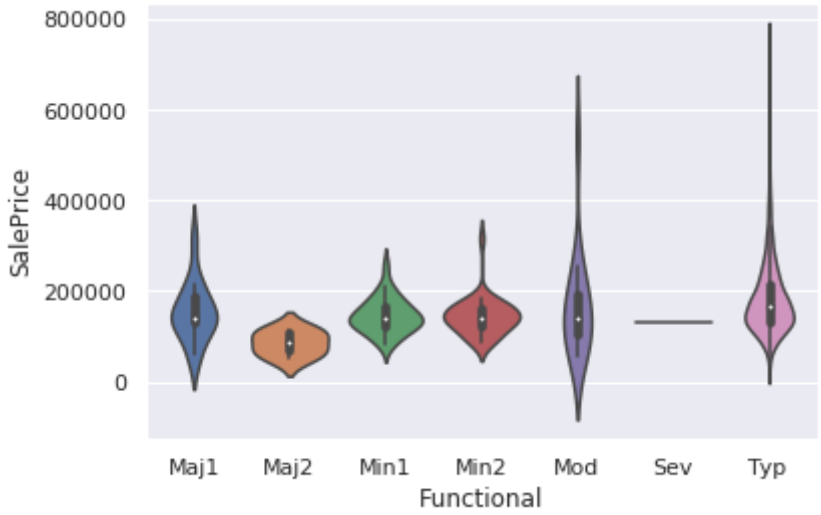
In [40]:

```
sns.violinplot('Functional', 'SalePrice', data = train)
```

```
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWa
rning: Using a non-tuple sequence for multidimensional indexing is depreca
ted; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will
be interpreted as an array index, `arr[np.array(seq)]`, which will result
either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[40]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f00bb1b7470>
```



FactorPlot - FirePlaceQC vs. SalePrice

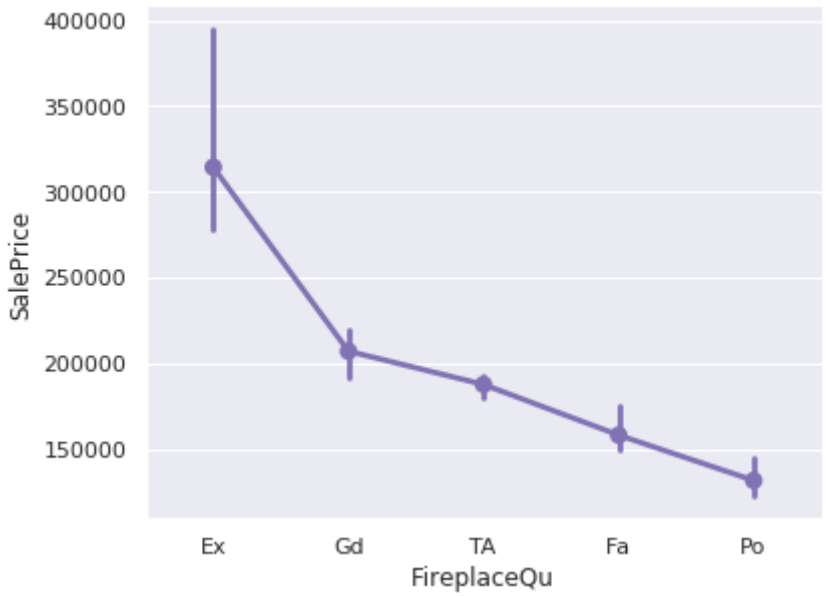
In [41]:

```
sns.factorplot('FireplaceQu', 'SalePrice', data = train, color = 'm', \
               estimator = np.median, order = ['Ex', 'Gd', 'TA', 'Fa', 'Po'], size = 4.
5,  aspect=1.35)
```

```
/opt/conda/lib/python3.6/site-packages/seaborn/categorical.py:3666: UserWa
rning: The `factorplot` function has been renamed to `catplot`. The origin
al name will be removed in a future release. Please update your code. Note
that the default `kind` in `factorplot` (`'point'`) has changed `'strip'`
in `catplot`.
  warnings.warn(msg)
/opt/conda/lib/python3.6/site-packages/seaborn/categorical.py:3672: UserWa
rning: The `size` paramter has been renamed to `height`; please update you
r code.
  warnings.warn(msg, UserWarning)
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWa
rning: Using a non-tuple sequence for multidimensional indexing is depreca
ted; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will
be interpreted as an array index, `arr[np.array(seq)]`, which will result
either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[41]:

```
<seaborn.axisgrid.FacetGrid at 0x7f00bb21bd30>
```



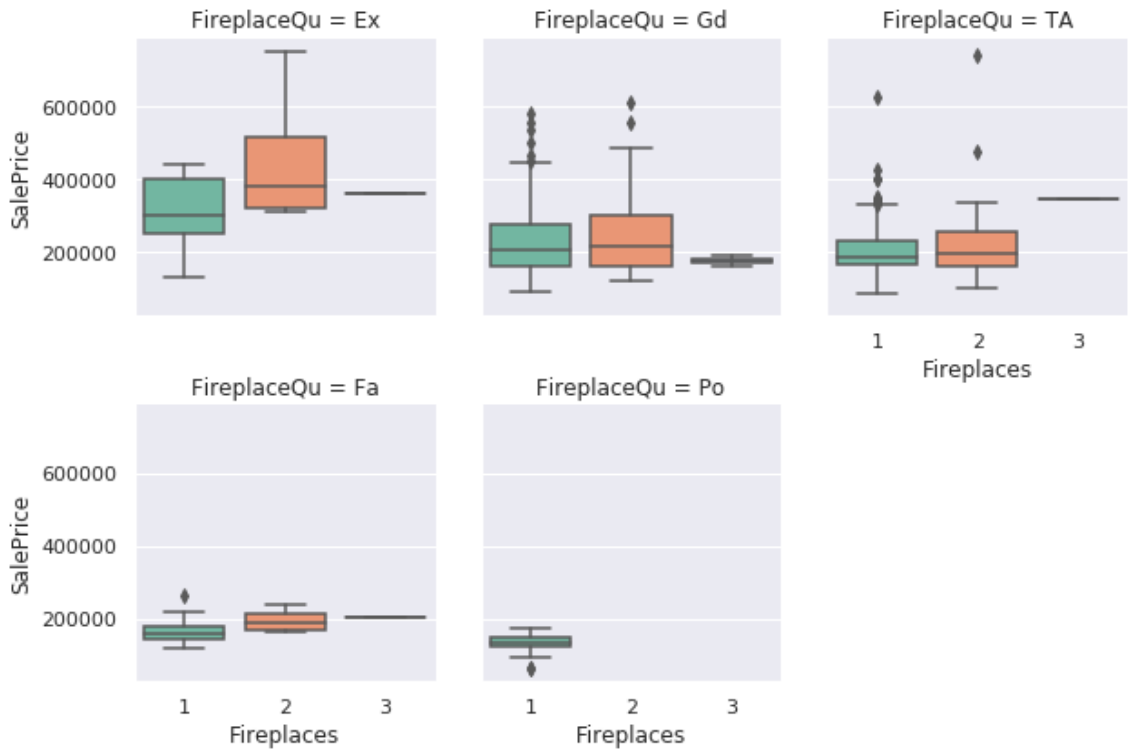
Facet Grid Plot - FirePlace QC vs.SalePrice

In [42]:

```
g = sns.FacetGrid(train, col = 'FireplaceQu', col_wrap = 3, col_order=['Ex', 'Gd', 'TA'
, 'Fa', 'Po'])
g.map(sns.boxplot, 'Fireplaces', 'SalePrice', order = [1, 2, 3], palette = 'Set2')
```

Out[42]:

```
<seaborn.axisgrid.FacetGrid at 0x7f00bc8edac8>
```

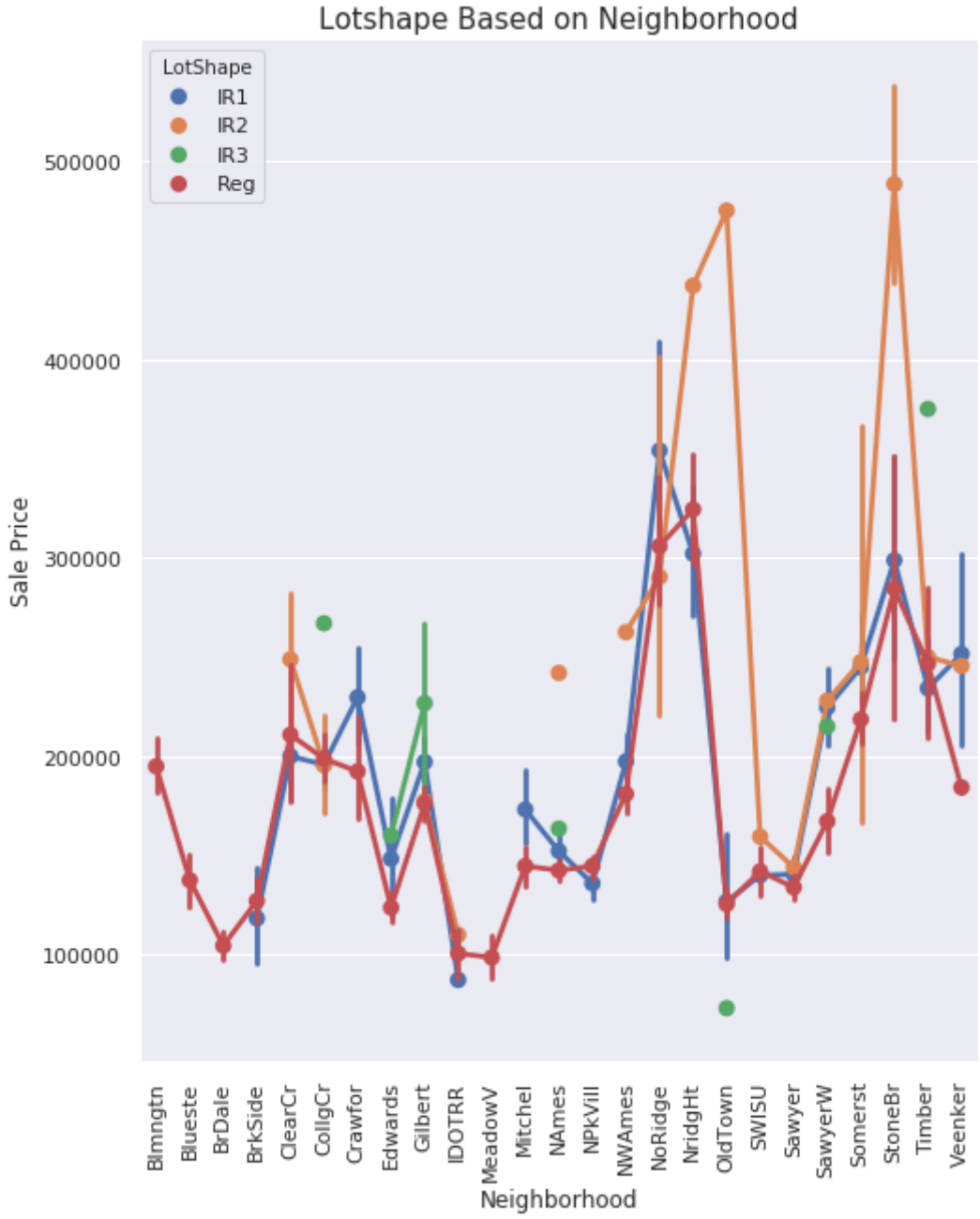


PointPlot

In [43]:

```
plt.figure(figsize=(8,10))
g1 = sns.pointplot(x='Neighborhood', y='SalePrice',
                   data=train, hue='LotShape')
g1.set_xticklabels(g1.get_xticklabels(),rotation=90)
g1.set_title("Lotshape Based on Neighborhood", fontsize=15)
g1.set_xlabel("Neighborhood")
g1.set_ylabel("Sale Price", fontsize=12)
plt.show()
```

```
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWa
rning: Using a non-tuple sequence for multidimensional indexing is depreca
ted; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will
be interpreted as an array index, `arr[np.array(seq)]`, which will result
either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



Missing Value Analysis

We will first check the percentage of missing values present in each feature

In [44]:

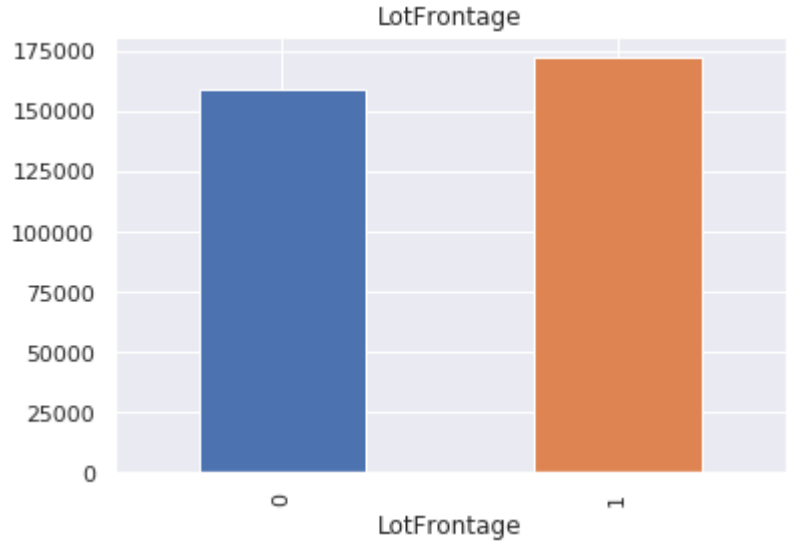
```
data = pd.read_csv("../input/train.csv")
features_with_na=[features for features in data.columns if data[features].isnull().sum
()>1]
for feature in features_with_na:
    print(feature, np.round(data[feature].isnull().mean(), 4),  ' % of Missing Values')
```

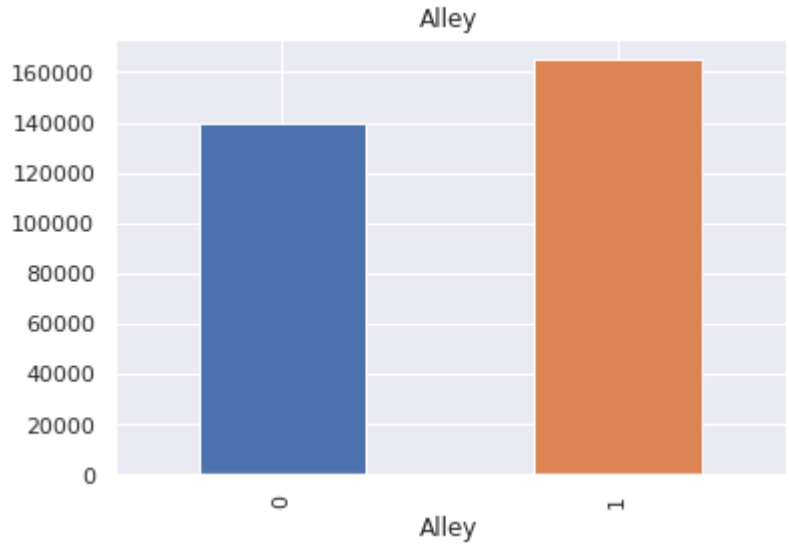
```
LotFrontage 0.1774  % of Missing Values
Alley 0.9377  % of Missing Values
MasVnrType 0.0055  % of Missing Values
MasVnrArea 0.0055  % of Missing Values
BsmtQual 0.0253  % of Missing Values
BsmtCond 0.0253  % of Missing Values
BsmtExposure 0.026  % of Missing Values
BsmtFinType1 0.0253  % of Missing Values
BsmtFinType2 0.026  % of Missing Values
FireplaceQu 0.4726  % of Missing Values
GarageType 0.0555  % of Missing Values
GarageYrBlt 0.0555  % of Missing Values
GarageFinish 0.0555  % of Missing Values
GarageQual 0.0555  % of Missing Values
GarageCond 0.0555  % of Missing Values
PoolQC 0.9952  % of Missing Values
Fence 0.8075  % of Missing Values
MiscFeature 0.963  % of Missing Values
```

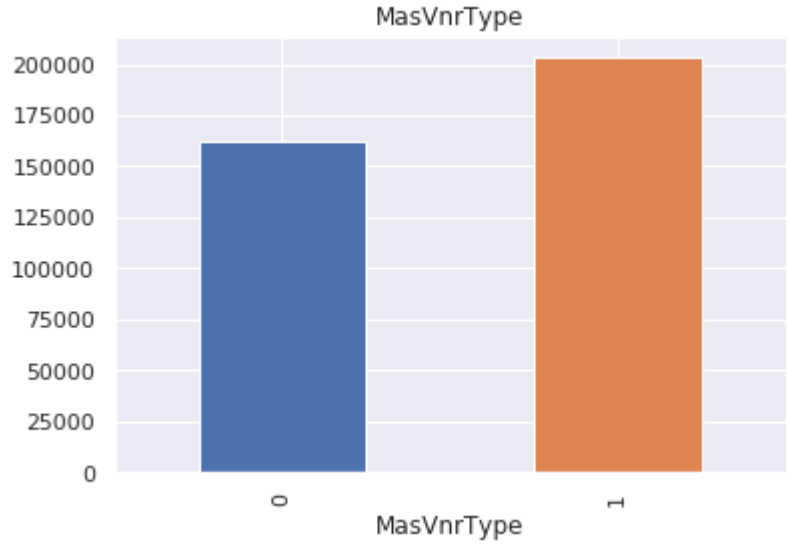
Now let us plot with the features with missing values vs Sales Price to find some insights about the data

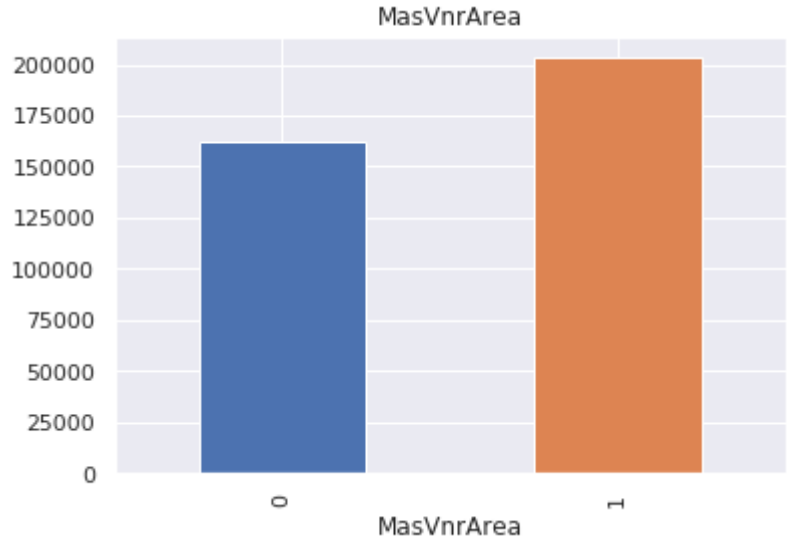
In [45]:

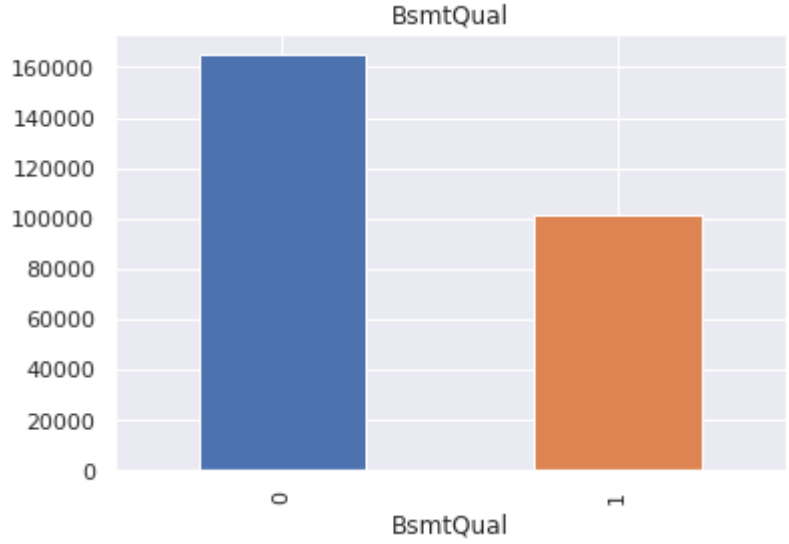
```
for feature in features_with_na:
    dataset = data.copy()
    dataset[feature] = np.where(dataset[feature].isnull(), 1, 0)
     # Calculate the mean of SalePrice where the information is missing or present
    dataset.groupby(feature)['SalePrice'].median().plot.bar()
    plt.title(feature)
    plt.show()
```

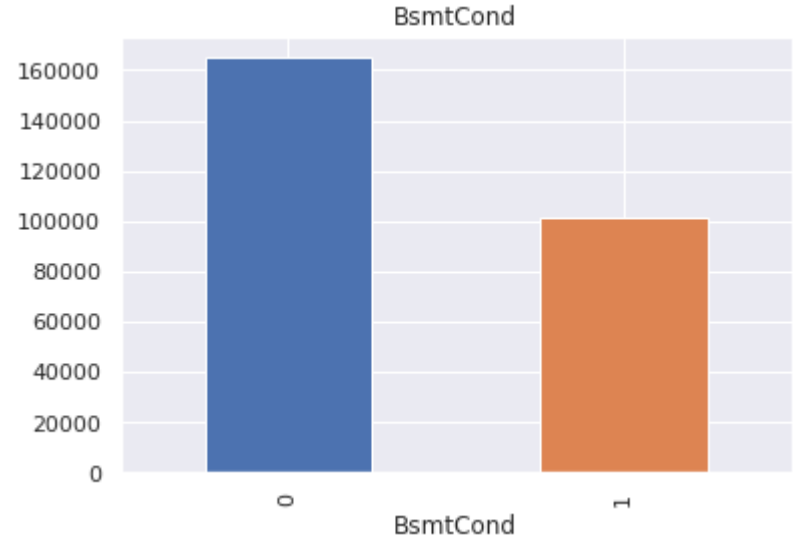

LotFrontage
175000
150000
125000
100000
75000
50000
25000
0
0
1
LotFrontage

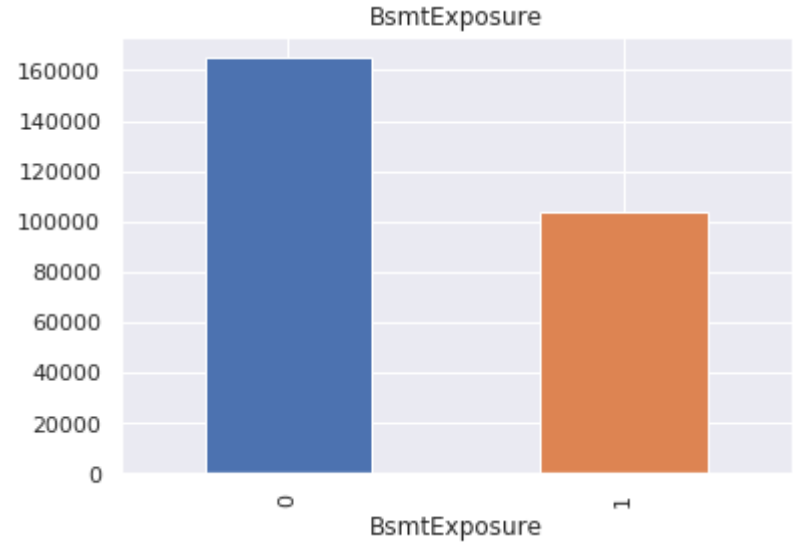

Alley
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
Alley

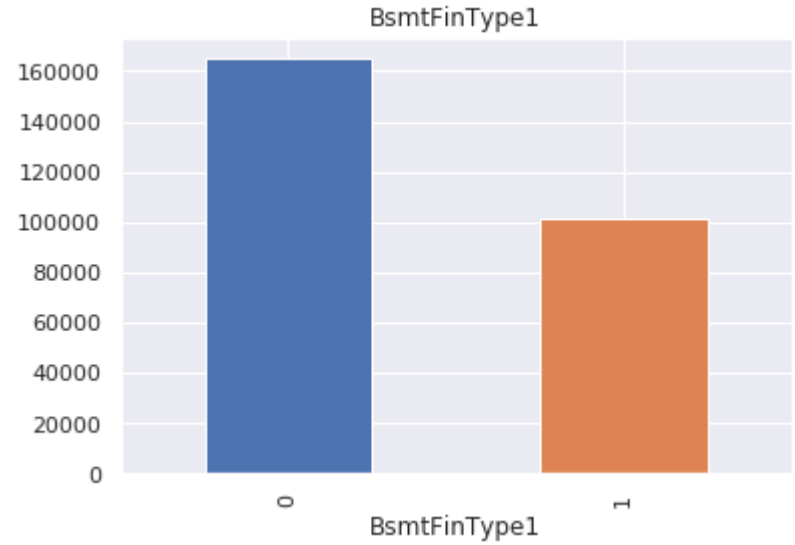

MasVnrType
200000
175000
150000
125000
100000
75000
50000
25000
0
0
1
MasVnrType

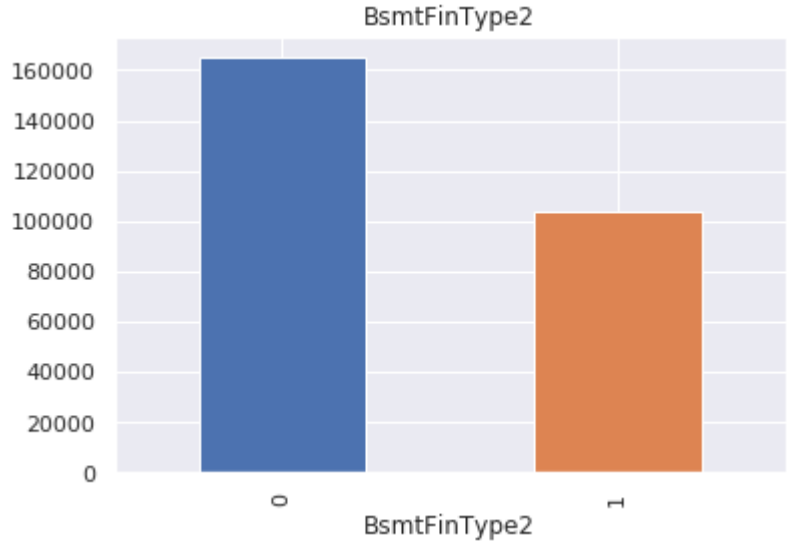

MasVnrArea
200000
175000
150000
125000
100000
75000
50000
25000
0
0
1
MasVnrArea

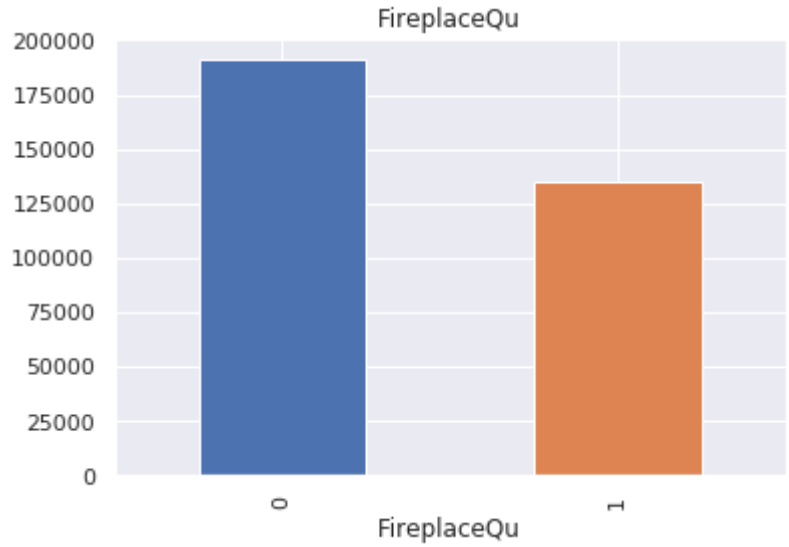

BsmtQual
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
BsmtQual

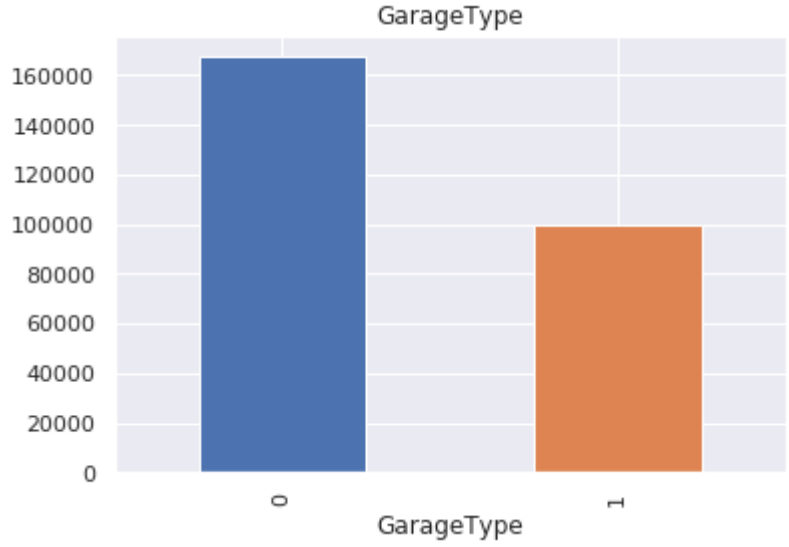

BsmtCond
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
BsmtCond

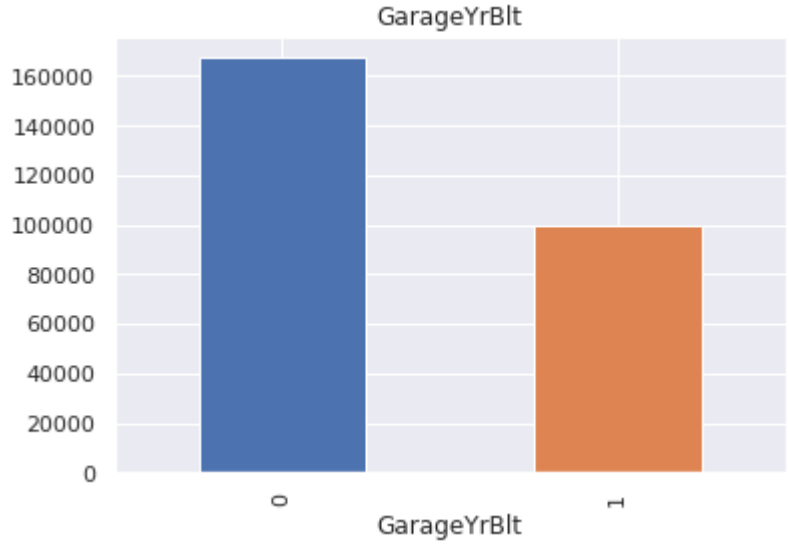

BsmtExposure
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
BsmtExposure

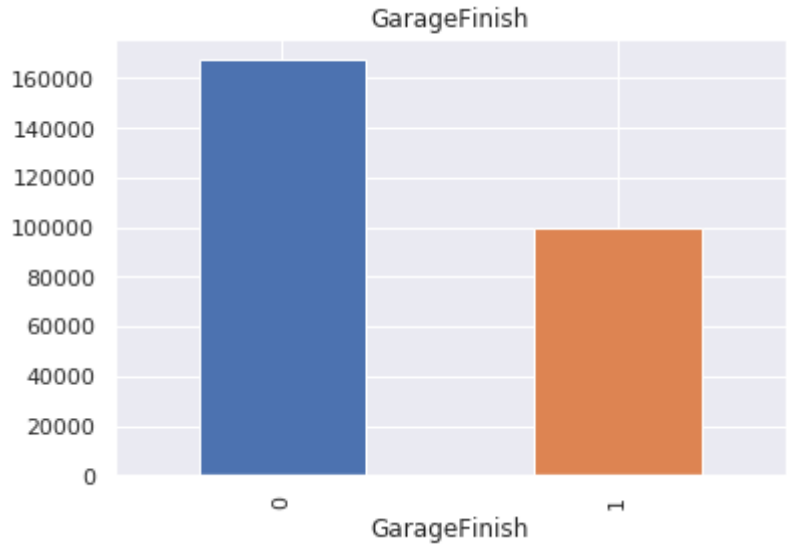

BsmtFinType1
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
BsmtFinType1

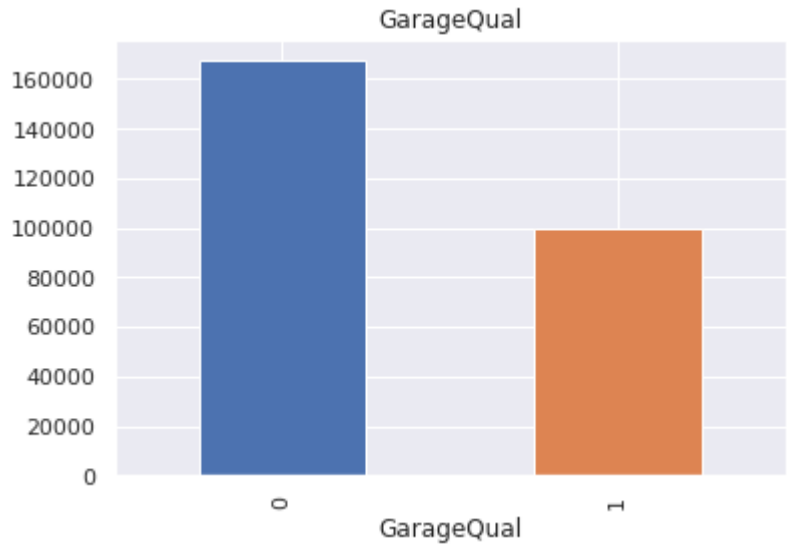

BsmtFinType2
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
BsmtFinType2

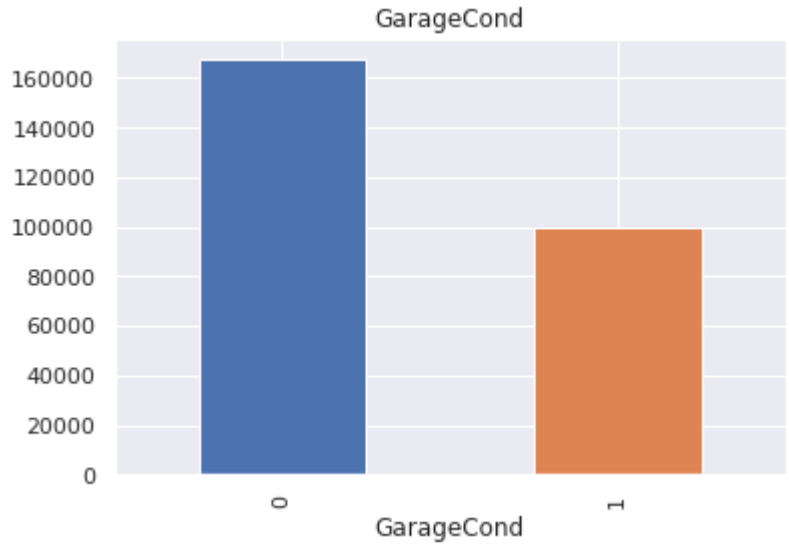

FireplaceQu
200000
175000
150000
125000
100000
75000
50000
25000
0
0
1
FireplaceQu

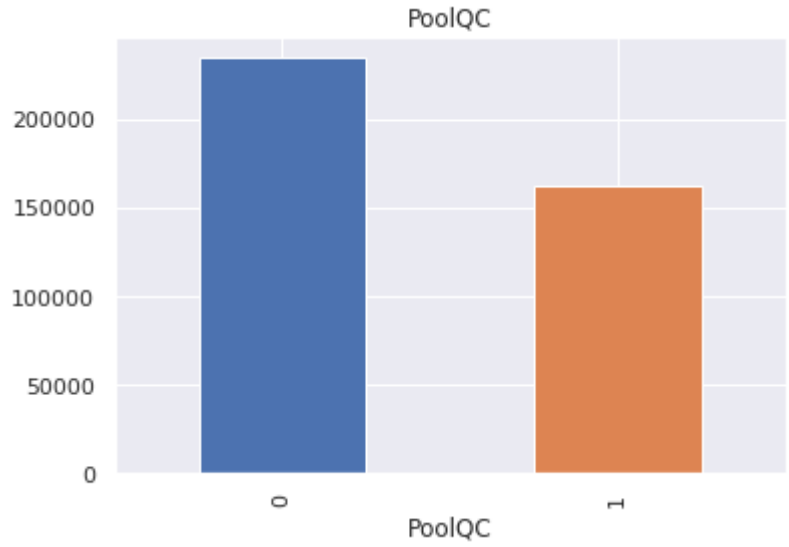

GarageType
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
GarageType

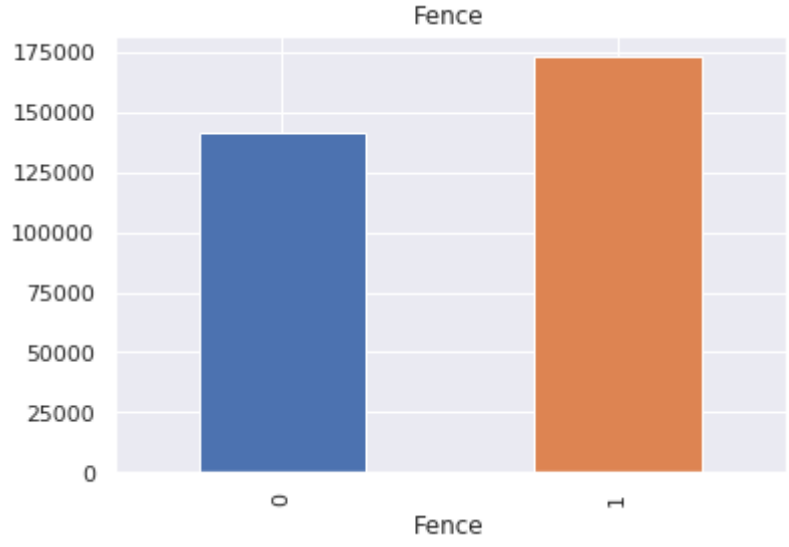

GarageYrBlt
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
GarageYrBlt

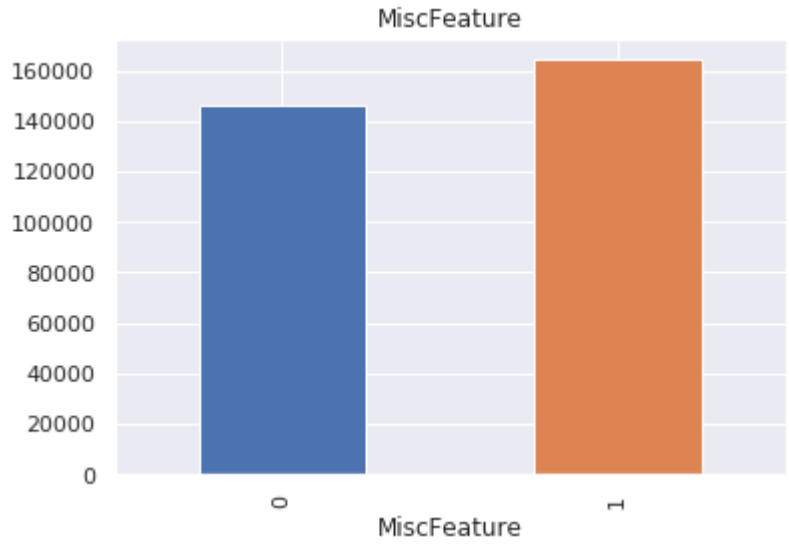

GarageFinish
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
GarageFinish


GarageQual
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
GarageQual


GarageCond
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
GarageCond


PoolQC
200000
150000
100000
50000
0
0
1
PoolQC


Fence
175000
150000
125000
100000
75000
50000
25000
0
0
1
Fence



Numeric Features

In [46]:

```
total = numeric_features.isnull().sum().sort_values(ascending=False)
percent = (numeric_features.isnull().sum()/numeric_features.isnull().count()).sort_valu
es(ascending=False)
missing_data = pd.concat([total, percent], axis=1,join='outer', keys=['Total Missing Co
unt', '% of Total Observations'])
missing_data.index.name =' Numeric Feature'
missing_data.head(20)
```

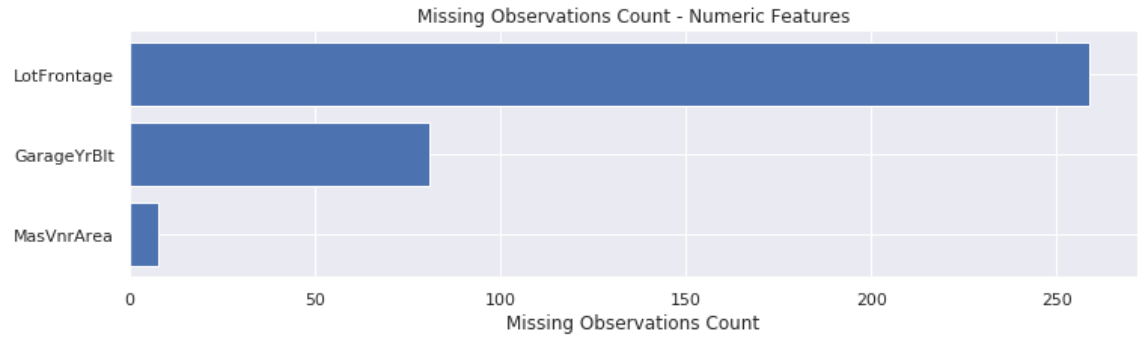
Out[46]:

Numeric Feature	Total Missing Count	% of Total Observations
LotFrontage	259	0.177397
GarageYrBlt	81	0.055479
MasVnrArea	8	0.005479
BsmtFinSF1	0	0.000000
LowQualFinSF	0	0.000000
2ndFlrSF	0	0.000000
1stFlrSF	0	0.000000
TotalBsmtSF	0	0.000000
BsmtUnfSF	0	0.000000
BsmtFinSF2	0	0.000000
SalePrice	0	0.000000
BsmtFullBath	0	0.000000
YearRemodAdd	0	0.000000
YearBuilt	0	0.000000
OverallCond	0	0.000000
OverallQual	0	0.000000
LotArea	0	0.000000
MSSubClass	0	0.000000
GrLivArea	0	0.000000
BsmtHalfBath	0	0.000000

Missing values for all numeric features in Bar chart Representation

In [47]:

```
missing_values = numeric_features.isnull().sum(axis=0).reset_index()
missing_values.columns = ['column_name', 'missing_count']
missing_values = missing_values.loc[missing_values['missing_count']>0]
missing_values = missing_values.sort_values(by='missing_count')
ind = np.arange(missing_values.shape[0])
width = 0.1
fig, ax = plt.subplots(figsize=(12,3))
rects = ax.barh(ind, missing_values.missing_count.values, color='b')
ax.set_yticks(ind)
ax.set_yticklabels(missing_values.column_name.values, rotation='horizontal')
ax.set_xlabel("Missing Observations Count")
ax.set_title("Missing Observations Count - Numeric Features")
plt.show()
```



Categorical Features

Let us look at the missing values in categorical features in detail

In [48]:

```
total = categorical_features.isnull().sum().sort_values(ascending=False)
percent = (categorical_features.isnull().sum()/categorical_features.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1,join='outer', keys=['Total Missing Count', ' % of Total Observations'])
missing_data.index.name ='Feature'
missing_data.head(20)
```

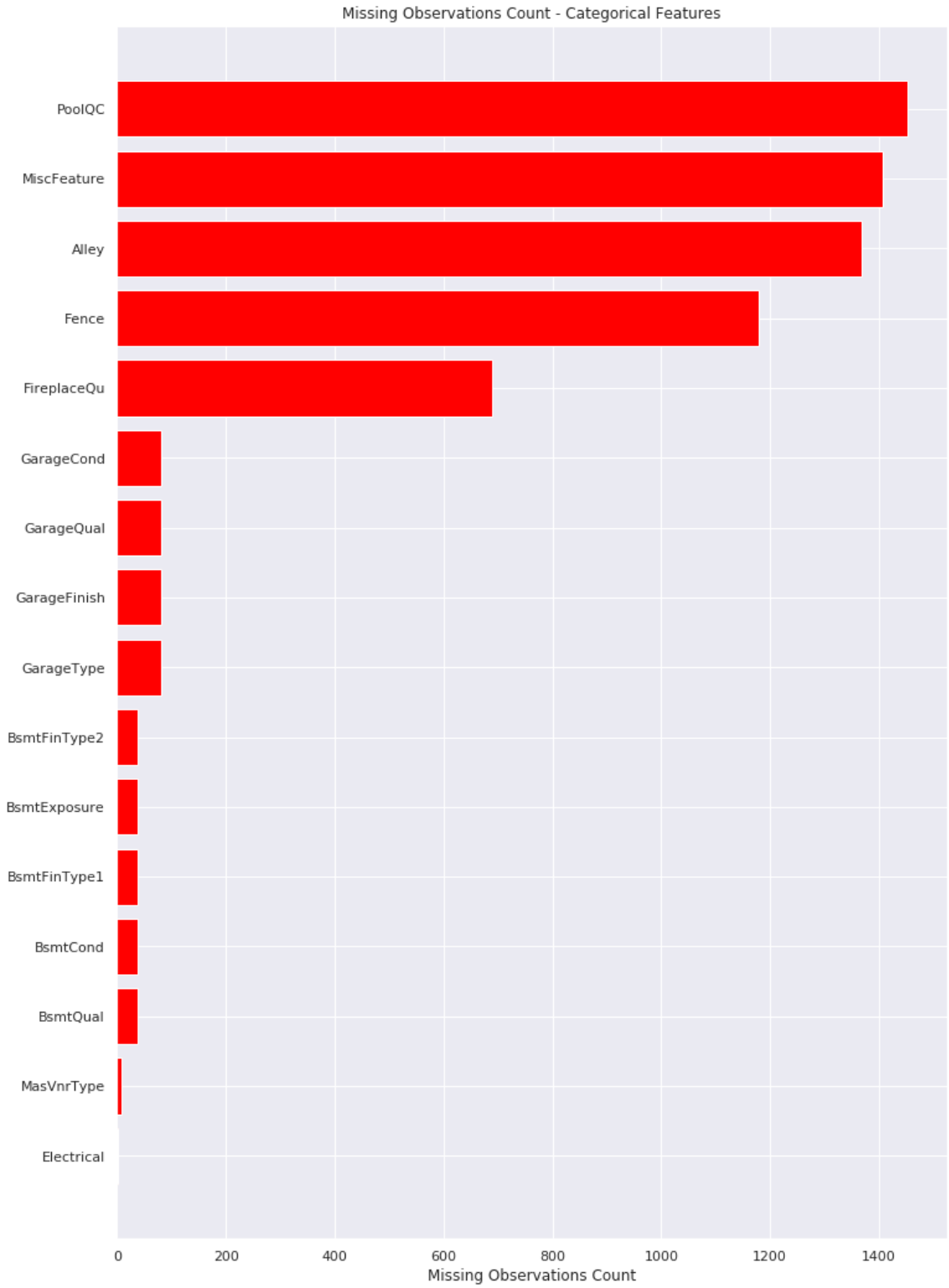
Out[48]:

Feature	Total Missing Count	% of Total Observations
PoolQC	1453	0.995205
MiscFeature	1406	0.963014
Alley	1369	0.937671
Fence	1179	0.807534
FireplaceQu	690	0.472603
GarageCond	81	0.055479
GarageQual	81	0.055479
GarageFinish	81	0.055479
GarageType	81	0.055479
BsmtFinType2	38	0.026027
BsmtExposure	38	0.026027
BsmtFinType1	37	0.025342
BsmtQual	37	0.025342
BsmtCond	37	0.025342
MasVnrType	8	0.005479
Electrical	1	0.000685
Condition2	0	0.000000
Condition1	0	0.000000
Neighborhood	0	0.000000
LandSlope	0	0.000000

Missing values for Categorical features in Bar chart Representation

In [49]:

```
missing_values = categorical_features.isnull().sum(axis=0).reset_index()
missing_values.columns = ['column_name', 'missing_count']
missing_values = missing_values.loc[missing_values['missing_count']>0]
missing_values = missing_values.sort_values(by='missing_count')

ind = np.arange(missing_values.shape[0])
width = 0.9
fig, ax = plt.subplots(figsize=(12,18))
rects = ax.barh(ind, missing_values.missing_count.values, color='red')
ax.set_yticks(ind)
ax.set_yticklabels(missing_values.column_name.values, rotation='horizontal')
ax.set_xlabel("Missing Observations Count")
ax.set_title("Missing Observations Count - Categorical Features")
plt.show()
```



Categorical Feature Exploration

Let us look at the unique values in categorical features in both train and test dataframes in detail

In [50]:

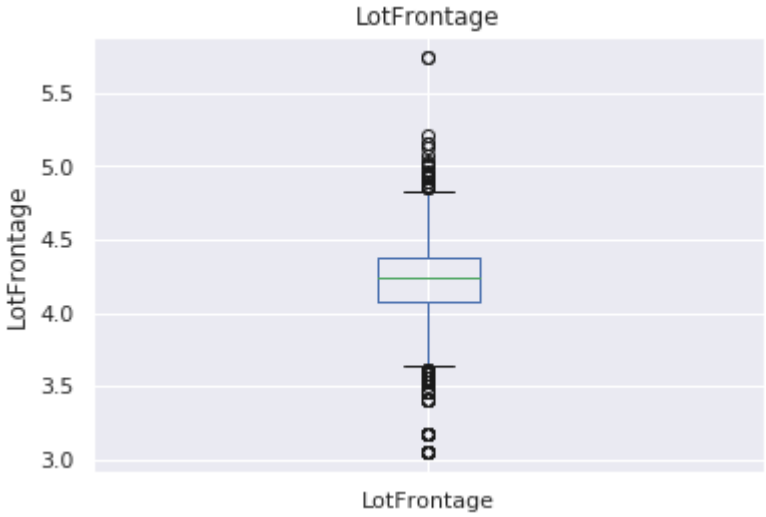
```
for column_name in train.columns:
    if train[column_name].dtypes == 'object':
        train[column_name] = train[column_name].fillna(train[column_name].mode().iloc[0
])
        unique_category = len(train[column_name].unique())
        print("Feature '{column_name}' has '{unique_category}' unique categories".forma
t(column_name = column_name,

unique_category=unique_category))

for column_name in test.columns:
    if test[column_name].dtypes == 'object':
        test[column_name] = test[column_name].fillna(test[column_name].mode().iloc[0])
        unique_category = len(test[column_name].unique())
        print("Features in test set '{column_name}' has '{unique_category}' unique cate
gories".format(column_name = column_name, unique_category=unique_category))
```

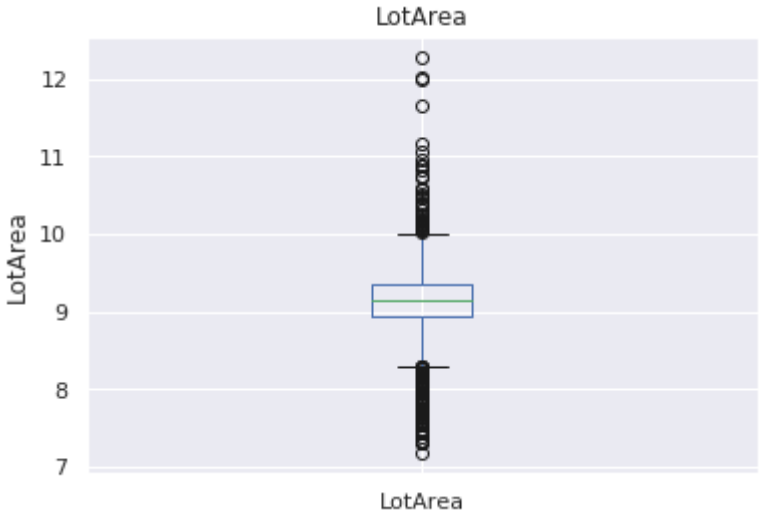
```
Features in test set 'MSZoning' has '5' unique categories
Features in test set 'Street' has '2' unique categories
Features in test set 'Alley' has '2' unique categories
Features in test set 'LotShape' has '4' unique categories
Features in test set 'LandContour' has '4' unique categories
Features in test set 'Utilities' has '1' unique categories
Features in test set 'LotConfig' has '5' unique categories
Features in test set 'LandSlope' has '3' unique categories
Features in test set 'Neighborhood' has '25' unique categories
Features in test set 'Condition1' has '9' unique categories
Features in test set 'Condition2' has '5' unique categories
Features in test set 'BldgType' has '5' unique categories
Features in test set 'HouseStyle' has '7' unique categories
Features in test set 'RoofStyle' has '6' unique categories
Features in test set 'RoofMatl' has '4' unique categories
Features in test set 'Exterior1st' has '13' unique categories
Features in test set 'Exterior2nd' has '15' unique categories
Features in test set 'MasVnrType' has '4' unique categories
Features in test set 'ExterQual' has '4' unique categories
Features in test set 'ExterCond' has '5' unique categories
Features in test set 'Foundation' has '6' unique categories
Features in test set 'BsmtQual' has '4' unique categories
Features in test set 'BsmtCond' has '4' unique categories
Features in test set 'BsmtExposure' has '4' unique categories
Features in test set 'BsmtFinType1' has '6' unique categories
Features in test set 'BsmtFinType2' has '6' unique categories
Features in test set 'Heating' has '4' unique categories
Features in test set 'HeatingQC' has '5' unique categories
Features in test set 'CentralAir' has '2' unique categories
Features in test set 'Electrical' has '4' unique categories
Features in test set 'KitchenQual' has '4' unique categories
Features in test set 'Functional' has '7' unique categories
Features in test set 'FireplaceQu' has '5' unique categories
Features in test set 'GarageType' has '6' unique categories
Features in test set 'GarageFinish' has '3' unique categories
Features in test set 'GarageQual' has '4' unique categories
Features in test set 'GarageCond' has '5' unique categories
Features in test set 'PavedDrive' has '3' unique categories
Features in test set 'PoolQC' has '2' unique categories
Features in test set 'Fence' has '4' unique categories
Features in test set 'MiscFeature' has '3' unique categories
Features in test set 'SaleType' has '9' unique categories
Features in test set 'SaleCondition' has '6' unique categories
```

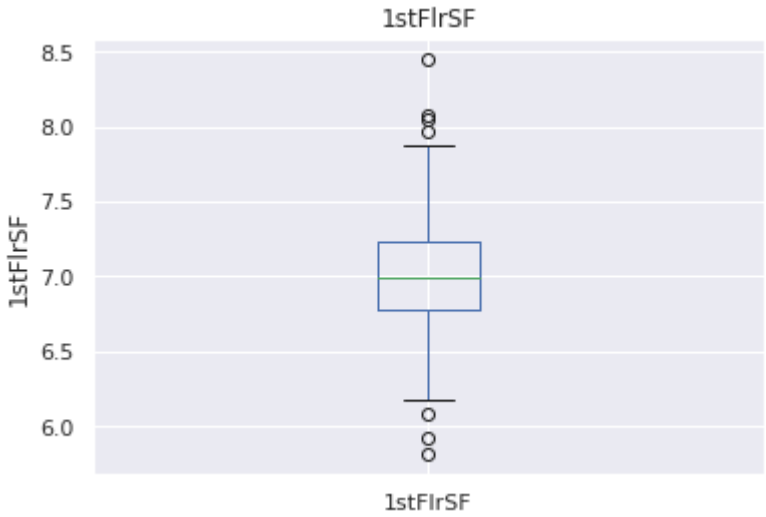
Outliers:

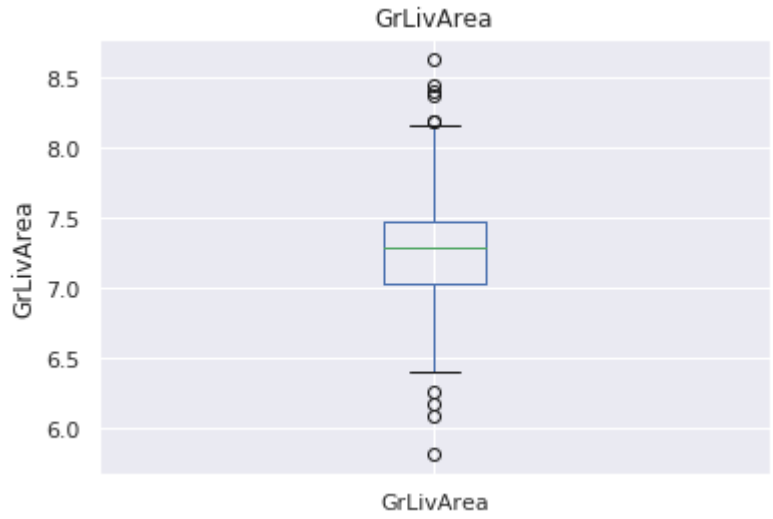
In [51]:

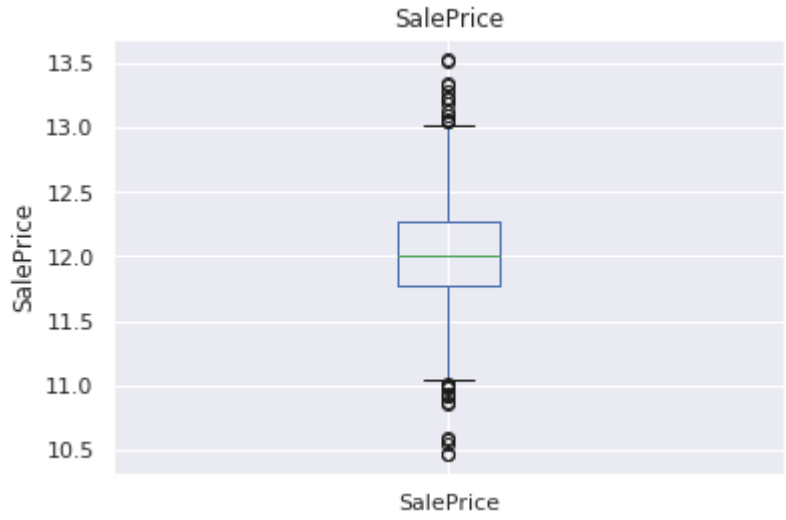
```
for feature in continuous_feature:
    data=train.copy()
    if 0 in data[feature].unique():
        pass
    else:
        data[feature]=np.log(data[feature])
        data.boxplot(column=feature)
        plt.ylabel(feature)
        plt.title(feature)
        plt.show()
```


LotFrontage
5.5
5.0
4.5
4.0
3.5
3.0
LotFrontage
LotFrontage


LotArea
12
11
10
9
8
7
LotArea
LotArea


1stFlrSF
8.5
8.0
7.5
7.0
6.5
6.0
1stFlrSF
1stFlrSF

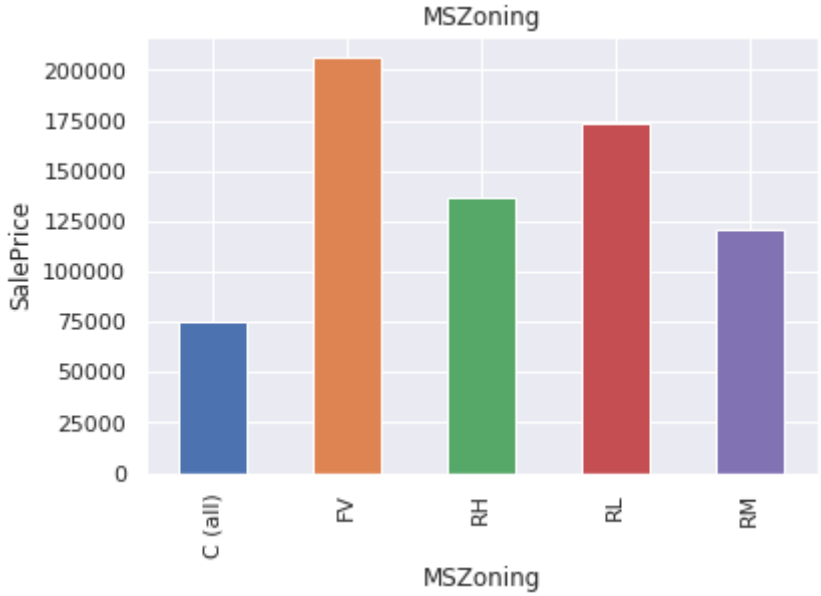


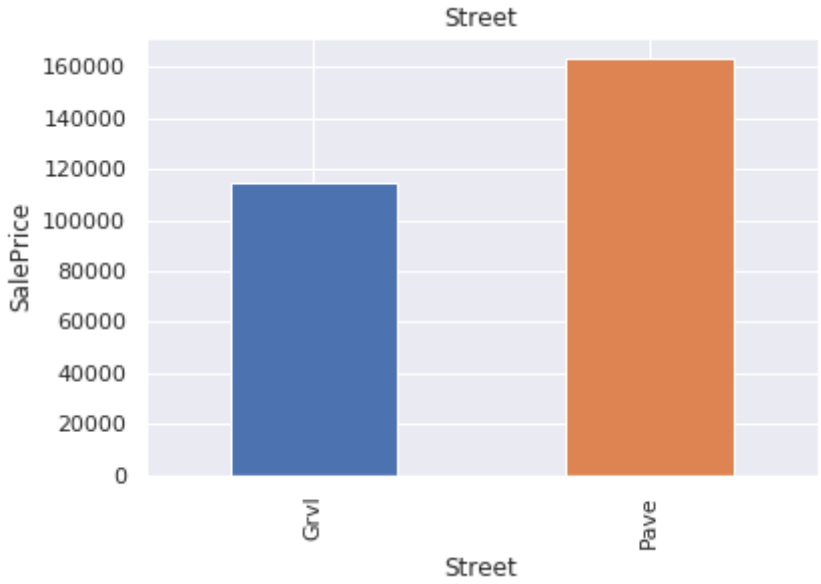


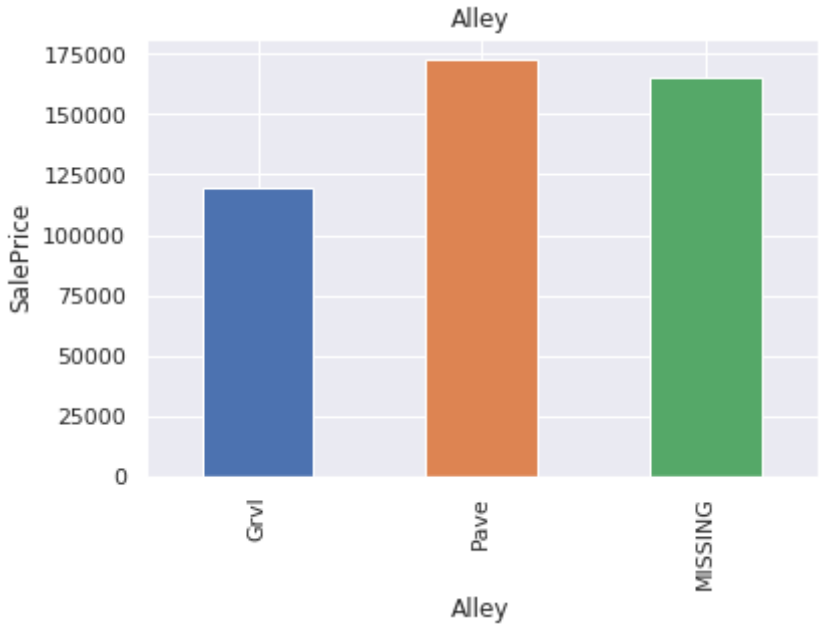
Let us find out the relationship between categorical variable and dependent feature SalesPrice

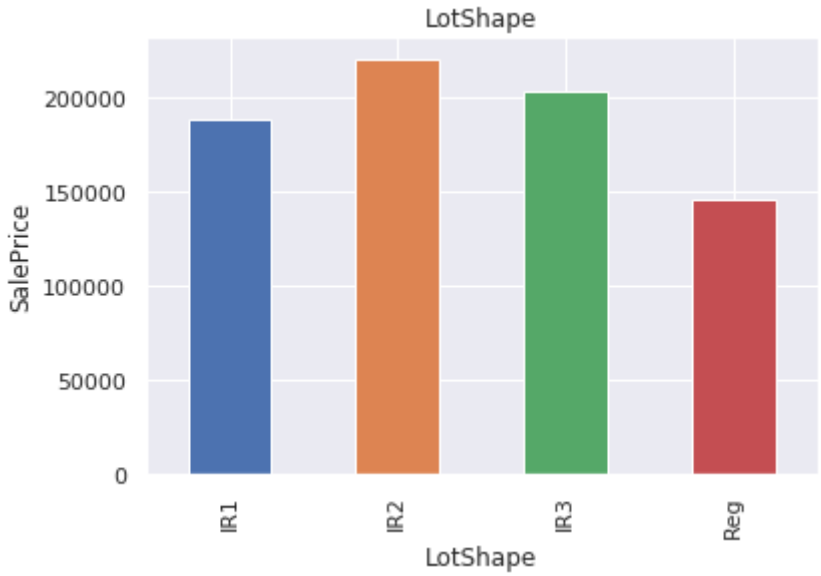
In [52]:

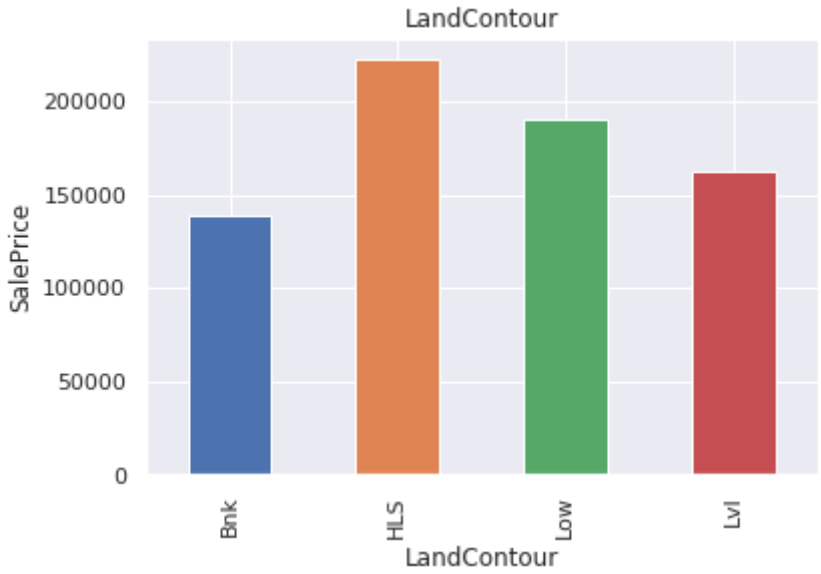
```
for feature in categorical_features:
    data=train.copy()
    data.groupby(feature)['SalePrice'].median().plot.bar()
    plt.xlabel(feature)
    plt.ylabel('SalePrice')
    plt.title(feature)
    plt.show()
```

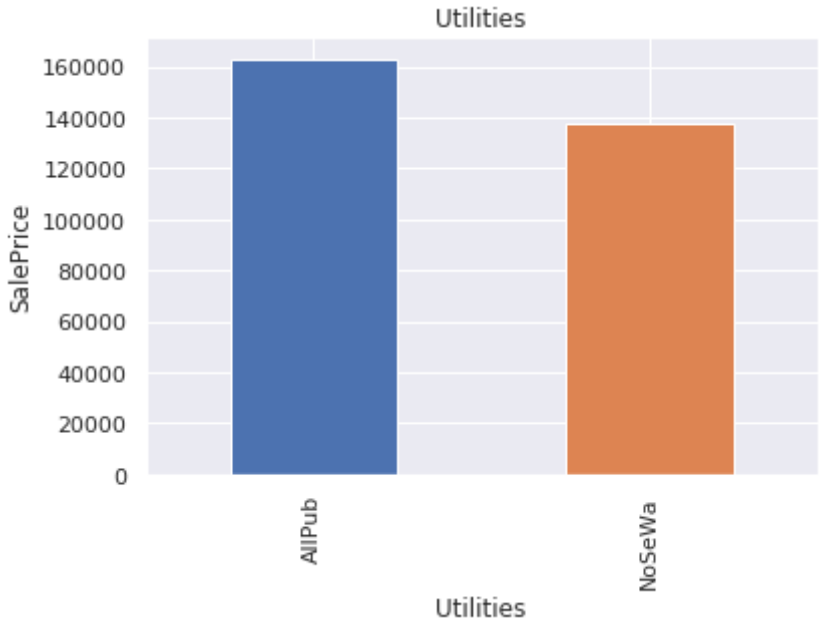

MSZoning
SalePrice
200000
175000
150000
125000
100000
75000
50000
25000
0
C (all)
FV
RH
RL
RM
MSZoning

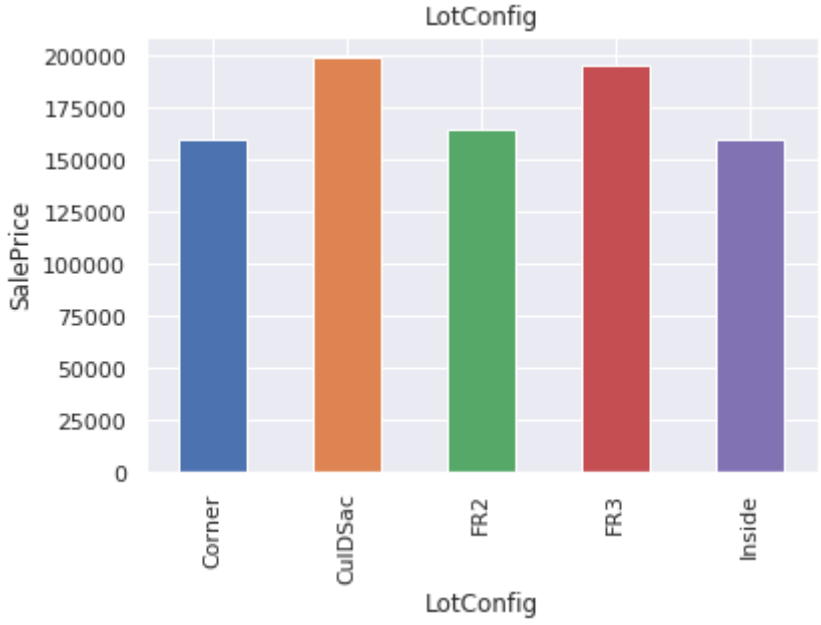

Street
SalePrice
160000
140000
120000
100000
80000
60000
40000
20000
0
Grvl
Pave
Street

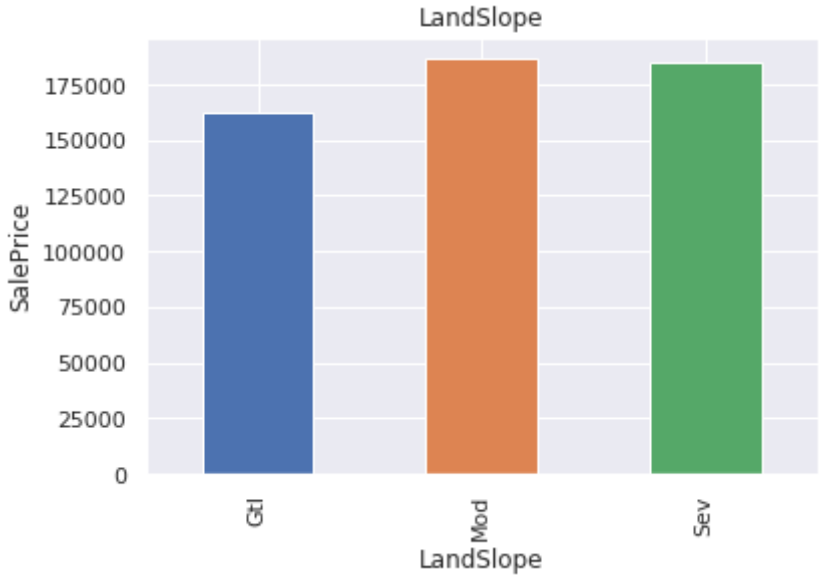

Alley
SalePrice
175000
150000
125000
100000
75000
50000
25000
0
Grvl
Pave
MISSING
Alley

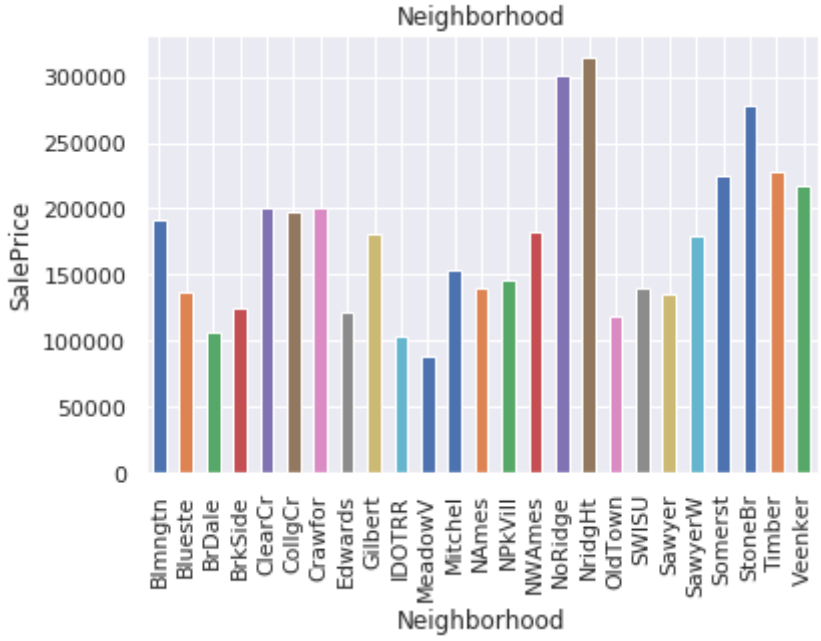

LotShape
SalePrice
200000
150000
100000
50000
0
IR1
IR2
IR3
Reg
LotShape

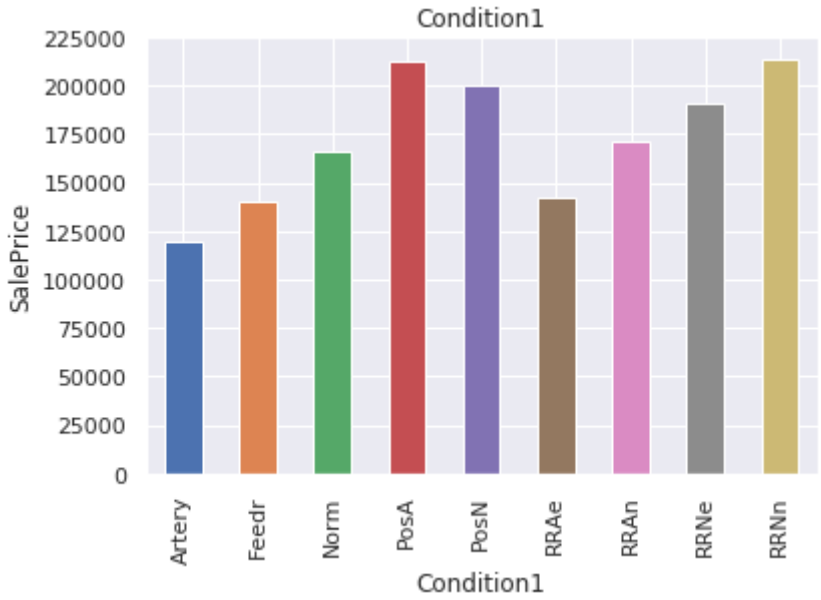

LandContour
SalePrice
200000
150000
100000
50000
0
Bnk
HLS
Low
Lvl
LandContour

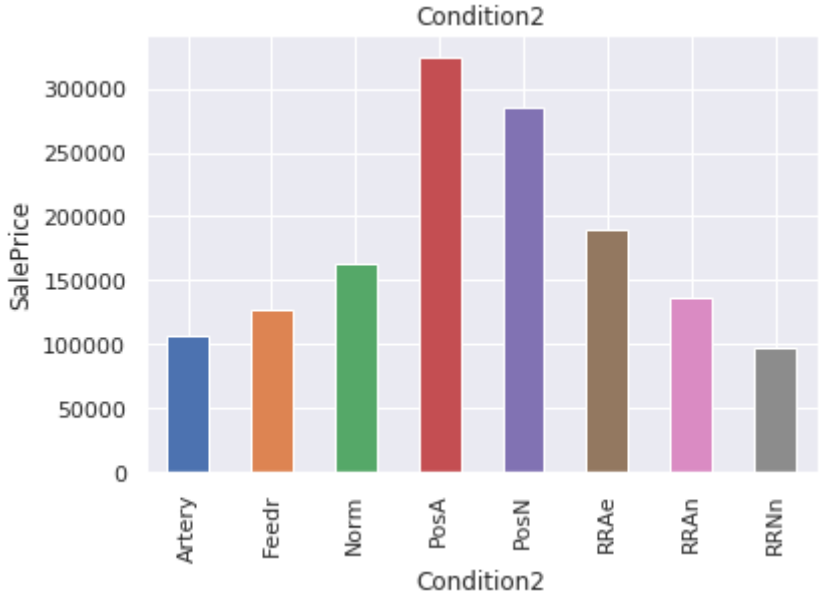

Utilities
SalePrice
160000
140000
120000
100000
80000
60000
40000
20000
0
AllPub
NoSeWa
Utilities

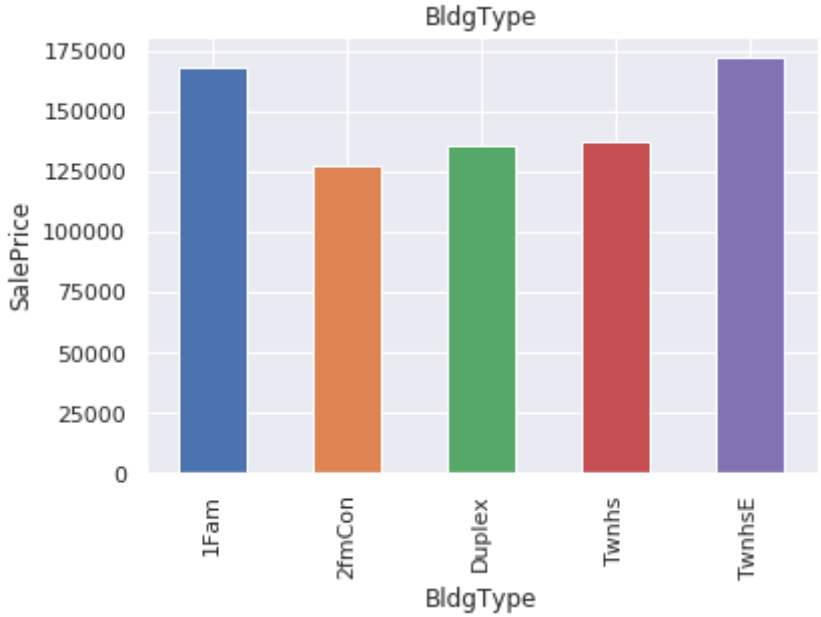

LotConfig
SalePrice
200000
175000
150000
125000
100000
75000
50000
25000
0
Corner
CulDSac
FR2
FR3
Inside
LotConfig

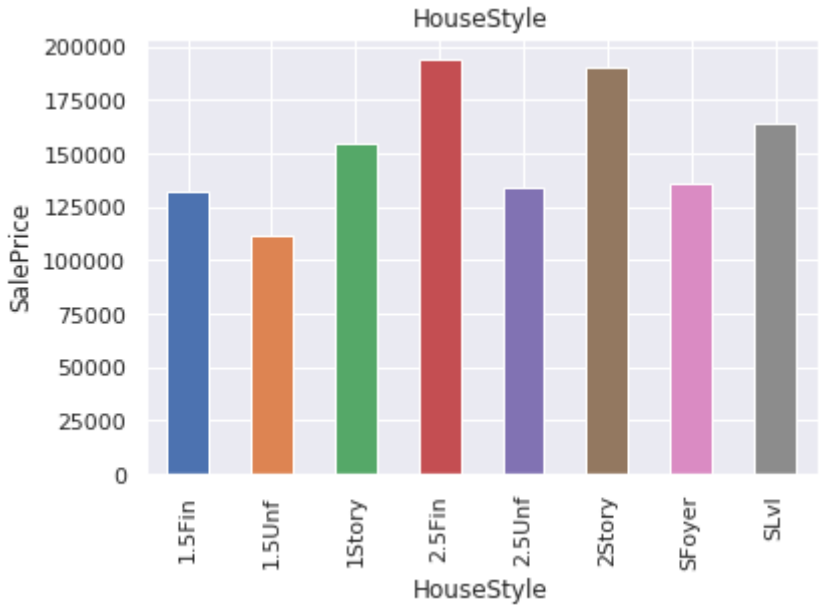

LandSlope
SalePrice
175000
150000
125000
100000
75000
50000
25000
0
Gtl
Mod
Sev
LandSlope

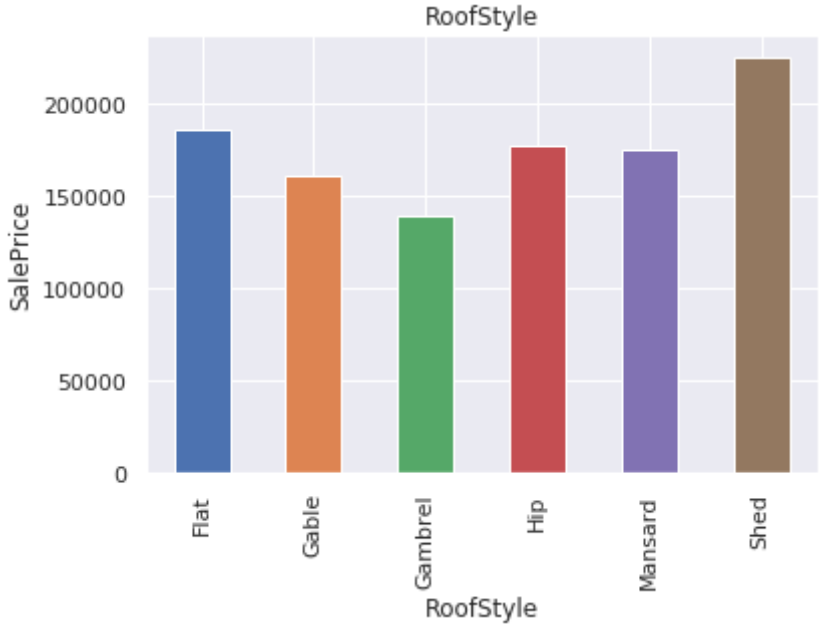

Neighborhood
SalePrice
300000
250000
200000
150000
100000
50000
0
Blmngtn
Blueste
BrDale
BrkSide
ClearCr
CollgCr
Crawfor
Edwards
Gilbert
IDOTRR
MeadowV
Mitchel
NAmes
NPkVill
NWAmes
NoRidge
NridgHt
OldTown
SWISU
Sawyer
SawyerW
Somerst
StoneBr
Timber
Veenker
Neighborhood

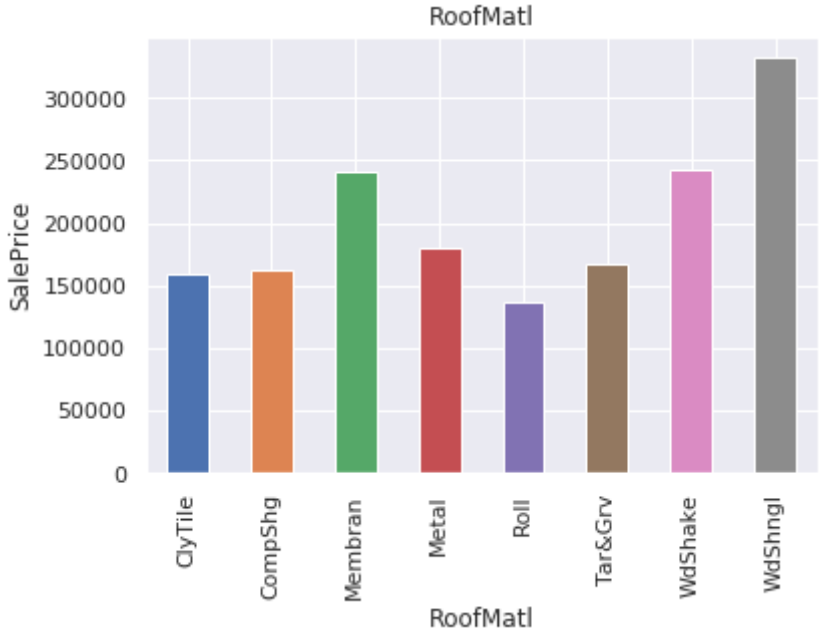

Condition1
SalePrice
225000
200000
175000
150000
125000
100000
75000
50000
25000
0
Artery
Feedr
Norm
PosA
PosN
RRAe
RRAn
RRNe
RRNn
Condition1

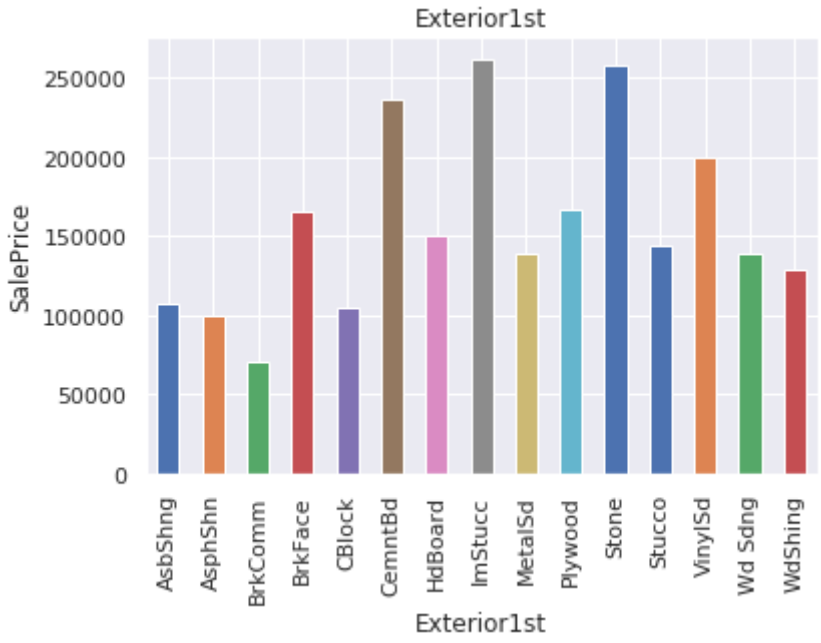

Condition2
SalePrice
300000
250000
200000
150000
100000
50000
0
Artery
Feedr
Norm
PosA
PosN
RRAe
RRAn
RRNn
Condition2

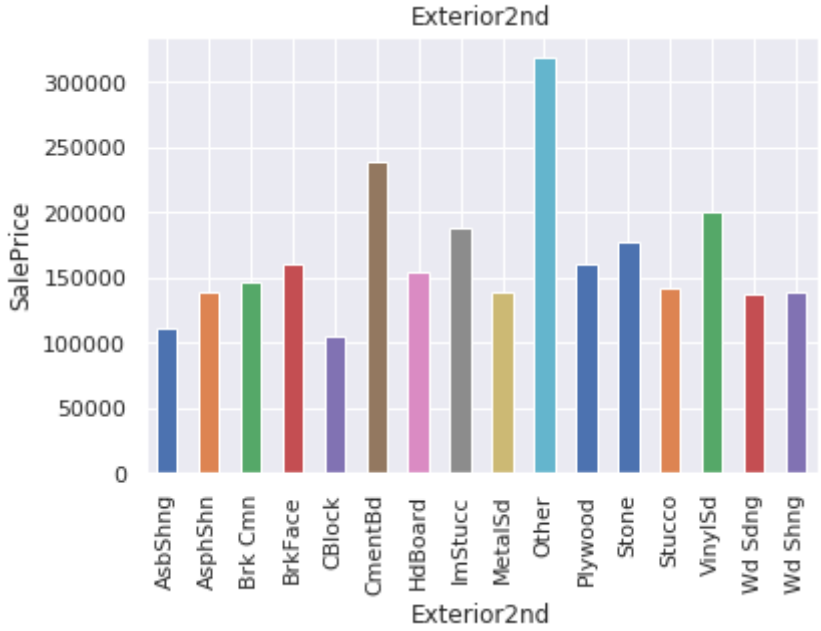

BldgType
SalePrice
175000
150000
125000
100000
75000
50000
25000
0
1Fam
2fmCon
Duplex
Twnhs
TwnhsE
BldgType

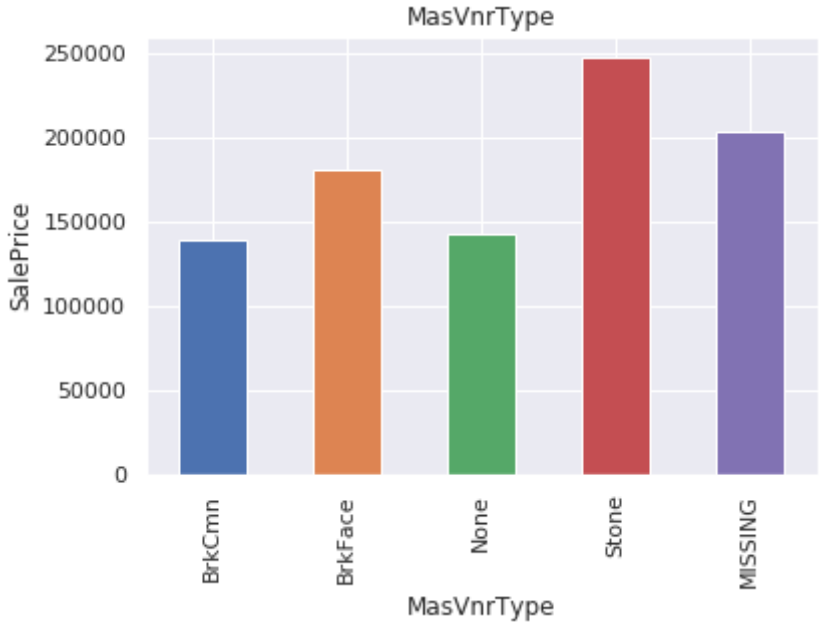

HouseStyle
SalePrice
200000
175000
150000
125000
100000
75000
50000
25000
0
1.5Fin
1.5Unf
1Story
2.5Fin
2.5Unf
2Story
SFoyer
SLvl
HouseStyle

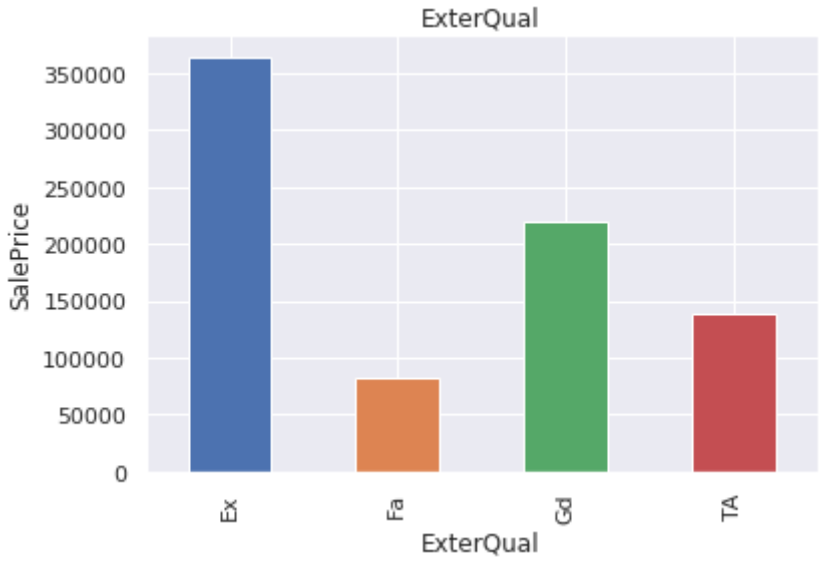

RoofStyle
SalePrice
200000
150000
100000
50000
0
Flat
Gable
Gambrel
Hip
Mansard
Shed
RoofStyle

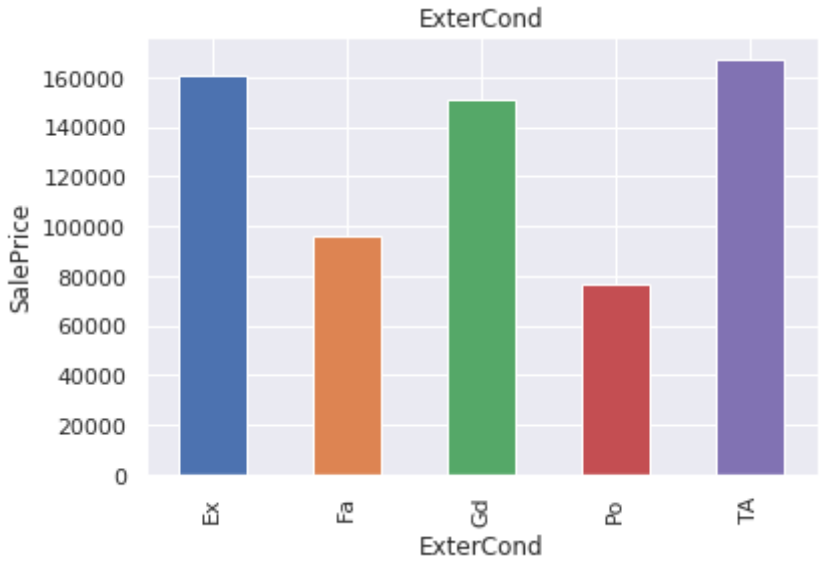

RoofMatl
SalePrice
300000
250000
200000
150000
100000
50000
0
ClyTile
CompShg
Membran
Metal
Roll
Tar&Grv
WdShake
WdShngl
RoofMatl

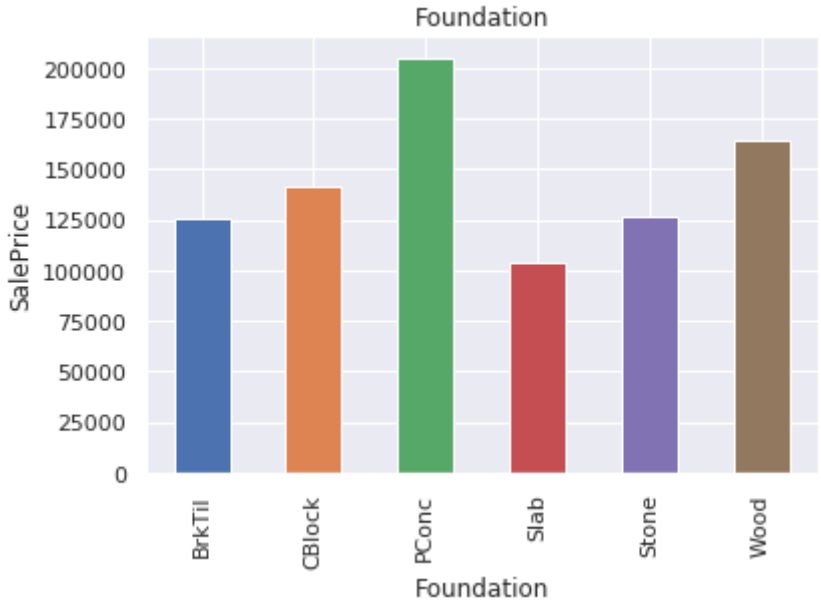

Exterior1st
SalePrice
250000
200000
150000
100000
50000
0
AsbShng
AsphShn
BrkComm
BrkFace
CBlock
CemntBd
HdBoard
ImStucc
MetalSd
Plywood
Stone
Stucco
VinylSd
Wd Sdng
WdShing
Exterior1st

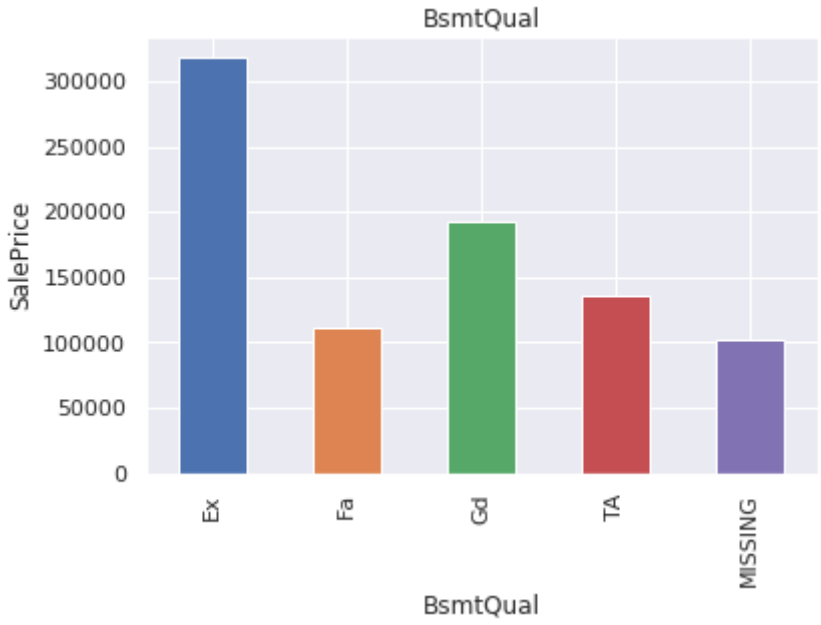

Exterior2nd
SalePrice
300000
250000
200000
150000
100000
50000
0
AsbShng
AsphShn
Brk Cmn
BrkFace
CBlock
CmentBd
HdBoard
ImStucc
MetalSd
Other
Plywood
Stone
Stucco
VinylSd
Wd Sdng
Wd Shng
Exterior2nd

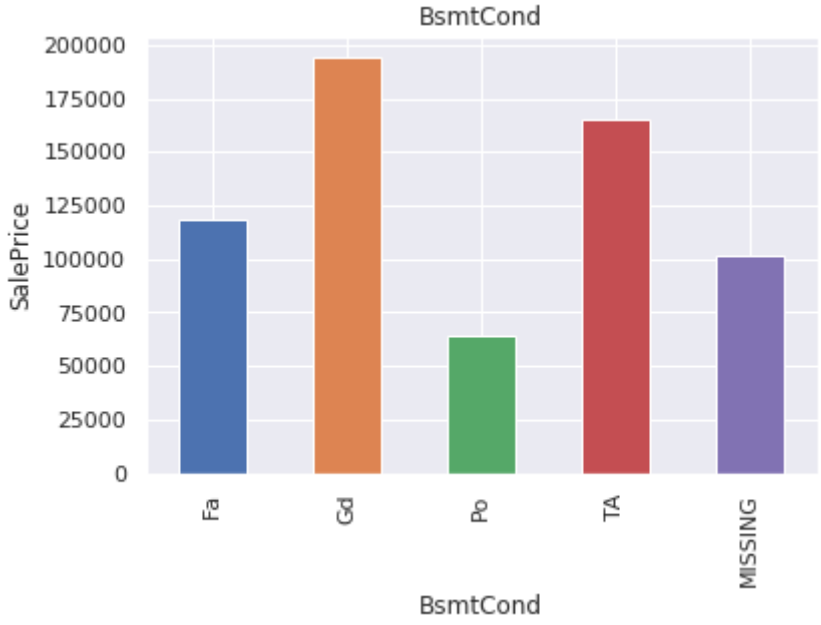

MasVnrType
SalePrice
250000
200000
150000
100000
50000
0
BrkCmn
BrkFace
None
Stone
MISSING
MasVnrType

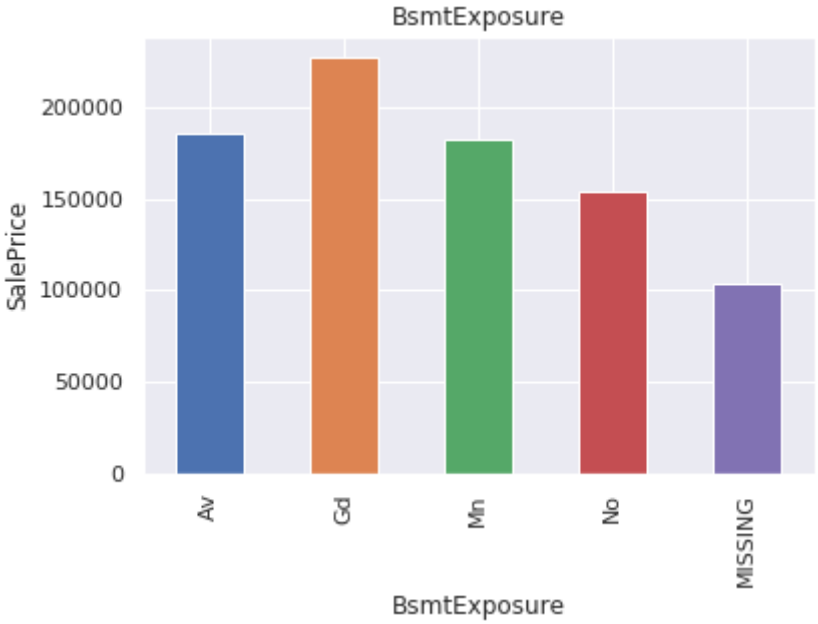

ExterQual
SalePrice
350000
300000
250000
200000
150000
100000
50000
0
Ex
Fa
Gd
TA
ExterQual

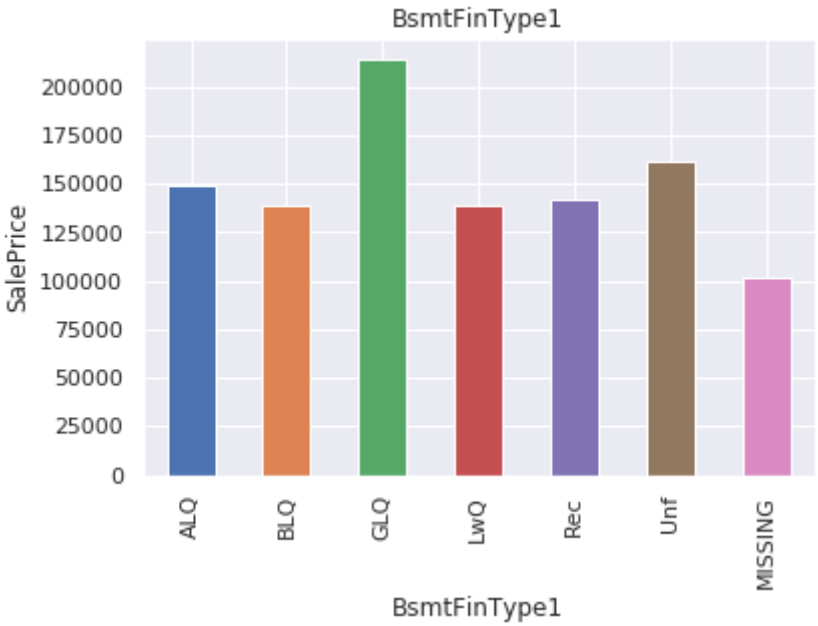

ExterCond
SalePrice
160000
140000
120000
100000
80000
60000
40000
20000
0
Ex
Fa
Gd
Po
TA
ExterCond

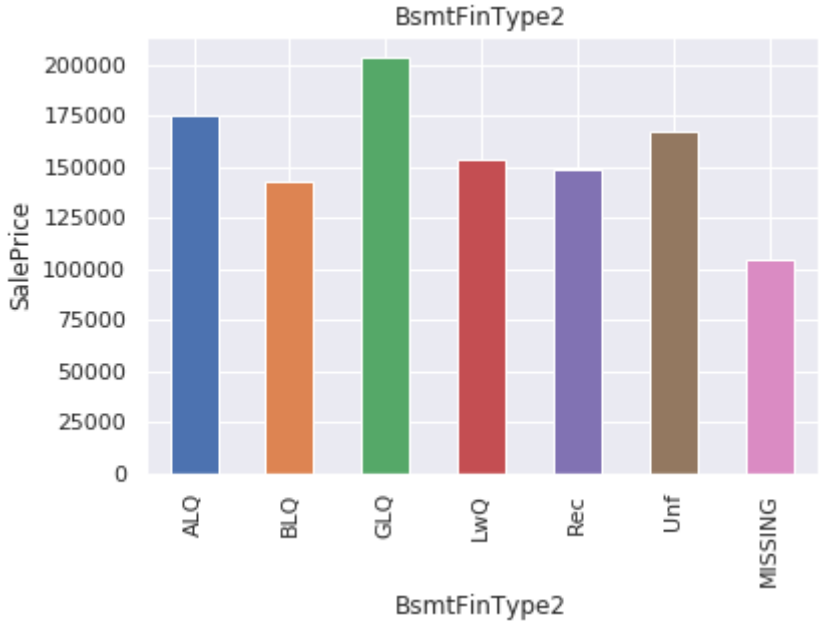

Foundation
SalePrice
200000
175000
150000
125000
100000
75000
50000
25000
0
BrkTil
CBlock
PConc
Slab
Stone
Wood
Foundation

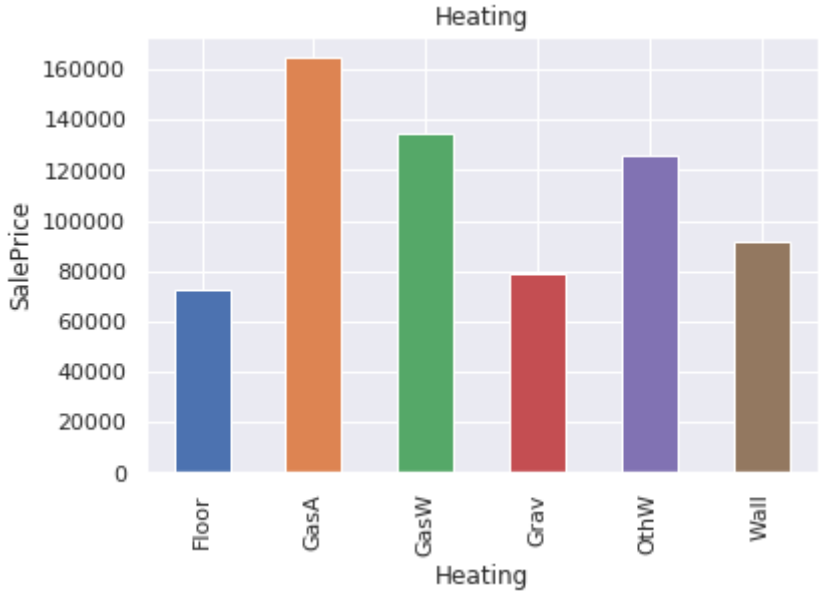

BsmtQual
SalePrice
300000
250000
200000
150000
100000
50000
0
Ex
Fa
Gd
TA
MISSING
BsmtQual

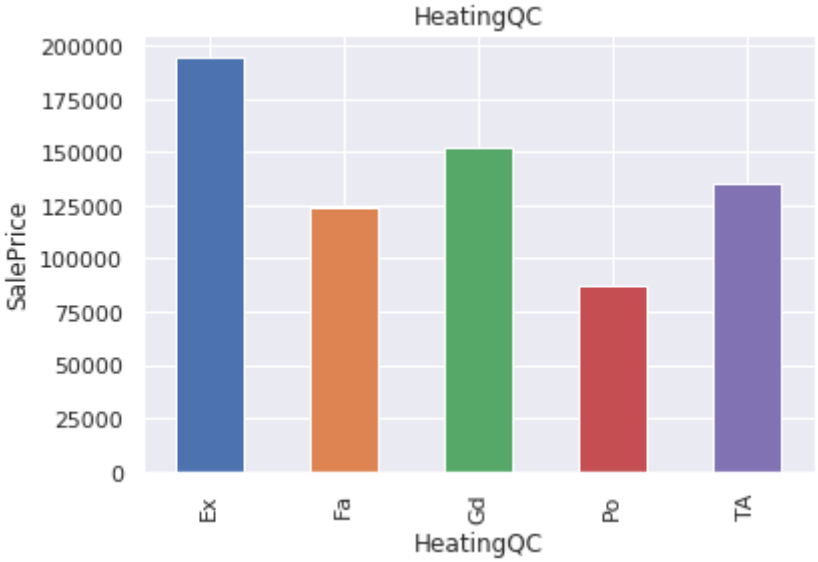

BsmtCond
SalePrice
200000
175000
150000
125000
100000
75000
50000
25000
0
Fa
Gd
Po
TA
MISSING
BsmtCond

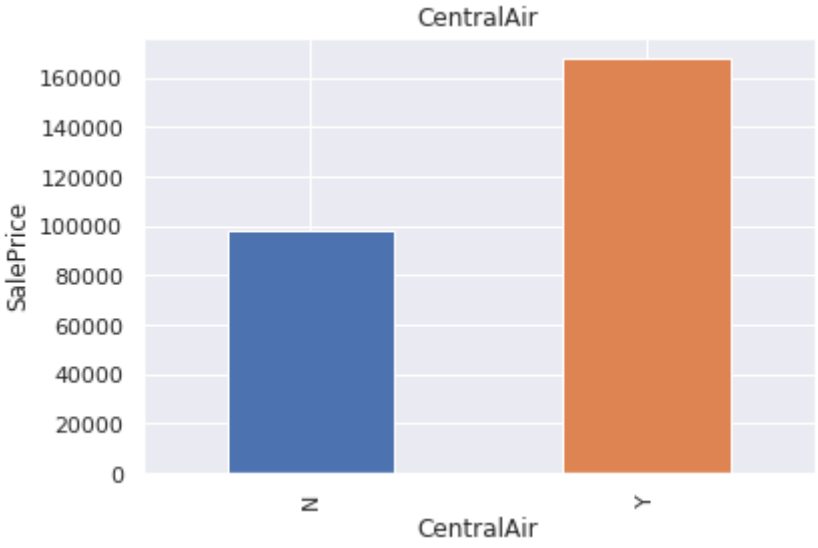

BsmtExposure
SalePrice
200000
150000
100000
50000
0
Av
Gd
Mn
No
MISSING
BsmtExposure

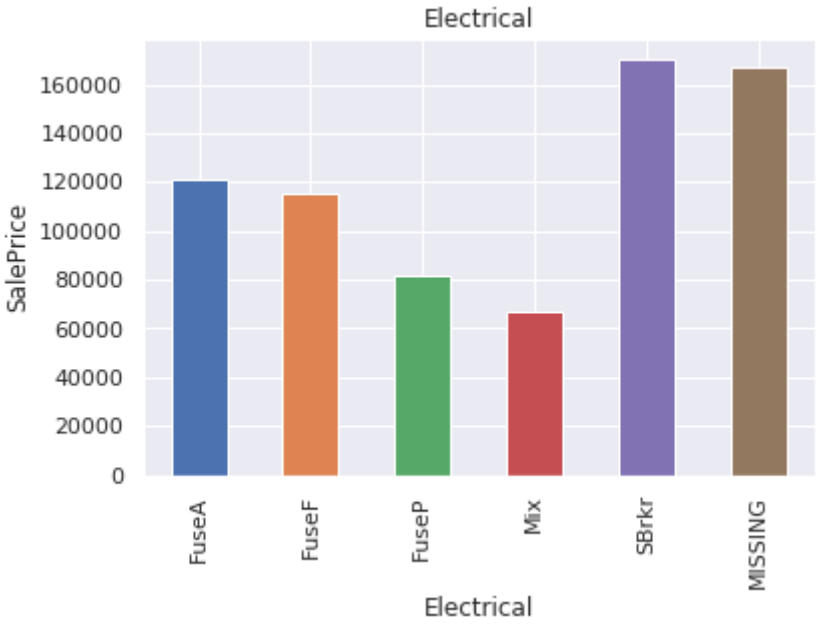

BsmtFinType1
SalePrice
200000
175000
150000
125000
100000
75000
50000
25000
0
ALQ
BLQ
GLQ
LwQ
Rec
Unf
MISSING
BsmtFinType1

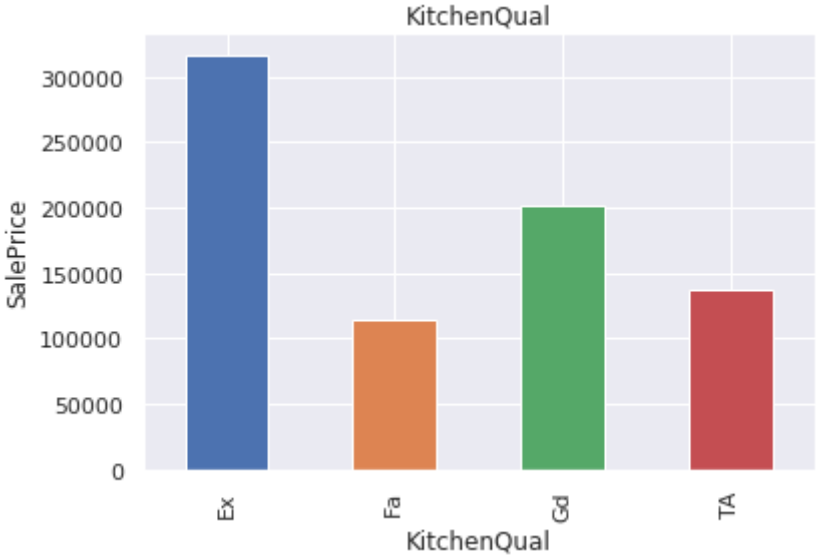

BsmtFinType2
SalePrice
200000
175000
150000
125000
100000
75000
50000
25000
0
ALQ
BLQ
GLQ
LwQ
Rec
Unf
MISSING
BsmtFinType2

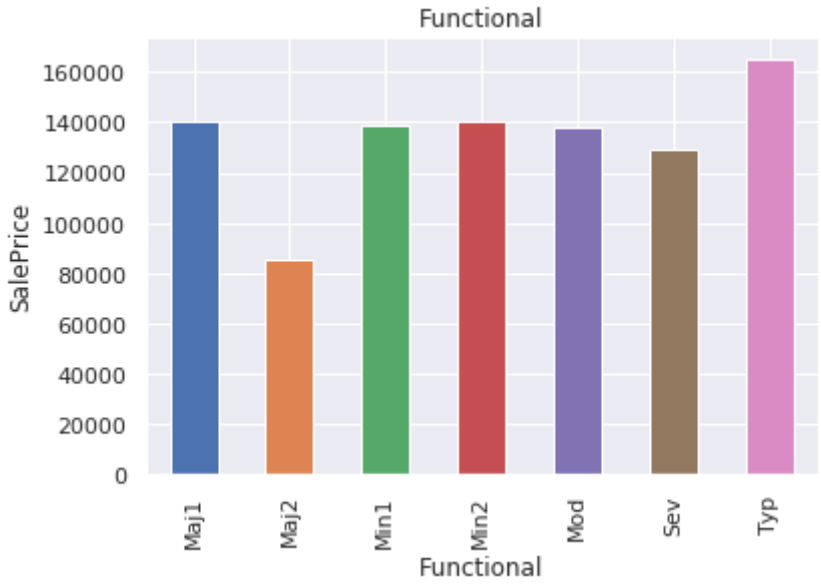

Heating
SalePrice
160000
140000
120000
100000
80000
60000
40000
20000
0
Floor
GasA
GasW
Grav
OthW
Wall
Heating

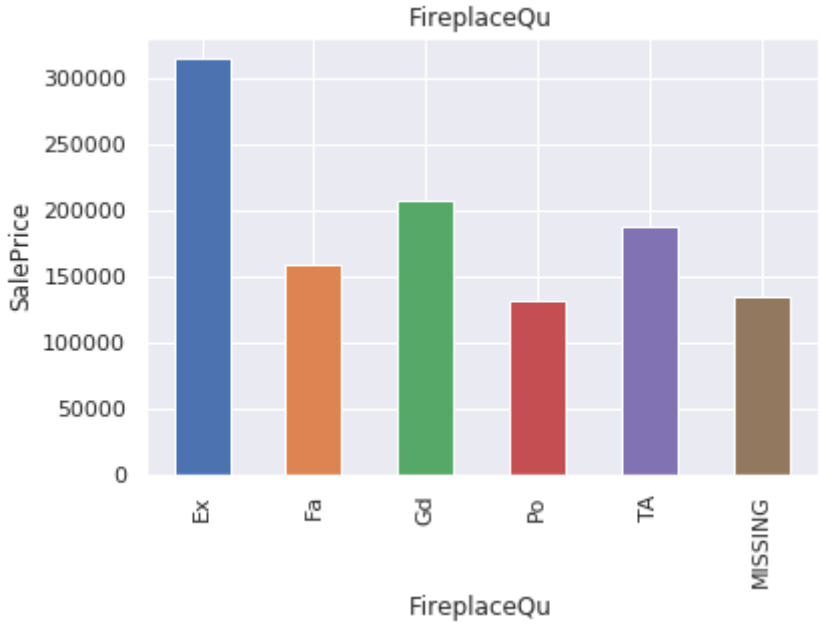

HeatingQC
200000
175000
150000
125000
100000
75000
50000
25000
0
SalePrice
Ex
Fa
Gd
Po
TA
HeatingQC

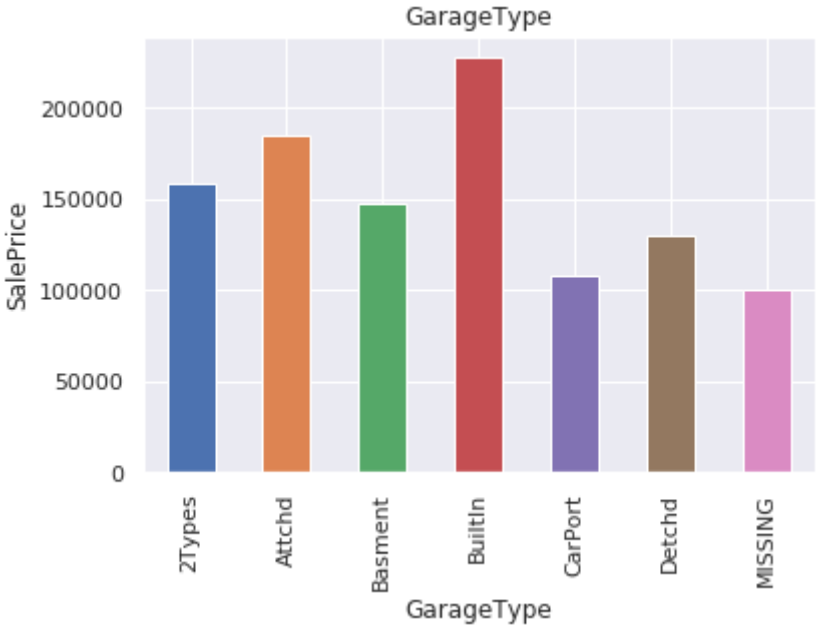

CentralAir
160000
140000
120000
100000
80000
60000
40000
20000
0
SalePrice
N
Y
CentralAir

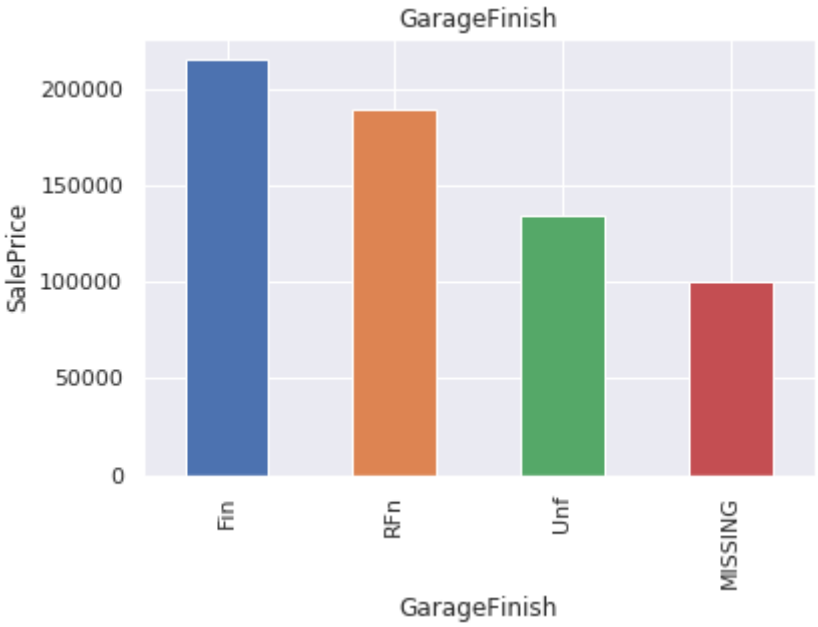

Electrical
SalePrice
160000
140000
120000
100000
80000
60000
40000
20000
0
FuseA
FuseF
FuseP
Mix
SBrkr
MISSING
Electrical

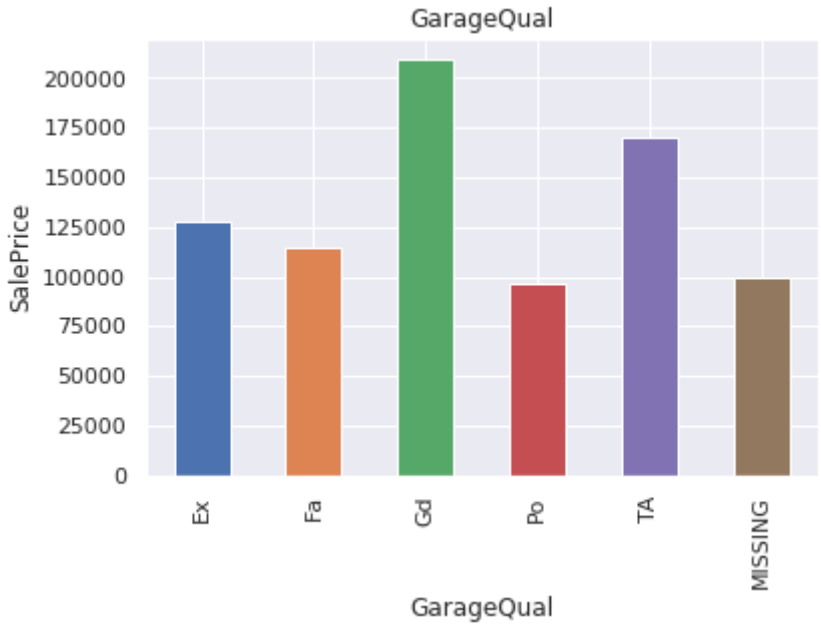

KitchenQual
SalePrice
300000
250000
200000
150000
100000
50000
0
Ex
Fa
Gd
TA
KitchenQual

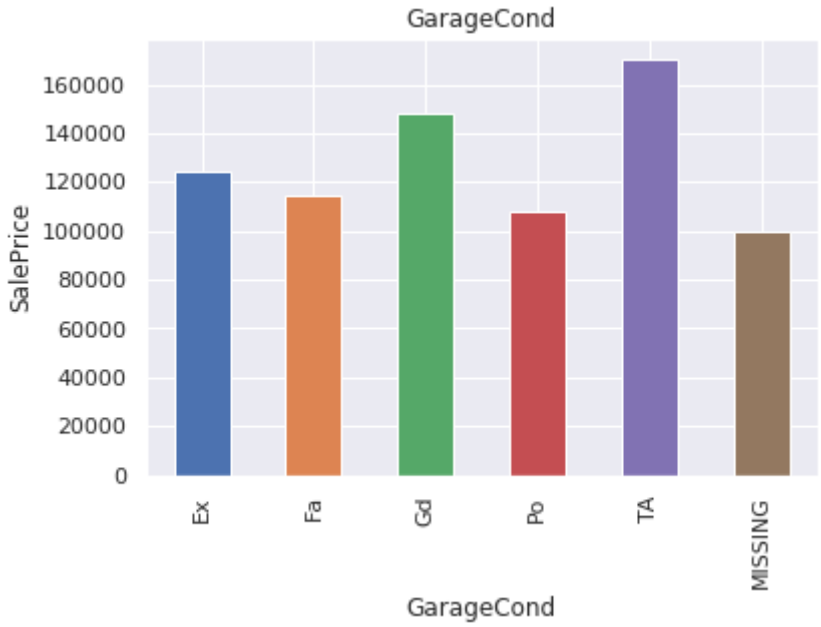

Functional
SalePrice
160000
140000
120000
100000
80000
60000
40000
20000
0
Maj1
Maj2
Min1
Min2
Mod
Sev
Typ
Functional

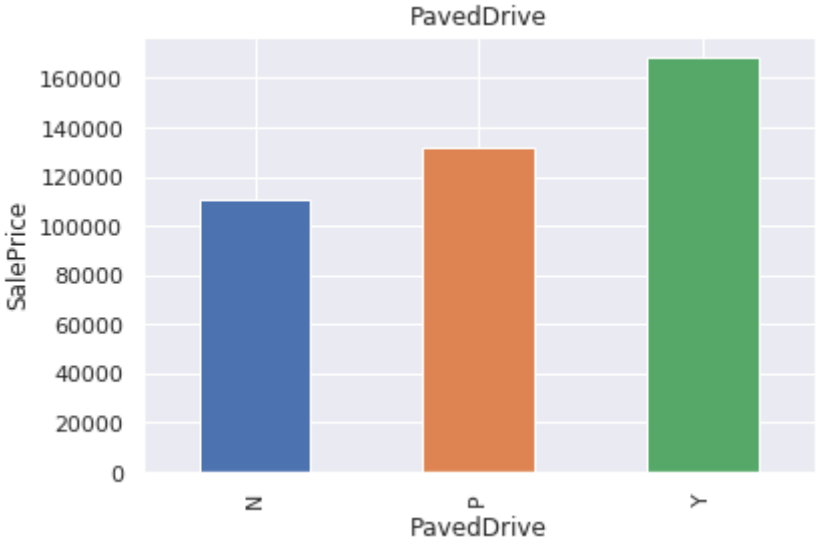

FireplaceQu
SalePrice
300000
250000
200000
150000
100000
50000
0
Ex
Fa
Gd
Po
TA
MISSING
FireplaceQu

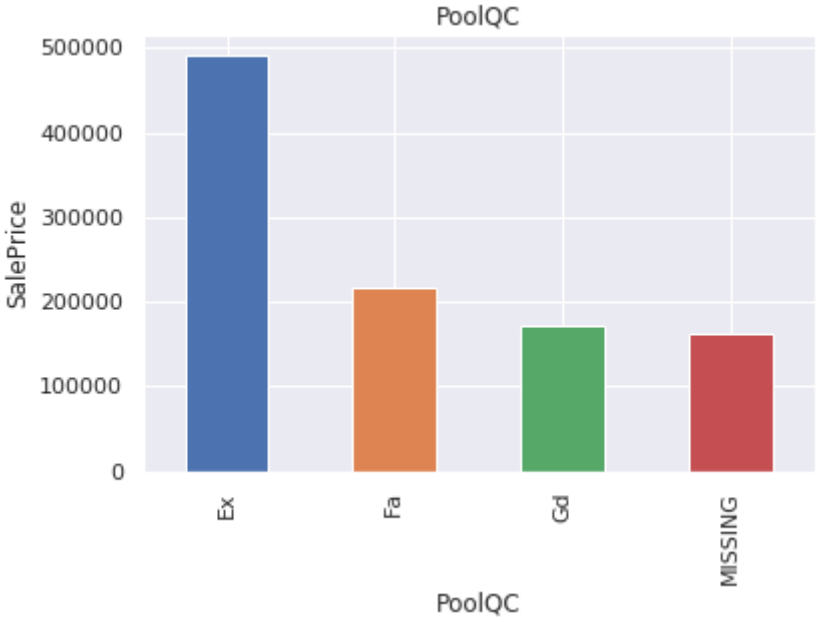

GarageType
SalePrice
200000
150000
100000
50000
0
2Types
Attchd
Basment
BuiltIn
CarPort
Detchd
MISSING
GarageType

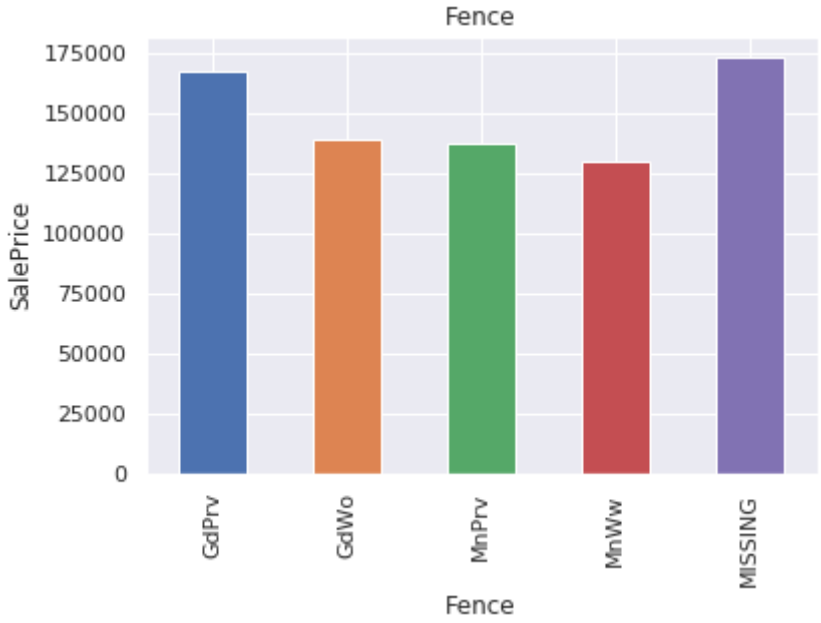

GarageFinish
SalePrice
200000
150000
100000
50000
0
Fin
RFn
Unf
MISSING
GarageFinish

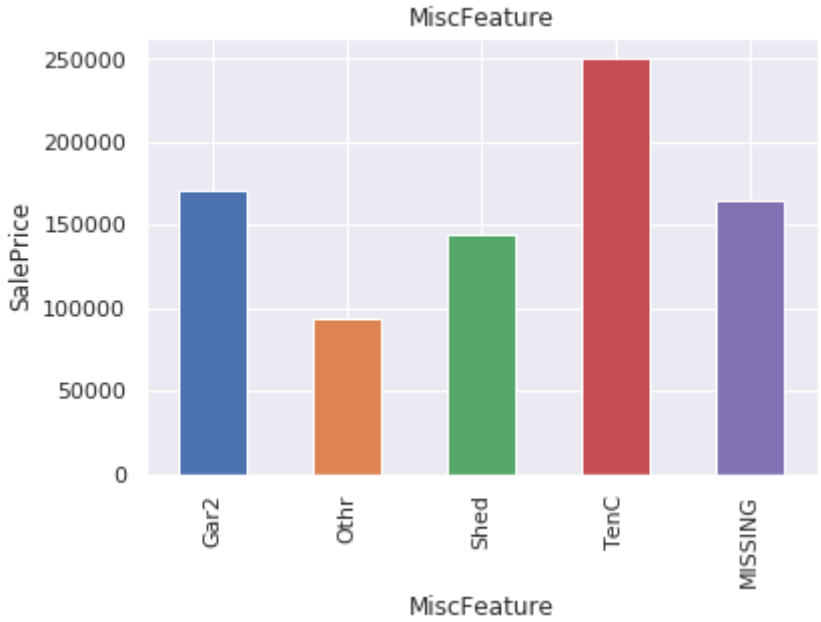

GarageQual
SalePrice
200000
175000
150000
125000
100000
75000
50000
25000
0
Ex
Fa
Gd
Po
TA
MISSING
GarageQual

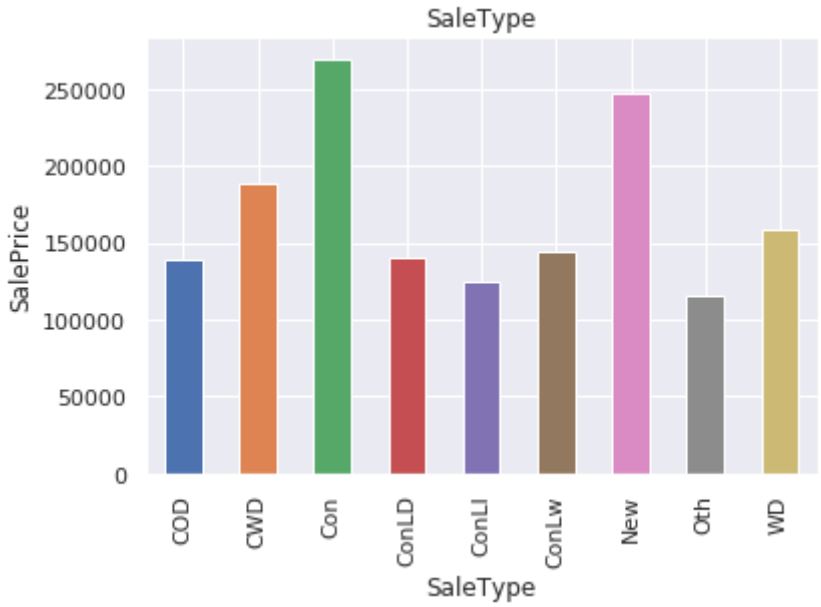

GarageCond
SalePrice
160000
140000
120000
100000
80000
60000
40000
20000
0
Ex
Fa
Gd
Po
TA
MISSING
GarageCond

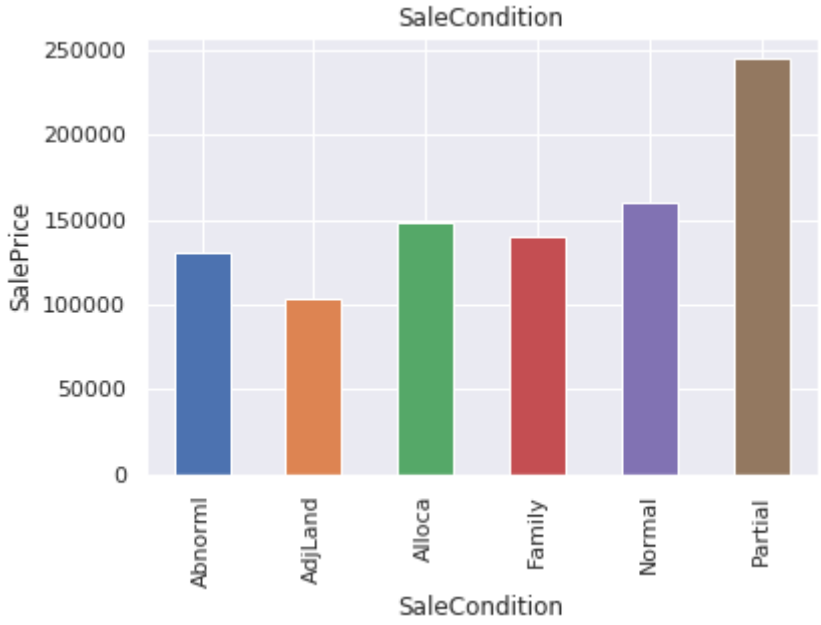

PavedDrive
160000
140000
120000
100000
80000
60000
40000
20000
0
SalePrice
N
P
Y
PavedDrive


PoolQC
500000
400000
300000
200000
100000
0
SalePrice
Ex
Fa
Gd
MISSING
PoolQC


Fence
SalePrice
175000
150000
125000
100000
75000
50000
25000
0
GdPrv
GdWo
MnPrv
MnWw
MISSING
Fence


MiscFeature
SalePrice
250000
200000
150000
100000
50000
0
Gar2
Othr
Shed
TenC
MISSING
MiscFeature





I hope you had a good feel of how to perform Exploratory Data Analysis with this kernel.

Greatly appreciate to leave your comments /suggestions and if you like this kernel please do UPVOTE.