Real Data

Assignment 4b - CH17B066

Load the Housing_Price.csv and see the feature_discription.txt for more insight (However you don't need to remove any unneccesary feature just use encoding to convert the categorical features)

In [1]:

```
#Loading libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
# load data
data = pd.read_csv('Housing_Price.csv')
data.head()
```

Out[2]:

	Id	MSSubClass	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	BsmtFinSI
0	1	60	8450	7	5	2003	2003	7(
1	2	20	9600	6	8	1976	1976	9
2	3	60	11250	7	5	2001	2002	4!
3	4	70	9550	7	5	1915	1970	2
4	5	60	14260	8	5	2000	2000	6!

5 rows × 42 columns

which are categorical and which are numerical features?

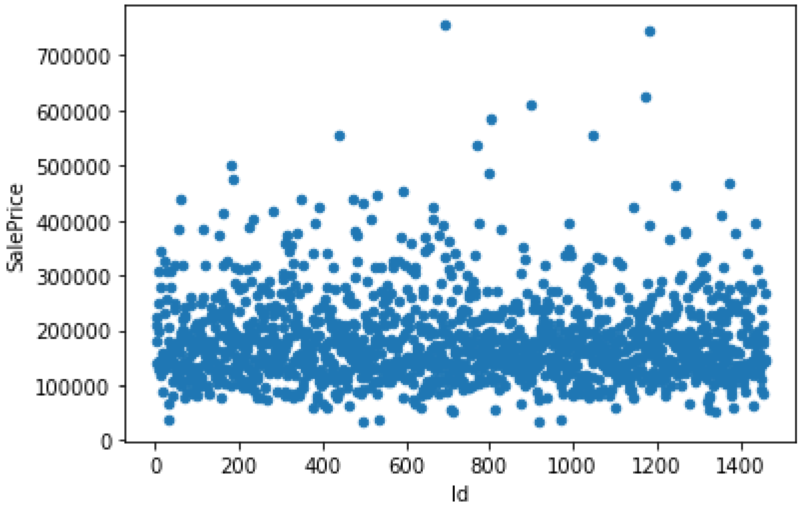
In [3]:

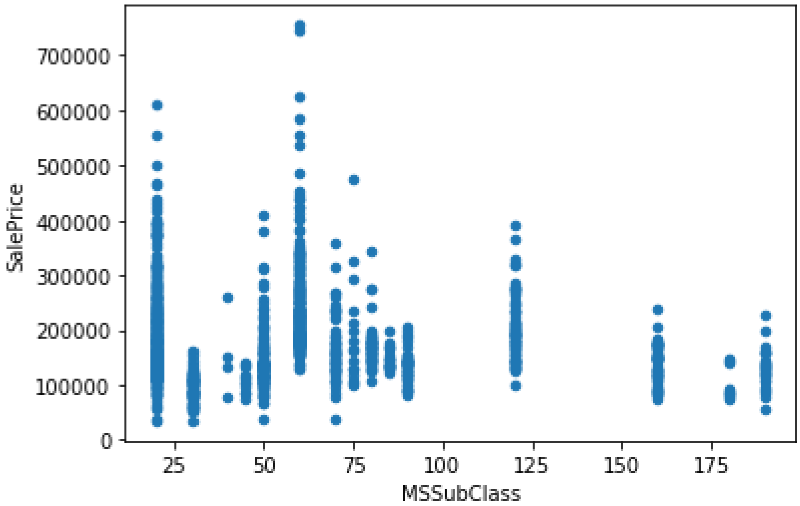
```
# you can print and analyse the data type to say about categorical features
data.info()
```

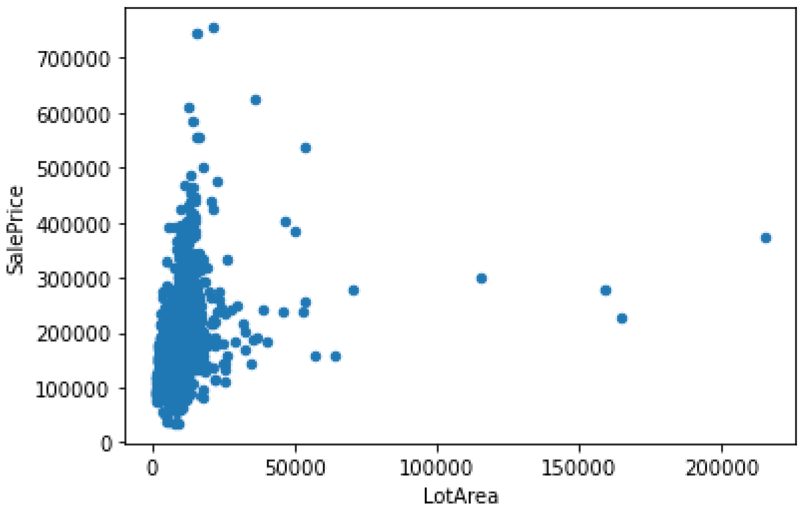
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 42 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Id             1460 non-null   int64 
 1   MSSubClass     1460 non-null   int64 
 2   LotArea        1460 non-null   int64 
 3   OverallQual    1460 non-null   int64 
 4   OverallCond    1460 non-null   int64 
 5   YearBuilt      1460 non-null   int64 
 6   YearRemodAdd   1460 non-null   int64 
 7   BsmtFinSF1     1460 non-null   int64 
 8   BsmtFinSF2     1460 non-null   int64 
 9   BsmtUnfSF      1460 non-null   int64 
 10  TotalBsmtSF    1460 non-null   int64 
 11  1stFlrSF       1460 non-null   int64 
 12  2ndFlrSF       1460 non-null   int64 
 13  LowQualFinSF   1460 non-null   int64 
 14  GrLivArea      1460 non-null   int64 
 15  BsmtFullBath   1460 non-null   int64 
 16  BsmtHalfBath   1460 non-null   int64 
 17  FullBath       1460 non-null   int64 
 18  HalfBath       1460 non-null   int64 
 19  BedroomAbvGr   1460 non-null   int64 
 20  KitchenAbvGr   1460 non-null   int64 
 21  TotRmsAbvGrd   1460 non-null   int64 
 22  Fireplaces     1460 non-null   int64 
 23  GarageCars     1460 non-null   int64 
 24  GarageArea     1460 non-null   int64 
 25  WoodDeckSF     1460 non-null   int64 
 26  OpenPorchSF    1460 non-null   int64 
 27  EnclosedPorch  1460 non-null   int64 
 28  3SsnPorch      1460 non-null   int64 
 29  ScreenPorch    1460 non-null   int64 
 30  PoolArea       1460 non-null   int64 
 31  MiscVal        1460 non-null   int64 
 32  MoSold         1460 non-null   int64 
 33  YrSold         1460 non-null   int64 
 34  SalePrice      1460 non-null   int64 
 35  Street         1460 non-null   object
 36  Condition1     1460 non-null   object
 37  Condition2     1460 non-null   object
 38  CentralAir     1460 non-null   object
 39  HeatingQC      1460 non-null   object
 40  LotShape       1460 non-null   object
 41  LandContour    1460 non-null   object
dtypes: int64(35), object(7)
memory usage: 479.2+ KB
```

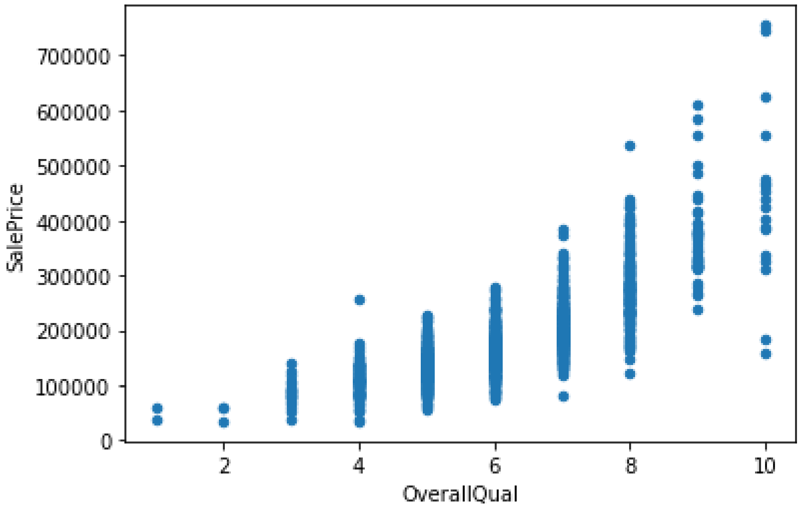
In [4]:

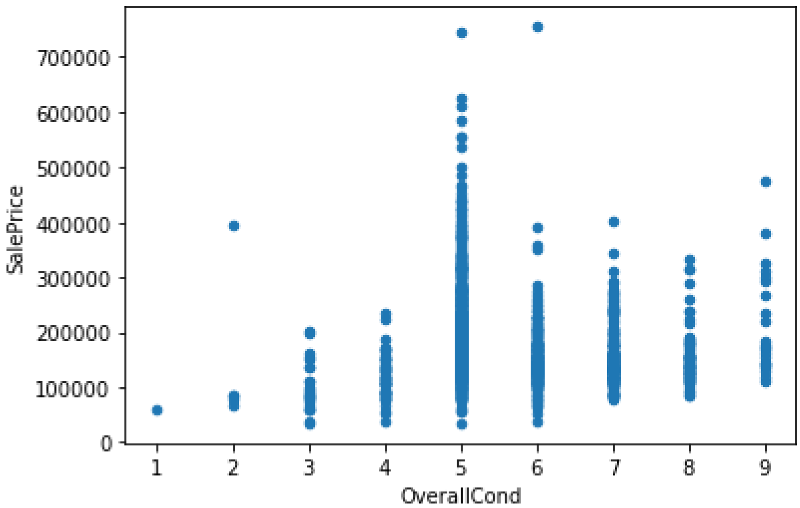
```
for i in range(len(data.columns)-1):
    data.plot(kind = 'scatter', x = data.columns[i], y= 'SalePrice')
```

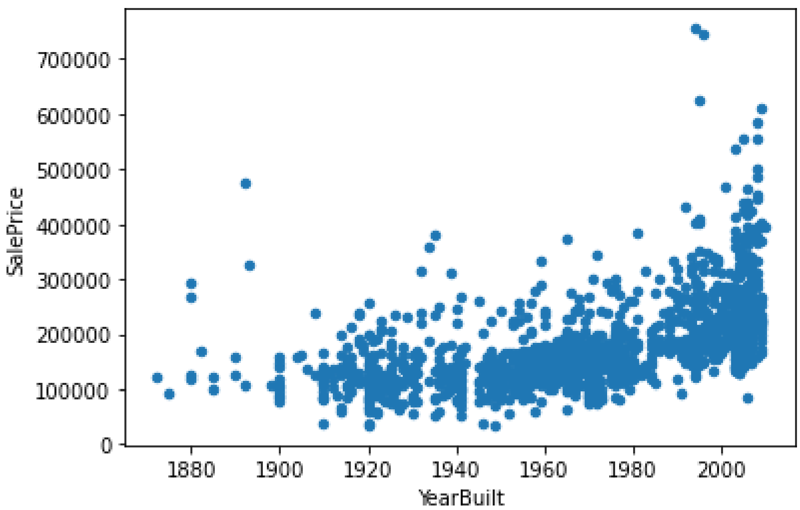

SalePrice
Id

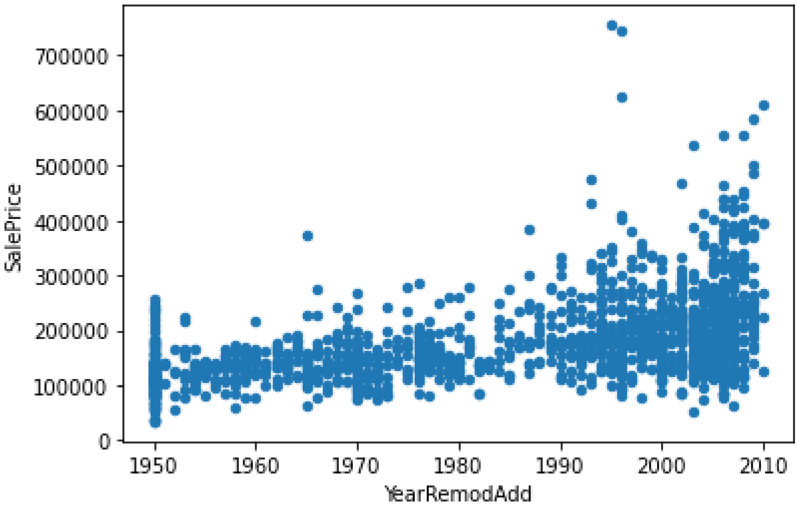

SalePrice
MSSubClass


SalePrice
LotArea

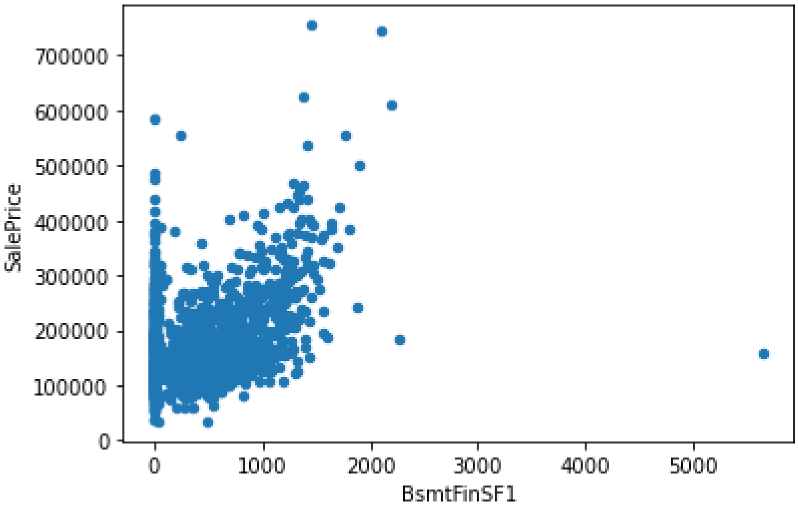

SalePrice
OverallQual
0
100000
200000
300000
400000
500000
600000
700000
2
4
6
8
10


SalePrice
OverallCond
0
100000
200000
300000
400000
500000
600000
700000
1
2
3
4
5
6
7
8
9

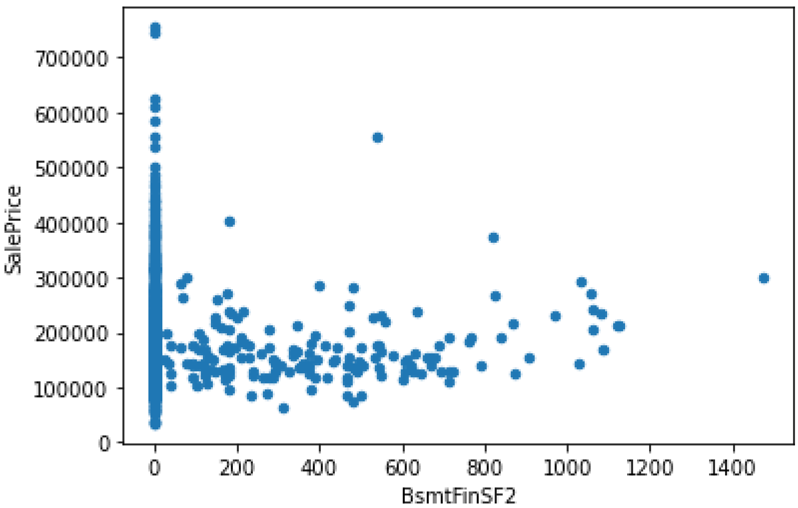

SalePrice
YearBuilt
0
100000
200000
300000
400000
500000
600000
700000
1880
1900
1920
1940
1960
1980
2000



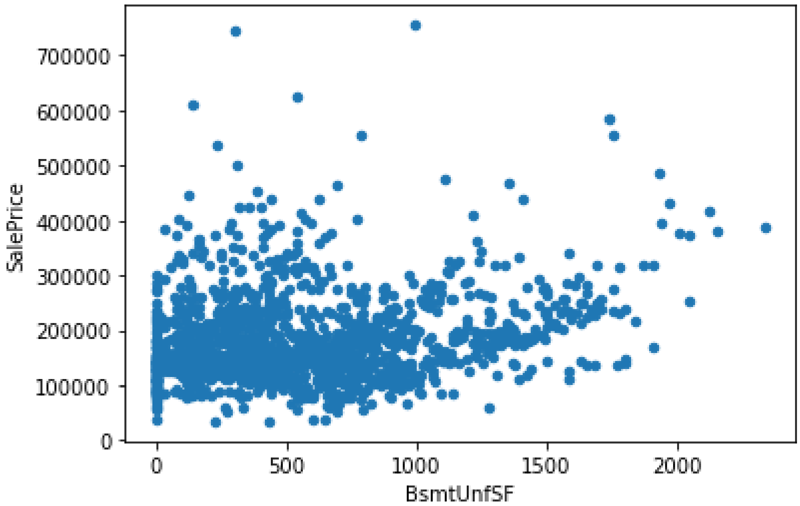
SalePrice
YearRemodAdd
0
100000
200000
300000
400000
500000
600000
700000
1950
1960
1970
1980
1990
2000
2010

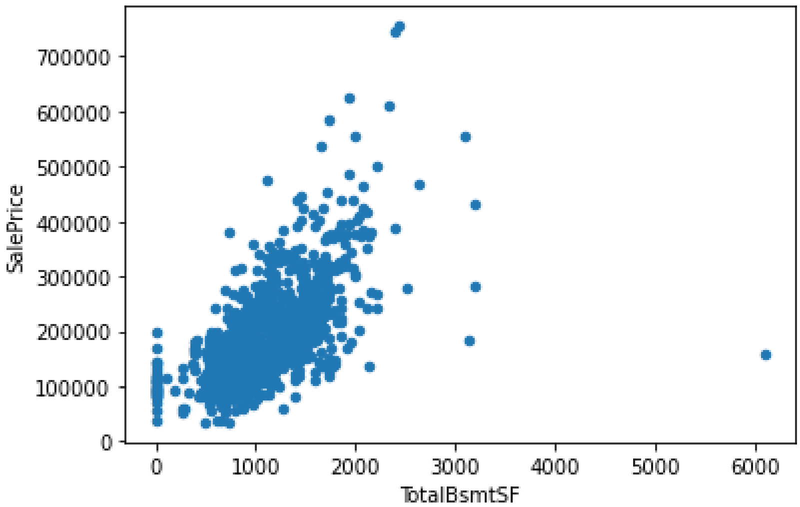


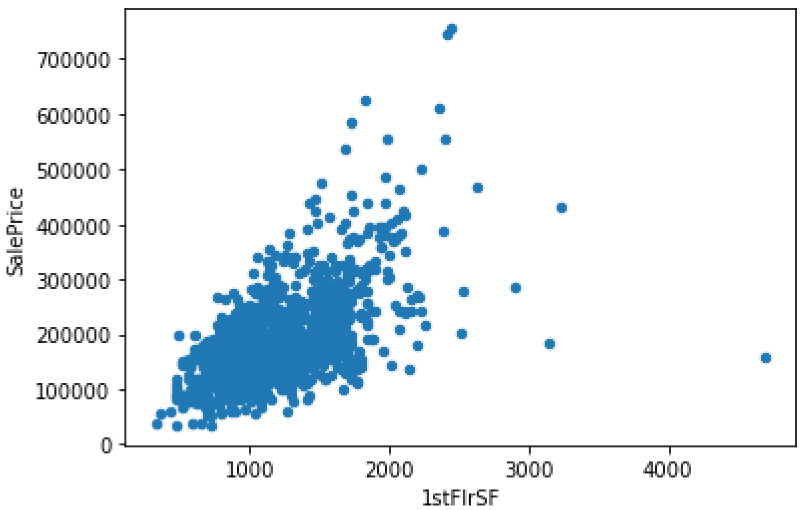
SalePrice
BsmtFinSF1
0
100000
200000
300000
400000
500000
600000
700000
0
1000
2000
3000
4000
5000

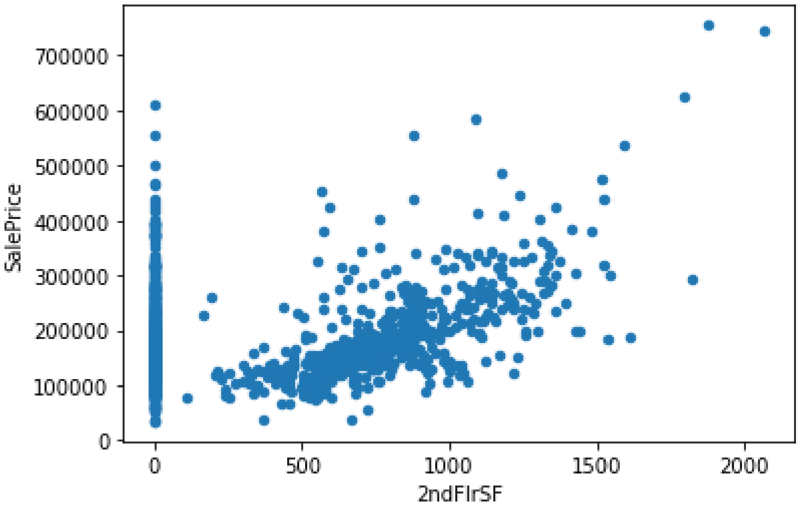


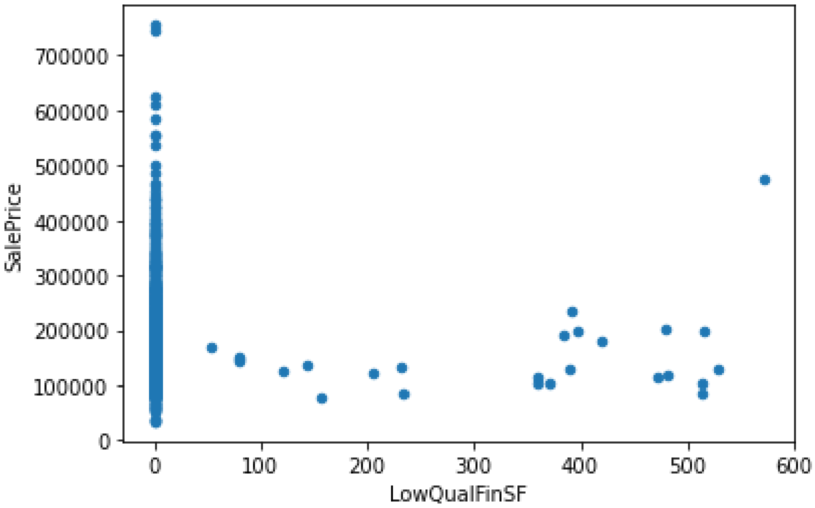
SalePrice
BsmtFinSF2
0
100000
200000
300000
400000
500000
600000
700000
0
200
400
600
800
1000
1200
1400

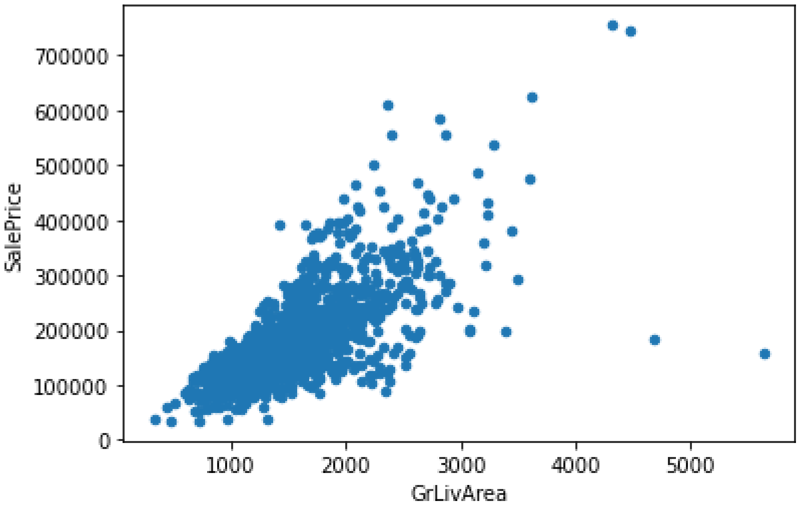

SalePrice
0
100000
200000
300000
400000
500000
600000
700000
0
500
1000
1500
2000
BsmtUnfSF

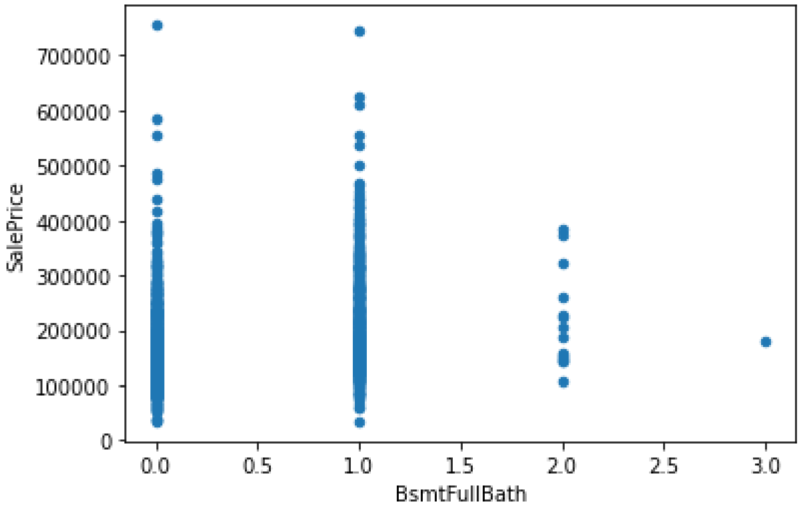

SalePrice
0
100000
200000
300000
400000
500000
600000
700000
0
1000
2000
3000
4000
5000
6000
TotalBsmtSF

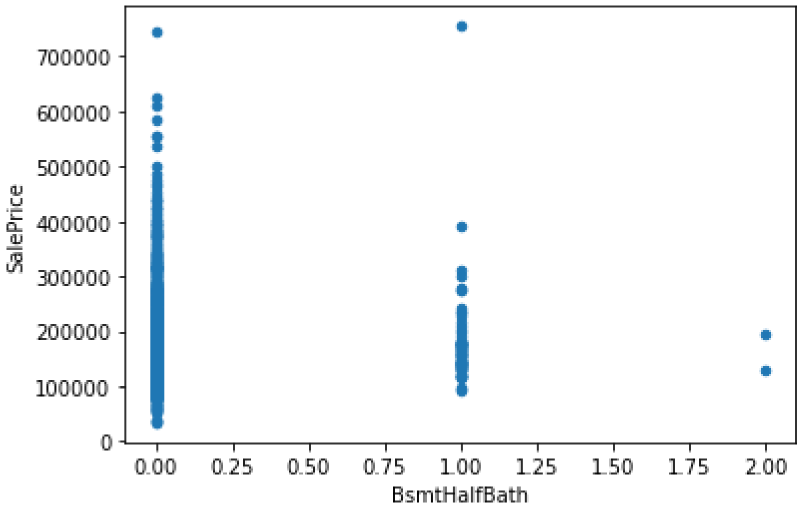

SalePrice
0
100000
200000
300000
400000
500000
600000
700000
1000
2000
3000
4000
1stFlrSF

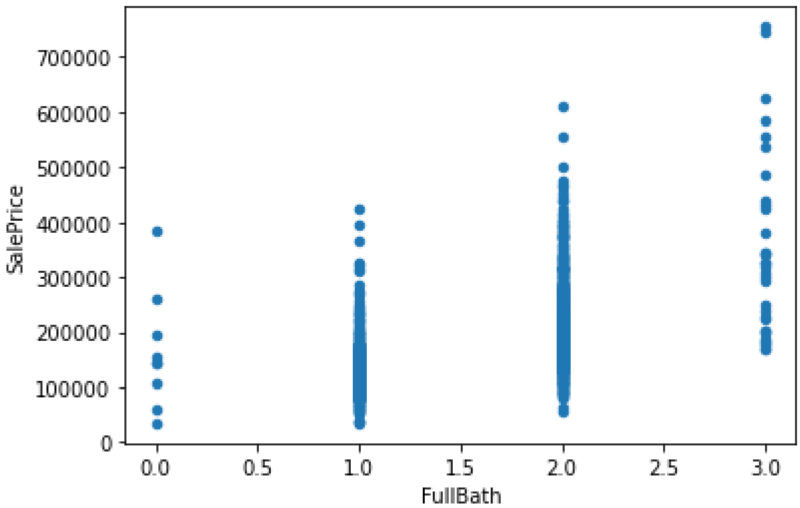

SalePrice
700000
600000
500000
400000
300000
200000
100000
0
0
500
1000
1500
2000
2ndFlrSF

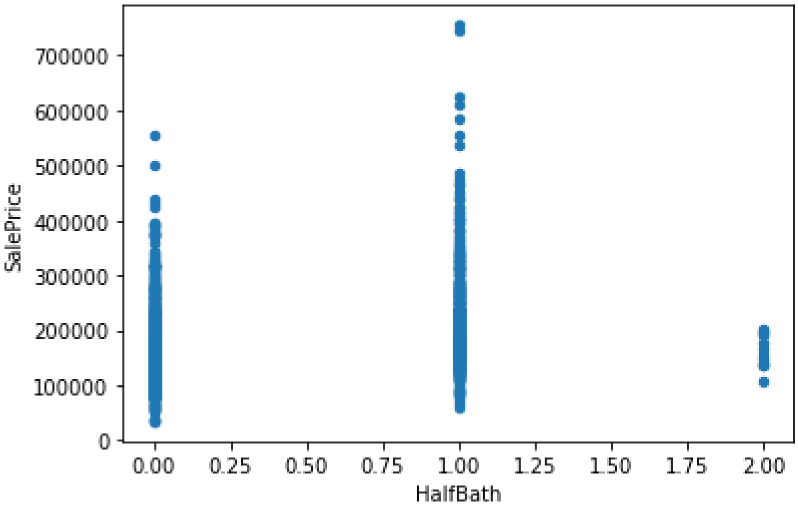

SalePrice
700000
600000
500000
400000
300000
200000
100000
0
0
100
200
300
400
500
600
LowQualFinSF

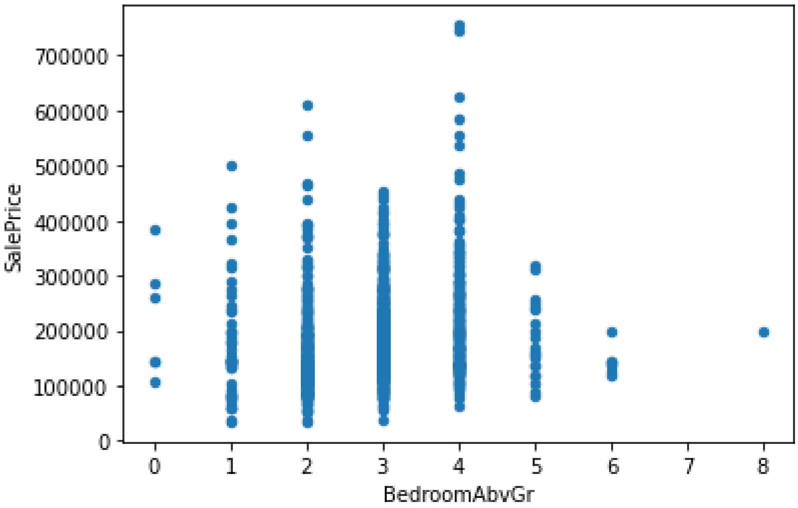

SalePrice
700000
600000
500000
400000
300000
200000
100000
0
1000
2000
3000
4000
5000
GrLivArea

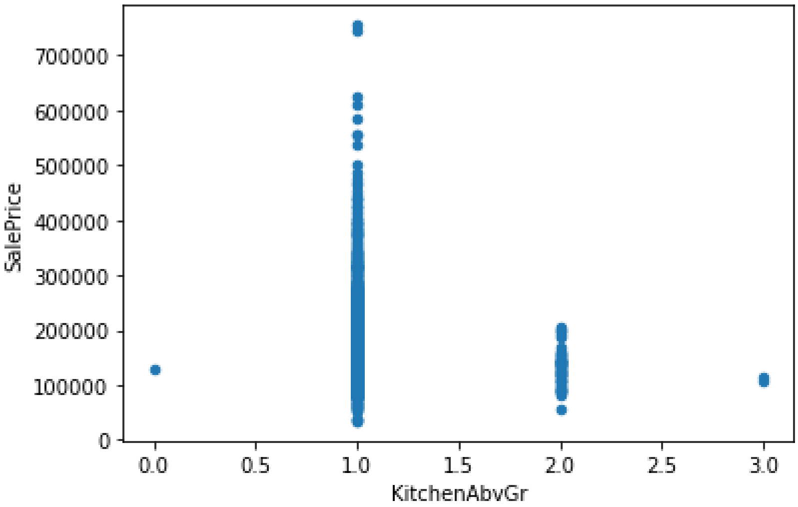

SalePrice
0
100000
200000
300000
400000
500000
600000
700000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
BsmtFullBath

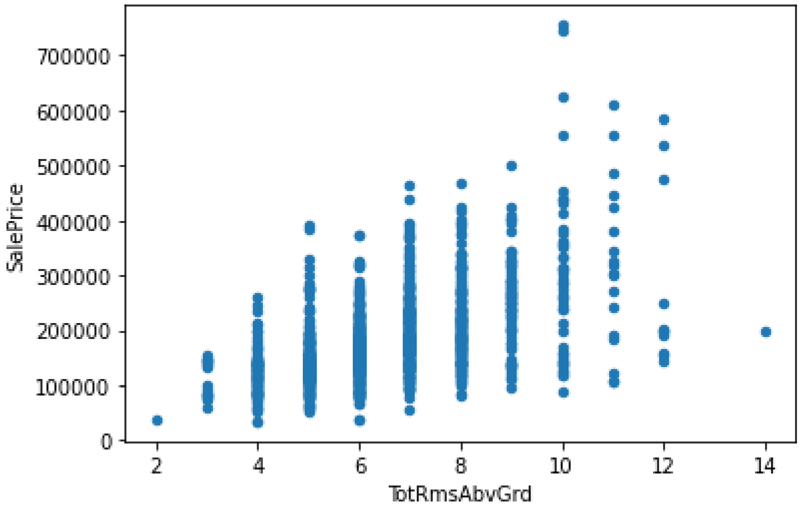

SalePrice
0
100000
200000
300000
400000
500000
600000
700000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
BsmtHalfBath

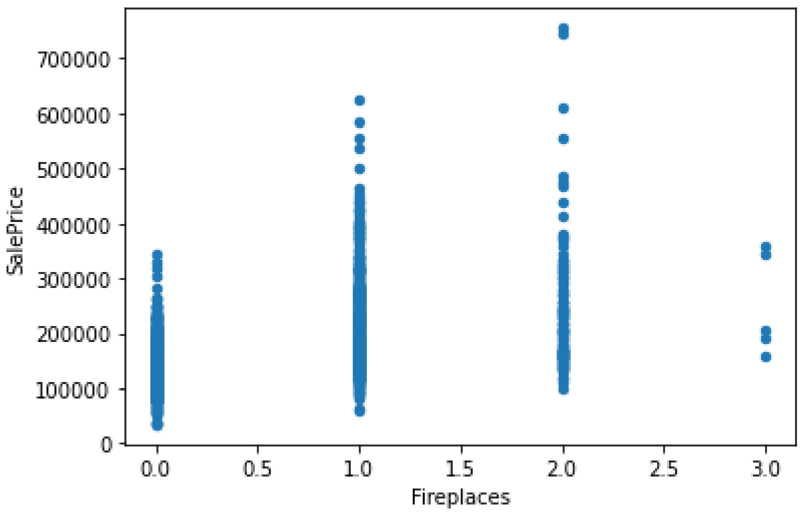

SalePrice
0
100000
200000
300000
400000
500000
600000
700000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
FullBath

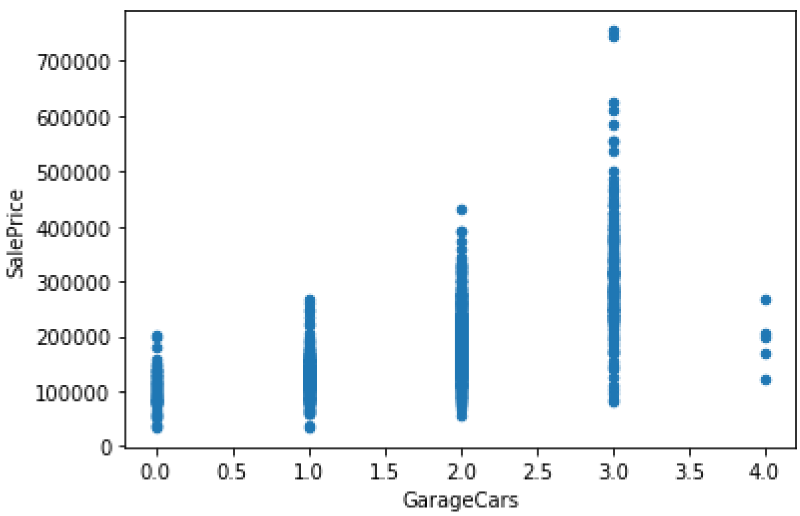

SalePrice
HalfBath
0
100000
200000
300000
400000
500000
600000
700000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

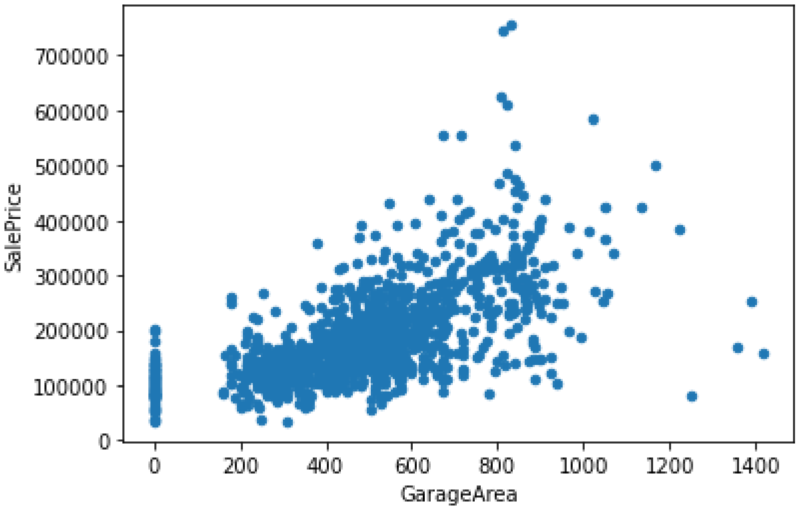

SalePrice
BedroomAbvGr
0
100000
200000
300000
400000
500000
600000
700000
0
1
2
3
4
5
6
7
8

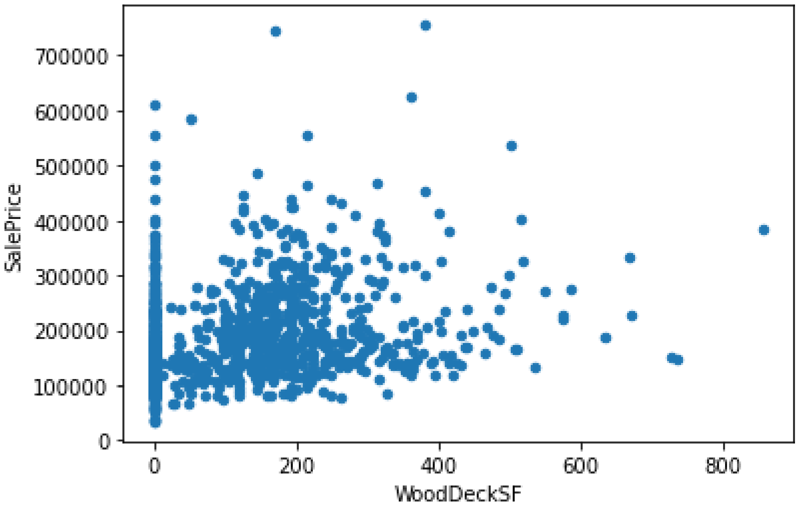

SalePrice
KitchenAbvGr
0
100000
200000
300000
400000
500000
600000
700000
0.0
0.5
1.0
1.5
2.0
2.5
3.0

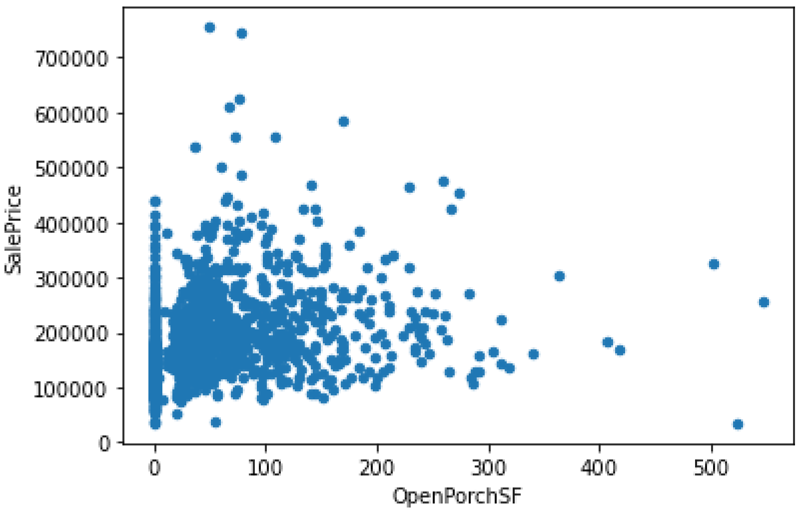

SalePrice
TotRmsAbvGrd
0
100000
200000
300000
400000
500000
600000
700000
2
4
6
8
10
12
14

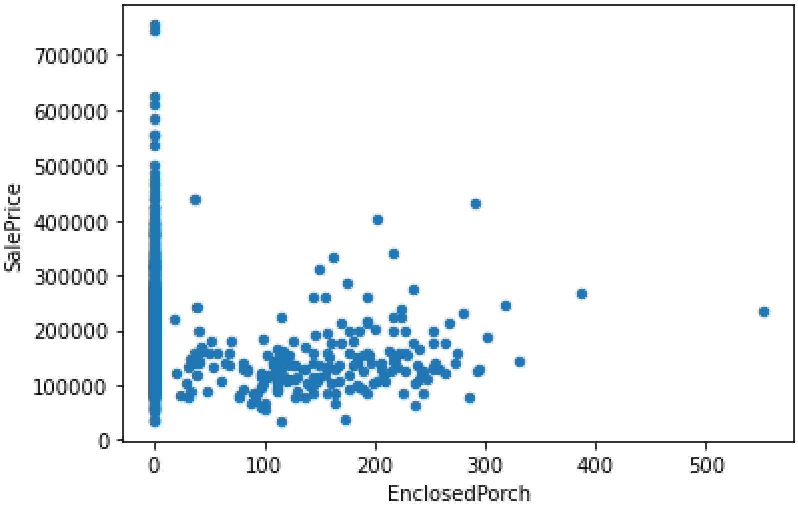

SalePrice
Fireplaces
0
100000
200000
300000
400000
500000
600000
700000
0.0
0.5
1.0
1.5
2.0
2.5
3.0

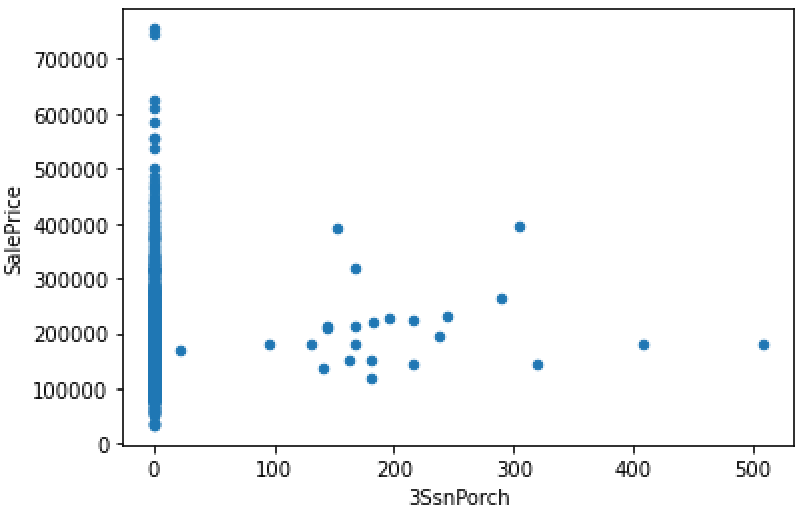

SalePrice
GarageCars
0
100000
200000
300000
400000
500000
600000
700000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

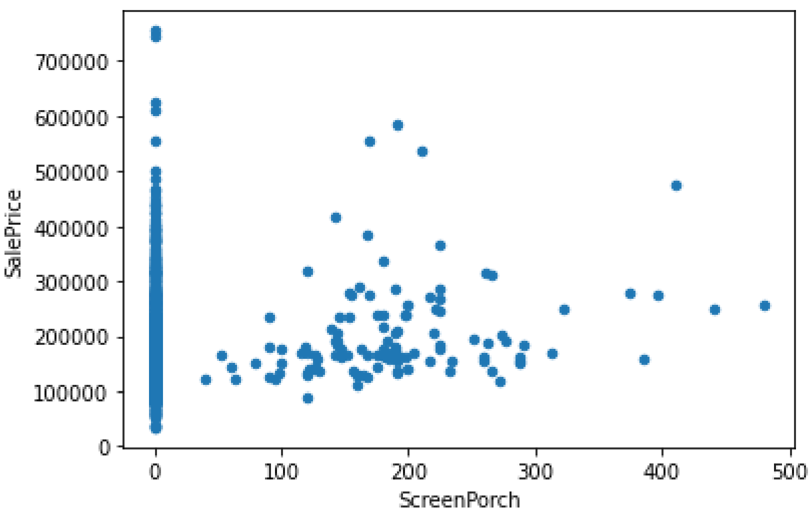

SalePrice
GarageArea
0
100000
200000
300000
400000
500000
600000
700000
0 200 400 600 800 1000 1200 1400

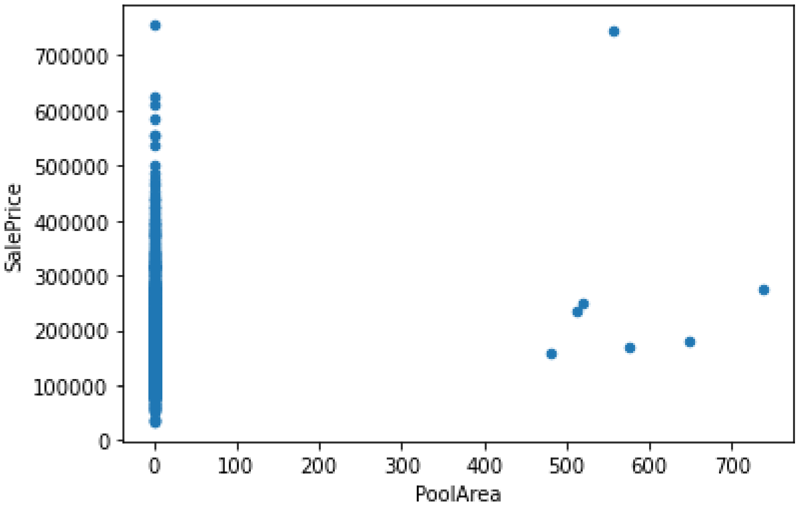

SalePrice
WoodDeckSF
0
100000
200000
300000
400000
500000
600000
700000
0 200 400 600 800

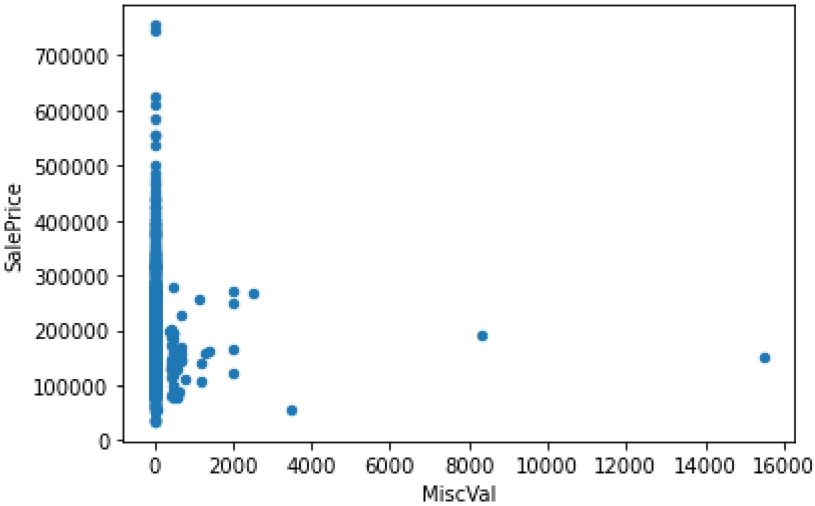

SalePrice
OpenPorchSF
0
100000
200000
300000
400000
500000
600000
700000
0 100 200 300 400 500

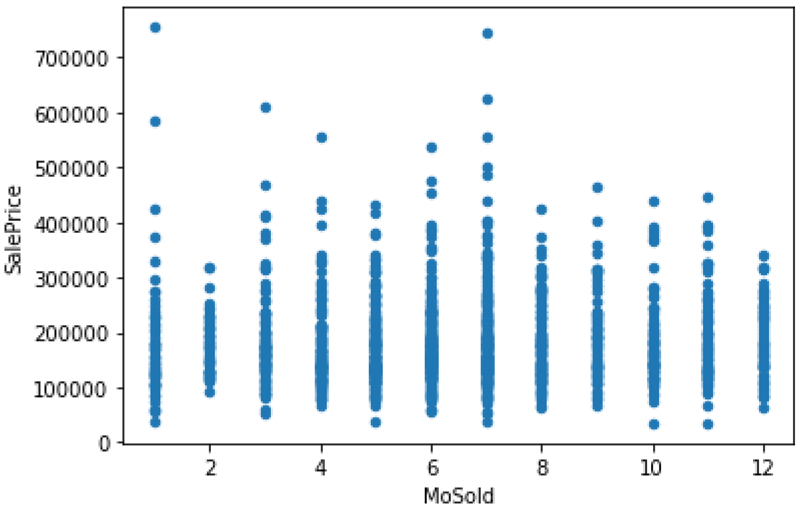

SalePrice
EnclosedPorch

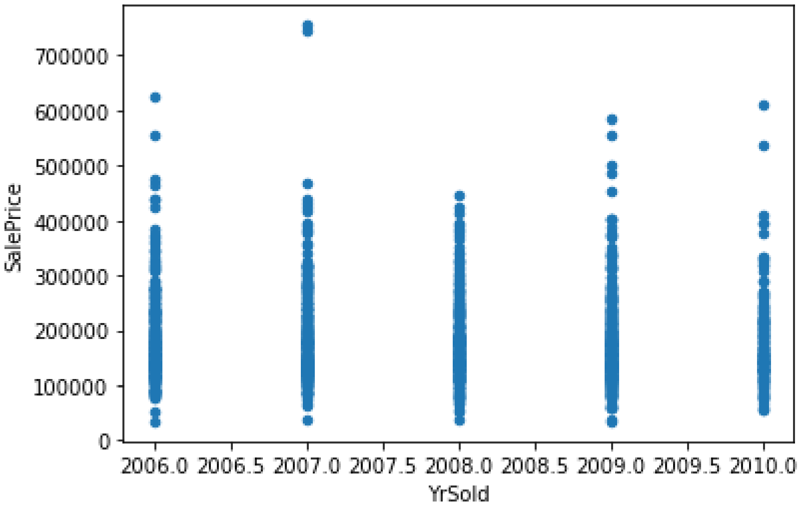

SalePrice
3SsnPorch


SalePrice
ScreenPorch

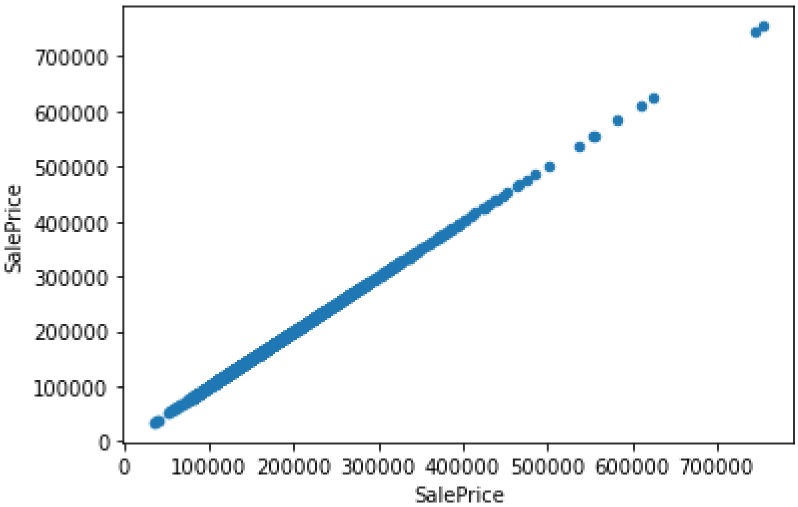

SalePrice
PoolArea
0
100000
200000
300000
400000
500000
600000
700000
0
100
200
300
400
500
600
700


SalePrice
MiscVal
0
100000
200000
300000
400000
500000
600000
700000
0
2000
4000
6000
8000
10000
12000
14000
16000

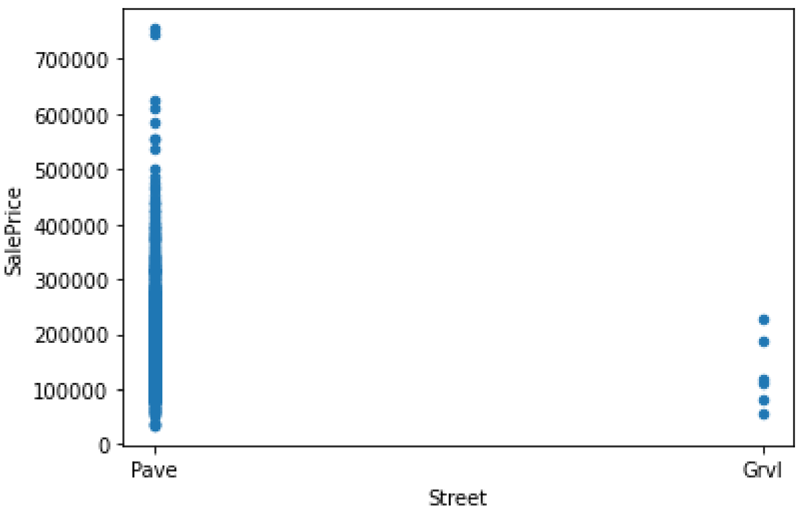

SalePrice
MoSold
0
100000
200000
300000
400000
500000
600000
700000
2
4
6
8
10
12



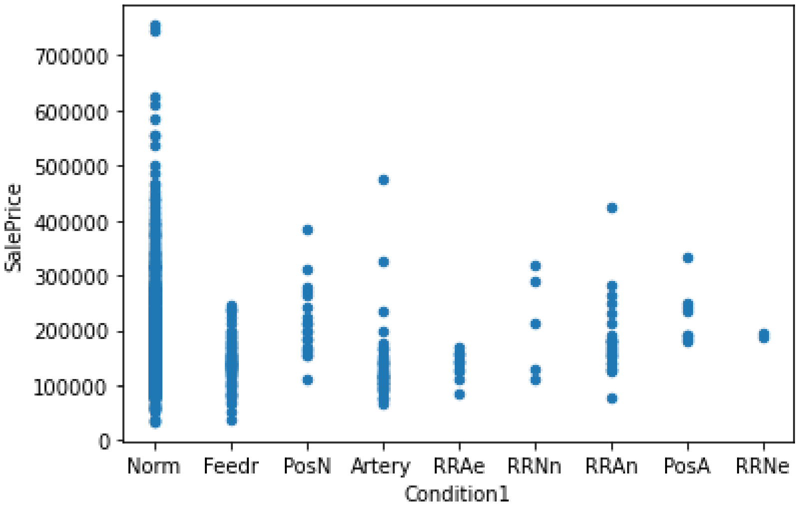
SalePrice
0
100000
200000
300000
400000
500000
600000
700000
2006.0 2006.5 2007.0 2007.5 2008.0 2008.5 2009.0 2009.5 2010.0
YrSold

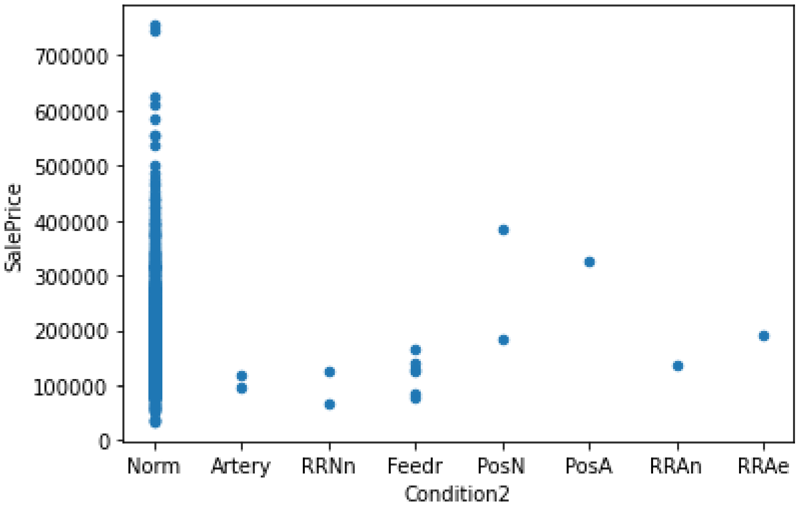


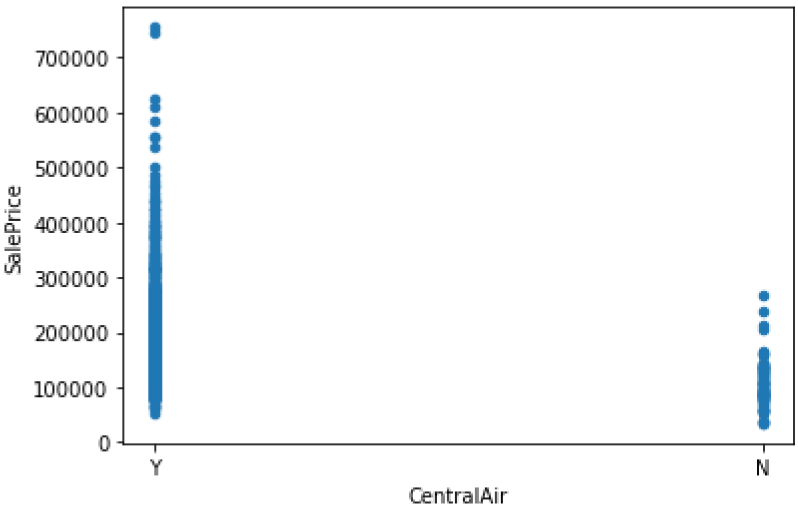
SalePrice
0
100000
200000
300000
400000
500000
600000
700000
0 100000 200000 300000 400000 500000 600000 700000
SalePrice

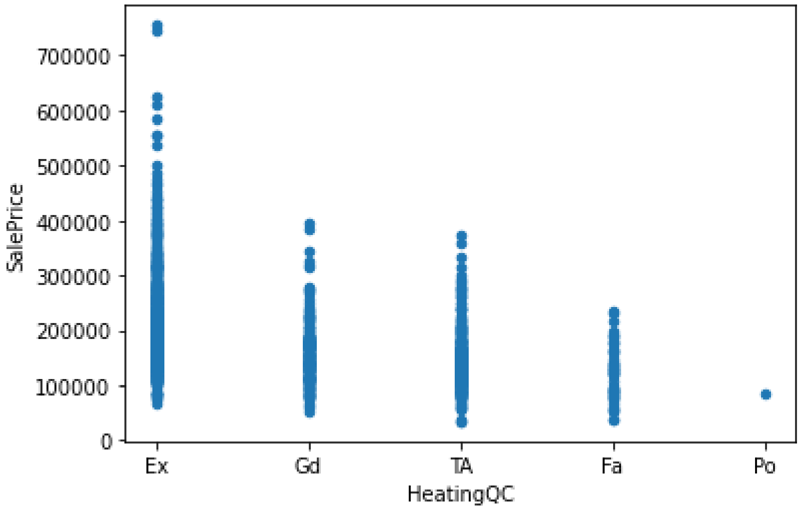


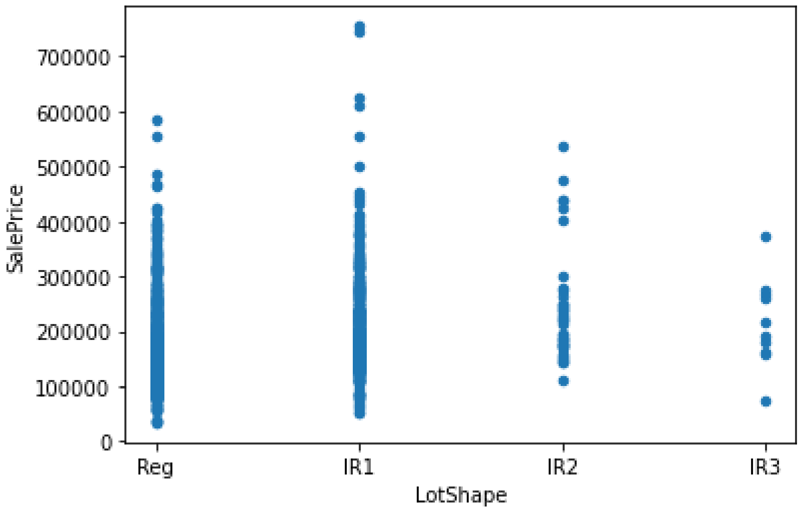
SalePrice
0
100000
200000
300000
400000
500000
600000
700000
Pave
Grvl
Street


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


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


SalePrice
0
100000
200000
300000
400000
500000
600000
700000
Y
N
CentralAir





Write a function for doing one hot encoding for all categorical features

Hint: Use pandas.get_dummies

In [5]:

```
CategorialFeatures = ['MSSubClass','OverallQual','OverallCond','Condition1','Condition
2','Street','CentralAir','HeatingQC','LotShape','LandContour']

def onehot_encoding(df,columns):
    '''
    Takes the dataframe
    columns which are corresponding to categorical features
    '''
    for col in columns:
        One_Hot_Encoded_Data = pd.get_dummies(df[col], prefix=col)
        df = df.drop(columns=[col])
        df = pd.concat([df,One_Hot_Encoded_Data],axis=1)
    return df


data = onehot_encoding(data,CategorialFeatures)
data.head()
```

Out[5]:

	Id	LotArea	YearBuilt	YearRemodAdd	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtS
0	1	8450	2003	2003	706	0	150	85
1	2	9600	1976	1976	978	0	284	126
2	3	11250	2001	2002	486	0	434	92
3	4	9550	1915	1970	216	0	540	75
4	5	14260	2000	2000	655	0	490	114

5 rows × 100 columns

Seperate the Label from the data, here it is 'SalePrice'

In [6]:

```
# Write your code here
SalePrice = data['SalePrice']
data = data.drop(columns=['Id','SalePrice']) #Id not needed for regression, not a featu
re
```

In [7]:

```
# Visualize the changed dataframe
data.head()
```

Out[7]:

	LotArea	YearBuilt	YearRemodAdd	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF
0	8450	2003	2003	706	0	150	856
1	9600	1976	1976	978	0	284	1262
2	11250	2001	2002	486	0	434	920
3	9550	1915	1970	216	0	540	756
4	14260	2000	2000	655	0	490	1145

5 rows × 98 columns

Split train test split with random state 42, test size 0.2

You can use sklearn module for this exercise

In [8]:

```
# Split data here
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data, SalePrice, test_size=0.2, ran
dom_state=42)
```

In [9]:

```
# import your Libraries
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
```

Lasso Regression

1. search for alphas in range of 0.1 to 1000 for Lasso rigression, choose the best which minimizes the mse
2. Plot the alphas vs MSE

Hint:

- cross validation score gives accuracy not the error convert to error appropriately (otherwise choose lambda which maximizes the score)

In the following cell, use training which was split earlier to cross validate (using cross_val_score) use cv = 5 (5 folds), then calculate the mean of the cross validation score for each alphas and plot λ vs cross_valiadtion_score or cross_validation_error. If you choose the accuracy then choose the λ which maximizes the cross_val_score.

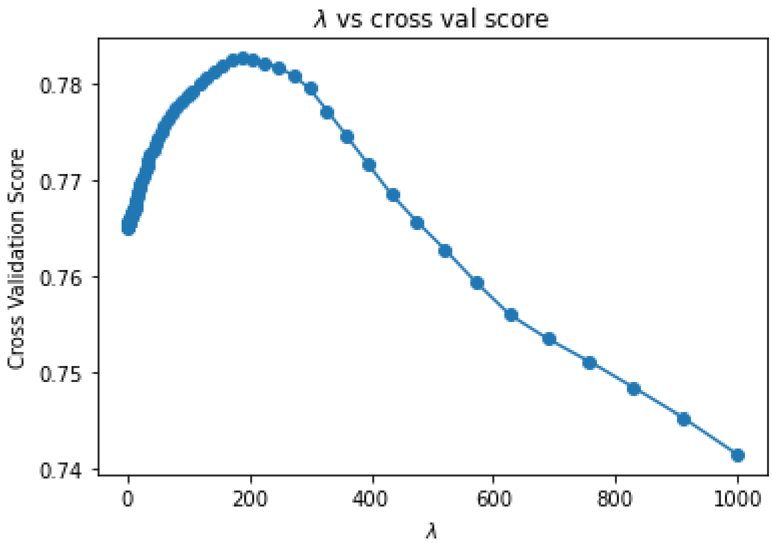
For range of alphas use alphas = np.logspace(-1, 3, 100)

Finally find the λ which maximizes the score (or minimizes the error) (appropriate value of λ is enough just by seeing the graph)

In [10]:

```
# write your code here and plot lambda vs cross validation score
alphas = np.logspace(-1,3,100)

score_history=[]

for alpha in alphas:
    model = Lasso(alpha=alpha)
    score = cross_val_score(model,X_train,y_train,cv=5)
    score_history.append(np.mean(score))


#plotting
plt.plot(alphas,score_history, linestyle='-', marker='o')
plt.xlabel(r'$\lambda$')
plt.ylabel('Cross Validation Score')
plt.title(r'$\lambda$'+' vs cross val score')

#Finding best lambda
Lasso_Best_Lambda = alphas[score_history.index(max(score_history))]
print('Lambda which maximises cross validation accuracy is: %.2f, Best Cross_val_score
 = %.2f' %(Lasso_Best_Lambda,max(score_history)))
```

```
Lambda which maximises cross validation accuracy is: 187.38, Best Cross_va
l_score = 0.78
```



From the graph also we can see that the best value of lambda lies around 200.

Ridge Regression

1. search for alphas in range of 0.1 to 100 for Ridge rigression, choose the best which minimizes the mse
2. Plot the alphas vs MSE

This similar to as explained for Lasso regression. Again plot λ vs accuracy (or error)

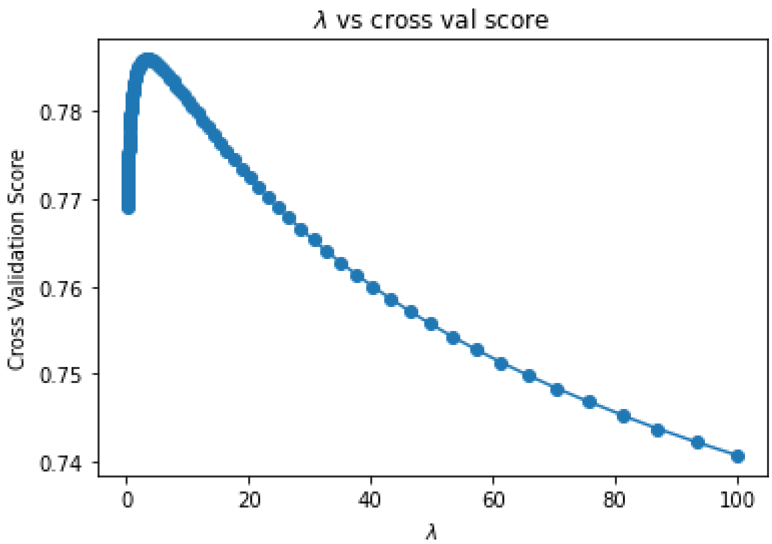
For range of alphas use alphas = np.logspace(-1, 2, 100)

Finally find the λ which maximizes the score (or minimizes the error) (appropriate value of λ is enough just by seeing the graph)

In [11]:

```
# write your code here and plot lambda vs cross validation score
alphas = np.logspace(-1,2,100)

score_history=[]

for alpha in alphas:
    model = Ridge(alpha=alpha)
    score = cross_val_score(model,X_train,y_train,cv=5)
    score_history.append(np.mean(score))


#plotting
plt.plot(alphas,score_history, linestyle='-', marker='o')
plt.xlabel(r'$\lambda$')
plt.ylabel('Cross Validation Score')
plt.title(r'$\lambda$'+' vs cross val score')

#Finding best lambda
Ridge_Best_Lambda = alphas[score_history.index(max(score_history))]
print('Lambda which maximises cross validation accuracy for ridge regression is: %.2f,
 Best Cross_val_score = %.2f' %(Ridge_Best_Lambda,max(score_history)))
```

```
Lambda which maximises cross validation accuracy for ridge regression is:
3.27, Best Cross_val_score = 0.79
```



From the graph also we can see that the best value of lambda lies around 5.

Now compare regularized models to linear Regression model

In the following cell, calculate cross validation score using Linear Regression model

In [12]:

```
# write your code here print cross validation score
LR_model = LinearRegression()
LR_score = np.mean(cross_val_score(LR_model,X_train,y_train,cv=5))
print('cross validation score using simple Linear Regression model = %.2f' %LR_score)
```

```
cross validation score using simple Linear Regression model = 0.76
```

Now you have λ values for both ridge and lasso regression, predict the model on the test data you created earlier

In the following cell use selected λ as the model parameter, predict on test data, compare among three models and report your findings.

Finally use lasso regression to find the important features and write your observations and also what do you observe when you compare both coefficients of Ridge and Lasso ? Do you see any property of Lasso which is used?

Hint:

- Check weights corresponding to each features

Note:

- Don't worry if you have huge error in prediction, it is possible, just compare among models and report which has lease error.

In [13]:

```
# predict on test data which you  splitted earlier, print coefficients of the learned m
odel, Mean square error. Report the model which gives the least MSE. Also commenton imp
ortant features

#Lasso, lambda/alpha = 187
Lasso_Model = Lasso(alpha=187)
Lasso_Model.fit(X_train,y_train)
y_test_pred = Lasso_Model.predict(X_test)
Lasso_MSE = mean_squared_error(y_test,y_test_pred)
print('Lasso MSE with lambda = 187: %.2f' %Lasso_MSE)

#Ridge, lambda/alpha = 3
Ridge_Model = Ridge(alpha=3)
Ridge_Model.fit(X_train,y_train)
y_test_pred = Ridge_Model.predict(X_test)
Ridge_MSE = mean_squared_error(y_test,y_test_pred)
print('Ridge MSE with lambda = 3: %.2f' %Ridge_MSE)

#Linear Regression Model
LR_model.fit(X_train,y_train)
y_test_pred = LR_model.predict(X_test)
LR_MSE = mean_squared_error(y_test,y_test_pred)
print('Simple Linear Regression MSE: %.2f' %LR_MSE)
```

```
Lasso MSE with lambda = 187: 1112733985.94
Ridge MSE with lambda = 3: 1111897197.85
Simple Linear Regression MSE: 1027400859.87
```

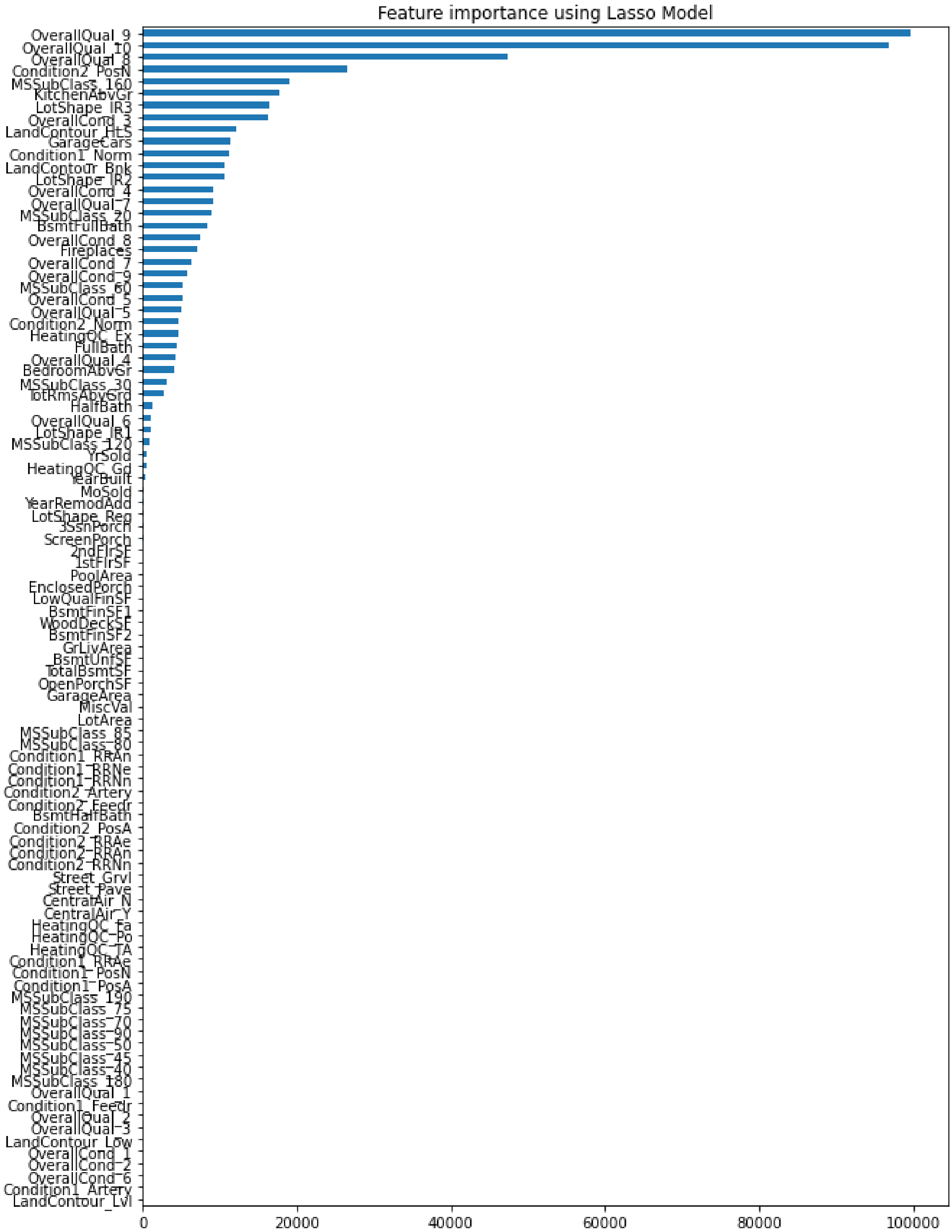
On test data, simple linear regression performed best, followed by Lasso regression.

In [16]:

```
lasso_abs_coef = pd.Series(abs(Lasso_Model.coef_), index = X_train.columns)

imp_coef = lasso_abs_coef.sort_values()
plt.rcParams['figure.figsize'] = (10.0, 15.0)
imp_coef.plot(kind = "barh")
plt.title("Feature importance using Lasso Model")
imp_coef
```

Out[16]:

```
LandContour_Lvl          0.000000
Condition1_Artery        0.000000
OverallCond_6            0.000000
OverallCond_2            0.000000
OverallCond_1            0.000000
                           ...
MSSubClass_160       19101.531975
Condition2_PosN      26592.799232
OverallQual_8        47343.023643
OverallQual_10       96747.014667
OverallQual_9        99559.009747
Length: 98, dtype: float64
```



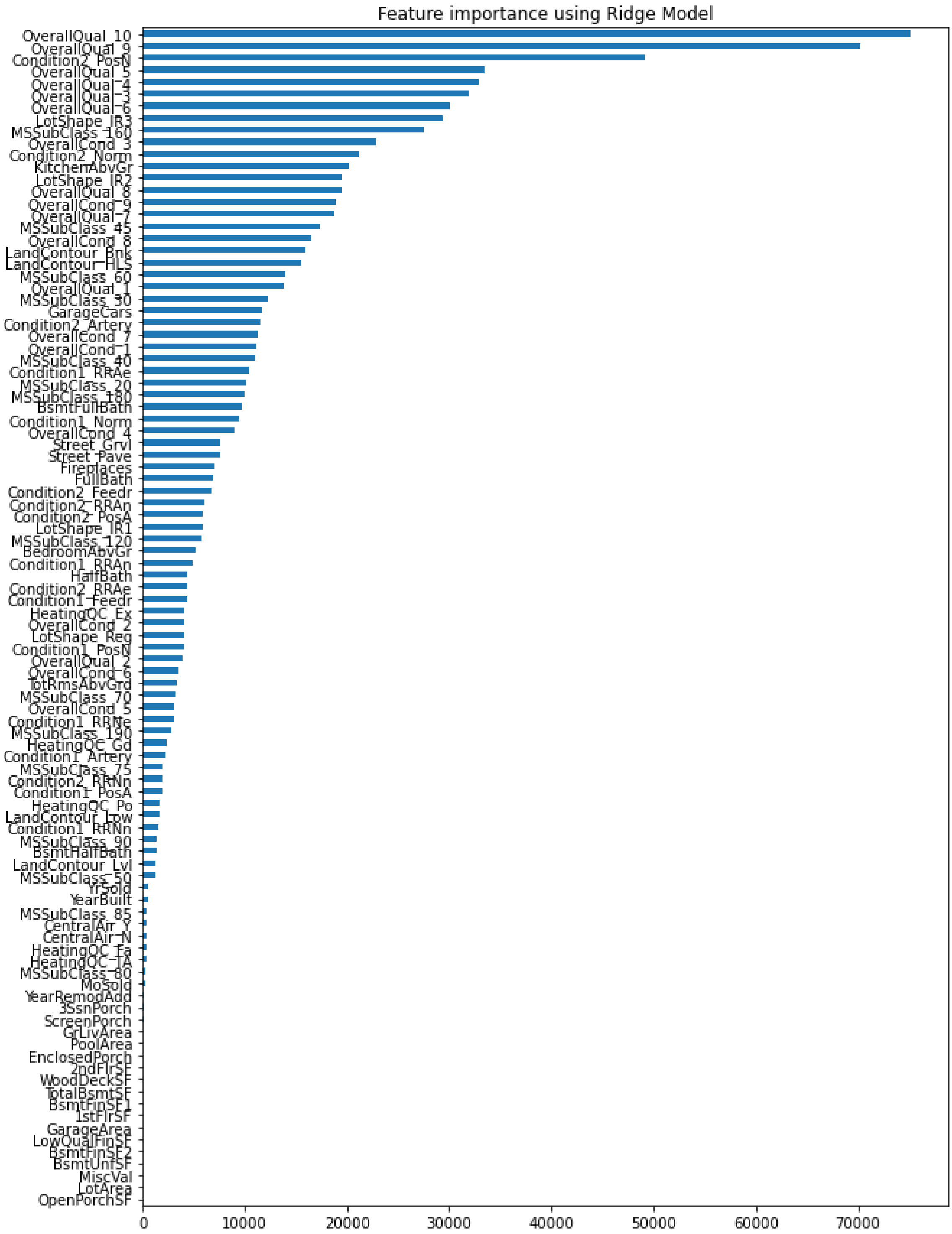
Variables like Overall quality, PosN (Condition2), MSSubclass_160, LotShape, and LandContour are given the most importance in lasso regression.

In [17]:

```
ridge_abs_coef = pd.Series(abs(Ridge_Model.coef_), index = X_train.columns)

imp_coef = ridge_abs_coef.sort_values()
plt.rcParams['figure.figsize'] = (10.0, 15.0)
imp_coef.plot(kind = "barh")
plt.title("Feature importance using Ridge Model")
imp_coef
```

Out[17]:

```
OpenPorchSF           0.365690
LotArea               0.405461
MiscVal               0.581808
BsmtUnfSF             0.751771
BsmtFinSF2            2.068937
                        ...
OverallQual_4     32970.179478
OverallQual_5     33480.689406
Condition2_PosN   49129.691629
OverallQual_9     70271.022412
OverallQual_10    75103.233468
Length: 98, dtype: float64
```



Variables like Overall quality, Condition2, MSSubclass, and LotShape are given the most importance in ridge regression.

What do you observe when you compare both coefficients of Ridge and Lasso ? Do you see any property of Lasso which is used?

Yes, lasso tends to make coefficients to absolute zero as compared to Ridge which never sets the value of coefficient to absolute zero. Therefore, we see a lot of variables have coefficients very close or equal to zero in lasso regression, whereas in ridge regression that is not the case.