In [122]:
```
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import Ridge

from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error
from sklearn.metrics import mean_squared_error

from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score
```

In [123]:
```
# Setting this option will print all collumns of a dataframe
pd.set_option('display.max_columns', None)
```

In [124]:
```
# Setting this option will print all of the data in a feature
pd.set_option('display.max_colwidth', None)
```

In [125]:
```
# Setting this option will print all rows di je none, jer mi se nije sve ispisivalo
#pd.set_option('max_rows', 100)
# za nove pande
pd.options.display.max_rows = 100
```

In [126]:
```
trainhouse = pd.read_csv('trainhouse.csv')
```

In [127]:
```
testhouse = pd.read_csv('testhouse.csv')
#50% uzeli za test
```

In [128]:
```
trainhouse.head(15)
```

Out[128]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Con
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	Veenker	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	
5	6	50	RL	85.0	14115	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	Mitchel	
6	7	20	RL	75.0	10084	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Somerst	
7	8	60	RL	NaN	10382	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	NWAmes	
8	9	50	RM	51.0	6120	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	OldTown	
9	10	190	RL	50.0	7420	Pave	NaN	Reg	Lvl	AllPub	Corner	Gtl	BrkSide	
10	11	20	RL	70.0	11200	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Sawyer	
11	12	60	RL	85.0	11924	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	NridgHt	
12	13	20	RL	NaN	12968	Pave	NaN	IR2	Lvl	AllPub	Inside	Gtl	Sawyer	
13	14	20	RL	91.0	10652	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	
14	15	20	RL	NaN	10920	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	NAmes	

In [129]:
```
trainhouse.isnull().sum()
```

Out[129]:
```
Id                 0
MSSubClass         0
MSZoning           0
LotFrontage      259
LotArea            0
Street             0
Alley           1369
LotShape           0
LandContour        0
Utilities          0
LotConfig          0
LandSlope          0
Neighborhood       0
Condition1         0
Condition2         0
BldgType           0
HouseStyle         0
OverallQual        0
OverallCond        0
```

In [130]:
```
# LotFrontage       259
# Alley            1369
# BsmtQual           37
# BsmtCond           37
# BsmtExposure       38
# BsmtFinType1       37
# BsmtFinType2       38
# Electrical          1
# FireplaceQu       690
# GarageType         81
# GarageYrBlt        81
# GarageFinish       81
# GarageQual         81
# GarageCond         81
# PoolQC           1453
# Fence            1179
# MiscFeature      1406
# MasVnrType          8
# MasVnrArea          8

#sve su tipovi za bins, mogu se grupirat preko encodea u svakoj koloni ili brojevi
```

In [131]:
```
#i will check on datatypes of columns
#i will use it do divide categorical from numeric variables and then tobe able to fill nan values with mode or median
#or some selected value i will see
trainhouse.dtypes
```

Out[131]:
```
Id                int64
MSSubClass        int64
MSZoning         object
LotFrontage     float64
LotArea           int64
Street           object
Alley            object
LotShape         object
LandContour      object
Utilities        object
LotConfig        object
LandSlope        object
Neighborhood     object
Condition1       object
Condition2       object
BldgType         object
HouseStyle       object
OverallQual       int64
OverallCond       int64
YearBuilt         int64
YearRemodAdd      int64
RoofStyle        object
RoofMatl         object
Exterior1st      object
Exterior2nd      object
MasVnrType       object
MasVnrArea      float64
ExterQual        object
ExterCond        object
Foundation       object
BsmtQual         object
BsmtCond         object
BsmtExposure     object
BsmtFinType1     object
BsmtFinSF1        int64
BsmtFinType2     object
BsmtFinSF2        int64
BsmtUnfSF         int64
TotalBsmtSF       int64
Heating          object
HeatingQC        object
```

```
CentralAir        object
Electrical        object
1stFlrSF           int64
2ndFlrSF           int64
LowQualFinSF       int64
GrLivArea          int64
BsmtFullBath       int64
BsmtHalfBath       int64
FullBath           int64
HalfBath           int64
BedroomAbvGr       int64
KitchenAbvGr       int64
KitchenQual       object
TotRmsAbvGrd       int64
Functional        object
Fireplaces         int64
FireplaceQu       object
GarageType        object
GarageYrBlt      float64
GarageFinish      object
GarageCars         int64
GarageArea         int64
GarageQual        object
GarageCond        object
PavedDrive        object
WoodDeckSF         int64
OpenPorchSF        int64
EnclosedPorch      int64
3SsnPorch          int64
ScreenPorch        int64
PoolArea           int64
PoolQC            object
Fence             object
MiscFeature       object
MiscVal            int64
MoSold             int64
YrSold             int64
SaleType          object
SaleCondition     object
SalePrice          int64
dtype: object
```

In [132]:
```
trainhouse[['LotFrontage','MasVnrArea','GarageYrBlt']] = trainhouse[['LotFrontage','MasVnrArea','GarageYrBlt']].fillna
```

In [133]:
```
trainhouse['Alley'] = trainhouse['Alley'].fillna(0)
#mode bas ove kolone
```

In [134]:
```
trainhouse['BsmtQual'] = trainhouse['BsmtQual'].fillna(0)
```

In [135]:
```
trainhouse['BsmtCond'] = trainhouse['BsmtCond'].fillna(0)
```

In [136]:
```
trainhouse['BsmtExposure'] = trainhouse['BsmtExposure'].fillna(0)
trainhouse['BsmtFinType1'] = trainhouse['BsmtFinType1'].fillna(0)
trainhouse['BsmtFinType2'] = trainhouse['BsmtFinType2'].fillna(0)
trainhouse['Electrical'] = trainhouse['Electrical'].fillna(0)
trainhouse['FireplaceQu'] = trainhouse['FireplaceQu'].fillna(0)
trainhouse['GarageType'] = trainhouse['GarageType'].fillna(0)
trainhouse['GarageYrBlt'] = trainhouse['GarageYrBlt'].fillna(0)
trainhouse['GarageFinish'] = trainhouse['GarageFinish'].fillna(0)
trainhouse['GarageQual'] = trainhouse['GarageQual'].fillna(0)
trainhouse['GarageCond'] = trainhouse['GarageCond'].fillna(0)
trainhouse['PoolQC'] = trainhouse['PoolQC'].fillna(0)
trainhouse['PoolQC'] = trainhouse['PoolQC'].fillna(0)
trainhouse['Fence'] = trainhouse['Fence'].fillna(0)
trainhouse['MiscFeature'] = trainhouse['MiscFeature'].fillna(0)
trainhouse['MasVnrType'] = trainhouse['MasVnrType'].fillna(0)
```

In [137]:
```
trainhouse.isnull().sum().sum()
```

Out[137]: 0

In [138]:
```
trainhouse.head(5)
```

Out[138]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Cond
0	1	60	RL	65.0	8450	Pave	0	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	
1	2	20	RL	80.0	9600	Pave	0	Reg	Lvl	AllPub	FR2	Gtl	Veenker	
2	3	60	RL	68.0	11250	Pave	0	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	
3	4	70	RL	60.0	9550	Pave	0	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	
4	5	60	RL	84.0	14260	Pave	0	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	

In [139]:
```
testhouse.head()
```

Out[139]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Co
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	NAmes	
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	NAmes	
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	Gilbert	
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	Gilbert	
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS	AllPub	Inside	Gtl	StoneBr	

In [140]:
```
testhouse.isnull().sum()
```

Out[140]:

```
Id                 0
MSSubClass         0
MSZoning           4
LotFrontage      227
LotArea            0
Street             0
Alley           1352
LotShape           0
LandContour        0
Utilities          2
LotConfig          0
LandSlope          0
Neighborhood       0
Condition1         0
Condition2         0
BldgType           0
HouseStyle         0
OverallQual        0
OverallCond        0
YearBuilt          0
YearRemodAdd       0
RoofStyle          0
RoofMatl           0
Exterior1st        1
Exterior2nd        1
MasVnrType        16
MasVnrArea        15
ExterQual          0
ExterCond          0
Foundation         0
BsmtQual          44
BsmtCond          45
BsmtExposure      44
BsmtFinType1      42
BsmtFinSF1         1
BsmtFinType2      42
BsmtFinSF2         1
BsmtUnfSF          1
TotalBsmtSF        1
Heating            0
HeatingQC          0
```

```
CentralAir          0
Electrical          0
1stFlrSF            0
2ndFlrSF            0
LowQualFinSF        0
GrLivArea           0
BsmtFullBath        2
BsmtHalfBath        2
FullBath            0
HalfBath            0
BedroomAbvGr        0
KitchenAbvGr        0
KitchenQual         1
TotRmsAbvGrd        0
Functional          2
Fireplaces          0
FireplaceQu       730
GarageType         76
GarageYrBlt        78
GarageFinish       78
GarageCars          1
GarageArea          1
GarageQual         78
GarageCond         78
PavedDrive          0
WoodDeckSF          0
OpenPorchSF         0
EnclosedPorch       0
3SsnPorch           0
ScreenPorch         0
PoolArea            0
PoolQC           1456
Fence            1169
MiscFeature      1408
MiscVal             0
MoSold              0
YrSold              0
SaleType            1
SaleCondition       0
dtype: int64
```

In [141]:
```
# MSZoning
# LotFrontage*
# Alley
# Utilities
# Exterior1st
# Exterior2nd
# MasVnrType
# MasVnrArea *
# BsmtQual
# BsmtCond
# BsmtExposure
# BsmtFinType1
# BsmtFinSF1 *
# BsmtFinType2
# BsmtFinSF2 *
# BsmtUnfSF *
# TotalBsmtSF *
# BsmtFullBath*
# BsmtHalfBath *
# KitchenQual
# Functional
# FireplaceQu
# GarageType
# GarageYrBlt *
# GarageFinish
# GarageCars  *
# GarageArea *
# GarageQual
# GarageCond
# PoolQC
# Fence
# MiscFeature
# SaleType
```

In [142]:
```
testhouse.dtypes
```

Out[142]:
```
Id                int64
MSSubClass        int64
MSZoning         object
LotFrontage     float64
LotArea           int64
Street           object
Alley            object
LotShape         object
LandContour      object
Utilities        object
LotConfig        object
LandSlope        object
Neighborhood     object
Condition1       object
Condition2       object
BldgType         object
HouseStyle       object
OverallQual       int64
OverallCond       int64
YearBuilt         int64
YearRemodAdd      int64
RoofStyle        object
RoofMatl         object
Exterior1st      object
Exterior2nd      object
MasVnrType       object
MasVnrArea      float64
ExterQual        object
ExterCond        object
Foundation       object
BsmtQual         object
BsmtCond         object
BsmtExposure     object
BsmtFinType1     object
BsmtFinSF1      float64
BsmtFinType2     object
BsmtFinSF2      float64
BsmtUnfSF       float64
TotalBsmtSF     float64
Heating          object
HeatingQC        object
```

```
CentralAir        object
Electrical        object
1stFlrSF           int64
2ndFlrSF           int64
LowQualFinSF       int64
GrLivArea          int64
BsmtFullBath     float64
BsmtHalfBath     float64
FullBath           int64
HalfBath           int64
BedroomAbvGr       int64
KitchenAbvGr       int64
KitchenQual       object
TotRmsAbvGrd       int64
Functional        object
Fireplaces         int64
FireplaceQu       object
GarageType        object
GarageYrBlt      float64
GarageFinish      object
GarageCars       float64
GarageArea       float64
GarageQual        object
GarageCond        object
PavedDrive        object
WoodDeckSF         int64
OpenPorchSF        int64
EnclosedPorch      int64
3SsnPorch          int64
ScreenPorch        int64
PoolArea           int64
PoolQC            object
Fence             object
MiscFeature       object
MiscVal            int64
MoSold             int64
YrSold             int64
SaleType          object
SaleCondition     object
dtype: object
```

In [143]:
```
for column in ['MSZoning','Alley','Utilities','Exterior1st','Exterior2nd','MasVnrType','BsmtQual','BsmtCond','BsmtExpo
    testhouse[column] = testhouse[column].fillna(0)
```

In [144]:
```
testhouse.isnull().sum()
```

```
Foundation         0
BsmtQual           0
BsmtCond           0
BsmtExposure       0
BsmtFinType1       0
BsmtFinSF1         1
BsmtFinType2       0
BsmtFinSF2         1
BsmtUnfSF          1
TotalBsmtSF        1
Heating            0
HeatingQC          0
CentralAir         0
Electrical         0
1stFlrSF           0
2ndFlrSF           0
LowQualFinSF       0
GrLivArea          0
BsmtFullBath       2
BsmtHalfBath       2
```

In [145]:
```
testhouse[['LotFrontage','MasVnrArea','BsmtFinSF1','BsmtFinSF2','BsmtUnfSF','TotalBsmtSF','BsmtFullBath','BsmtHalfBath
```

In [146]:
```
testhouse.isnull().sum().sum()
```

Out[146]:
```
0
```

In [147]:
```
#encoding
```

In [148]:
```
trainhouse.head(5)
```

Out[148]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Cond
0	1	60	RL	65.0	8450	Pave	0	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	
1	2	20	RL	80.0	9600	Pave	0	Reg	Lvl	AllPub	FR2	Gtl	Veenker	
2	3	60	RL	68.0	11250	Pave	0	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	
3	4	70	RL	60.0	9550	Pave	0	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	
4	5	60	RL	84.0	14260	Pave	0	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	

In [ ]:

In [149]:
```
#another way
#i took categorical variables which had null, added those that needed encoding and even didnt have nulls before and al
#encoded month year and some numeric variables, so that they just belong to different category and dont bring differer
#the result (same as different numbers from 1 to 5 maybe should just mean different style and not bring
#bigger value if style is 4th and not 2nd)
# encodecolstrain= ['MSSubClass','MSZoning','Street','Alley','LotShape','LandContour', 'Utilities','LotConfig', 'LandS

# for col in encodecolstrain:
#     trainhouse[col]=le.fit_transform(trainhouse[col])
#     print(le.classes_)

# trainhouse.head(5)
```

In [150]:
```
trainhouse = pd.get_dummies(data=trainhouse, columns=['MSSubClass','MSZoning','Street','Alley','LotShape','LandContour
```

In [151]:
```
trainhouse.head(5)
```

Out[151]:

	Id	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	Low
0	1	65.0	8450	7	5	196.0	706	0	150	856	856	854	
1	2	80.0	9600	6	8	0.0	978	0	284	1262	1262	0	
2	3	68.0	11250	7	5	162.0	486	0	434	920	920	866	
3	4	60.0	9550	7	5	0.0	216	0	540	756	961	756	
4	5	84.0	14260	8	5	350.0	655	0	490	1145	1145	1053	

In [152]:
```
len(trainhouse.columns)
```

Out[152]: 603

In [153]:
```
#another way
#i took categorical variables which had null, added those that needed encoding and even didnt have nulls before and al
#encoded month year and some numeric variables, so that they just belong to different category and dont bring differer
#the result (same as different numbers from 1 to 5 maybe should just mean different style and not bring
#bigger value if style is 4th and not 2nd)
# encodecolstest= ['MSSubClass','MSZoning','Street','Alley','LotShape','LandContour', 'Utilities','LotConfig', 'LandSl

# for col in encodecolstest:
#     testhouse[col]=le.fit_transform(testhouse[col])
#     print(le.classes_)

# testhouse.head(5)
```

In [154]:
```
testhouse = pd.get_dummies(data=testhouse, columns=['MSSubClass','MSZoning','Street','Alley','LotShape','LandContour',
```

In [155]:
```
testhouse.head(5)
```

Out[155]:

	Id	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	L
0	1461	80.0	11622	5	6	0.0	468.0	144.0	270.0	882.0	896	0	
1	1462	81.0	14267	6	6	108.0	923.0	0.0	406.0	1329.0	1329	0	
2	1463	74.0	13830	5	5	0.0	791.0	0.0	137.0	928.0	928	701	
3	1464	78.0	9978	6	6	20.0	602.0	0.0	324.0	926.0	926	678	
4	1465	43.0	5005	8	5	0.0	263.0	0.0	1017.0	1280.0	1280	0	

In [156]:
```
len(testhouse.columns)
```

Out[156]: 585

In [157]:
```
#***beacause of some category that train data has and test doesnt there are less columns in test then in train data
#and some are intest and not in train
```

In [158]:
```
#Check for missing columns
train_cols = trainhouse.columns
test_cols = testhouse.columns

common_cols = train_cols.intersection(test_cols)
train_not_test = train_cols.difference(test_cols)
test_not_train = test_cols.difference(train_cols)

print("Train not test", train_not_test)
print("Test not train", test_not_train)
```

```
Train not test Index(['Condition2_RRAe', 'Condition2_RRAn', 'Condition2_RRNn', 'Electrical_0',
       'Electrical_Mix', 'Exterior1st_ImStucc', 'Exterior1st_Stone',
       'Exterior2nd_Other', 'GarageQual_Ex', 'GarageYrBlt_1906.0',
       'GarageYrBlt_1908.0', 'GarageYrBlt_1914.0', 'GarageYrBlt_1929.0',
       'GarageYrBlt_1931.0', 'GarageYrBlt_1933.0', 'Heating_Floor',
       'Heating_OthW', 'HouseStyle_2.5Fin', 'MiscFeature_TenC', 'PoolQC_Fa',
       'RoofMatl_ClyTile', 'RoofMatl_Membran', 'RoofMatl_Metal',
       'RoofMatl_Roll', 'SalePrice', 'Utilities_NoSeWa', 'YearBuilt_1872',
       'YearBuilt_1875', 'YearBuilt_1882', 'YearBuilt_1885', 'YearBuilt_1892',
       'YearBuilt_1893', 'YearBuilt_1898', 'YearBuilt_1904', 'YearBuilt_1906',
       'YearBuilt_1908', 'YearBuilt_1911', 'YearBuilt_1913'],
      dtype='object')
Test not train Index(['Exterior1st_0', 'Exterior2nd_0', 'Functional_0', 'GarageYrBlt_1895.0',
       'GarageYrBlt_1896.0', 'GarageYrBlt_1917.0', 'GarageYrBlt_1919.0',
       'GarageYrBlt_1943.0', 'GarageYrBlt_2207.0', 'KitchenQual_0',
       'MSSubClass_150', 'MSZoning_0', 'SaleType_0', 'Utilities_0',
       'YearBuilt_1879', 'YearBuilt_1895', 'YearBuilt_1896', 'YearBuilt_1901',
       'YearBuilt_1902', 'YearBuilt_1907'],
      dtype='object')
```

In [159]:
```
#i will add to test those who are in train and i can drop from test those who are not in train because
#i cant find parameter ot add also missing ones
```

In [160]:
```
testhouse[['Condition2_RRAe', 'Condition2_RRAn', 'Condition2_RRNn', 'Electrical_0',
       'Electrical_Mix', 'Exterior1st_ImStucc', 'Exterior1st_Stone',
       'Exterior2nd_Other', 'GarageQual_Ex', 'GarageYrBlt_1906.0',
       'GarageYrBlt_1908.0', 'GarageYrBlt_1914.0', 'GarageYrBlt_1929.0',
       'GarageYrBlt_1931.0', 'GarageYrBlt_1933.0', 'Heating_Floor',
       'Heating_OthW', 'HouseStyle_2.5Fin', 'MiscFeature_TenC', 'PoolQC_Fa',
       'RoofMatl_ClyTile', 'RoofMatl_Membran', 'RoofMatl_Metal',
       'RoofMatl_Roll', 'Utilities_NoSeWa', 'YearBuilt_1872',
       'YearBuilt_1875', 'YearBuilt_1882', 'YearBuilt_1885', 'YearBuilt_1892',
       'YearBuilt_1893', 'YearBuilt_1898', 'YearBuilt_1904', 'YearBuilt_1906',
       'YearBuilt_1908', 'YearBuilt_1911', 'YearBuilt_1913']] = 0

#with no Saleprice as target
```

In [161]:
```
trainhouse[['Exterior1st_0', 'Exterior2nd_0', 'Functional_0', 'GarageYrBlt_1895.0',
       'GarageYrBlt_1896.0', 'GarageYrBlt_1917.0', 'GarageYrBlt_1919.0',
       'GarageYrBlt_1943.0', 'GarageYrBlt_2207.0', 'KitchenQual_0',
       'MSSubClass_150', 'MSZoning_0', 'SaleType_0', 'Utilities_0',
       'YearBuilt_1879', 'YearBuilt_1895', 'YearBuilt_1896', 'YearBuilt_1901',
       'YearBuilt_1902', 'YearBuilt_1907']] = 0
```

In [162]:
```
len(trainhouse.columns)
trainhouse.shape
```

Out[162]:
```
(1460, 623)
```

In [163]:
```
len(testhouse.columns)
testhouse.shape
```

Out[163]:
```
(1459, 622)
```

In [164]:
```
#Check for missing columns
train_cols = trainhouse.columns
test_cols = testhouse.columns

common_cols = train_cols.intersection(test_cols)
train_not_test = train_cols.difference(test_cols)
test_not_train = test_cols.difference(train_cols)

print("Train not test", train_not_test)
print("Test not train", test_not_train)
```

```
Train not test Index(['SalePrice'], dtype='object')
Test not train Index([], dtype='object')
```

In [165]:
```
# # from sklearn.preprocessing import StandardScaler
# scaler = StandardScaler()
# trainhouse[['LotFrontage','LotArea', 'MasVnrArea','BsmtFinSF1','BsmtFinSF2','BsmtUnfSF','TotalBsmtSF','1stFlrSF','Gr
```

In [166]:
```
trainhouse.head()
```

Out[166]:

	Id	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	Low
0	1	65.0	8450	7	5	196.0	706	0	150	856	856	854	
1	2	80.0	9600	6	8	0.0	978	0	284	1262	1262	0	
2	3	68.0	11250	7	5	162.0	486	0	434	920	920	866	
3	4	60.0	9550	7	5	0.0	216	0	540	756	961	756	
4	5	84.0	14260	8	5	350.0	655	0	490	1145	1145	1053	

In [167]:
```
# def log (x):
#     return np.log1p(x)
#i created a function to normalize data with log, but with logplus1 because od 0 values in some cells, so log would gi
#an error, it's not possible because it is undefined.if base is biggerthen 1 it is minus infonity

# trainhouse1 = trainhouse[['LotFrontage','LotArea', 'MasVnrArea','BsmtFinSF1','BsmtFinSF2','BsmtUnfSF','TotalBsmtSF',
```

In [ ]:

In [168]:
```
trainhouse.head()
```

Out[168]:

	Id	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	Low
0	1	65.0	8450	7	5	196.0	706	0	150	856	856	854	
1	2	80.0	9600	6	8	0.0	978	0	284	1262	1262	0	
2	3	68.0	11250	7	5	162.0	486	0	434	920	920	866	
3	4	60.0	9550	7	5	0.0	216	0	540	756	961	756	
4	5	84.0	14260	8	5	350.0	655	0	490	1145	1145	1053	

In [169]:
```
# trainhouse[['LotFrontage','LotArea', 'MasVnrArea','BsmtFinSF1','BsmtFinSF2','BsmtUnfSF','TotalBsmtSF','1stFlrSF','Gr
```

In [170]:
```
trainhouse.head()
```

Out[170]:

	Id	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	Low
0	1	65.0	8450	7	5	196.0	706	0	150	856	856	854	
1	2	80.0	9600	6	8	0.0	978	0	284	1262	1262	0	
2	3	68.0	11250	7	5	162.0	486	0	434	920	920	866	
3	4	60.0	9550	7	5	0.0	216	0	540	756	961	756	
4	5	84.0	14260	8	5	350.0	655	0	490	1145	1145	1053	

In [171]:
```
#*************************saleprice i will arrange the target value but later for measuring accuracy or results i will
#***************************so opposite of log is exponent
```

In [172]:
```
len(trainhouse.columns)
```

Out[172]: 623

In [173]:
```
# # from sklearn.preprocessing import StandardScaler
# #scaler = StandardScaler()
# testhouse[['LotFrontage','LotArea', 'MasVnrArea','BsmtFinSF1','BsmtFinSF2','BsmtUnfSF','TotalBsmtSF','1stFlrSF','GrL
```

In [ ]:

In [174]:
```
# # def log (x):
# #     return np.log1p(x)

# testhouse1 = testhouse[['LotFrontage','LotArea', 'MasVnrArea','BsmtFinSF1','BsmtFinSF2','BsmtUnfSF','TotalBsmtSF','1
```

In [175]:
```
# testhouse1.head()
```

In [ ]:

In [176]:
```
testhouse.head()
```

Out[176]:

	Id	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	L
0	1461	80.0	11622	5	6	0.0	468.0	144.0	270.0	882.0	896	0	
1	1462	81.0	14267	6	6	108.0	923.0	0.0	406.0	1329.0	1329	0	
2	1463	74.0	13830	5	5	0.0	791.0	0.0	137.0	928.0	928	701	
3	1464	78.0	9978	6	6	20.0	602.0	0.0	324.0	926.0	926	678	
4	1465	43.0	5005	8	5	0.0	263.0	0.0	1017.0	1280.0	1280	0	

In [177]:
```
len(testhouse.columns)
```

Out[177]: 622

In [178]:
```
#iz train zadanog dataseta X će biti Xtrain koji ću podijeliti na X_train i X_val tj Y_train i Y_val, na cijelom
#datasetu ću primijeniti crossvalidation umejsto traintestsplita, prije toga izdvojiti Ykolonu
#iz Y zadanog dataseta test data odvojeni će ostati za testiranje kao X_test, nema Y kolone

# Xa=trainhouse.iloc[:,1:80]
# print(Xa)
#izbacila sam kolonu indeksa  i kolonu cijena koja je Y
```

In [179]:
```
X=trainhouse.drop(['Id', 'SalePrice'],axis=1)
print(X)
```

```
1456         85.0    13175            6            6        119.0         790
1457         66.0     9042            7            9          0.0         275
1458         68.0     9717            5            6          0.0          49
1459         75.0     9937            5            6          0.0         830

      BsmtFinSF2  BsmtUnfSF  TotalBsmtSF  1stFlrSF  2ndFlrSF  LowQualFinSF  \
0              0        150          856       856       854             0
1              0        284         1262      1262         0             0
2              0        434          920       920       866             0
3              0        540          756       961       756             0
4              0        490         1145      1145      1053             0
...          ...        ...          ...       ...       ...           ...
1455           0        953          953       953       694             0
1456         163        589         1542      2073         0             0
1457           0        877         1152      1188      1152             0
1458        1029          0         1078      1078         0             0
1459         290        136         1256      1256         0             0

      GrLivArea  BsmtFullBath  BsmtHalfBath  FullBath  HalfBath  BedroomAbvGr  \
0          1710             1             0         2         1             3
```

In [180]:
```
# Xa.head()
```

In [181]:
```
#trainhouse nepromijenjena pa iz nje mogu kopirat, u x i y
```

In [182]:
```
#len(X.columns)
```

In [183]:
```
# Ya=trainhouse.iloc[:,80]
# print(Ya)
```

In [184]:
```
# from sklearn.preprocessing import StandardScaler
# mms = MinMaxScaler()
# trainhouse[['SalePrice']] = mms.fit_transform(trainhouse[['SalePrice']])
```

In [185]:
```
Y=trainhouse['SalePrice'].copy()
print(Y)
#sve iz traindataseta i x i y
```

```
0       208500
1       181500
2       223500
3       140000
4       250000
         ...
1455    175000
1456    210000
1457    266500
1458    142125
1459    147500
Name: SalePrice, Length: 1460, dtype: int64
```

In [186]:
```
X_test1 = testhouse.iloc[:,1:]
#izbacila sam kolonu indeksa samo, ovo je test data koji ne diramo
```

In [187]:
```
X_test1.head()
```

Out[187]:

	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	LowQua
0	80.0	11622	5	6	0.0	468.0	144.0	270.0	882.0	896	0	
1	81.0	14267	6	6	108.0	923.0	0.0	406.0	1329.0	1329	0	
2	74.0	13830	5	5	0.0	791.0	0.0	137.0	928.0	928	701	
3	78.0	9978	6	6	20.0	602.0	0.0	324.0	926.0	926	678	
4	43.0	5005	8	5	0.0	263.0	0.0	1017.0	1280.0	1280	0	

In [188]:
```
len(X_test1.columns)
```

Out[188]:
```
621
```

In [189]:
```
#I will split X on train i val part using cross validation, not train test split here
```

In [190]:
```
#kf = KFold(n_splits=5)
#kf.get_n_splits(trainhouse)
#X i Y su mi dio train data koji zelim podilit na cross val dio, xtest nisam jos dirala,

#SVE OK, PODIJELIS KOLONE NA X I Y I ONDA MEDJU NJIJMA CROSS VALIDATION, TJ NA X-U A INDEKSI X-A VUKU Y;
#SAMO SAM JA OVDJE XI Y NAPRAVILA OD TRAIN DATASETA DA BUDU ZAPRAVO TRAIN I VALIDATION..ALI MOGU JA IZDVOJENI TEST DIO
#ZVATI VALIDATION, CESTO TAKO I IDE
#DA SAM UZELA CIJELI TRAINDATA NEKE TABLICE ZA TRAIN I TESTDATA ZA TEST, DA IMAM YTEST PODATKE, SVE BI DAKLE
#DIJELILA PREKO CROSS VALIDATIONA  PO INDEKSIMA TRAINA A TEST
# ne znam jesam prvo morala podilit cross val pa izdvojit y onda mozda, ne i u primjeru je ok tako samo si ti sve
#prije izdvojila iz train dataseta ali ista stvar
```

In [191]:
```
#print(kf)
k=5
kf = KFold(n_splits=5, random_state=None, shuffle=False)
#ovo je sam objekt
# for i, (train_index, val_index) in enumerate(kf.split(trainhouse)):
#         print(f"Fold {i}:")
#         print(f"  Train: index={train_index}")
#         print(f"  Val:  index={val_index}")

        #preko x-a dobijas indekse za x i y dio
```

In [192]:
```
# kf = KFold(n_splits=5)
# kf.get_n_splits(trainhouse)
# #X i Y su mi dio train data koji zelim podilit na cross val dio, xtest nisam jos dirala,
# #ne znam jesam prvo morala podilit cross val pa izdvojit y onda mozda
# #ne dole prikaz bilo bi isto isti indeksi
```

In [193]:
```
# print(kf)
# KFold(n_splits=5, random_state=None, shuffle=False)
# for i, (train_index, test_index) in enumerate(kf.split(trainhouse)):
#         print(f"Fold {i}:")
#         print(f"  Train: index={train_index}")
#         print(f"  Test:  index={test_index}")
```

In [194]:
```

#kf.split( )

k = 5
kf = KFold(n_splits=k, random_state=None)

#po indeksima dijelis xcili train i y kolonu
```

In [195]:
```
for train_index , test_index in kf.split(X):
    X_train , X_test = X.iloc[train_index,:],X.iloc[test_index,:]
    Y_train , Y_test = Y[train_index] , Y[test_index]
```

In [196]:
```
# for train_index, val_index in kf.split(X,Y)
#     break
```

In [197]:
```
# train_index
```

In [198]:
```
# test_index
```

In [199]:
```
X_train.shape
```

Out[199]:
```
(1168, 621)
```

In [200]:
```
Y_train.shape
```

Out[200]:
```
(1168,)
```

In [201]:
```
# for train_index, val_index in kf.split(trainhouse):
#     X_train, X_val = X.iloc[train_index,:], X.iloc[train_index,:]
#     Y_train, Y_val = Y[val_index], Y[val_index]

#     ili
# X_tr=trainhouse.loc[train_index]
# Y_tr=trainhouse.loc[train_index]

# X_vl=trainhouse.loc[val_index]
# Y_vl=trainhouse.loc[val_index]
```

In [202]:
```
X_test.shape
```

Out[202]:
```
(292, 621)
```

In [203]:
```
Y_test.shape
```

Out[203]:
```
(292,)
```

In [ ]:

In [204]:
```

#I'll try to lower a multicollinearity (correlation matrix in normalized data example attachment) with adding
# regularization which adds penalty on coefficients impact, by implementing Ridge regression
model = Ridge()

#model = LinearRegression()
```

In [205]:
```
model.fit(X_train, Y_train)
```

Out[205]:
```
Ridge()
```

In [206]:
```
ytrainpredict = model.predict(X_test)
```

In [207]:
```
print(ytrainpredict)
```

```
[192361.82171974 487028.17454775 111873.37091412 168889.45799963
 160518.05829812 264134.10703201 251584.77770304 322370.92580749
 100053.02619134 104216.61232585 138039.18011763  89808.44871427
 222812.23830778 312181.65997885 470236.00458908 149144.65927184
 201689.49944976 129032.14007378  27938.71764066 289239.68210929
 188726.02596164 189327.57635694 188471.12915194 176267.04746749
 148828.10010444 153953.15765606 154913.68742621 169483.3334427
 234196.36327972 142906.40494237 178089.60935586 135762.63695637
  93059.77413298 189479.6782171  134937.62124269 177513.510153
 135772.00556531 252630.67144318 128296.97720338 219048.61039298
 130974.96263979 320567.79001707 204505.80809437 212454.47670249
  64058.17236459 135413.39021226 127204.55461329 116218.85739833
 113193.89440316 251663.25763362  72792.67646823 119001.59872091
  74090.12092394 107313.25086646 175958.394211   173485.86829826
 204268.23866104 140346.54304926 237969.36710473 146678.86982376
 380375.10884053 125776.19195953 171741.3509911  152038.78448667
  90909.47968303 132821.95192504 117488.43906857 136649.45624828
 174770.69822756 199580.01734197 135172.39813907 229733.51406007
 214508.67708977 254053.0602334  189162.44192023 389723.68929104
```

In [ ]:

In [208]:
```
Y_test.to_numpy()
```

Out[208]:
```
array([235000, 625000, 171000, 163000, 171900, 200500, 239000, 285000,
       119500, 115000, 154900,  93000, 250000, 392500, 745000, 120000,
       186700, 104900,  95000, 262000, 195000, 189000, 168000, 174000,
       125000, 165000, 158000, 176000, 219210, 144000, 178000, 148000,
       116050, 197900, 117000, 213000, 153500, 271900, 107000, 200000,
       140000, 290000, 189000, 164000, 113000, 145000, 134500, 125000,
       112000, 229456,  80500,  91500, 115000, 134000, 143000, 137900,
       184000, 145000, 214000, 147000, 367294, 127000, 190000, 132500,
       101800, 142000, 130000, 138887, 175500, 195000, 142500, 265900,
       224900, 248328, 170000, 465000, 230000, 178000, 186500, 169900,
       129500, 119000, 244000, 171750, 130000, 294000, 165400, 127500,
       301500,  99900, 190000, 151000, 181000, 128900, 161500, 180500,
       181000, 183900, 122000, 378500, 381000, 144000, 260000, 185750,
       137000, 177000, 139000, 137000, 162000, 197900, 237000,  68400,
       227000, 180000, 150500, 139000, 169000, 132500, 143000, 190000,
       278000, 281000, 180500, 119500, 107500, 162900, 115000, 138500,
       155000, 140000, 160000, 154000, 225000, 177500, 290000, 232000,
       130000, 325000, 202500, 138000, 147000, 179200, 335000, 203000,
       302000, 333168, 119000, 206900, 295493, 208900, 275000, 111000,
       156500,  72500, 190000,  82500, 147000,  55000,  79000, 130500,
       256000, 176500, 227000, 132500, 100000, 125500, 125000, 167900,
       135000,  52500, 200000, 128500, 123000, 155000, 228500, 177000,
       155835, 108500, 262500, 283463, 215000, 122000, 200000, 171000,
       134900, 410000, 235000, 170000, 110000, 149900, 177500, 315000,
       189000, 260000, 104900, 156932, 144152, 216000, 193000, 127000,
       144000, 232000, 105000, 165500, 274300, 466500, 250000, 239000,
        91000, 117000,  83000, 167500,  58500, 237500, 157000, 112000,
       105000, 125500, 250000, 136000, 377500, 131000, 235000, 124000,
       123000, 163000, 246578, 281213, 160000, 137500, 138000, 137450,
       120000, 193000, 193879, 282922, 105000, 275000, 133000, 112000,
       125500, 215000, 230000, 140000,  90000, 257000, 207000, 175900,
       122500, 340000, 124000, 223000, 179900, 127500, 136500, 274970,
       144000, 142000, 271000, 140000, 119000, 182900, 192140, 143750,
        64500, 186500, 160000, 174000, 120500, 394617, 149700, 197000,
       191000, 149300, 310000, 121000, 179600, 129000, 157900, 240000,
       112000,  92000, 136000, 287090, 145000,  84500, 185000, 175000,
       210000, 266500, 142125, 147500], dtype=int64)
```

In [209]:
```
#Y_val ranije gore definirala
```

In [210]:
```
# accuracy_score(Y_test, ytrainpredict)
```

accuracy_score(Y_test, ytrainpredict)

In [211]:
```
mean_squared_error(Y_test, ytrainpredict,squared=False)
```

Out[211]:
```
47398.78600468732
```

In [212]:
```
r_squared =  r2_score(Y_test, ytrainpredict)
#jer nije klasifikacijs ne more accuracy
#1 is the best but 0.7 is good! better then linear or knn!
```

In [213]:
```
print(r_squared)
```

```
0.6649483809269205
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: