House Prices dataset: Feature Engineering

In the following cells, we will engineer / pre-process the variables of the House Price Dataset from Kaggle. We will engineer the variables so that we tackle:

1. Missing values
2. Temporal variables
3. Non-Gaussian distributed variables
4. Categorical variables: remove rare labels
5. Categorical variables: convert strings to numbers
6. Standarise the values of the variables to the same range

Setting the seed

It is important to note that we are engineering variables and pre-processing data with the idea of deploying the model. Therefore, from now on, for each step that includes some element of randomness, it is extremely important that we **set the seed**. This way, we can obtain reproducibility between our development and production code.

Import necessary libraries

In [21]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import MinMaxScaler

pd.set_option('display.max_columns',None)

import warnings
warnings.simplefilter(action='ignore')
```

Load Dataset

In [22]:
```
df = pd.read_csv('houseprice.csv')
print(df.shape)
df.head()
```

(1460, 81)

Out[22]:

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

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub

Split Data into Training & Test Set:

In [23]:
```
X_train,X_test,y_train,y_test = train_test_split(df,df['SalePrice'],test_size=0.1,rando
```

In [24]:
```
X_train.shape, X_test.shape
```

Out[24]:
```
((1314, 81), (146, 81))
```

Missing Values:

Categorical variables

In [25]:
```
# make a list of the categorical variables that contain missing values

mis_var = [
    var for var in df.columns
    if X_train[var].isnull().sum() > 0 and X_train[var].dtypes == 'O'
]

# print percentage of missing values per variable
X_train[mis_var].isnull().mean()
```

Out[25]:
```
Alley           0.938356
MasVnrType      0.004566
BsmtQual        0.024353
BsmtCond        0.024353
BsmtExposure    0.025114
BsmtFinType1    0.024353
BsmtFinType2    0.025114
Electrical      0.000761
FireplaceQu     0.472603
GarageType      0.056317
GarageFinish    0.056317
GarageQual      0.056317
GarageCond      0.056317
PoolQC          0.995434
Fence           0.814307
MiscFeature     0.961187
dtype: float64
```

In [26]:
```
X_train[mis_var] = X_train[mis_var].fillna('Missing')
X_test[mis_var] = X_test[mis_var].fillna('Missing')
```

In [27]:
```
X_train[mis_var].isnull().sum()
```

Out[27]:
```
Alley           0
MasVnrType      0
BsmtQual        0
BsmtCond        0
BsmtExposure    0
BsmtFinType1    0
BsmtFinType2    0
Electrical      0
```

```
FireplaceQu     0
GarageType      0
GarageFinish    0
GarageQual      0
GarageCond      0
PoolQC          0
Fence           0
MiscFeature     0
dtype: int64
```

In [28]:
```
[var for var in mis_var if X_test[var].isnull().sum() > 0]
```

Out[28]:
```
[]
```

Numerical variables

To deal with missing values in numerical variables, we will:

- **add a binary missing value indicator variable**
- **and then replace the missing values in the original variable with the mode**

In [29]:
```
# make a list with the numerical variables that contain missing values
mis_var_num = [
    var for var in df.columns
    if X_train[var].isnull().sum() > 0 and X_train[var].dtypes != 'O'
]

# print percentage of missing values per variable
X_train[mis_var_num].isnull().mean()
```

Out[29]:
```
LotFrontage    0.177321
MasVnrArea     0.004566
GarageYrBlt    0.056317
dtype: float64
```

In [30]:
```
# replace engineer missing values as we described above

for var in mis_var_num:

    # calculate the mode using the train set
    mode_val = X_train[var].mode()[0]

    # add binary missing indicator (in train and test)
    X_train[var+'_na'] = np.where(X_train[var].isnull(), 1, 0)
    X_test[var+'_na'] = np.where(X_test[var].isnull(), 1, 0)

    # replace missing values by the mode
    # (in train and test)
    X_train[var] = X_train[var].fillna(mode_val)
    X_test[var] = X_test[var].fillna(mode_val)

# check that we have no more missing values in the engineered variables
X_train[mis_var_num].isnull().sum()
```

Out[30]:
```
LotFrontage    0
MasVnrArea     0
GarageYrBlt    0
dtype: int64
```

In [31]:
```
[var for var in mis_var_num if X_test[var].isnull().sum() > 0]
```

Out[31]: []

In [32]:
```
# check the binary missing indicator variables

X_train[['LotFrontage_na', 'MasVnrArea_na', 'GarageYrBlt_na']].head()
```

Out[32]:

	LotFrontage_na	MasVnrArea_na	GarageYrBlt_na
930	0	0	0
656	0	0	0
45	0	0	0
1348	1	0	0
55	0	0	0

Temporal Variables:

Capture Elapsed Time

In [33]:
```
def elapsed_years(df, var):
    # capture difference between the year variable
    # and the year in which the house was sold
    df[var] = df['YrSold'] - df[var]
    return df
```

In [34]:
```
for var in ['YearBuilt', 'YearRemodAdd', 'GarageYrBlt']:
    X_train = elapsed_years(X_train, var)
    X_test = elapsed_years(X_test, var)
```

Numerical Variable Transformation:

In [35]:
```
for var in ['LotFrontage', 'LotArea', '1stFlrSF', 'GrLivArea', 'SalePrice']:
    X_train[var] = np.log(X_train[var])
    X_test[var] = np.log(X_test[var])
```

In [36]:
```
# check that test set does not contain null values in the feature engineered variables
[var for var in ['LotFrontage', 'LotArea', '1stFlrSF',
                 'GrLivArea', 'SalePrice'] if X_test[var].isnull().sum() > 0]
```

Out[36]: []

In [37]:
```
[var for var in ['LotFrontage', 'LotArea', '1stFlrSF',
                 'GrLivArea', 'SalePrice'] if X_train[var].isnull().sum() > 0]
```

Out[37]: []

Categorical variables

Removing rare labels

First, we will group those categories within variables that are present in less than 1% of the observations. That is, all values of categorical variables that are shared by less than 1% of

houses, well be replaced by the string "Rare".

In [38]:
```
cat_vars = [var for var in X_train.columns if X_train[var].dtype == 'O']
```

In [39]:
```
def find_frequent_labels(df, var, rare_perc):

    # function finds the labels that are shared by more than
    # a certain % of the houses in the dataset

    df = df.copy()

    tmp = df.groupby(var)['SalePrice'].count() / len(df)

    return tmp[tmp > rare_perc].index


for var in cat_vars:

    # find the frequent categories
    frequent_ls = find_frequent_labels(X_train, var, 0.01)

    # replace rare categories by the string "Rare"
    X_train[var] = np.where(X_train[var].isin(
        frequent_ls), X_train[var], 'Rare')

    X_test[var] = np.where(X_test[var].isin(
        frequent_ls), X_test[var], 'Rare')
```

Encoding Categorical Variables

In [40]:
```
def replace_categories(train, test, var, target):

    # order the categories in a variable from that with the lowest
    # house sale price, to that with the highest
    ordered_labels = train.groupby([var])[target].mean().sort_values().index

    # create a dictionary of ordered categories to integer values
    ordinal_label = {k: i for i, k in enumerate(ordered_labels, 0)}

    # use the dictionary to replace the categorical strings by integers
    train[var] = train[var].map(ordinal_label)
    test[var] = test[var].map(ordinal_label)
```

In [41]:
```
for var in cat_vars:
    replace_categories(X_train, X_test, var, 'SalePrice')
```

In [42]:
```
[var for var in X_train.columns if X_train[var].isnull().sum() > 0]
```

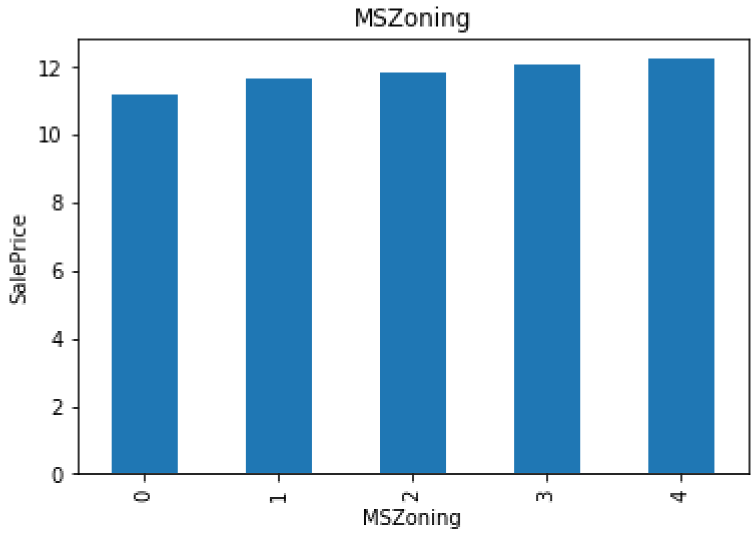
Out[42]: []

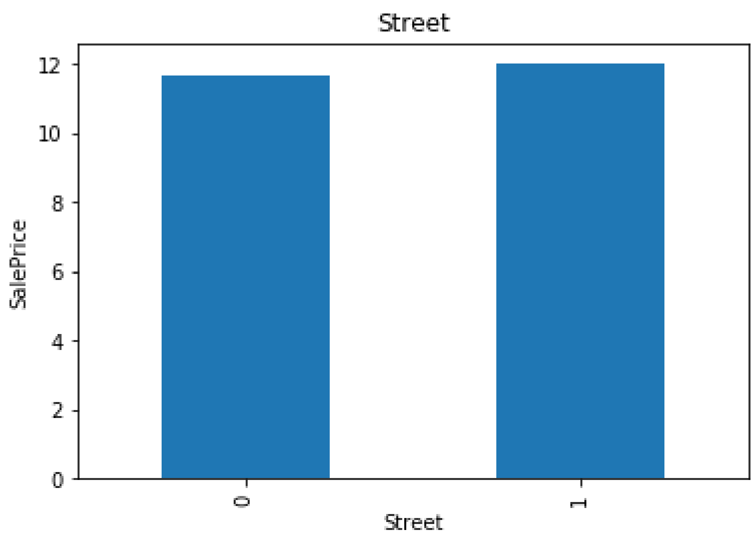
In [43]:
```
[var for var in X_test.columns if X_test[var].isnull().sum() > 0]
```

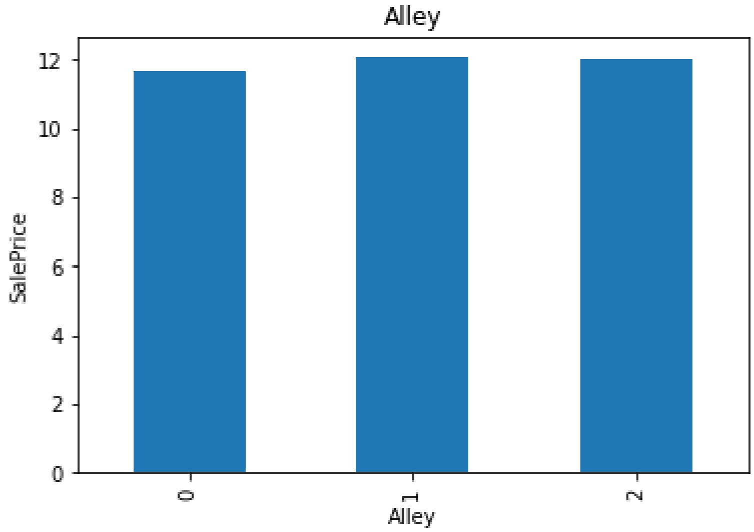
Out[43]: []

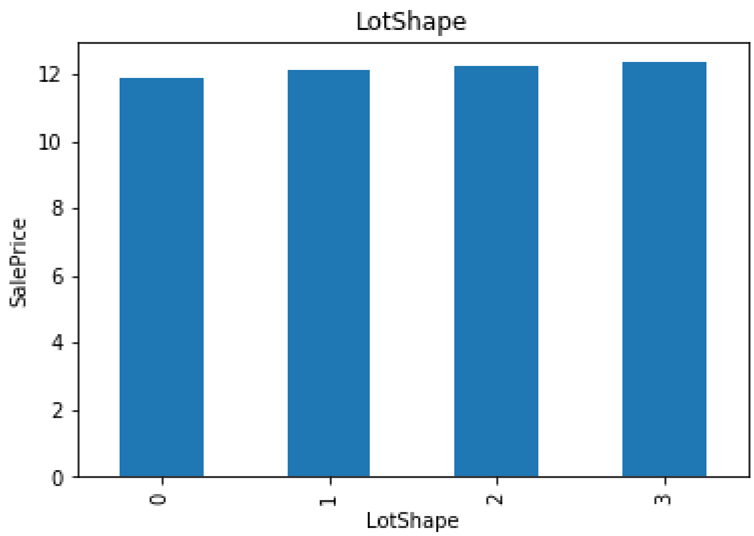
In [44]:
```
def analyse_vars(df, var):

    # function plots median house sale price per encoded
```

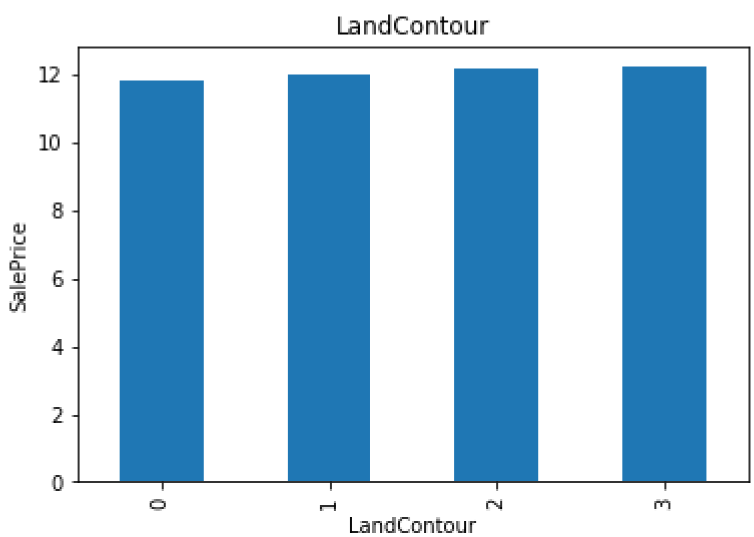
```
    # category

    df = df.copy()
    df.groupby(var)['SalePrice'].median().plot.bar()
    plt.title(var)
    plt.ylabel('SalePrice')
    plt.show()

for var in cat_vars:
    analyse_vars(X_train, var)
```

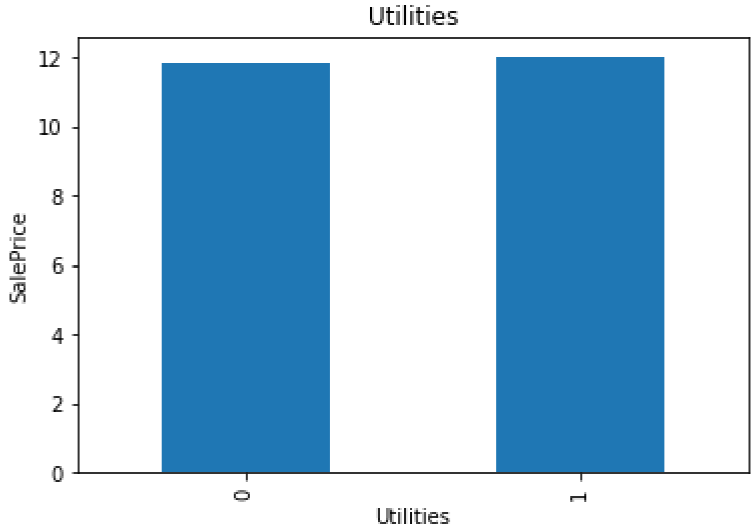


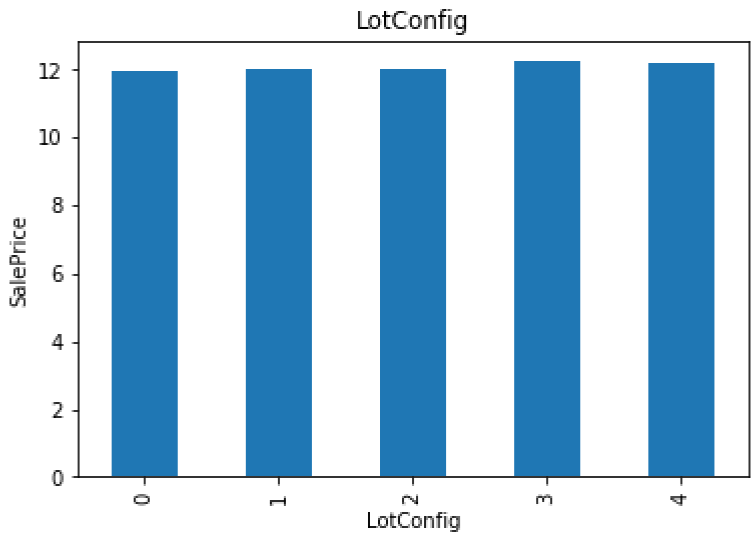


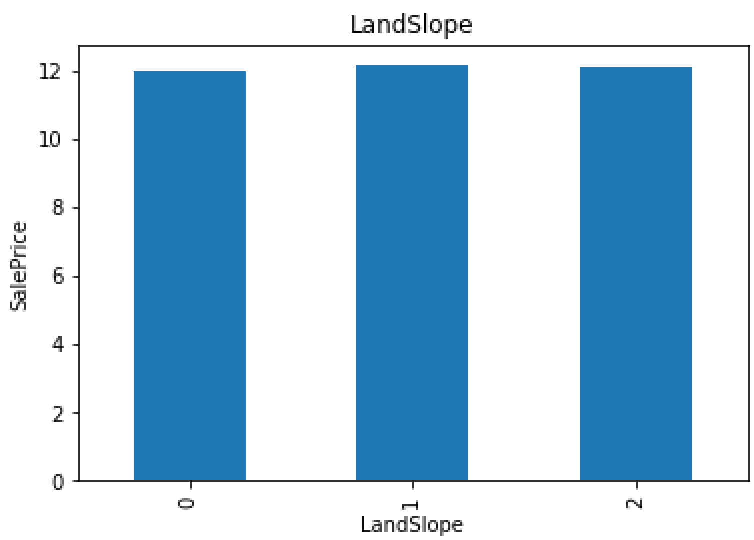

Alley
SalePrice
12
10
8
6
4
2
0
0
1
2
Alley

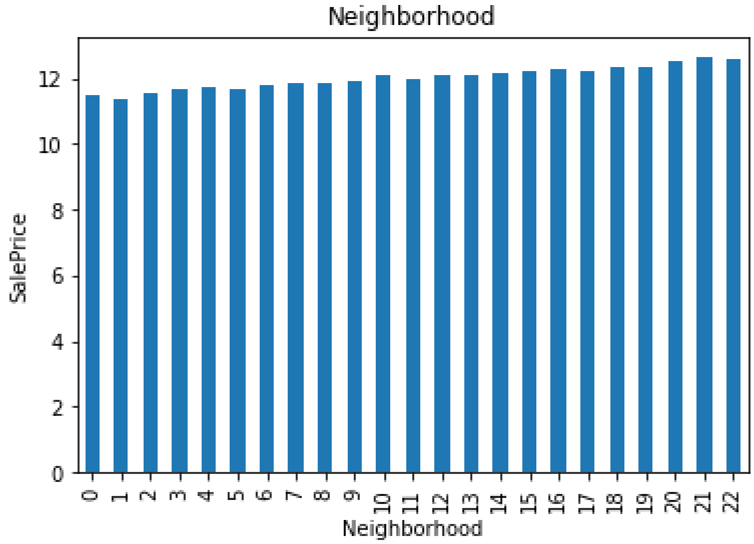

LotShape
SalePrice
12
10
8
6
4
2
0
0
1
2
3
LotShape

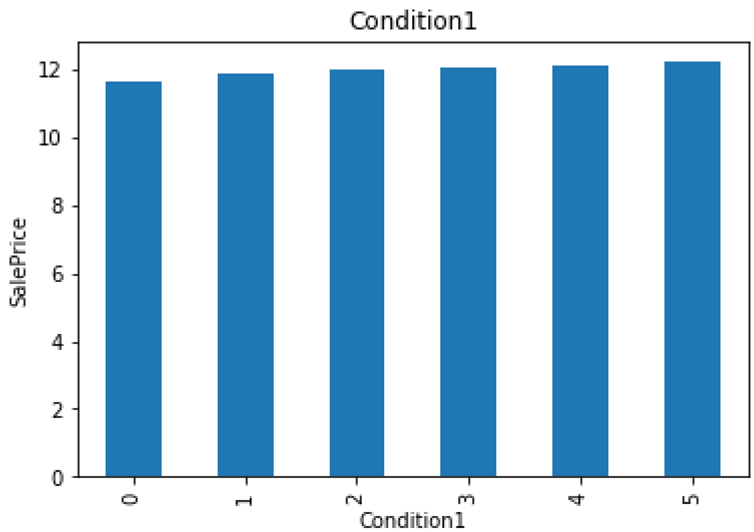

LandContour
SalePrice
12
10
8
6
4
2
0
0
1
2
3
LandContour

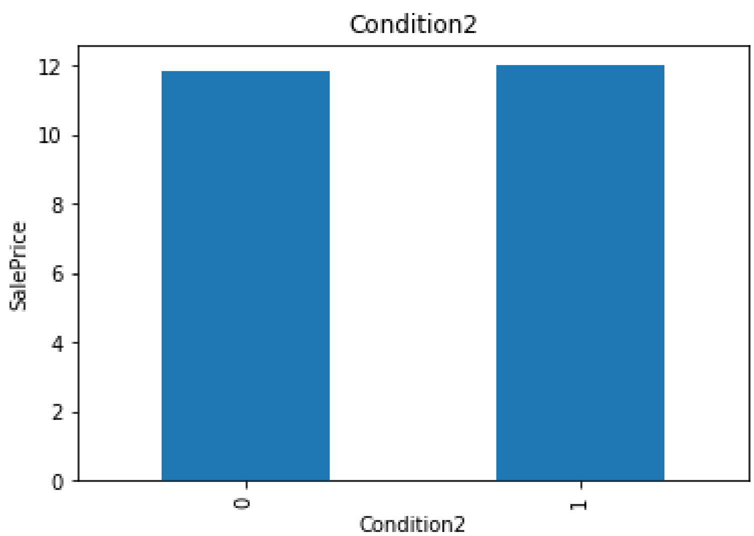

Utilities
SalePrice
Utilities
0
1
0
2
4
6
8
10
12

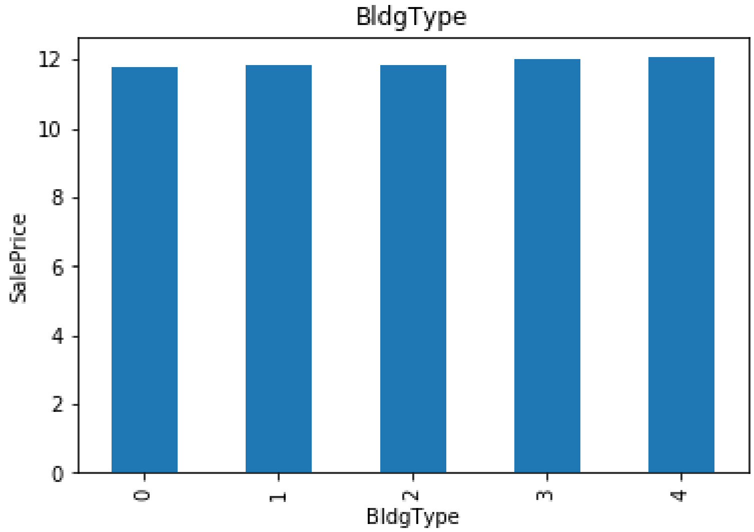

LotConfig
SalePrice
LotConfig
0
1
2
3
4
0
2
4
6
8
10
12

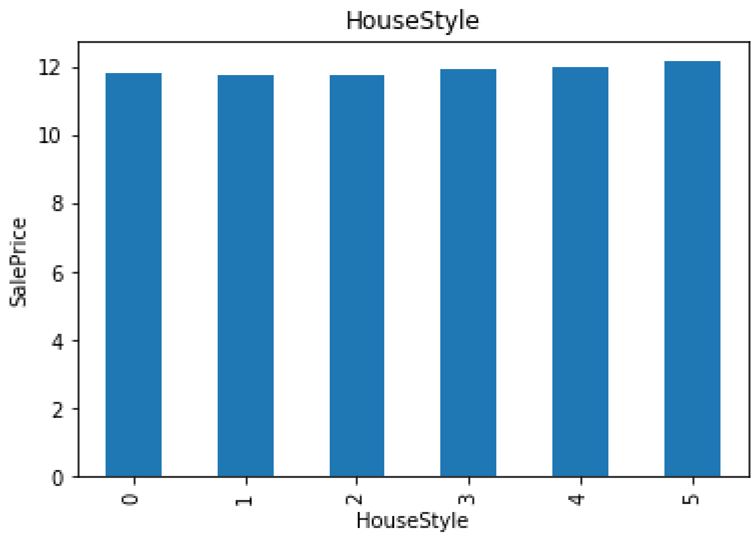

LandSlope
SalePrice
LandSlope
0
1
2
0
2
4
6
8
10
12

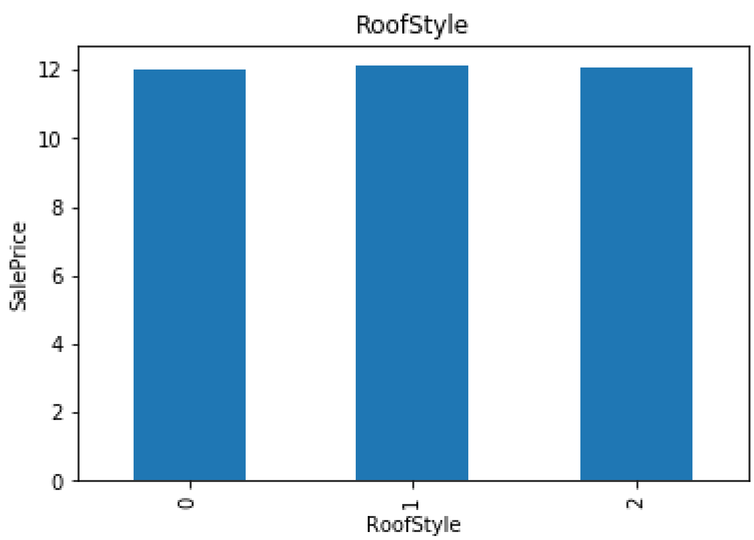

Neighborhood
SalePrice
0
2
4
6
8
10
12
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
Neighborhood

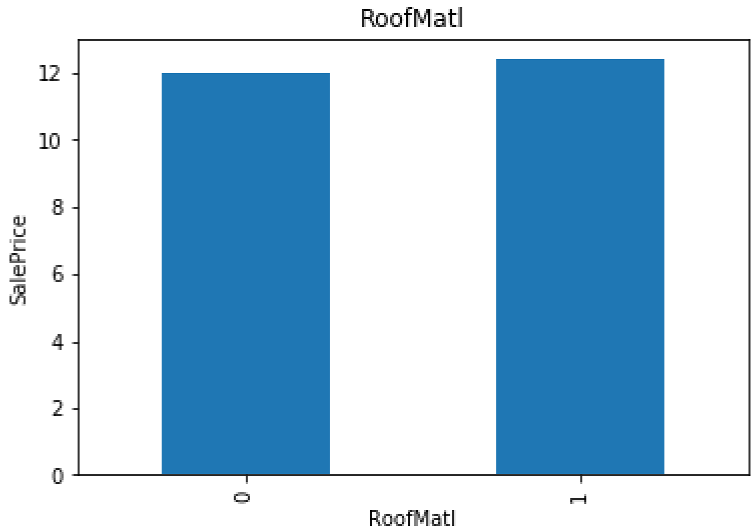

Condition1
SalePrice
0
2
4
6
8
10
12
0 1 2 3 4 5
Condition1

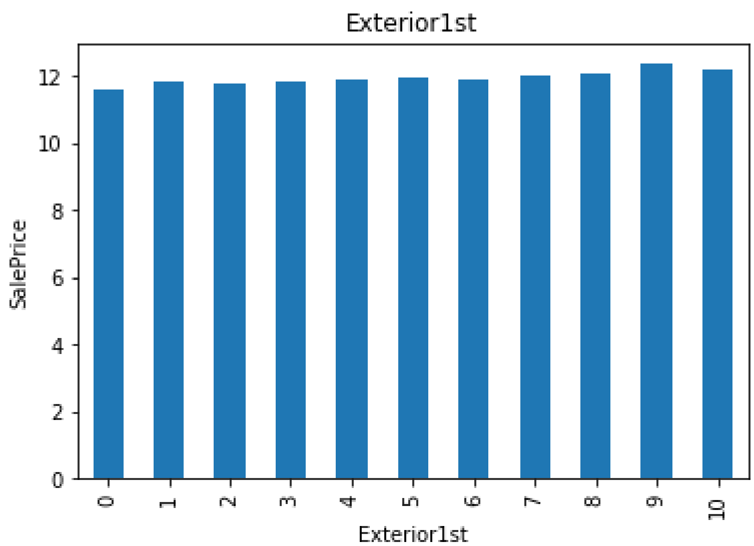

Condition2
SalePrice
0
2
4
6
8
10
12
0 1
Condition2

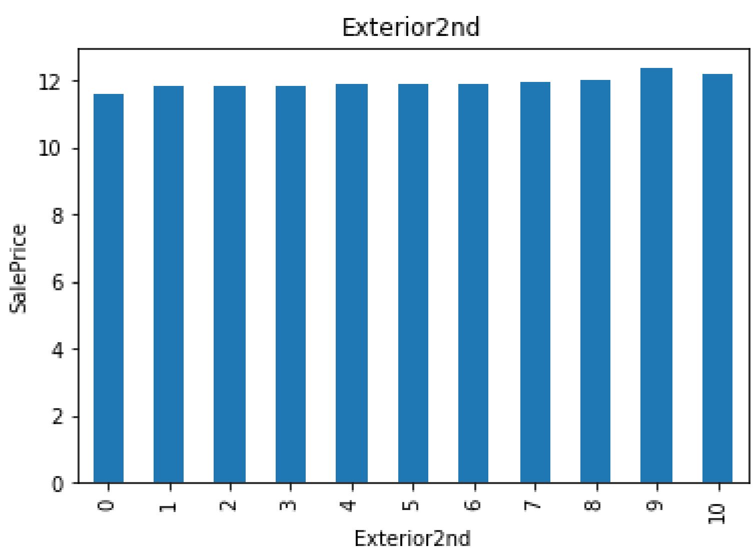

BldgType
12
10
8
6
4
2
0
SalePrice
0
1
2
3
4
BldgType

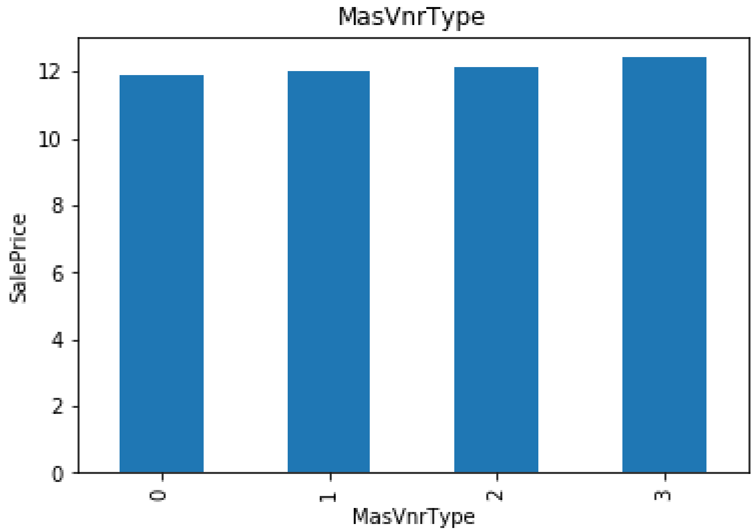

HouseStyle
12
10
8
6
4
2
0
SalePrice
0
1
2
3
4
5
HouseStyle

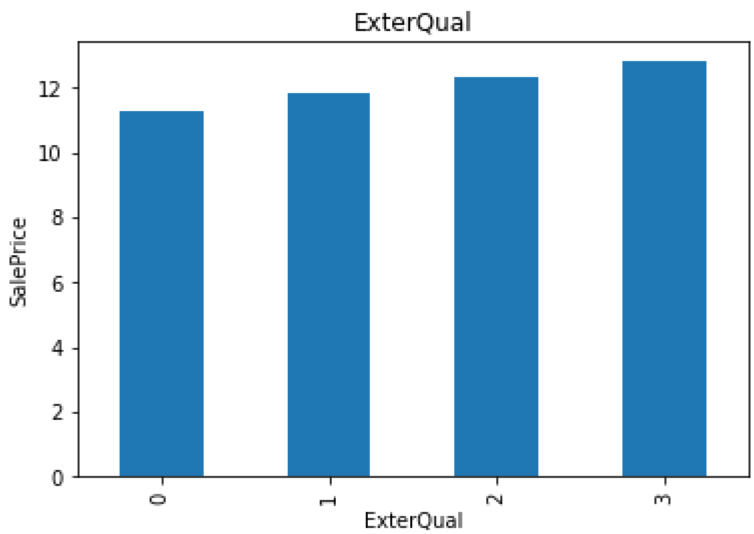

RoofStyle
12
10
8
6
4
2
0
SalePrice
0
1
2
RoofStyle

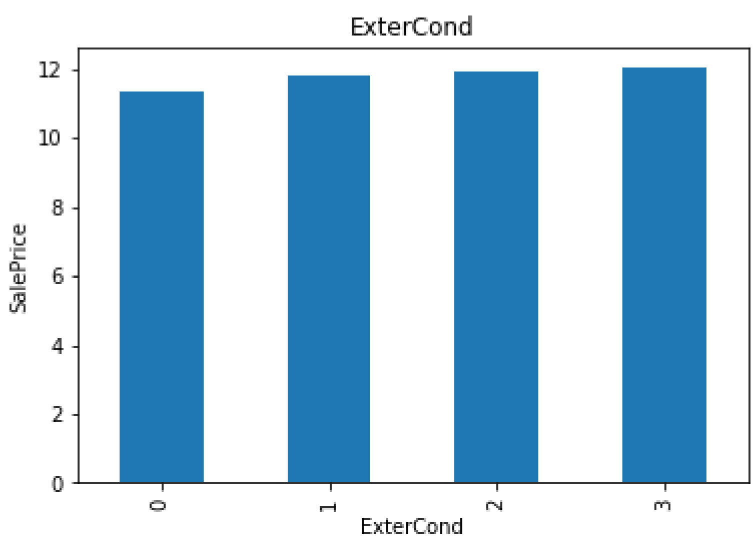

RoofMatl
SalePrice
0
2
4
6
8
10
12
0
1
RoofMatl

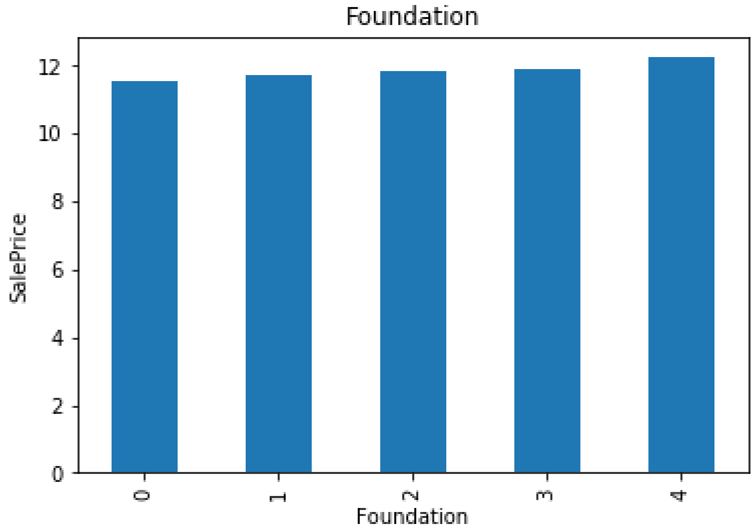

Exterior1st
SalePrice
0
2
4
6
8
10
12
0
1
2
3
4
5
6
7
8
9
10
Exterior1st

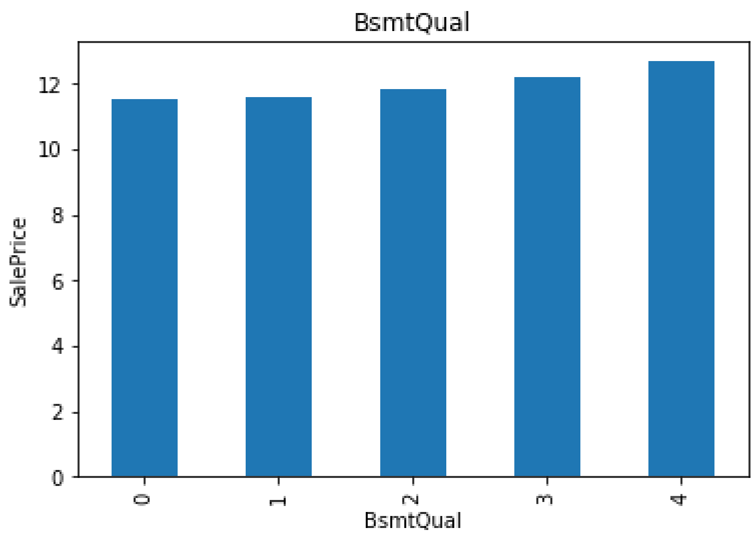

Exterior2nd
SalePrice
0
2
4
6
8
10
12
0
1
2
3
4
5
6
7
8
9
10
Exterior2nd

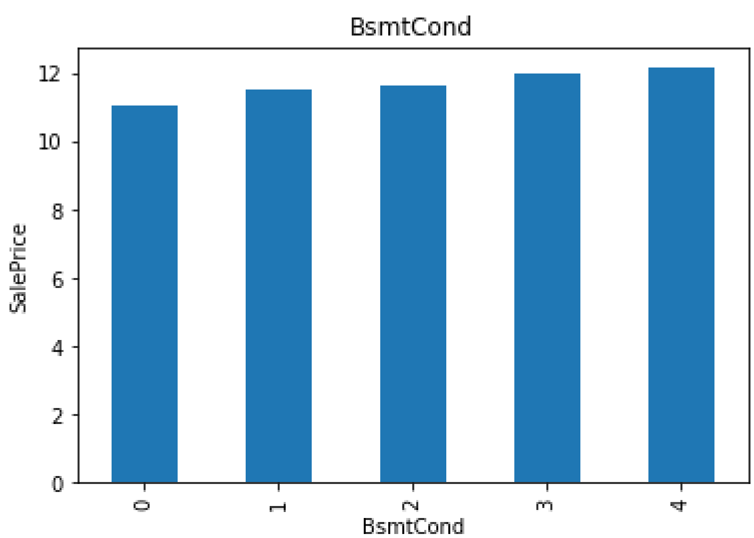

MasVnrType
SalePrice
0
2
4
6
8
10
12
0
1
2
3
MasVnrType

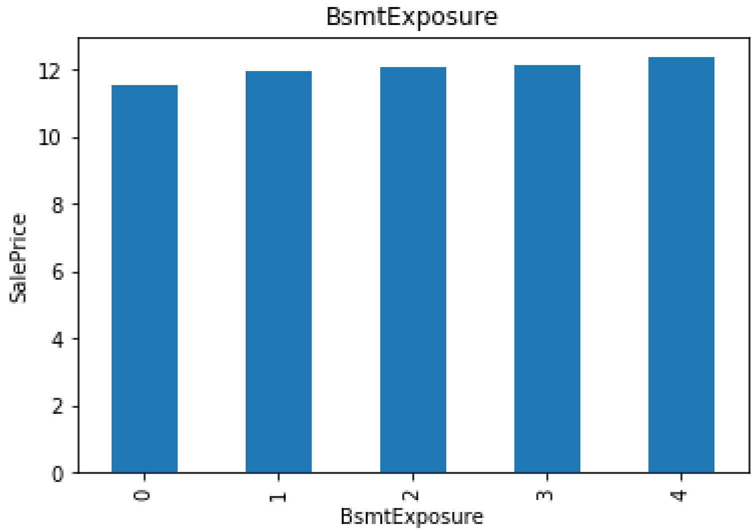

ExterQual
SalePrice
0
2
4
6
8
10
12
0
1
2
3
ExterQual

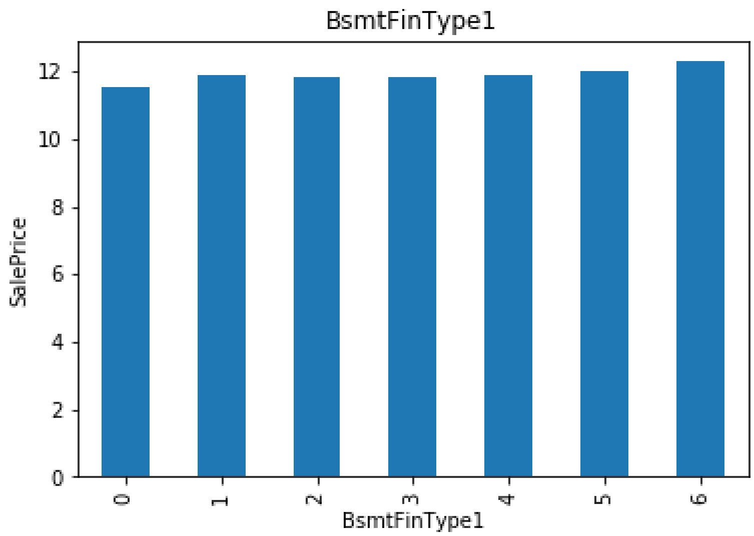

ExterCond
SalePrice
0
2
4
6
8
10
12
0
1
2
3
ExterCond

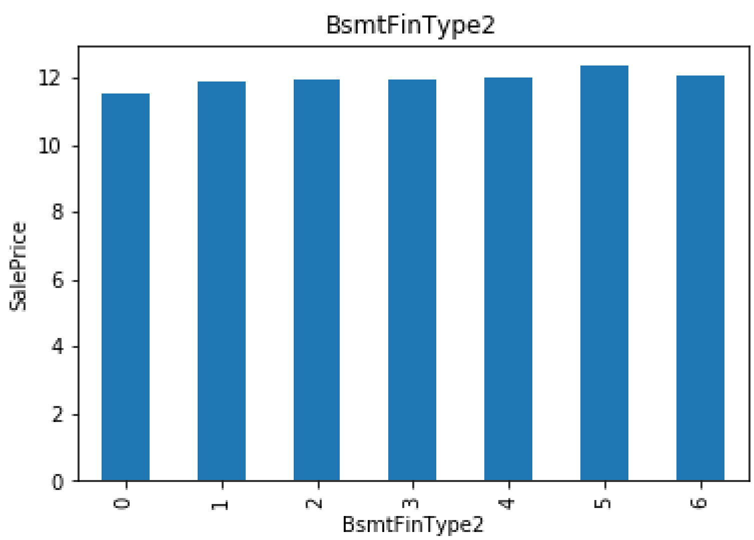

Foundation
SalePrice
0
2
4
6
8
10
12
0
1
2
3
4
Foundation

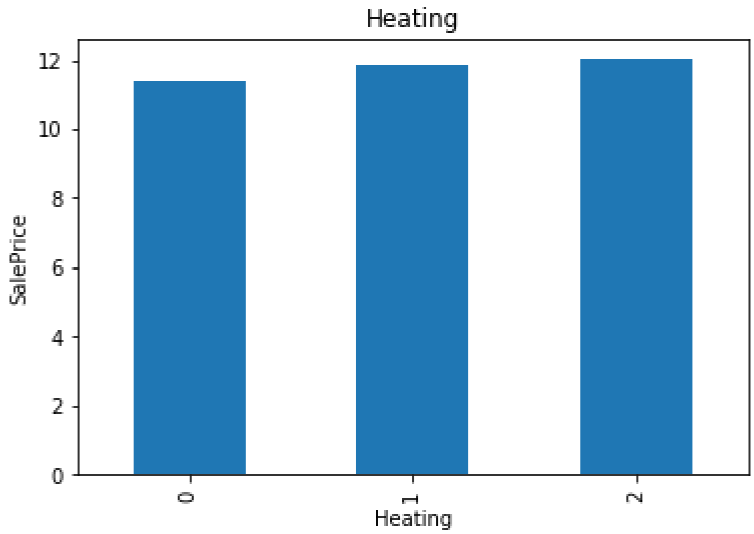

BsmtQual
SalePrice
0
2
4
6
8
10
12
0
1
2
3
4
BsmtQual

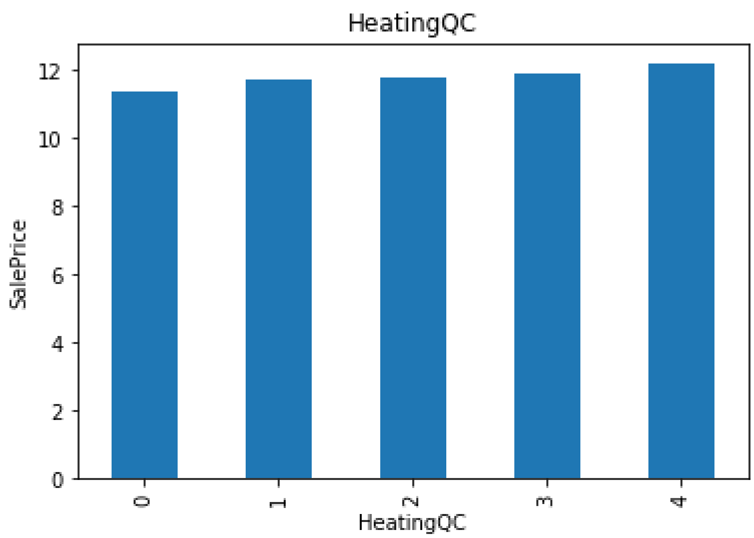

BsmtCond
SalePrice
0
2
4
6
8
10
12
0
1
2
3
4
BsmtCond

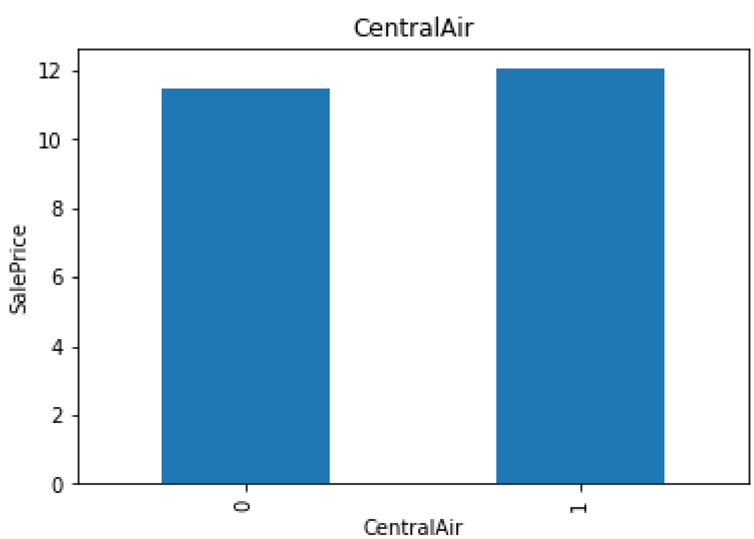

BsmtExposure
SalePrice
0
2
4
6
8
10
12
0
1
2
3
4
BsmtExposure

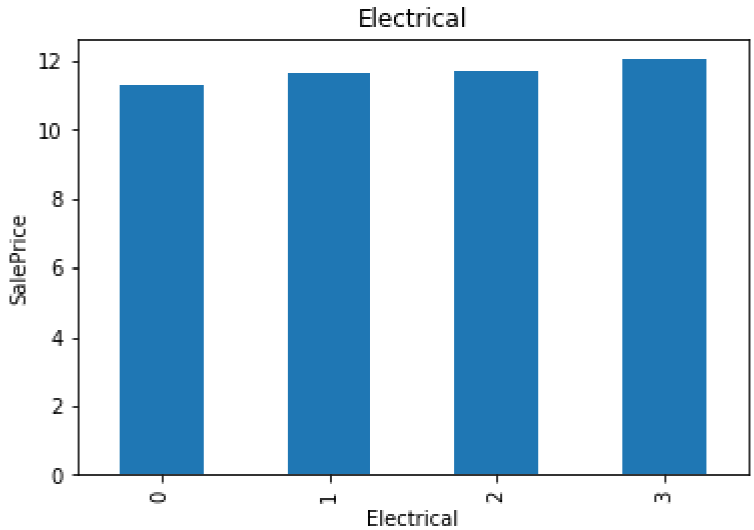

BsmtFinType1
SalePrice
0
2
4
6
8
10
12
0
1
2
3
4
5
6
BsmtFinType1

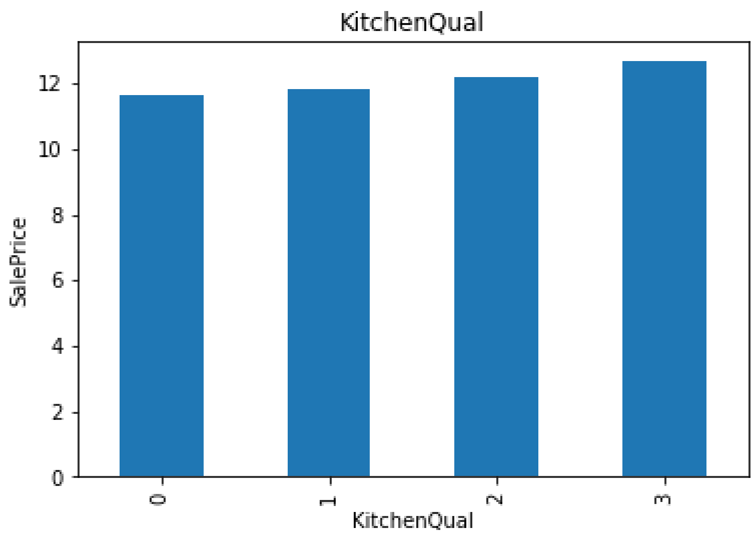

BsmtFinType2
SalePrice
0
2
4
6
8
10
12
0
1
2
3
4
5
6
BsmtFinType2

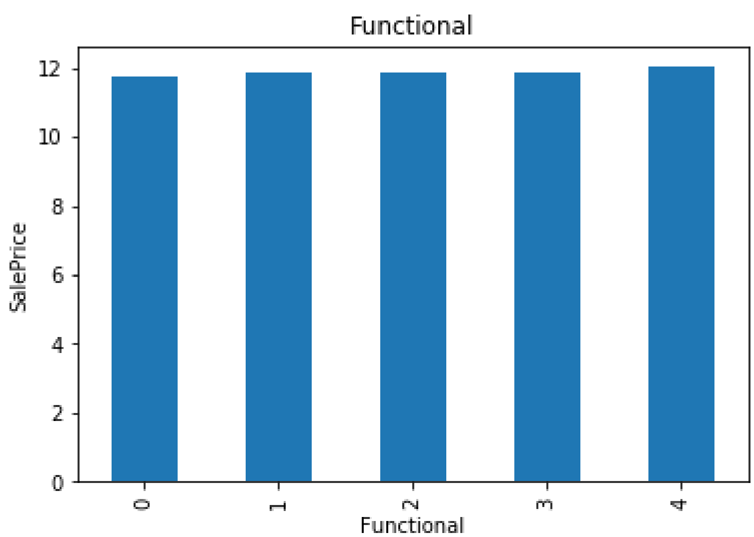

Heating
SalePrice
Heating
0
1
2
12
10
8
6
4
2
0

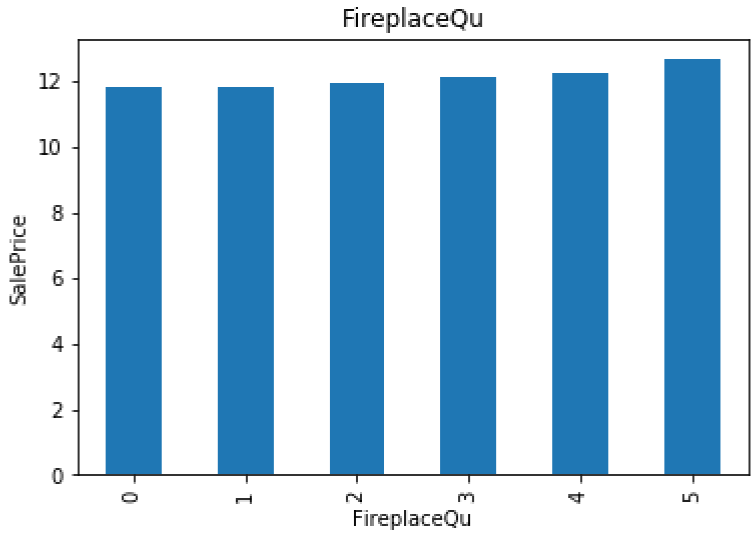

HeatingQC
SalePrice
HeatingQC
0
1
2
3
4
12
10
8
6
4
2
0

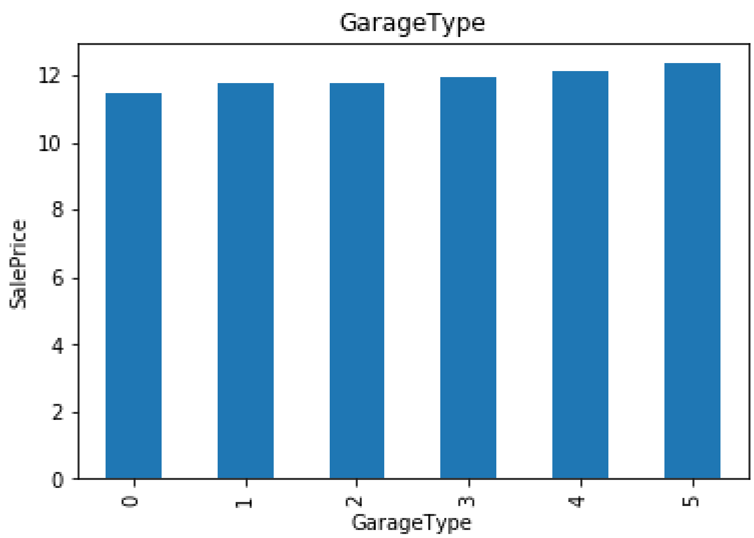

CentralAir
SalePrice
CentralAir
0
1
12
10
8
6
4
2
0

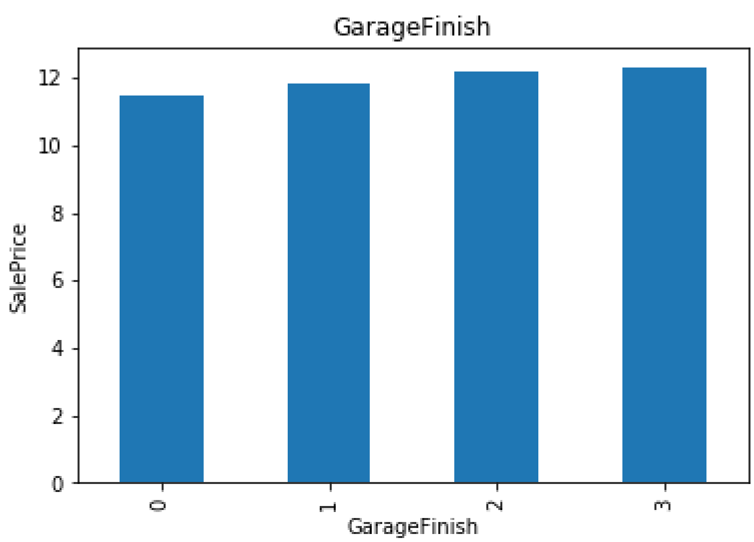

Electrical
12
10
8
6
4
2
0
SalePrice
0
1
2
3
Electrical

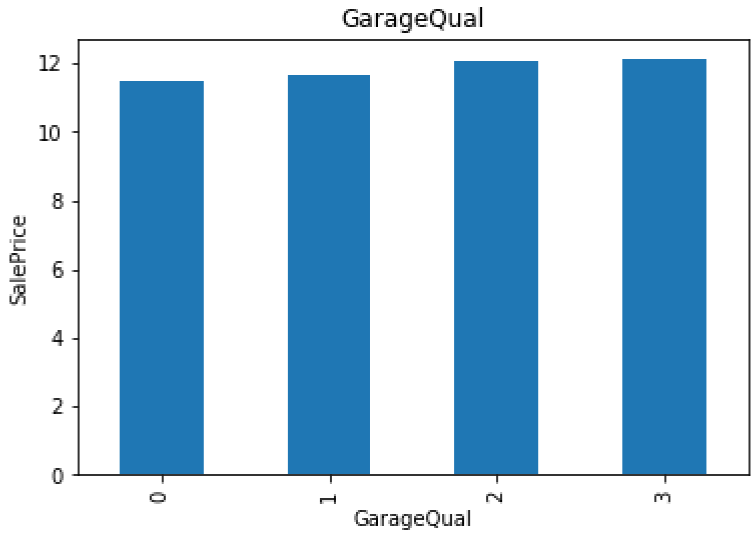

KitchenQual
12
10
8
6
4
2
0
SalePrice
0
1
2
3
KitchenQual

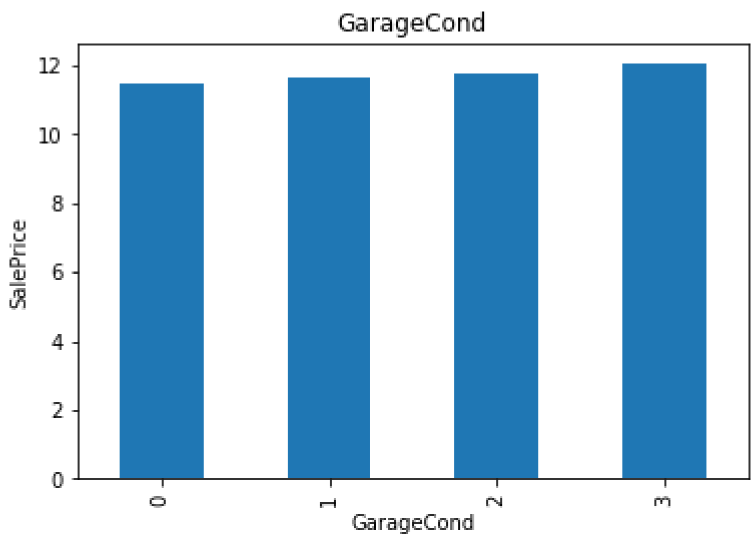

Functional
12
10
8
6
4
2
0
SalePrice
0
1
2
3
4
Functional

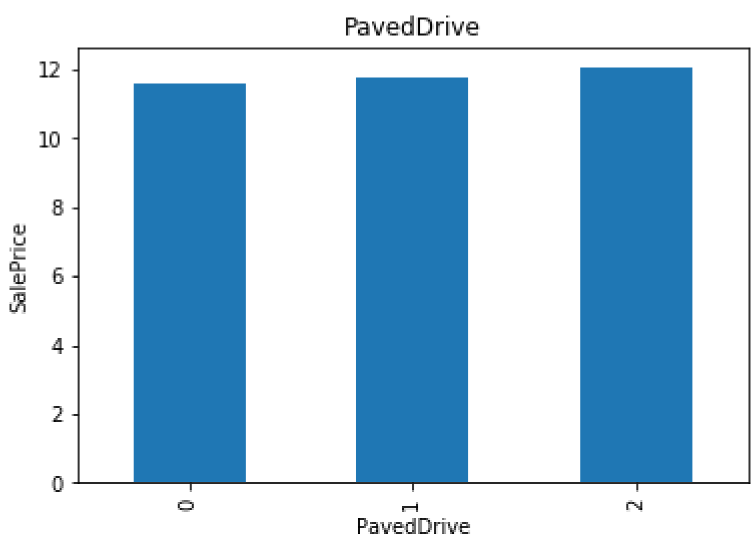

FireplaceQu
SalePrice
0
2
4
6
8
10
12
0
1
2
3
4
5
FireplaceQu

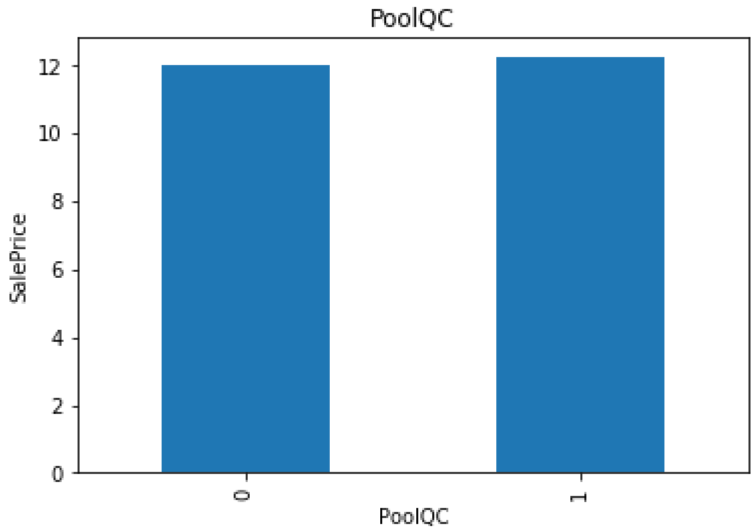

GarageType
SalePrice
0
2
4
6
8
10
12
0
1
2
3
4
5
GarageType

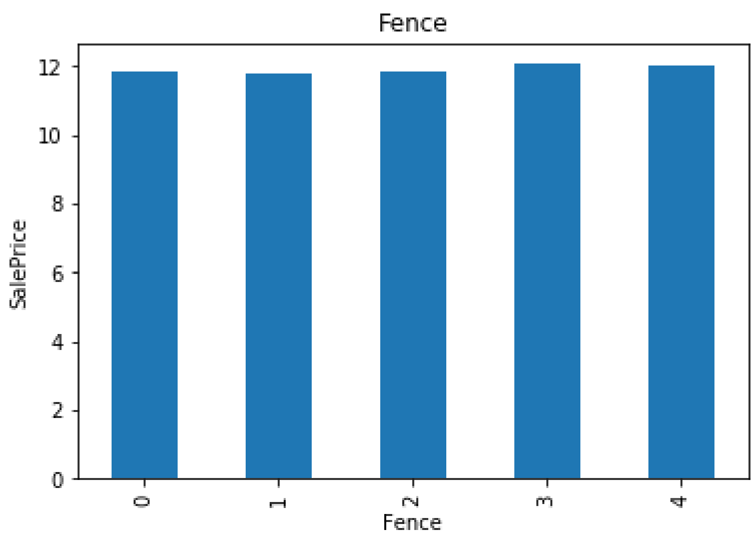

GarageFinish
SalePrice
0
2
4
6
8
10
12
0
1
2
3
GarageFinish

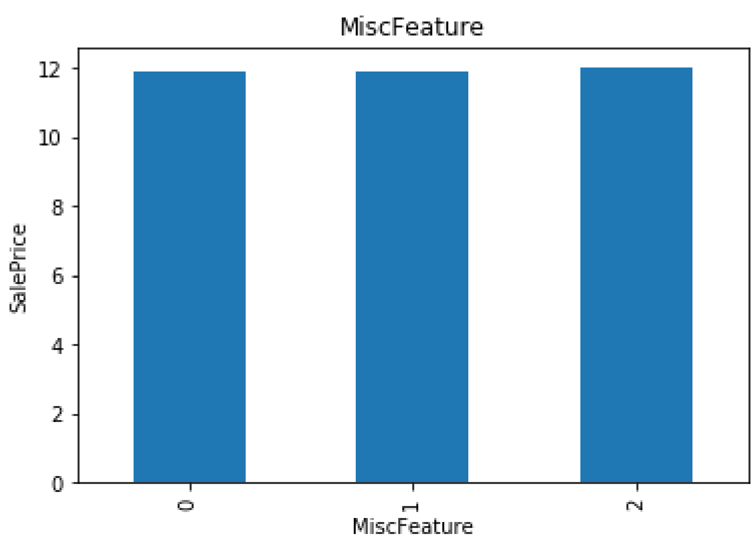

GarageQual
SalePrice
0
2
4
6
8
10
12
0
1
2
3
GarageQual

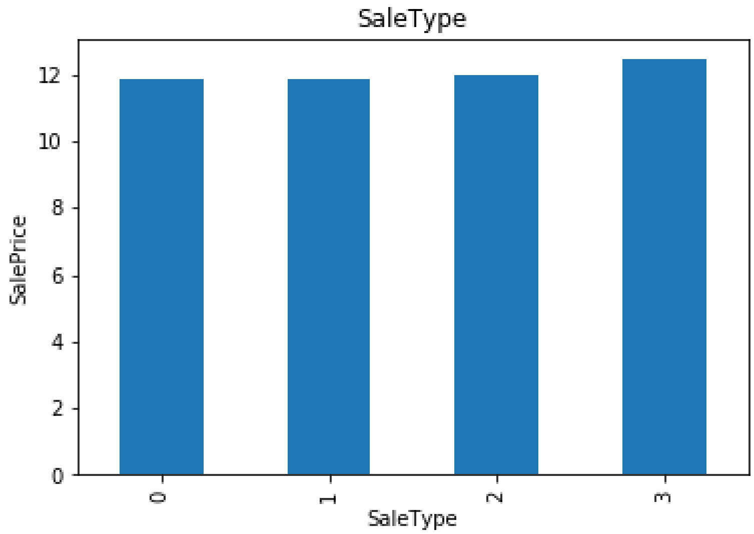

GarageCond
SalePrice
0
2
4
6
8
10
12
0
1
2
3
GarageCond

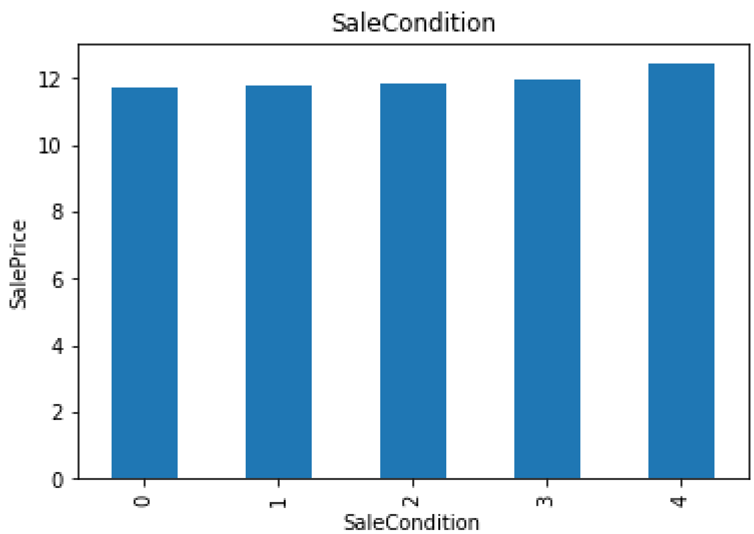

PavedDrive
SalePrice
0
2
4
6
8
10
12
0
1
2
PavedDrive


PoolQC
12
10
8
6
4
2
0
SalePrice
0
1
PoolQC


Fence
12
10
8
6
4
2
0
SalePrice
0
1
2
3
4
Fence


MiscFeature
12
10
8
6
4
2
0
SalePrice
0
1
2
MiscFeature





Feature Scaling

For use in linear models, features need to be either scaled or normalised.

In [45]:
```
train_vars = [var for var in X_train.columns if var not in ['Id', 'SalePrice']]

# count number of variables
len(train_vars)
```

Out[45]: 82

In [46]:
```
# create scaler
scaler = MinMaxScaler()

#  fit  the scaler to the train set
scaler.fit(X_train[train_vars])

# transform the train and test set
X_train[train_vars] = scaler.transform(X_train[train_vars])

X_test[train_vars] = scaler.transform(X_test[train_vars])
```

In [47]:
```
X_train.head()
```

Out[47]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Ut
930	931	0.000000	0.75	0.461171	0.377048	1.0	1.0	0.333333	1.000000	
656	657	0.000000	0.75	0.456066	0.399443	1.0	1.0	0.333333	0.333333	
45	46	0.588235	0.75	0.394699	0.347082	1.0	1.0	0.000000	0.333333	
1348	1349	0.000000	0.75	0.388581	0.493677	1.0	1.0	0.666667	0.666667	
55	56	0.000000	0.75	0.577658	0.402702	1.0	1.0	0.333333	0.333333	

In [48]:

```
X_train.to_csv('xtrain.csv', index=False)
X_test.to_csv('xtest.csv', index=False)
```

In [ ]: