Homework 3 - Ames Housing Dataset

For all parts below, answer all parts as shown in the Google document for Homework 3. Be sure to include both code that justifies your answer as well as text to answer the questions. We also ask that code be commented to make it easier to follow.

In [381]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
from scipy import stats
warnings.filterwarnings('ignore')
```

In [382]:
```
x_train = pd.read_csv('train.csv')
x_test = pd.read_csv('test.csv')
print("train : ", x_train.shape)
print("test : " , x_test.shape)
print(x_train.head(5))
```

```
train :  (1460, 81)
test :  (1459, 80)
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotSha
pe  \
0   1          60       RL         65.0     8450   Pave   NaN      R
eg
1   2          20       RL         80.0     9600   Pave   NaN      R
eg
2   3          60       RL         68.0    11250   Pave   NaN      I
R1
3   4          70       RL         60.0     9550   Pave   NaN      I
R1
4   5          60       RL         84.0    14260   Pave   NaN      I
R1

  LandContour Utilities  ... PoolArea PoolQC Fence MiscFeature MiscV
al MoSold  \
0         Lvl    AllPub  ...        0    NaN   NaN         NaN
0      2
1         Lvl    AllPub  ...        0    NaN   NaN         NaN
0      5
2         Lvl    AllPub  ...        0    NaN   NaN         NaN
0      9
3         Lvl    AllPub  ...        0    NaN   NaN         NaN
0      2
4         Lvl    AllPub  ...        0    NaN   NaN         NaN
0     12

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

[5 rows x 81 columns]
```

In [383]:

```python
train_ID = x_train['Id']
test_ID = x_test['Id']
# Drop Id column from train and test data sets
x_train.drop("Id", axis = 1, inplace = True)
x_test.drop("Id", axis = 1, inplace = True)
```

In [384]:

```
train_missing = pd.isna(x_train).sum()
test_missing = pd.isna(x_test).sum()

missing = pd.concat([train_missing, test_missing], axis=1, keys=["Trai
n_Missing", "Test_Missing"])
missing = missing[missing.sum(axis=1) > 0]
missing
```

Out[384]:

	Train_Missing	Test_Missing
Alley	1369	1352.0
BsmtCond	37	45.0
BsmtExposure	38	44.0
BsmtFinSF1	0	1.0
BsmtFinSF2	0	1.0
BsmtFinType1	37	42.0
BsmtFinType2	38	42.0
BsmtFullBath	0	2.0
BsmtHalfBath	0	2.0
BsmtQual	37	44.0
BsmtUnfSF	0	1.0
Electrical	1	0.0
Exterior1st	0	1.0
Exterior2nd	0	1.0
Fence	1179	1169.0
FireplaceQu	690	730.0
Functional	0	2.0
GarageArea	0	1.0
GarageCars	0	1.0
GarageCond	81	78.0
GarageFinish	81	78.0
GarageQual	81	78.0
GarageType	81	76.0
GarageYrBlt	81	78.0
KitchenQual	0	1.0

LotFrontage	259	227.0
MSZoning	0	4.0
MasVnrArea	8	15.0
MasVnrType	8	16.0
MiscFeature	1406	1408.0
PoolQC	1453	1456.0
SaleType	0	1.0
TotalBsmtSF	0	1.0
Utilities	0	2.0

In [385]:

```
#There're some meaningful missing values in data which can be understo
od from description of data.

meaningful_val_miss = ["Alley", "BsmtQual", "BsmtCond", "BsmtExposure"
, "BsmtFinType1",
                     "BsmtFinType2", "FireplaceQu", "GarageType", "Gara
geFinish", "GarageQual",
                     "GarageCond", "Fence","PoolQC", "MiscFeature","Mas
VnrType"]
for each_col in meaningful_val_miss:
    x_train[each_col].fillna("None", inplace=True)
    x_test[each_col].fillna("None", inplace=True)
```

In [386]:
```
# Some of the numerical features in data seems categorical.
#MSSubClass and MoSold (Month sold)
#Converting them into categorical
x_train = x_train.replace({"MSSubClass" : {20 : "SC20", 30 : "SC30", 40 : "SC40", 45 : "SC45",
                                        50 : "SC50", 60 : "SC60", 70 : "SC70", 75 : "SC75",
                                        80 : "SC80", 85 : "SC85", 90 : "SC90", 120 : "SC120",
                                        150 : "SC150", 160 : "SC160", 180 : "SC180", 190 : "SC190"},
                       "MoSold" : {1 : "Jan", 2 : "Feb", 3 : "Mar", 4 : "Apr", 5 : "May", 6 : "Jun",
                                   7 : "Jul", 8 : "Aug", 9 : "Sep", 10 : "Oct", 11 : "Nov", 12 : "Dec"}
                      })

x_test = x_test.replace({"MSSubClass" : {20 : "SC20", 30 : "SC30", 40 : "SC40", 45 : "SC45",
                                        50 : "SC50", 60 : "SC60", 70 : "SC70", 75 : "SC75",
                                        80 : "SC80", 85 : "SC85", 90 : "SC90", 120 : "SC120",
                                        150 : "SC150", 160 : "SC160", 180 : "SC180", 190 : "SC190"},
                       "MoSold" : {1 : "Jan", 2 : "Feb", 3 : "Mar", 4 : "Apr", 5 : "May", 6 : "Jun",
                                   7 : "Jul", 8 : "Aug", 9 : "Sep", 10 : "Oct", 11 : "Nov", 12 : "Dec"}
                      })
```

In [387]:
```
numeric_train = x_train.select_dtypes(include=[np.number])
numeric_test = x_test.select_dtypes(include=[np.number])
print(numeric_train.columns)
```

```
Index(['LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'Year
Built',
       'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'Bs
mtUnfSF',
       'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLiv
Area',
       'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'Bedr
oomAbvGr',
       'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageYrBlt',
       'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'Mis
cVal',
       'YrSold', 'SalePrice'],
      dtype='object')
```

In [388]:
```
# filling NA's of numerical variables with mean
x_train.fillna(numeric_train.mean(),inplace=True)
x_test.fillna(numeric_test.mean(),inplace=True)
```

In [389]:
```
numeric_train.fillna(numeric_train.mean(),inplace=True)
numeric_test.fillna(numeric_test.mean(),inplace=True)
```

In [390]:
```
#LotFrontage which is Numeric value has many missing values so we remo
ve it
x_train.drop("LotFrontage", axis=1, inplace=True)
x_test.drop("LotFrontage", axis=1, inplace=True)

numeric_train.drop("LotFrontage", axis=1, inplace=True)
numeric_test.drop("LotFrontage", axis=1, inplace=True)
```

In [391]:
```
# filling categorical variables with mode
x_train['Electrical'] = x_train['Electrical'].fillna(x_train['Electric
al'].mode()[0])
```

In [392]:
```
x_test['Exterior1st'] = x_test['Exterior1st'].fillna(x_test['Exterior1
st'].mode()[0])
x_test['Exterior2nd'] = x_test['Exterior2nd'].fillna(x_test['Exterior2
nd'].mode()[0])
x_test['Functional'] = x_test['Functional'].fillna(x_test['Functional'
].mode()[0])
x_test['KitchenQual'] = x_test['KitchenQual'].fillna(x_test['KitchenQu
al'].mode()[0])
x_test['MSZoning'] = x_test['MSZoning'].fillna(x_test['MSZoning'].mode
()[0])
x_test['SaleType'] = x_test['SaleType'].fillna(x_test['SaleType'].mode
()[0])
x_test['Utilities'] = x_test['Utilities'].fillna(x_test['Utilities'].m
ode()[0])
```

In [393]:
```
#Adding new features to train and test data
# Overall quality of the house
# Total number of bathrooms
numeric_train["TotalBath"] = numeric_train["BsmtFullBath"] + (0.5 * nu
meric_train["BsmtHalfBath"]) + \
numeric_train["FullBath"] + (0.5 * numeric_train["HalfBath"])

numeric_test["TotalBath"] = numeric_test["BsmtFullBath"] + (0.5 * nume
ric_test["BsmtHalfBath"]) + \
numeric_test["FullBath"] + (0.5 * numeric_test["HalfBath"])

# Total SF for house (incl. basement)
numeric_train["AllSF"] = numeric_train["GrLivArea"] + numeric_train["T
otalBsmtSF"]

numeric_test["AllSF"] = numeric_test["GrLivArea"] + numeric_test["Tota
lBsmtSF"]

# Total SF for 1st + 2nd floors
numeric_train["AllFlrsSF"] = numeric_train["1stFlrSF"] + numeric_train
["2ndFlrSF"]

numeric_test["AllFlrsSF"] = numeric_test["1stFlrSF"] + numeric_test["2
ndFlrSF"]

# Total SF for porch
numeric_train["AllPorchSF"] = numeric_train["OpenPorchSF"] + numeric_t
rain["EnclosedPorch"] + \
numeric_train["3SsnPorch"] + numeric_train["ScreenPorch"]

numeric_test["AllPorchSF"] = numeric_test["OpenPorchSF"] + numeric_tes
t["EnclosedPorch"] + \
numeric_test["3SsnPorch"] + numeric_test["ScreenPorch"]
```

In [394]:
```
y_train_label = np.log(numeric_train['SalePrice'])
numeric_train.drop(['SalePrice'], axis = 1, inplace=True)
```

In [395]:
```
categoric_train = x_train.select_dtypes(include=[np.object])
categoric_test = x_test.select_dtypes(include=[np.object])
print(categoric_train.columns)
print(categoric_test.columns)
```

```
Index(['MSSubClass', 'MSZoning', 'Street', 'Alley', 'LotShape', 'Lan
dContour',
       'Utilities', 'LotConfig', 'LandSlope', 'Neighborhood', 'Condi
tion1',
       'Condition2', 'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMat
l',
       'Exterior1st', 'Exterior2nd', 'MasVnrType', 'ExterQual', 'Ext
erCond',
       'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFi
nType1',
       'BsmtFinType2', 'Heating', 'HeatingQC', 'CentralAir', 'Electr
ical',
       'KitchenQual', 'Functional', 'FireplaceQu', 'GarageType',
       'GarageFinish', 'GarageQual', 'GarageCond', 'PavedDrive', 'Po
olQC',
       'Fence', 'MiscFeature', 'MoSold', 'SaleType', 'SaleCondition'
],
      dtype='object')
Index(['MSSubClass', 'MSZoning', 'Street', 'Alley', 'LotShape', 'Lan
dContour',
       'Utilities', 'LotConfig', 'LandSlope', 'Neighborhood', 'Condi
tion1',
       'Condition2', 'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMat
l',
       'Exterior1st', 'Exterior2nd', 'MasVnrType', 'ExterQual', 'Ext
erCond',
       'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFi
nType1',
       'BsmtFinType2', 'Heating', 'HeatingQC', 'CentralAir', 'Electr
ical',
       'KitchenQual', 'Functional', 'FireplaceQu', 'GarageType',
       'GarageFinish', 'GarageQual', 'GarageCond', 'PavedDrive', 'Po
olQC',
       'Fence', 'MiscFeature', 'MoSold', 'SaleType', 'SaleCondition'
],
      dtype='object')
```

In [396]:
```
categoric_train_test = pd.concat([categoric_train , categoric_test])
```

In [397]:
```
ntrain = categoric_train.shape[0]
ntrain
```

Out[397]: 1460

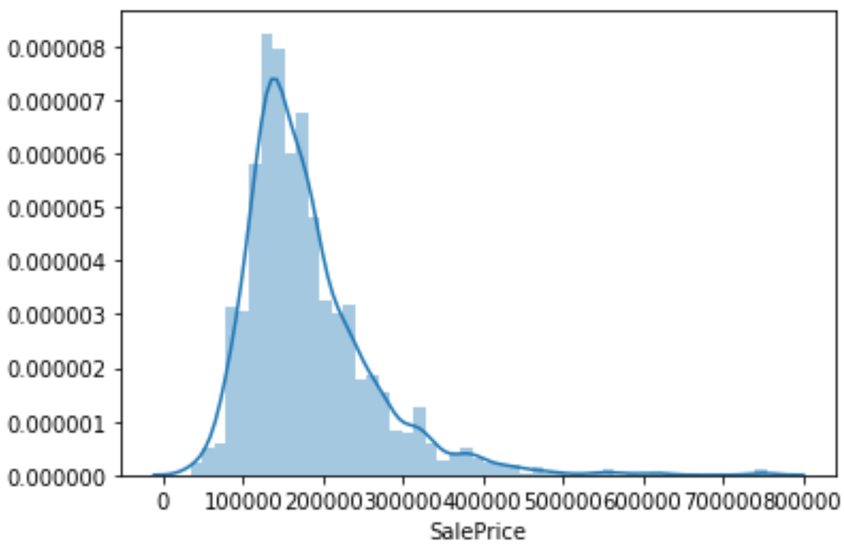
In [398]:
```
categoric_onehot = pd.get_dummies(categoric_train_test, columns=catego
ric_train_test.columns)
```

In [399]:
```
#Separating Categorical features of Train and test data sets after enc
oding
categoric_train_encod = categoric_onehot[:ntrain]
categoric_test_encod = categoric_onehot[ntrain:]
```

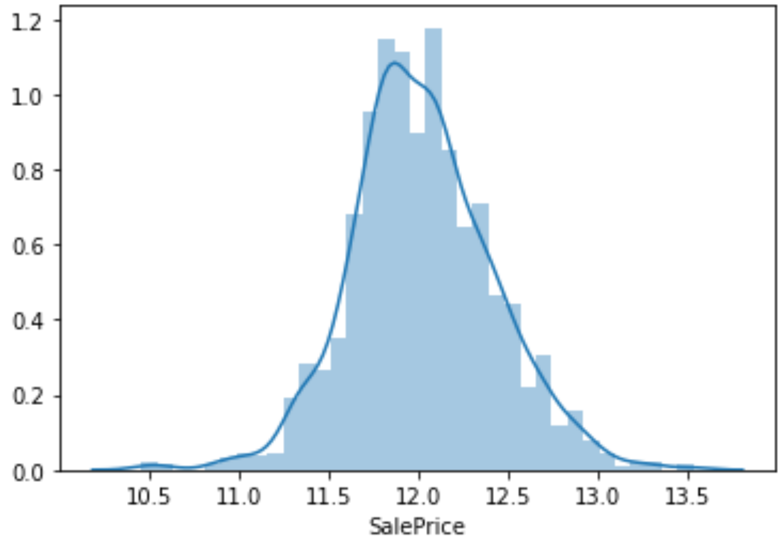
In [400]:
```
train_final = pd.concat([numeric_train, categoric_train_encod],axis=1)
test_final = pd.concat([numeric_test, categoric_test_encod],axis=1)
```

Part 1 - Pairwise Correlations

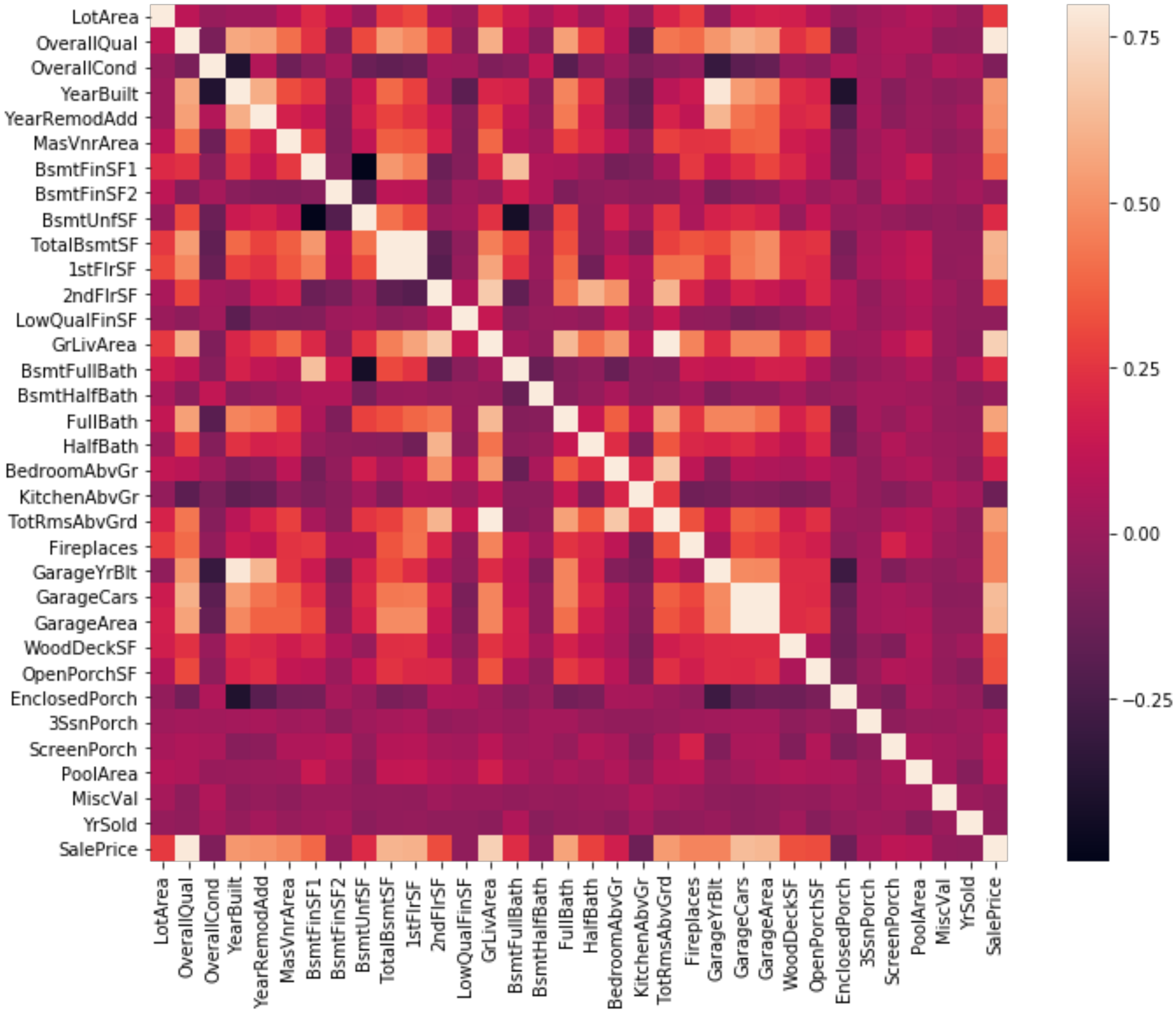
In [401]:
```
sns.distplot(x_train['SalePrice']);
```



In [402]:
```
sns.distplot(np.log(x_train['SalePrice']));
```



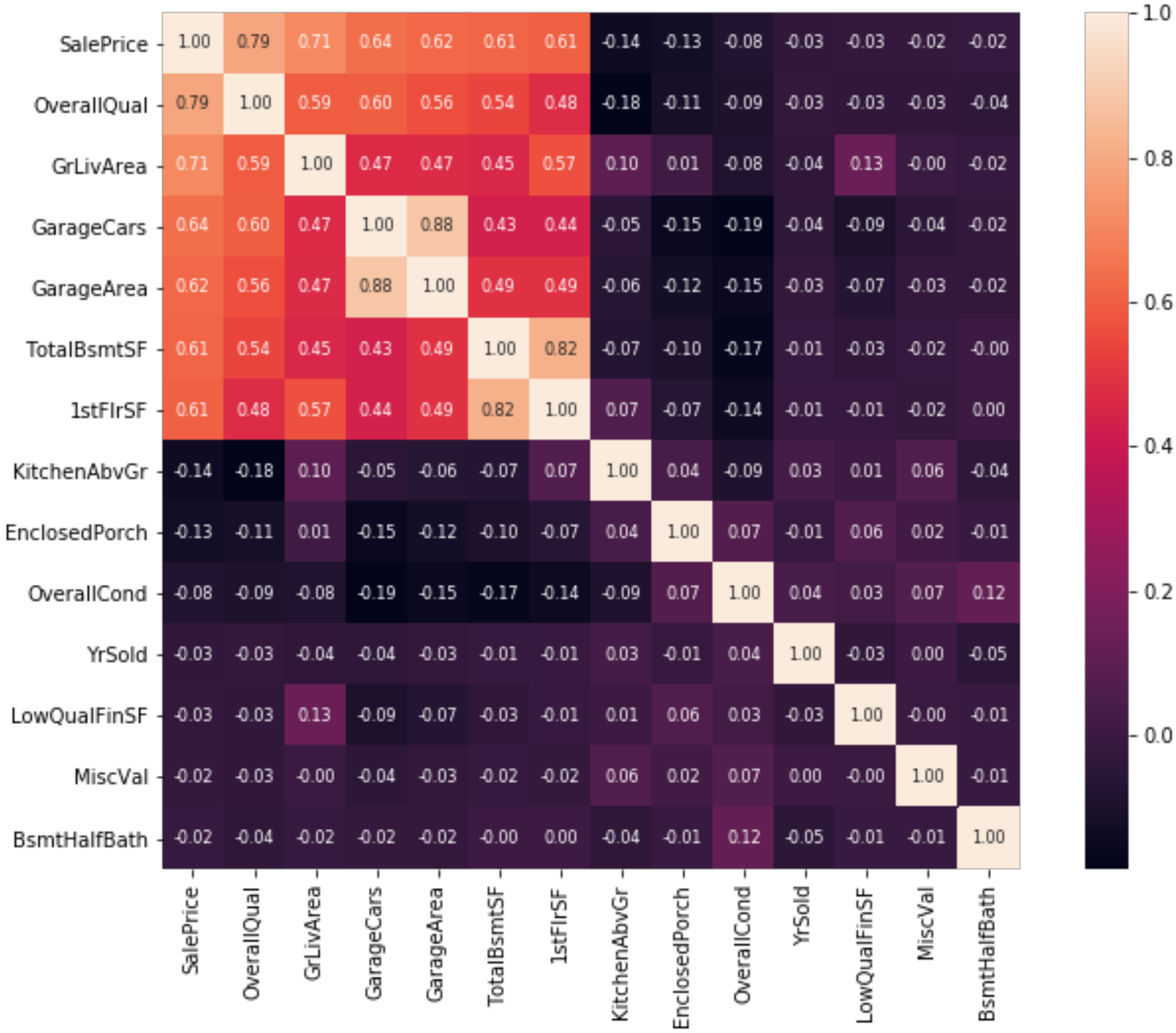
In [403]:
```
#correlation matrix
corrmat = x_train.corr(method = 'pearson')
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True);
```



In [404]:
```
cols = corrmat.nlargest(7, 'SalePrice')['SalePrice'].index
smallest_cols = corrmat.nsmallest(7, 'SalePrice')['SalePrice'].index
print(list(cols) + list(smallest_cols))
merged_cols = list(cols) + list(smallest_cols)
```

```
['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'GarageArea'
, 'TotalBsmtSF', '1stFlrSF', 'KitchenAbvGr', 'EnclosedPorch', 'Overa
llCond', 'YrSold', 'LowQualFinSF', 'MiscVal', 'BsmtHalfBath']
```

In [405]:
```
f,ax = plt.subplots(figsize = (12,8))
# np.corrcoef returns pearson corr coefficients
cm = np.corrcoef(x_train[merged_cols].values.T)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', an
not_kws={'size': 8}, yticklabels=merged_cols, xticklabels=merged_cols)
plt.show()
```



Discuss most positive and negative correlations.

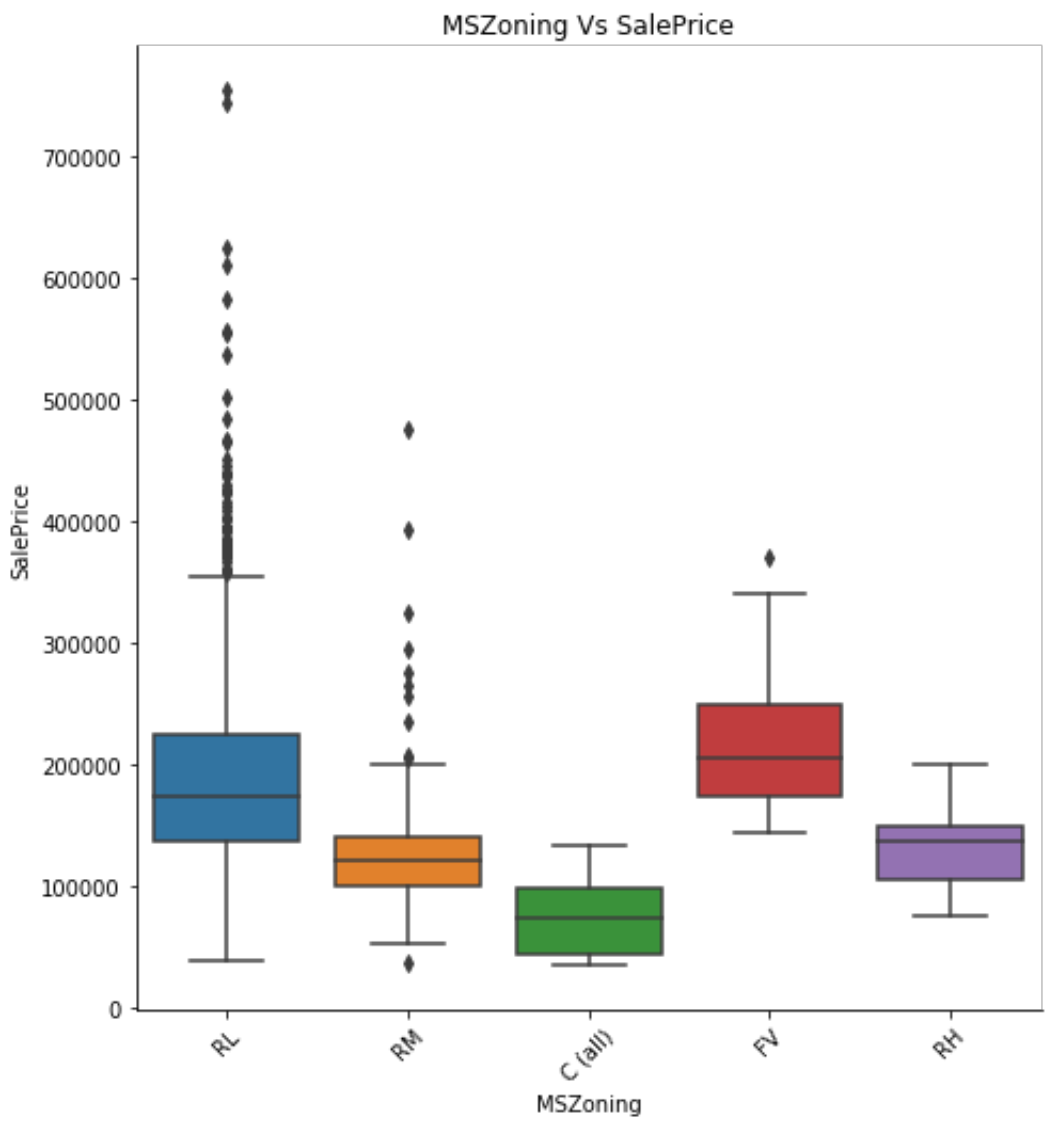
I chose 14 features having 7 highest and 7 lowest pearson co-relation coefficent with respect to SalePrice. From the above heatmap, we can get the features having highest positive co-relation. 0.88 for GarageCars and GarageArea. Least co-relation is for MSSubClass and 1stFlrSF with value -0.25.

Part 2 - Informative Plots

What interesting properties does Plot 1 reveal?

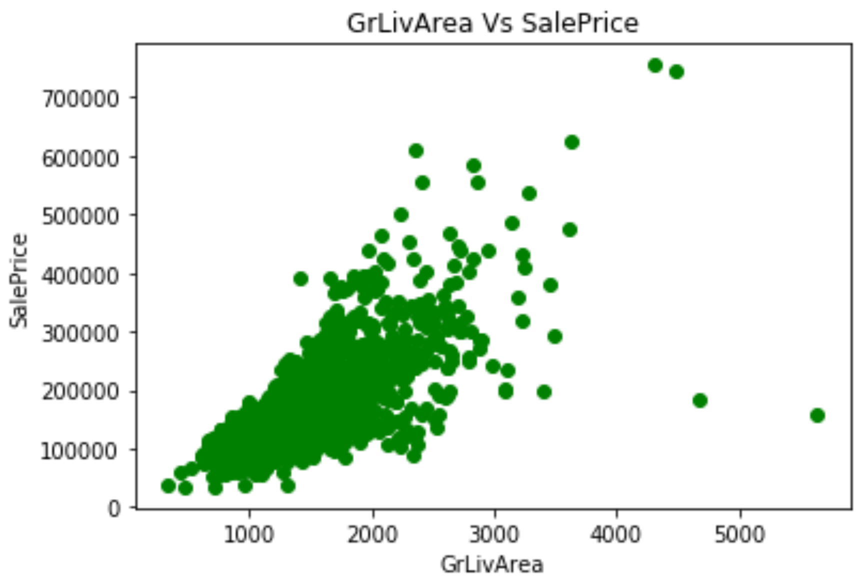
In [406]:
```
sns.catplot(x="MSZoning", y="SalePrice",
            kind="box", dodge=False, data=x_train, height=7)
plt.xticks(rotation=45);
plt.title('MSZoning Vs SalePrice')
```

Out[406]: Text(0.5, 1, 'MSZoning Vs SalePrice')



Generally, Commercial housing should be having highest saleprice as the houses might have good and luxurious facilities. But from the above plot, Commercial housing has less saleprice average which is interesting. Floating Villages as the name suggests might be near lakeside so the expectation is correct that it might have higher saleprice. One more interesting observation is that RL is having higher saleprice average than RH. It's interesting because generally low residential density areas might be having less saleprice as proper facilities and neighborhoods might not be available compared to high residential density areas.

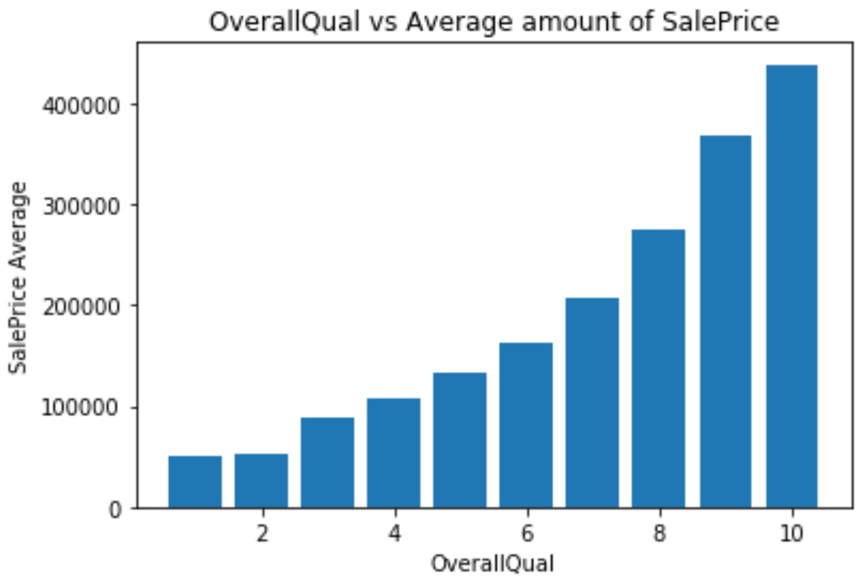
In [407]:
```
plt.scatter(x_train['GrLivArea'], x_train['SalePrice'], c='g')
plt.title("GrLivArea Vs SalePrice")
plt.ylabel("SalePrice")
plt.xlabel("GrLivArea")
plt.show()
```



There exists almost linear relationship between GrLivArea and SalePrice. One interesting observation is that there're 2 outliers whose GrLivArea is above 4000 and Saleprice < 200000. These two points seems strange. This might be some barren or agricultural land where there's no residential area nearby. So we can remove these 2 points.

In [408]:
```
overall_qual_mean = x_train['SalePrice'].groupby(x_train['OverallQual'
]).mean()
plt.bar(range(1,11),overall_qual_mean)
plt.xlabel('OverallQual')
plt.ylabel('SalePrice Average')
plt.title('OverallQual vs Average amount of SalePrice')
```

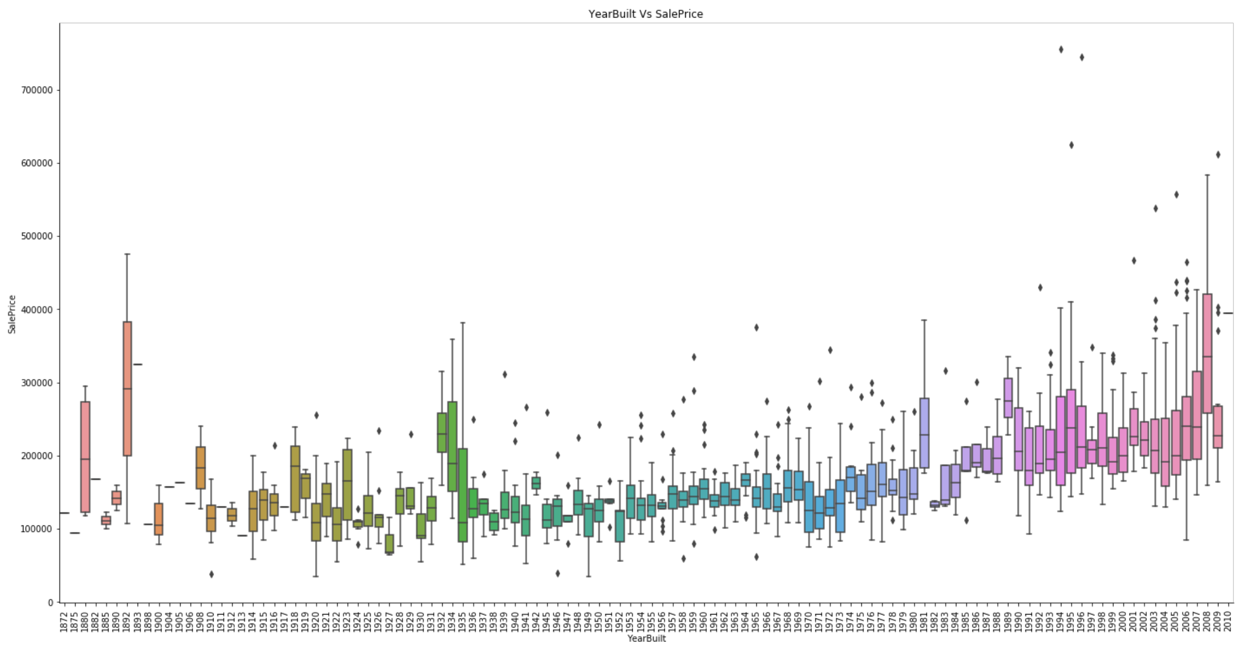
Out[408]:
```
Text(0.5, 1.0, 'OverallQual vs Average amount of SalePrice')
```



As Overall quality of house increases, sale price of house increases. From the above plot we can observe that for a house with high quality, average of saleprice is higher. OverallQual almost follows x^2 with SalePrice which is not linear.

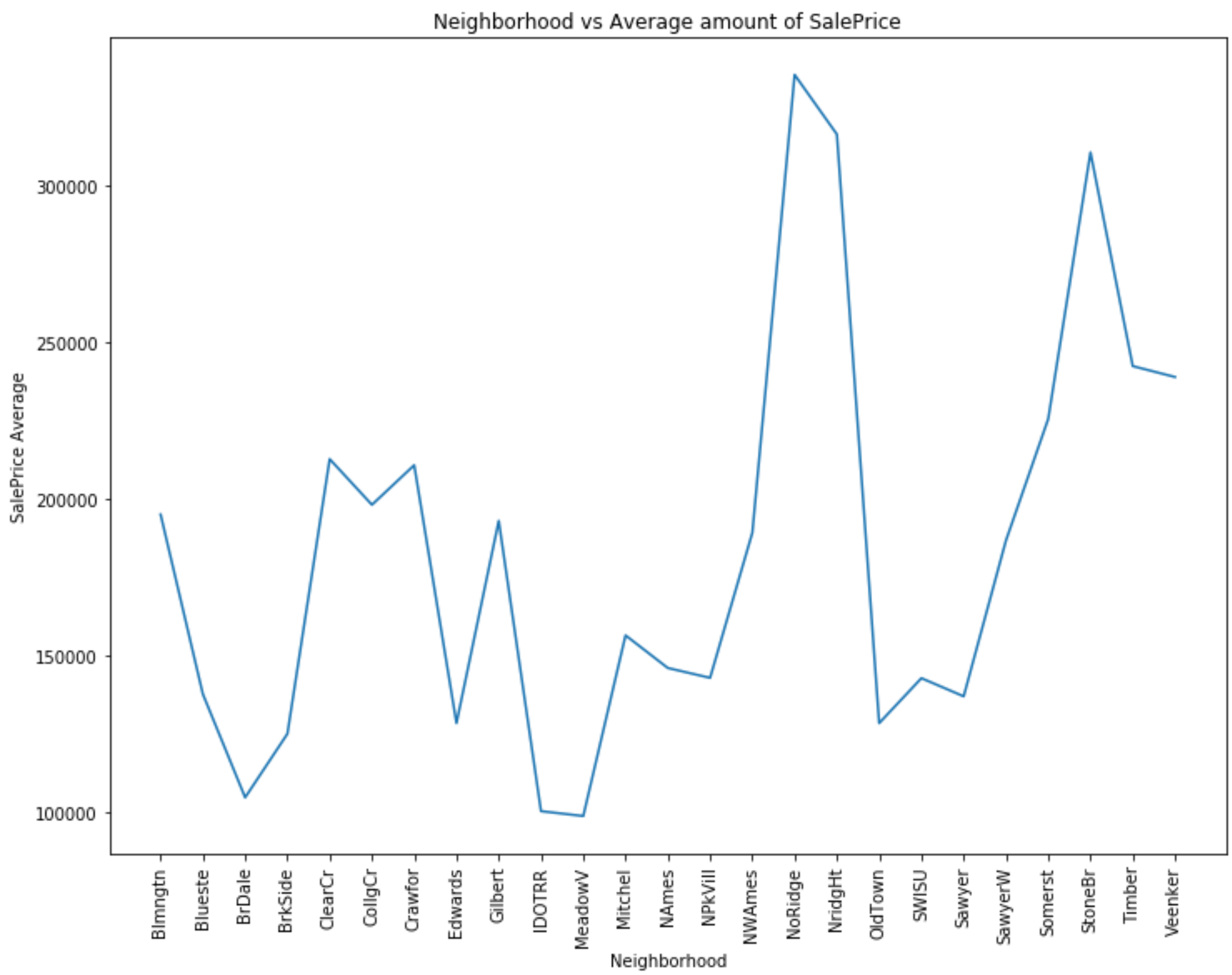
In [343]:
```
sns.catplot(x="YearBuilt", y="SalePrice",
               kind="box", dodge=False, data=x_train, height=10, aspect =
2)
plt.xticks(rotation=90);
plt.title('YearBuilt Vs SalePrice')
```

Out[343]: `Text(0.5, 1, 'YearBuilt Vs SalePrice')`



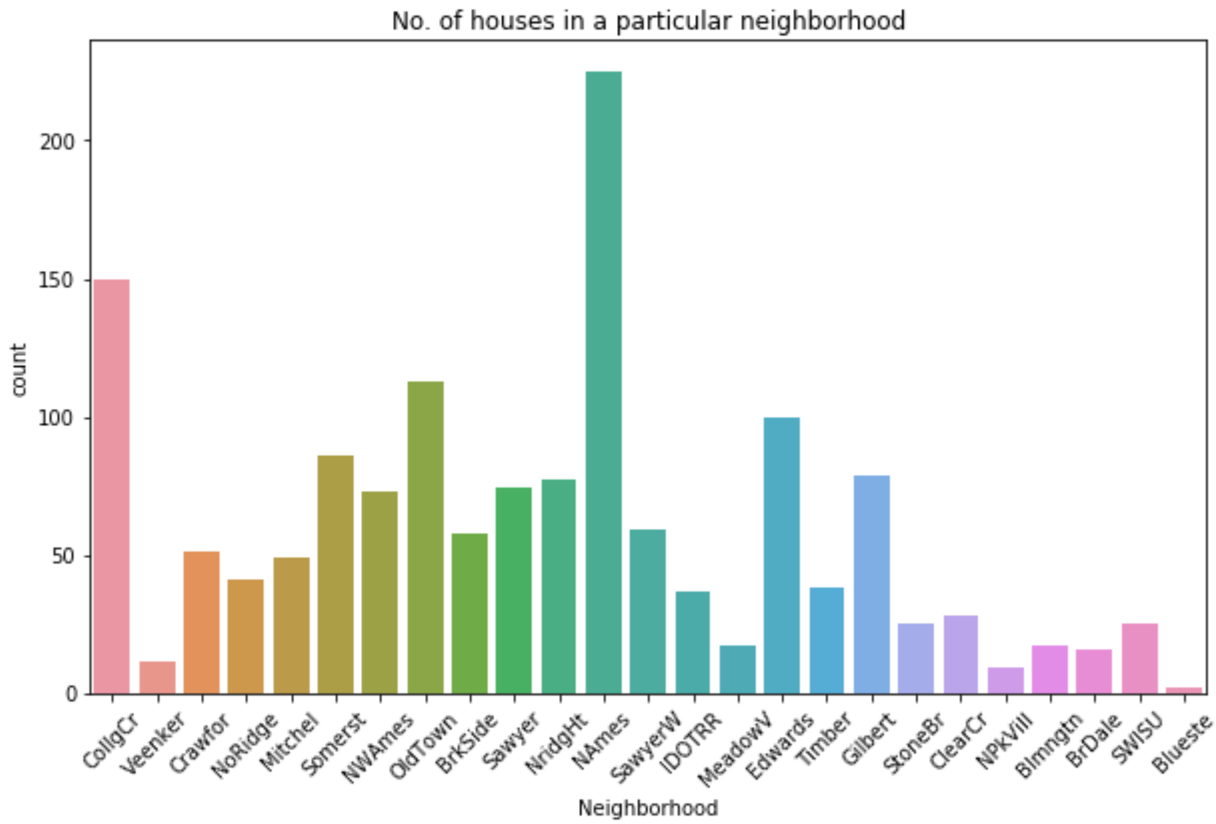
Though there's not much strong tendency, we can say that newly built houses has higher saleprice compared to old houses. The increase or decrease in saleprice amount also depends on economy and land value in that particular year.

In [413]:
```
f, ax = plt.subplots(figsize=(12, 9))
neighborhood = np.unique(x_train["Neighborhood"])
neighborhood_mean = x_train['SalePrice'].groupby(x_train['Neighborhood'
']).mean()
plt.plot(neighborhood, neighborhood_mean)
plt.xlabel('Neighborhood')
plt.ylabel('SalePrice Average')
plt.title('Neighborhood vs Average amount of SalePrice')
plt.xticks(rotation=90);
```



In [414]:
```
plt.figure(figsize = (10, 6))
sns.countplot(x ='Neighborhood', data = x_train)
plt.xticks(rotation=45)
plt.title('No. of houses in a particular neighborhood')
```

Out[414]: Text(0.5, 1.0, 'No. of houses in a particular neighborhood')



From the above two plots, we can infer that neighborhood plays an important role in justifying the sale price. Though the number of houses at NrindgHt and NoRidge are less compared to NAmes and CollgCr, the average saleprice is higher for NrindgHt and NoRidge. From this we can infer that houses near NrindgHt, NoRidge are costly and houses near NAmes and CollgCr are less costly.

Part 3 - Handcrafted Scoring Function

In [415]:
```
enumeration = {'RL':4, 'RM':2,'C (all)':1,'FV':5,'RH':3}
enum_keyset = enumeration.keys()
train_copy = pd.DataFrame(x_train['MSZoning'])
for en in enum_keyset:
    train_copy['MSZoning'] = train_copy['MSZoning'].replace(en,enumeration.get(en))

train_copy['OverallQual'] = (numeric_train['OverallQual'])**2
train_copy['GrLivArea'] = numeric_train['GrLivArea']
train_copy['YearBuilt'] = numeric_train['YearBuilt']
train_copy['GarageArea'] = numeric_train['GarageArea']
```

In [416]:
```
train_copy['desirability'] = train_copy['OverallQual']*(1/np.mean(train_copy['OverallQual']))
+train_copy['MSZoning']*(1/np.mean(train_copy['MSZoning'])) + x_train['GrLivArea']*(1/np.mean(x_train['GrLivArea']))
+x_train['YearBuilt']*(1/np.mean(x_train['YearBuilt'])) + x_train['GarageArea']*(1/np.mean(x_train['GarageArea']))
```

Out[416]:
```
0       2.174708
1       1.974957
2       2.300549
3       2.328807
4       2.782092
5       2.025866
6       2.361270
7       2.024178
8       1.969043
9       1.417053
10      1.808694
11      2.573203
12      1.739516
13      2.793592
14      1.738501
15      2.196368
16      2.014199
17      2.088790
18      2.234415
19      1.614860
20      2.820570
21      1.571056
22      2.144602
23      2.211754
24      1.569191
25      2.899812
26      2.207529
27      2.650330
28      1.667209
29      1.484964
```

```
             ...
1430    1.803614
1431    1.932672
1432    1.434222
1433    1.968104
1434    2.026207
1435    1.972084
1436    2.116190
1437    2.655066
1438    2.944218
1439    2.166762
1440    2.395786
1441    1.904591
1442    2.735408
1443    1.377900
1444    2.340127
1445    1.504749
1446    1.654946
1447    2.187564
1448    1.780793
1449    0.999357
1450    1.001386
1451    2.794607
1452    2.127095
1453    1.017619
1454    1.862306
1455    1.986625
1456    2.060542
1457    1.517437
1458    1.496632
1459    1.580354
Length: 1460, dtype: float64
```

In [417]:

```
train_copy['desirability'].corr(x_train['SalePrice'])
```

Out[417]:

```
0.8171684436128853
```

In [418]:
```
train_copy['SalePrice'] = x_train['SalePrice']
large_ten = train_copy.nlargest(10, "desirability")
print('Top 10 most desirable houses')
print(large_ten)
```

```
Top 10 most desirable houses
     MSZoning  OverallQual  GrLivArea  YearBuilt  GarageArea  desira
bility  \
58          4          100       2945       2006         641       2.
556694
185         2          100       3608       1892         840       2.
556694
224         4          100       2392       2003         968       2.
556694
389         4          100       2332       2007         846       2.
556694
440         4          100       2402       2008         672       2.
556694
515         4          100       2020       2009         900       2.
556694
523         4          100       4676       2007         884       2.
556694
583         2          100       2775       1893         880       2.
556694
591         4          100       2296       2008         842       2.
556694
691         4          100       4316       1994         832       2.
556694

     SalePrice
58      438780
185     475000
224     386250
389     426000
440     555000
515     402861
523     184750
583     325000
591     451950
691     755000
```

In [419]:
```
least_ten = train_copy.nsmallest(10, "desirability")
print('Least 10 most desirable houses')
print(least_ten)
```

```
Least 10 most desirable houses
      MSZoning  OverallQual  GrLivArea  YearBuilt  GarageArea  desir
ability  \
375          4            1        904       1922           0       0
.025567
533          4            1        334       1946           0       0
.025567
636          2            4        800       1936           0       0
.102268
916          1            4        480       1949         308       0
.102268
1100         4            4        438       1920         246       0
.102268
74           2            9       1605       1915         379       0
.230102
88           1            9       1526       1915           0       0
.230102
250          4            9       1306       1940           0       0
.230102
323          2            9       1163       1955         220       0
.230102
342          4            9       1040       1949         400       0
.230102

      SalePrice
375       61000
533       39300
636       60000
916       35311
1100      60000
74       107400
88        85000
250       76500
323      126175
342       87500
```

I considered variables OverallQual, GarageArea, MSZoning, GrLivArea, YearBuilt for scoring function as these features seems to have greater co-relation with SalePrice. I considered different weights for each feature with value: 1/mean(feature). Summation of all the features multiplied with it's respective weights gave the desirability value. Desirability values are highly co-related with SalePrice with value of 0.81.

Part 4 - Pairwise Distance Function

In [420]:
```
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn import preprocessing

#x = train_data.values #returns a numpy array
scaler_min_max = preprocessing.MinMaxScaler()
train_scaled = scaler_min_max.fit_transform(train_final)

train_data = pd.DataFrame(train_scaled,columns=train_final.columns)

pca_model = PCA(n_components=50)
comp = pca_model.fit_transform(train_data)
principal_df = pd.DataFrame(data = comp)
tsne_dimen_reduc = TSNE(n_components=2).fit_transform(principal_df)
tsne_dimen_reduc.shape
```

Out[420]:
```
(1460, 2)
```

In [421]:
```
tsne_dimen_reduc
```

Out[421]:
```
array([[  2.7790134, -36.17673  ],
       [-19.873207 ,  23.003153 ],
       [ 12.359818 , -37.911915 ],
       ...,
       [ -2.4816294, -29.969608 ],
       [-13.216659 ,  24.471989 ],
       [ -2.2494123,   8.493286 ]], dtype=float32)
```

In [422]:
```
def pairwise_euc(x,y):
    if len(x) != len(y):
        return False
    else:
        return np.sqrt(sum([(a-b)**2 for a,b in zip(x,y)]))

distance = []
for each_i in range(len(tsne_dimen_reduc)):
    dist = []
    for each_j in range(len(tsne_dimen_reduc)):
        dist.append(pairwise_euc(tsne_dimen_reduc[each_i], tsne_dimen_
reduc[each_j]))
    distance.append(dist)
```

In [423]:
```
distance_array = np.array(distance)
print(distance_array.shape)
print(distance_array)
```

```
(1460, 1460)
[[ 0.         63.36704152  9.73666697 ...  8.13650652 62.72263341
  44.95214684]
 [63.36704152  0.         68.91744029 ... 55.75464676  6.8166793
  22.82836759]
 [ 9.73666697 68.91744029  0.         ... 16.83296785 67.42334607
  48.65051188]
 ...
 [ 8.13650652 55.75464676 16.83296785 ...  0.         55.48989399
  38.46359543]
 [62.72263341  6.8166793  67.42334607 ... 55.48989399  0.
  19.38038754]
 [44.95214684 22.82836759 48.65051188 ... 38.46359543 19.38038754
   0.        ]]
```

In [424]:
```
neighborhood_feat = x_train['Neighborhood']
true_count = 0
false_count = 0
for i in range(len(distance_array)):
    unsorted_arr = distance_array[i]
    sorted_arr = np.sort(distance_array[i])
    index_val = np.where(unsorted_arr == sorted_arr[1])[0]#ignoring 0
    if neighborhood_feat[index_val[0]]==neighborhood_feat[i]:
        true_count=true_count+1
    else:
        false_count=false_count+1
```

In [425]:
```
(true_count/(true_count+false_count))*100
```

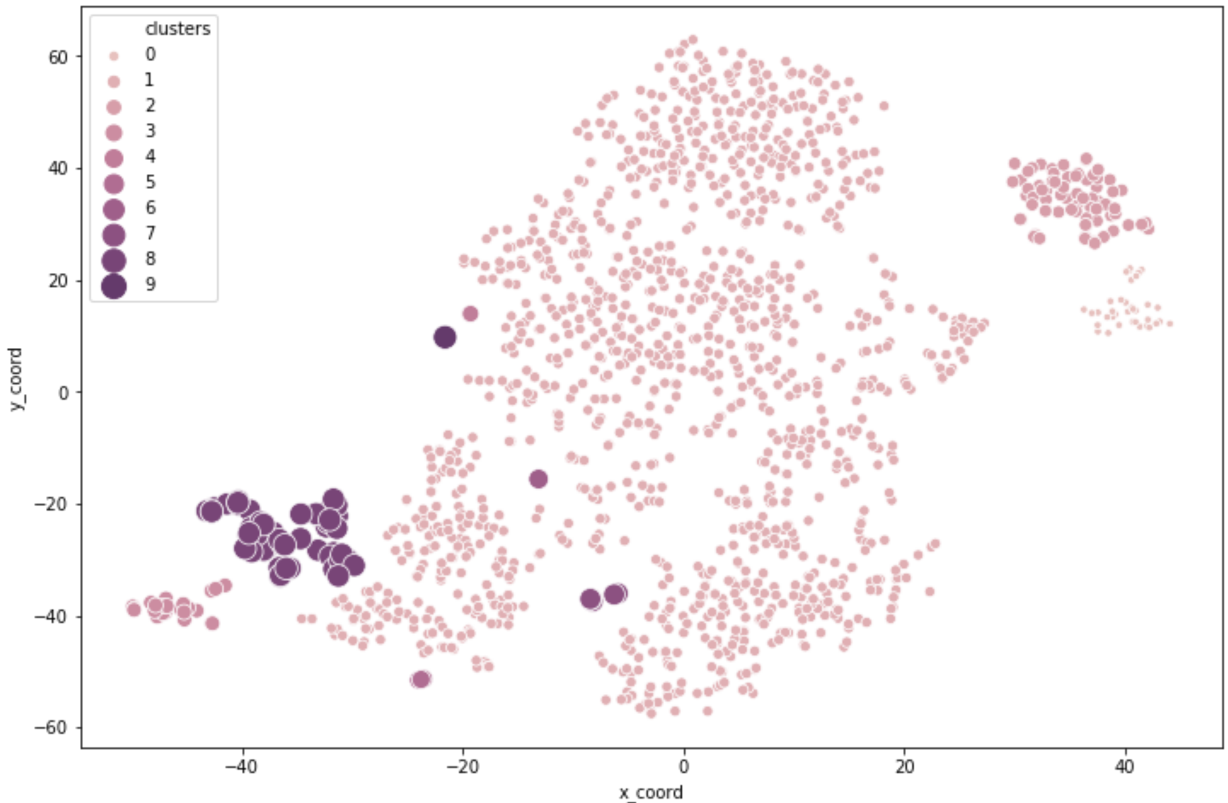
Out[425]:
```
52.26027397260275
```

Implementation: Reduced dimensionality from about 330 features to 50 features using PCA and later reduced it to 2 dimensions using T-SNE. Designed a handcrafted pair-wise function using euclidean. Validate whether the smallest distance pairs contain same neighborhood or not and increment the true and false counts. Sort each distance array and take the one with lowest pair-wise distance, and get the actual index of the element from the original dataset. Compare neighborhood labels with original value of dataset and calculated value. Based on the above score, 52.2 percent of data matches with correct neighborhood

Part 5 - Clustering

In [426]:
```
from sklearn.cluster import AgglomerativeClustering
cluster = AgglomerativeClustering(affinity='precomputed', compute_full
_tree='auto',
                        connectivity=None, distance_threshold=None,
                        linkage='single', memory=None, n_clusters=10,
                        pooling_func='deprecated').fit(distance_array)
x_tsne = pd.DataFrame(tsne_dimen_reduc)
to_plot = pd.DataFrame()
to_plot['Id'] = train_ID
to_plot['x_coord'] = x_tsne[0]
to_plot['y_coord'] = x_tsne[1]
to_plot['clusters'] = cluster.labels_
```

In [427]:
```
import seaborn as sns
plt.figure(figsize=[12,8])
cmap = sns.cubehelix_palette(dark=.3, light=.8, as_cmap=True)
g=sns.scatterplot(x="x_coord", y="y_coord",hue="clusters",data=to_plot
,sizes=(20, 200),size="clusters", palette=cmap, legend="full");
```



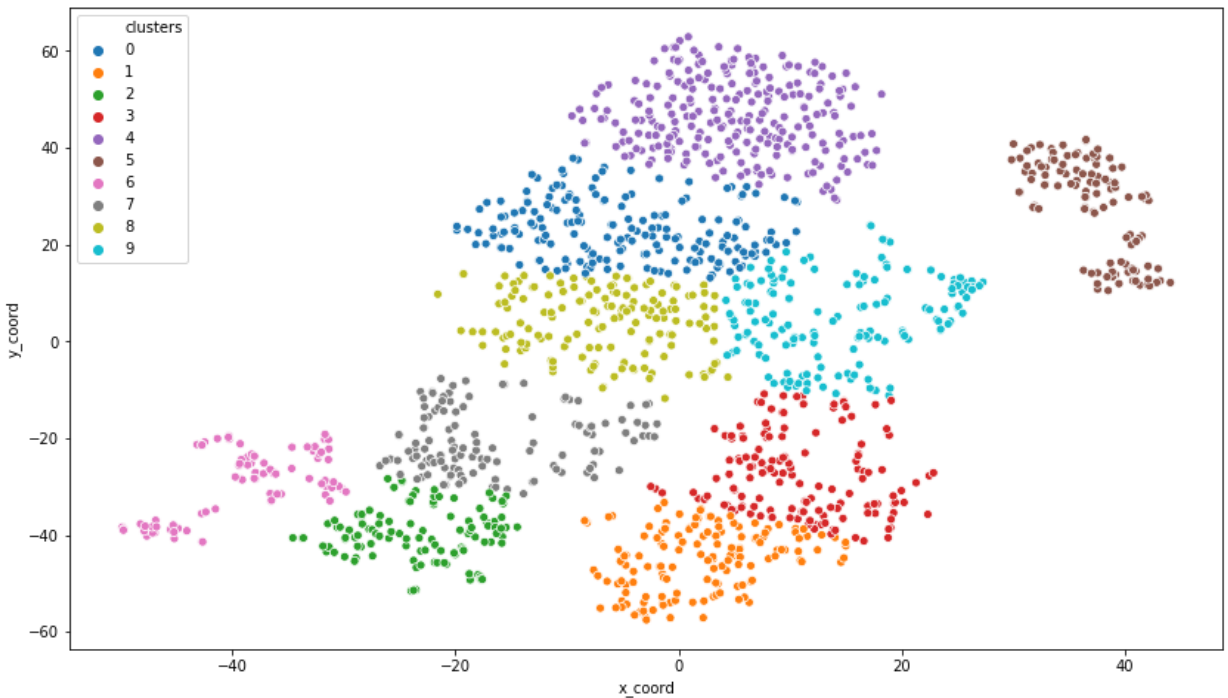
In [428]:
```
import operator

for i in range(0,10):
    neighbor = neighborhood_feat[cluster.labels_==i]
    unique_val, total_count = np.unique(neighbor, return_counts=True)
    dict_neighborhood = dict(zip(unique_val, total_count))
    sorted_dict = sorted(dict_neighborhood.items(), key=operator.itemg
etter(1),reverse=True)
    percent = sorted_dict[0][1] * 100.0/sum(dict_neighborhood.values()
)
    neighborhood = sorted_dict[0][0]
    count = sorted_dict[0][1]
    data_neighborhood = pd.DataFrame({'Clust_centre': i,'Neighborhood'
: neighborhood,'Count':count,'Percentage':percent}, index = [i])
    print(data_neighborhood)
```

```
   Clust_centre Neighborhood  Count  Percentage
0             0      Edwards     10   27.777778
   Clust_centre Neighborhood  Count  Percentage
1             1        NAmes    206       16.48
   Clust_centre Neighborhood  Count  Percentage
2             2      Edwards     19   25.675676
   Clust_centre Neighborhood  Count  Percentage
3             3      Somerst     25       100.0
   Clust_centre Neighborhood  Count  Percentage
4             4      Crawfor      1        50.0
   Clust_centre Neighborhood  Count  Percentage
5             5      Crawfor      3       100.0
   Clust_centre Neighborhood  Count  Percentage
6             6      Crawfor      1       100.0
   Clust_centre Neighborhood  Count  Percentage
7             7      Edwards      4   57.142857
   Clust_centre Neighborhood  Count  Percentage
8             8      Blmngtn     16   26.229508
   Clust_centre Neighborhood  Count  Percentage
9             9      OldTown      1       100.0
```

The above clustering was done using Agglomerative approach. This takes distance matrix as input which was calculated using pair-wise function above and gives out the cluster_labels, from which we can find out the cluster to which it belongs to. From the above details we can see that most of the datapoints belonging to Somerst, Crawfor are classified correctly. From the graph, it's understood that clusters have a clearer separation of data.

In [429]:
```
from sklearn.cluster import KMeans
k_means = KMeans(n_clusters=10, random_state=0).fit(tsne_dimen_reduc)
k_means_plot = pd.DataFrame()
k_means_plot['Id'] = train_ID
k_means_plot['x_coord'] = x_tsne[0]
k_means_plot['y_coord'] = x_tsne[1]
k_means_plot['clusters'] = k_means.labels_
```

In [430]:
```
import seaborn as sns
plt.figure(figsize=[14,8])
ax = sns.scatterplot(x="x_coord", y="y_coord",hue="clusters",legend="f
ull",palette=sns.color_palette()
                     ,data=k_means_plot);
```



In [432]:
```
for i in range(0,10):
    neighbor = neighborhood_feat[k_means.labels_==i]
    unique_val, total_count = np.unique(neighbor, return_counts=True)
    dict_neighborhood = dict(zip(unique_val, total_count))
    sorted_dict = sorted(dict_neighborhood.items(), key=operator.itemg
etter(1),reverse=True)
    percent = sorted_dict[0][1] * 100.0/sum(dict_neighborhood.values()
)
    neighborhood = sorted_dict[0][0]
    count = sorted_dict[0][1]
    data_neighborhood = pd.DataFrame({'Clust_centre': i,'Neighborhood'
: neighborhood,'Count':count,'Percentage':percent}, index = [i])
    print(data_neighborhood)
```

```
   Clust_centre Neighborhood  Count  Percentage
0             0        NAmes    103   56.593407
   Clust_centre Neighborhood  Count  Percentage
1             1      CollgCr     32    21.47651
   Clust_centre Neighborhood  Count  Percentage
2             2      NridgHt     36   31.858407
   Clust_centre Neighborhood  Count  Percentage
3             3      Gilbert     35   24.137931
   Clust_centre Neighborhood  Count  Percentage
4             4      OldTown     89   36.326531
   Clust_centre Neighborhood  Count  Percentage
5             5      Edwards     29   26.363636
   Clust_centre Neighborhood  Count  Percentage
6             6      Somerst     25   29.069767
   Clust_centre Neighborhood  Count  Percentage
7             7      CollgCr     47   37.903226
   Clust_centre Neighborhood  Count  Percentage
8             8        NAmes     41   26.973684
   Clust_centre Neighborhood  Count  Percentage
9             9        NAmes     40   25.974026
```

Kmeans takes input as tsne_dimen_reduc which is the data obtained after performing dimensionality reduction using t-sne and gives the clustering labels as output. In Kmeans almost all the clusters stays the same as data is divided into k sets and assign all items to the cluster whose representative is closer, and cluster mean is calculated to get new representative. So the results appear almost uniformly distributed. From the above dataframe, NAmes is having highest percentage of data with values split between cluster_centre 0,8 and 9.

Part 6 - Linear Regression

In [433]:
```
print(train_final.shape)
print(test_final.shape)
```

```
(1460, 331)
(1459, 331)
```

In [434]:
```
from sklearn.model_selection import train_test_split
x_train_split, x_val_split, y_train_split, y_val_split = train_test_sp
lit(train_final, y_train_label, test_size = 0.3, random_state = 0)
print("X_train : " + str(x_train_split.shape))
print("X_test : " + str(x_val_split.shape))
print("y_train : " + str(y_train_split.shape))
print("y_test : " + str(y_val_split.shape))
```

```
X_train : (1022, 331)
X_test : (438, 331)
y_train : (1022,)
y_test : (438,)
```

Baseline Model

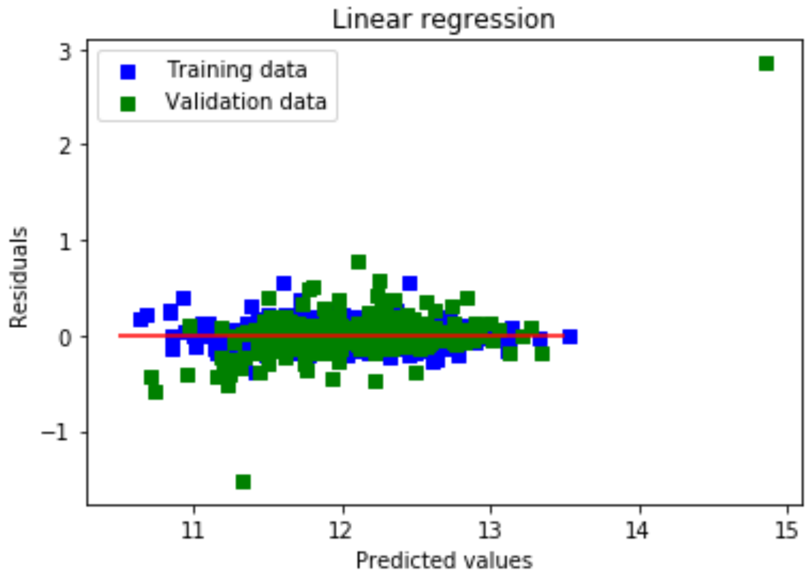
In [435]:
```
from sklearn.linear_model import LinearRegression

lm = LinearRegression()
lm.fit(x_train_split,y_train_split)
y_train_pred = lm.predict(x_train_split)
```

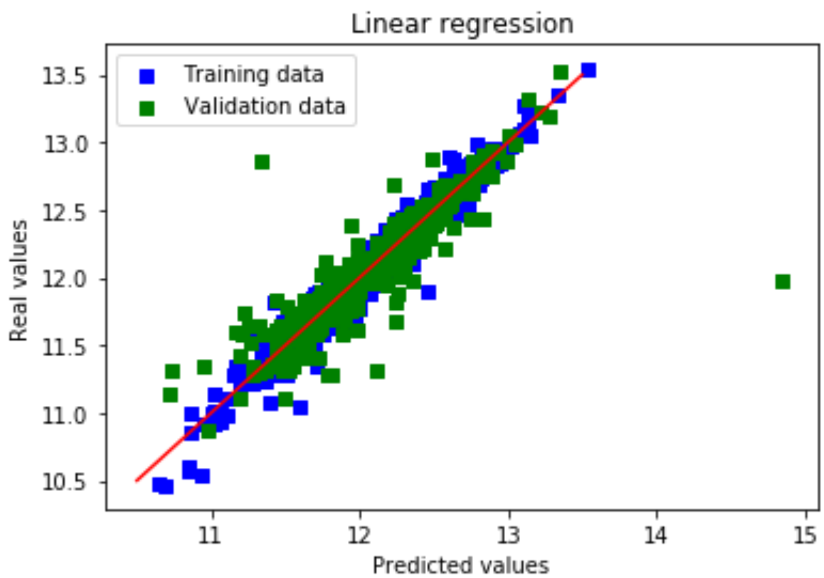
In [436]:
```
y_test_pred = lm.predict(x_val_split)
```

In [437]:
```
mse_train = (np.mean((y_train_pred - y_train_split)**2))
print("MSE on Training set : ", mse_train )


mse_test = (np.mean((y_test_pred - y_val_split)**2))

print("MSE on Test set : ", mse_test )
print(lm.score(x_val_split,y_val_split))
```

```
MSE on Training set :  0.006858881214507823
MSE on Test set :  0.0441451639851263
0.7142913744192403
```

In [438]:
```
# Plot residuals
plt.scatter(y_train_pred, y_train_pred - y_train_split, c = "blue", ma
rker = "s", label = "Training data")
plt.scatter(y_test_pred, y_test_pred - y_val_split, c = "green", marke
r = "s", label = "Validation data")
plt.title("Linear regression")
plt.xlabel("Predicted values")
plt.ylabel("Residuals")
plt.legend(loc = "upper left")
plt.hlines(y = 0, xmin = 10.5, xmax = 13.5, color = "red")
plt.show()

# Plot predictions
plt.scatter(y_train_pred, y_train_split, c = "blue", marker = "s", lab
el = "Training data")
plt.scatter(y_test_pred, y_val_split, c = "green", marker = "s", label
= "Validation data")
plt.title("Linear regression")
plt.xlabel("Predicted values")
plt.ylabel("Real values")
plt.legend(loc = "upper left")
plt.plot([10.5, 13.5], [10.5, 13.5], c = "red")
plt.show()
```





In [453]:
```
import operator
print(lm.coef_.shape)

index_max, value_max = max(enumerate(abs(lm.coef_)), key=operator.item
getter(1))
print(value_max)
print(index_max)
print('Important feature from Linear regression model: ',train_final.c
olumns[index_max])
```

```
(331,)
1.230586503760977
119
Important feature from Linear regression model:  Condition2_PosN
```

By appling log on y_train, RMSE for linear regression gave a value of 0.044 on test set. Linear Regression worked pretty well by taking logarithm of SalePrice and converting ypred back to original format by taking exponentiation. Most important feature in linear regression model is the one with highest absolute value of coefficient vector. Most of the weights are the ones which were on-hot-encoded. The above graph gives a detail view of top 10 and least 10 important variables.

Part 7 - External Dataset

In [454]:
```
#https://en.wikipedia.org/wiki/Crime_in_the_United_States - national a
verage rates
#https://www.addressreport.com/report/neighborhood/ames-ia/somerset-am
es-ia/?display=true

train_merged = pd.DataFrame(train_final)
add_data = {}
robbery_rate_nation = 98
rape_rate_nation = 41.7
murder_rate_nation = 5.3
car_theft_rate_nation = 237.4

#last two properties in the list is percentage of owners in particular
neighborhood and household income

ames_average = [(1-0.94)*robbery_rate_nation,(1-0.0)*rape_rate_nation,
                (1-0.81)*murder_rate_nation,(1-0.67)*car_theft_rate_na
tion, 0.43, 46358]

add_data['Blmngtn'] = [(1-0.98)*robbery_rate_nation,(1-0.54)*rape_rate
_nation,
                      (1-0.79)*murder_rate_nation,(1-0.77)*car_theft_r
ate_nation, 0.83, 95256]

add_data['Blueste'] = ames_average

add_data['BrDale'] = [(1-0.94)*robbery_rate_nation,(1-0.26)*rape_rate_
nation,
                      (1-0.89)*murder_rate_nation,(1-0.70)*car_theft_r
ate_nation, 0.39, 45558]

add_data['BrkSide'] = ames_average

add_data['ClearCr'] = ames_average

add_data['CollgCr'] = [(1-0.93)*robbery_rate_nation,(1+0.11)*rape_rate
_nation,
```

```
              (1-0.92)*murder_rate_nation,(1-0.54)*car_theft_rate_nation
, 0.54, 66875]

add_data['Crawfor'] = ames_average

add_data['Edwards'] = ames_average

add_data['Gilbert'] = ames_average

add_data['IDOTRR'] = ames_average

add_data['MeadowV'] = [(1-0.92)*robbery_rate_nation,(1+0.22)*rape_rate
_nation,
              (1-0.78)*murder_rate_nation,(1-0.46)*car_theft_rate_nation
, 0.5, 53962]

add_data['Mitchel'] = ames_average

add_data['NAmes'] = ames_average

add_data['NoRidge'] = ames_average

add_data['NPkVill'] = ames_average

add_data['NridgHt'] = [(1-0.98)*robbery_rate_nation,(1-0.54)*rape_rate
_nation,
              (1-0.79)*murder_rate_nation,(1-0.77)*car_theft_rate_nation
, 0.83, 95256]

add_data['NWAmes'] = ames_average

add_data['OldTown'] = ames_average

add_data['SWISU'] = ames_average

add_data['Sawyer'] = ames_average

add_data['SawyerW'] = ames_average

add_data['Somerst'] = [(1-0.97)*robbery_rate_nation,(1-0.39)*rape_rate
_nation,
              (1-0.78)*murder_rate_nation,(1-0.69)*car_theft_rate_nation
, 0.61, 84600]

add_data['StoneBr'] = [(1-0.98)*robbery_rate_nation,(1-0.54)*rape_rate
_nation,
              (1-0.79)*murder_rate_nation,(1-0.77)*car_theft_rate_nation
, 0.83, 95256]

add_data['Timber'] = ames_average
```

```
add_data['Veenker'] = ames_average

train_merged['robbery_rate'] = 0
train_merged['rape_rate'] = 0
train_merged['murder_rate'] = 0
train_merged['car_theft_rate'] = 0

keyset = ['rate_of_robbery','rape_rate','rate_murder','rate_car_theft'
, 'owned_property', 'household_income']

robbery_rate = []
rape_rate = []
murder_rate = []
car_theft_rate = []
owned_prop = []
income = []

for i in range(len(train_merged)):
    neighborhood = x_train.iloc[i]['Neighborhood']
    robbery_rate.append(add_data[neighborhood][0])
    rape_rate.append(add_data[neighborhood][1])
    murder_rate.append(add_data[neighborhood][2])
    car_theft_rate.append(add_data[neighborhood][3])
    owned_prop.append(add_data[neighborhood][4])
    income.append(add_data[neighborhood][4])


train_merged['rate_of_robbery']=robbery_rate
train_merged['rape_rate']=rape_rate
train_merged['rate_murder']=murder_rate
train_merged['rate_car_theft']=car_theft_rate
train_merged['owned_property']=owned_prop
train_merged['household_income']=income
```

In [455]:
```
print(train_merged.shape)
```

```
(1460, 340)
```

In [456]:
```
x_train_merged_split, x_val_merged_split, y_train_merged_split, y_val_
merged_split = train_test_split(train_merged, y_train_label, test_size
= 0.3, random_state = 0)
```

In [457]:
```
lm.fit(x_train_merged_split,y_train_merged_split)
y_train_pred_merge = lm.predict(x_train_merged_split)
mse_train_merged = (np.mean((y_train_pred_merge - y_train_merged_split
)**2))
print("MSE on Training set after adding new dataset : ", mse_train_mer
ged )
y_val_pred_merge = lm.predict(x_val_merged_split)
mse_test_merged = (np.mean((y_val_pred_merge - y_val_merged_split)**2)
)
print("MSE on Test set after adding new dataset : ", mse_test_merged )
```

```
MSE on Training set after adding new dataset :  0.006858881214507824
5
MSE on Test set after adding new dataset :  0.044145163985125496
```

Assumption is that sale_price will be less in the areas where the crime_rate is high. These features might help in predicting salePrice but there's not much difference in MSE after adding external dataset related to crime_rate in the neighborhood area like robbery, rape_rate, murder_rate, car_theft rate. Some other features like percentage of owned property currently occupied by owners and also income of household.

Part 8 - Permutation Test

In [458]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.permutation_test_score.html
from sklearn.model_selection import permutation_test_score

lm = LinearRegression()
ten_variables = ['OverallQual', 'GrLivArea','GarageCars','GarageArea',
'TotalBsmtSF','1stFlrSF','FullBath',
                'YearBuilt','TotRmsAbvGrd','YearRemodAdd']

for each_col in ten_variables:
    train_each_feat = np.array(train_final[each_col])
    train_each_feat = np.reshape(train_each_feat, (train_each_feat.sha
pe[0],1))
    model = lm.fit(train_each_feat,y_train_label)
    p_value = permutation_test_score(model, train_each_feat, y_train_l
abel,n_permutations=100)
    print(p_value)
```

```
(0.6662141142239526, array([-2.02756334e-03, -6.90362761e-03, -2.763
02822e-03, -3.20248134e-05,
       -1.73549804e-03, -3.65261092e-04, -4.29779841e-03, -1.0305944
2e-03,
       -5.59690431e-03, -8.91864518e-03, -2.17393594e-03, -1.0751910
```

```
3e-03,
        -4.28092880e-03, -4.95740262e-03, -6.31858375e-03, -1.4659421
8e-03,
        -3.69676729e-03, -7.42087989e-03, -5.92302887e-03, -1.0080062
6e-03,
        -1.10959652e-02, -5.94655675e-03, -5.80500479e-03, -6.3801021
7e-03,
        -2.49115151e-03, -2.12385487e-03, -3.65991819e-03, -5.7007125
7e-04,
        -7.54425074e-03, -3.29459214e-03, -6.96533492e-03, -2.2417708
5e-03,
        -6.78704158e-03, -9.81676320e-03, -4.32468506e-03, -1.7423505
6e-03,
        -1.01697479e-02, -3.64507296e-03,  1.40456887e-03, -2.8699278
9e-03,
         2.22059695e-04, -3.34137835e-03, -7.35697146e-03, -2.2468828
2e-03,
         7.34864175e-04, -3.29791046e-03, -4.67343358e-03, -2.5709465
2e-02,
        -1.19000533e-02, -3.02189407e-03,  3.29546578e-03, -1.9737342
1e-03,
        -2.81011262e-03, -2.54797111e-03, -7.86019320e-03, -6.6982989
8e-03,
        -8.27481306e-03, -5.95713001e-03, -1.95021036e-03, -6.7321832
1e-03,
        -6.28636816e-03, -2.09772194e-03, -2.02107398e-03, -1.2349992
8e-02,
        -5.47012151e-03, -1.10045768e-02, -8.39030090e-04, -3.0697481
2e-03,
        -3.32266168e-03, -6.79022340e-03, -1.94211032e-04,  1.2987317
2e-03,
        -1.28578178e-03, -1.26885265e-04, -1.04635315e-03,  4.2119448
3e-06,
        -4.71599706e-04, -1.03879536e-03, -1.25440483e-03, -2.8288185
2e-03,
        -1.49852124e-03, -1.26391841e-03, -6.97069123e-03, -4.2152321
8e-03,
        -5.95452211e-03, -9.88867005e-03, -7.24711561e-04, -3.2935170
2e-03,
        -1.28372651e-03, -1.72271533e-03, -1.77800403e-03, -2.4151190
0e-03,
        -1.01838923e-02, -3.97350531e-03, -1.42360902e-03, -2.6240837
5e-03,
        -4.49925076e-03, -2.61899255e-03,  1.17370517e-05, -7.0895754
1e-03]), 0.009900990099009901)
(0.47791328554743356, array([-4.67389525e-05, -6.84398277e-03, -3.30
837045e-03, -3.53525856e-03,
        -7.41875945e-04, -1.26304867e-03, -5.19112675e-03, -5.5646900
5e-03,
        -3.77319691e-03, -8.88801420e-03, -5.90713183e-03, -1.0375424
```

```
6e-03,
       -1.07241999e-02, -2.11982548e-03, -7.97112649e-03,  1.8754967
6e-04,
       -5.09422836e-03, -8.02568159e-03, -9.31452628e-03,  4.0152484
8e-04,
       -1.02069211e-02, -6.91141121e-03, -7.49052738e-03, -5.3575867
3e-03,
       -2.57556951e-03, -2.25610547e-03, -3.79257351e-03, -1.0702838
5e-03,
       -3.92868340e-03, -8.65957988e-04, -1.04395870e-02, -7.6935007
4e-04,
       -8.56937717e-03, -7.07419844e-03, -2.23330510e-03, -7.7873533
0e-04,
       -1.11071123e-02, -1.71516777e-03, -3.64571862e-03, -4.7164209
3e-03,
       -3.42919758e-04, -3.58556811e-03, -8.86545039e-03, -2.8267221
6e-03,
       -8.15531414e-04, -6.75115779e-03, -3.29836968e-03, -2.7553785
9e-02,
       -1.17278169e-02, -2.19899989e-03, -1.38093482e-03, -9.8906560
6e-04,
       -1.07317768e-03, -4.23503735e-03, -9.29105458e-03, -7.3599189
5e-04,
       -1.03529648e-02, -6.94919282e-03, -3.15010292e-03, -6.4573244
9e-03,
       -7.23927577e-03, -3.16463179e-03, -3.91444360e-03, -1.1651581
5e-02,
       -4.70108570e-03, -9.75604931e-03, -3.89956253e-03, -6.5646116
2e-04,
       -1.17430664e-03, -9.97246169e-03, -2.58239402e-03,  7.0953743
5e-04,
        5.94171243e-05, -4.71111641e-04, -1.56221714e-03, -1.7717839
1e-03,
       -1.02372623e-03, -2.66426258e-03, -2.62566446e-03, -1.7138455
3e-03,
       -1.39548907e-03, -5.64979265e-04, -4.42065267e-03, -5.4801313
5e-03,
       -7.33659891e-03, -8.06255752e-03, -2.28784250e-03, -3.0298037
1e-03,
       -1.86056842e-03, -5.67492310e-03, -1.08101315e-03, -1.3627311
5e-03,
       -9.24996797e-03, -4.57955738e-03, -3.00730802e-04, -9.2874303
9e-03,
       -3.77564164e-03, -2.10591925e-03,  6.52657432e-03, -6.4015664
2e-03]), 0.009900990099009901)
(0.46010392375143994, array([-8.61964270e-04, -6.32973940e-03, -1.65
795255e-03, -3.81699395e-03,
       -1.09099577e-03, -3.59730511e-04, -4.30094908e-03, -5.6602763
7e-03,
       -6.73542564e-03, -7.20940257e-03, -5.81139946e-03, -1.6320687
```

```
8e-06,
        -1.35281871e-03, -4.87545419e-03, -4.50857547e-03, -4.0320490
1e-04,
        -3.70172749e-03, -7.51852505e-03, -6.95444793e-03, -1.0805154
2e-03,
        -9.94586231e-03, -8.47085369e-03, -7.59217450e-03, -3.6429851
5e-03,
        -7.54297998e-04, -7.16004702e-03, -4.63945731e-03, -1.2553463
0e-03,
        -8.11453418e-03, -4.76141712e-03, -8.25336611e-03,  1.1807841
7e-03,
        -7.48034939e-03, -5.11836982e-03, -2.01509337e-03, -6.2365713
7e-04,
        -1.20760130e-02, -2.54846221e-03, -4.57758835e-03, -9.3250324
8e-03,
         1.71101125e-04, -6.70865918e-03, -6.43823395e-03, -7.1694569
8e-03,
        -6.77907824e-04, -2.88445483e-03, -1.11814933e-02, -2.5021418
8e-02,
        -1.42429930e-02,  3.83814684e-04, -1.78316879e-03, -2.6752266
8e-03,
        -2.01164429e-03, -2.95903343e-03, -9.05921635e-03, -2.3946541
6e-03,
        -5.91217251e-03, -5.13806923e-03, -3.53751782e-03, -6.5579185
2e-03,
        -7.61370621e-03, -1.30123765e-03, -4.81501157e-03, -1.0068297
3e-02,
        -4.37136218e-03, -1.17689179e-02, -8.42659621e-04, -4.2592699
0e-03,
        -7.55787103e-04, -3.27035825e-03, -1.42139330e-04, -2.9365273
1e-03,
        -9.59849453e-04, -1.66824127e-03, -6.92843085e-04, -3.0328524
9e-03,
        -1.85998248e-03, -4.65618347e-03, -3.65308972e-03, -2.3109565
3e-03,
        -1.95699346e-03, -8.47713669e-04, -4.26708981e-03, -3.9900179
0e-03,
        -6.28071175e-03, -7.70203719e-03, -7.94473792e-04, -3.0874496
7e-03,
        -4.20873354e-04, -1.02954966e-03, -5.39376718e-03, -3.2901008
1e-03,
        -1.01740531e-02, -5.47492006e-03, -2.00548889e-03, -5.9359068
2e-03,
        -5.56955188e-03, -1.22787090e-03, -4.40187765e-04, -6.4380508
6e-03]), 0.009900990099009901)
(0.4163606862775098, array([-0.00057852, -0.00511996, -0.00275775, -
0.00329453, -0.00244458,
        -0.00088817, -0.00455772, -0.0047295 , -0.00651798, -0.008473
11,
        -0.00422129,  0.00050265, -0.00211002, -0.00421799, -0.004180
```

```
59,
       -0.00010424, -0.00212349, -0.00719471, -0.00976843, -0.001378
6 ,
       -0.00969827, -0.00466952, -0.00615121, -0.00340859, -0.000935
48,
       -0.00821896, -0.00487301, -0.00112883, -0.00819968, -0.003449
29,
       -0.00781665, -0.00105148, -0.00724916, -0.00625466, -0.002651
64,
       -0.00093021, -0.01038011, -0.00211887, -0.0064422 , -0.006761
58,
        0.00016167, -0.00505403, -0.00596588, -0.00793154, -0.002119
87,
       -0.00293212, -0.01172494, -0.02707124, -0.01547461, -0.000504
,
        0.00046976, -0.00276583, -0.00129702, -0.00291052, -0.007158
73,
       -0.00092459, -0.00635286, -0.0044321 , -0.00267022, -0.007061
58,
       -0.00751481, -0.00253878, -0.00447347, -0.00904321, -0.004493
06,
       -0.01194926, -0.0007559 , -0.00226258, -0.00149126, -0.002624
64,
       -0.00079881, -0.0028385 , -0.00512423, -0.00103581, -0.001621
04,
       -0.00074775, -0.0029217 , -0.00508567, -0.00392739, -0.002005
,
       -0.00078879, -0.00094167, -0.00607693, -0.00397662, -0.007298
59,
       -0.008965  , -0.00119229, -0.0044708 , -0.00013213, -0.001162
66,
       -0.00777423, -0.00299712, -0.00984491, -0.00415695, -0.000897
27,
       -0.00405935, -0.00558166, -0.00138172, -0.00049004, -0.007090
78]), 0.009900990099009901)
(0.36381300464568067, array([-0.00163398, -0.00383374, -0.00335363,
-0.01285209, -0.00185876,
       -0.00082711, -0.00408359, -0.00591877, -0.00344053, -0.008049
96,
       -0.00449682,  0.00161778, -0.00352238, -0.00696726, -0.006000
34,
       -0.00526628, -0.00412635, -0.00743742, -0.00633408, -0.002536
19,
       -0.00709668, -0.00655566, -0.00516459, -0.00509843, -0.004948
72,
       -0.00571743, -0.00198463, -0.0014704 , -0.00380574, -0.000860
98,
       -0.00908927, -0.00443001, -0.00818223, -0.00736967, -0.001177
17,
       -0.00109211, -0.00993737, -0.00823593, -0.00181361, -0.002247
```

```
16,
        0.00052877, -0.00550252, -0.00348656, -0.00260855, -0.000773
53,
       -0.00259138, -0.00826924, -0.02268668, -0.01281074, -0.000617
37,
       -0.00060713, -0.00264117, -0.00191019, -0.00287422, -0.004369
4 ,
       -0.0001752 , -0.00560091, -0.00724977, -0.00151635, -0.007300
97,
       -0.00694953, -0.00502485, -0.00348929, -0.01210623, -0.005528
19,
       -0.0165282 , -0.00099367, -0.00321685,  0.00039809, -0.012245
98,
        0.0002892 , -0.00531371, -0.00242942,  0.00080666, -0.000394
91,
       -0.0012503 , -0.00025173, -0.0036006 , -0.00381348, -0.003199
29,
       -0.00196841, -0.00071529, -0.00404911, -0.0038775 , -0.007391
34,
       -0.00821603, -0.00139504, -0.00217716, -0.00166127, -0.003935
66,
       -0.00504635, -0.00265525, -0.01149414, -0.00536921,  0.000257
29,
       -0.00355535, -0.00519843, -0.00047104, -0.00040944, -0.007292
23]), 0.009900990099009901)
(0.34811127945711773, array([ 0.00261247, -0.00366721, -0.00296434,
-0.00610886, -0.00388074,
       -0.00164239, -0.00453493, -0.00616054, -0.00306994, -0.007583
06,
       -0.00739667, -0.00099793, -0.00300327, -0.00330107, -0.007091
68,
       -0.00722355, -0.00093152, -0.00744742, -0.00673672, -0.003571
65,
       -0.00877695, -0.01061073, -0.0052714 , -0.00442366, -0.004797
63,
       -0.00248377, -0.00343825, -0.00329637, -0.00528941, -0.000427
35,
       -0.01080339, -0.0019792 , -0.00874897, -0.0073968 , -0.001216
52,
       -0.00194538, -0.0086483 , -0.00556355, -0.00200707, -0.006701
2 ,
       -0.00017747, -0.00939609, -0.00624273, -0.00210152, -0.002348
63,
       -0.00392707, -0.00349252, -0.02450075, -0.01320986, -0.001331
41,
       -0.00166187, -0.00292224, -0.00256785, -0.00261349, -0.006344
05,
       -0.00038391, -0.00553147, -0.00423557, -0.0024136 , -0.006894
98,
       -0.00770518, -0.00510518, -0.0006732 , -0.01153512, -0.005473
```

```
09,
       -0.01489312, -0.00056312, -0.0009589 ,  0.000336  , -0.016729
49,
       -0.00326074, -0.00238481, -0.00139721,  0.00132722, -0.000359
84,
        0.00026351, -0.00072741, -0.00135986, -0.00350547, -0.001524
32,
       -0.00085938, -0.00174222, -0.00605266, -0.00427261, -0.008276
85,
       -0.01070729, -0.00210054, -0.00295651, -0.00312674, -0.007587
82,
       -0.00273843, -0.00374445, -0.01050167, -0.00596339, -0.000422
53,
       -0.0031506 , -0.00375691, -0.00018139,  0.00288196, -0.006604
45]), 0.009900990099009901)
(0.35178576192334315, array([-0.00396584, -0.00463686, -0.00365379,
-0.0024781 , -0.00149306,
       -0.00180163, -0.00372296, -0.00329111, -0.00739232, -0.006991
88,
       -0.00455476, -0.00068557, -0.00394741, -0.0011483 , -0.007266
45,
       -0.00316551, -0.00205591, -0.0069429 , -0.01091456, -0.003902
59,
       -0.01111194, -0.00743126, -0.00447155, -0.005691  , -0.002789
97,
       -0.00334452, -0.00262323, -0.00243162, -0.00437529, -0.000990
61,
       -0.00555949, -0.0014447 , -0.00749846, -0.00480538, -0.002744
49,
       -0.00057704, -0.0140717 , -0.00246597,  0.00051122, -0.002671
3 ,
       -0.00332985, -0.00128685, -0.00490344, -0.00186486, -0.002809
2 ,
       -0.00428507, -0.00688727, -0.02421271, -0.01082453, -0.002000
49,
        0.00038878, -0.00314202, -0.00138424, -0.00406817, -0.009124
18,
       -0.00073362, -0.00615087, -0.00722834, -0.00522881, -0.005860
31,
       -0.00912155, -0.00353516, -0.00150794, -0.01295196, -0.005541
97,
       -0.00905143, -0.00178269,  0.00114859, -0.00119998, -0.003611
98,
       -0.00099644, -0.00218277, -0.00150172, -0.00398744, -0.001661
04,
       -0.00067368, -0.00222949,  0.00068195, -0.00214784, -0.002402
62,
       -0.00188421, -0.00351862, -0.00209904, -0.00260371, -0.005438
8 ,
       -0.00957657, -0.00073974, -0.00483064,  0.00219994, -0.001230
```

```
11,
       -0.00212953, -0.00169904, -0.01214722, -0.00848818, -0.001253
46,
       -0.00150793, -0.00488396, -0.00219933,  0.00452896, -0.008388
16]), 0.009900990099009901)
(0.342290077614468, array([-2.52538386e-03, -3.00911365e-03, -2.2125
1393e-05, -7.37395480e-03,
       -3.89513110e-04, -1.62944433e-03, -4.48922803e-03, -5.7621077
1e-03,
       -8.88047970e-03, -8.48633641e-03, -1.46773054e-03, -2.9089781
7e-04,
       -1.81519811e-03, -2.27602760e-03, -7.97037349e-03, -1.1790137
2e-03,
       -2.26858992e-03, -7.37447863e-03, -4.43450105e-03, -2.7342605
6e-03,
       -1.47167573e-02, -1.82345264e-03, -1.29690271e-02, -5.0745967
8e-03,
       -1.98430766e-03, -4.97234948e-03, -3.25898026e-03, -1.0008135
1e-03,
       -4.30435162e-03, -7.15562712e-03, -5.90424030e-03, -6.7478963
2e-03,
       -6.75543718e-03, -6.28370542e-03, -9.57164368e-03, -4.1239280
9e-03,
       -1.06391366e-02, -5.15798285e-03, -5.12072684e-03, -4.7497662
2e-03,
        4.14666491e-04,  1.98364706e-03, -6.69631484e-03, -2.6694381
4e-03,
       -2.17885373e-03, -1.98016843e-03, -6.93379006e-03, -2.6325507
1e-02,
       -1.16251820e-02, -1.00761298e-03, -1.22135014e-04, -2.1973037
4e-03,
       -2.08187295e-03, -3.60007982e-03, -8.24594723e-03,  9.3114522
4e-04,
       -6.59456190e-03, -1.03043392e-02, -3.04441889e-03, -6.5025536
6e-03,
       -8.99065915e-03, -3.85196377e-03, -1.41107275e-03, -1.1768633
4e-02,
       -5.15679489e-03, -1.02840434e-02, -3.05988064e-03, -1.7653719
7e-03,
       -4.20945945e-03, -1.28182823e-03, -5.03500419e-03, -1.0479905
6e-03,
       -9.31999865e-05, -1.72569433e-03, -1.38245817e-03, -1.4882630
1e-03,
        1.14426460e-04, -4.01933667e-03, -2.25737038e-03, -1.5529901
6e-03,
       -2.23286299e-03, -1.67001862e-03, -3.89776841e-03, -1.0122797
6e-03,
       -9.95130282e-03, -8.16712702e-03, -5.79760074e-03, -4.7225458
1e-03,
       -3.12649332e-03, -1.03519126e-03, -1.22807894e-03, -9.0188914
```

```
1e-04,
       -6.51433018e-03, -4.47772058e-03, -7.58773003e-04, -4.0697621
2e-03,
       -6.45457205e-03, -2.48493956e-03, -9.15831068e-04, -8.0405559
4e-03]), 0.009900990099009901)
(0.2766168412909925, array([-6.98652716e-04, -6.37613565e-03, -3.994
25683e-03, -3.79835753e-03,
       -4.95833716e-04, -6.41183935e-04, -5.69495314e-03, -8.3589201
7e-03,
       -2.95573125e-03, -8.37933297e-03, -5.33737992e-03, -1.9216540
8e-03,
       -5.10313056e-03, -4.14050669e-03, -7.96716171e-03, -4.8555799
9e-03,
       -3.05231516e-03, -9.05602529e-03, -9.09164816e-03, -2.7070078
7e-03,
       -1.00082191e-02, -6.66340669e-03, -7.15936125e-03, -3.8661974
6e-03,
       -2.82880690e-03,  8.24110225e-05, -4.09963589e-03, -8.1357390
3e-04,
       -4.10736545e-03, -1.65245156e-03, -7.62585326e-03, -1.9471874
7e-03,
       -8.72731221e-03, -7.22818767e-03, -4.89765515e-04, -9.6315365
4e-04,
       -1.36624840e-02, -4.35227221e-04, -2.44201601e-03, -6.9058909
2e-03,
       -7.62974046e-04, -2.90776433e-03, -1.07647761e-02, -2.5270104
9e-03,
       -1.55717635e-03, -5.50539771e-03,  4.57912071e-04, -2.5124034
3e-02,
       -1.20427277e-02, -5.86487780e-03, -1.32482741e-03, -1.5639405
7e-03,
       -1.61674164e-03, -3.35319264e-03, -1.06046412e-02, -1.2585685
6e-03,
       -8.80744049e-03, -8.35574388e-03, -3.09404763e-03, -7.9880363
1e-03,
       -7.62675511e-03, -4.02519647e-03, -2.28110237e-03, -1.2920667
9e-02,
       -4.94491090e-03, -1.02579454e-02, -5.69368329e-03, -2.8233267
4e-04,
       -2.11228054e-03, -6.61436702e-03, -1.11211161e-03, -9.2603489
8e-04,
       -3.50105919e-04, -1.17177171e-03, -1.07986132e-03, -1.9881261
1e-03,
       -1.07225814e-03, -2.69672673e-03, -3.43768037e-03, -4.0963860
0e-03,
       -3.64207278e-03,  1.78921965e-04, -2.68015910e-03, -4.8553979
0e-03,
       -6.85747784e-03, -8.79353927e-03, -7.28507941e-04, -3.1523193
4e-03,
       -1.15136354e-03, -9.64399271e-03, -1.27817390e-03,  8.6336677
```

```
4e-04,
       -8.85338365e-03, -6.14748661e-03,  1.55573525e-03, -1.0330380
5e-02,
       -4.30443347e-03, -4.61378365e-03,  5.39061350e-03, -6.5828454
6e-03]), 0.009900990099009901)
(0.31557841790382707, array([-1.24825833e-03, -4.64436021e-03, -1.16
773068e-03, -3.06250884e-03,
       -2.57558421e-03,  3.15747809e-04, -4.47824450e-03, -4.6562987
2e-03,
       -7.51905147e-03, -9.80959368e-03, -5.31649486e-03, -7.6473129
8e-04,
       -3.31112167e-05, -7.37040082e-03, -6.37491642e-03,  1.1212596
3e-03,
       -1.47135796e-03, -8.26747257e-03, -7.05571109e-03, -4.1573192
4e-04,
       -7.67949261e-03, -2.78775154e-03, -5.95110355e-03, -5.7453532
8e-03,
       -8.00086643e-04, -2.69677673e-03, -2.67244759e-03, -5.6693929
0e-04,
       -4.13399109e-03, -3.18138020e-03, -5.84694484e-03, -2.8519690
9e-03,
       -6.18581113e-03, -6.89402387e-03, -6.92682538e-03, -1.3047429
5e-03,
       -1.44859441e-02, -2.63591832e-03, -6.56625921e-04, -2.5989178
5e-03,
        2.63561985e-03, -3.96205354e-04, -6.18861274e-03, -2.0698511
9e-03,
       -5.41445371e-03, -3.60432609e-03, -6.81364037e-03, -2.6499903
5e-02,
       -1.22567031e-02, -1.12536357e-03, -1.47216184e-04, -3.2076626
0e-03,
       -2.29164877e-03, -3.03665828e-03, -6.07898527e-03, -1.5980895
1e-03,
       -7.09267594e-03, -8.55746294e-03, -2.11630565e-03, -7.0603313
0e-03,
       -7.78398071e-03, -2.02250939e-03, -1.79363465e-03, -1.5211605
1e-02,
       -5.00117169e-03, -9.71290541e-03, -3.47380366e-03,  1.5248245
3e-03,
       -8.14731898e-03, -2.62577425e-03, -8.10253326e-04, -3.2988332
2e-03,
       -9.96769368e-04, -3.61986106e-03, -1.21874119e-03,  1.0182797
5e-03,
       -1.31613613e-03, -1.95845866e-03, -5.66529129e-03, -2.2258023
5e-03,
       -2.44429074e-03, -6.57071922e-04, -5.58223434e-03, -2.8614712
3e-03,
       -6.71406044e-03, -7.95669439e-03, -3.06795728e-03, -2.1921531
3e-03,
       -3.55750294e-03, -3.66528364e-03, -1.23605534e-03, -1.5412686
```

```
3e-03,
       -8.78821683e-03, -4.37977205e-03, -1.21503523e-03, -2.3399991
7e-03,
       -5.02214499e-03, -2.36435777e-03, -8.42818529e-04, -6.5119456
6e-03]), 0.009900990099009901)
```

The p-value, which approximates the probability that the score would be obtained by chance. This is calculated as:

(C + 1) / (n_permutations + 1) Where C is the number of permutations whose score >= the true score.

In the first iteration with feature OverallQual, true score observed is 0.66. But all the permutation values are < true score so in this case C value is 0. In similar way, for all the considered features the true value is larger than all the permutations. So C is 0 in all cases.

The p-value is therefore 1/(100 + 1) = 0.0099 as obtained

Part 9 - Final Result

1. Applied Linear Regression as baseline-model.
2. Compared to Lasso, Ridge regression is giving better Mean squared error for test set.
3. Done hyper parameter tuning for both Ridge and Lasso regression

In [459]:

```
from sklearn.linear_model import RidgeCV

# hyper parameter tuning
ridge_model = RidgeCV(alphas = [1e-12, 1e-10, 1e-8, 1e-6, 1e-4,1e-2, 1
, 5, 8, 9, 10, 10.2, 10.5,12,14, 15, 20, 25, 30, 40, 50, 60, 100])
ridge_model.fit(x_train_split, y_train_split)
lamda = ridge_model.alpha_
print("Best lamda :", lamda)
```

```
Best lamda : 15.0
```

In [460]:

```
ridge_model.fit(x_train_split, y_train_split)
ytrain_rid_pred = ridge_model.predict(x_train_split)
```

In [461]:

```
ytest_rid_pred = ridge_model.predict(x_val_split)
```

In [462]:
```
mse_train = (np.mean((ytrain_rid_pred - y_train_split)**2))
print("MSE on Training set after L2 reg: ", mse_train )


mse_test = (np.mean((ytest_rid_pred - y_val_split)**2))

print("MSE on Test set after L2 reg : ", mse_test )

# MSE on the test set slightly increased after applying L2 Regularizat
ion
```

```
MSE on Training set after L2 reg:  0.010927692491958208
MSE on Test set after L2 reg :  0.02595027602207179
```

In [463]:
```
coefs = pd.Series(ridge_model.coef_, index = x_train_split.columns)
print("ridge model picked " + str(sum(coefs != 0)) + " features and el
iminated the other " +  \
      str(sum(coefs == 0)) + " features")
```

```
ridge model picked 322 features and eliminated the other 9 features
```

In [464]:
```
from sklearn.linear_model import LassoCV

lasso = LassoCV(alphas = [0.0001, 0.00045,0.0004,0.00048 , 0.001, 0.00
2,0.004, 0.006, 0.01, 0.02,0.04, 0.06, 0.1,
                          0.4, 0.6, 1],
                max_iter = 50000, cv = 10)
lasso.fit(x_train_split, y_train_split)
best_alpha = lasso.alpha_
print("best_alpha :", best_alpha)
```

```
best_alpha : 0.0004
```

In [465]:
```
lasso.fit(x_train_split, y_train_split)
ytrain_l1_pred = lasso.predict(x_train_split)
ytest_l1_pred = lasso.predict(x_val_split)
```

In [466]:
```
mse_train = (np.mean((ytrain_l1_pred - y_train_split)**2))
print("MSE on Training set after L1 reg : ", mse_train )


mse_test = (np.mean((ytest_l1_pred - y_val_split)**2))

print("MSE on Test set after L1 reg : ", mse_test )
```

```
MSE on Training set after L1 reg :  0.009165805932448604
MSE on Test set after L1 reg :  0.031299195582668526
```

In [467]:
```
coefs = pd.Series(lasso.coef_, index = x_train_split.columns)
print("lasso model picked " + str(sum(coefs != 0)) + " features and el
iminated the other " +  \
      str(sum(coefs == 0)) + " features")
```

```
lasso model picked 135 features and eliminated the other 196 feature
s
```

In [468]:
```
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV
# A parameter grid for XGBoost
params = {'min_child_weight':[4,5], 'gamma':[i/10.0 for i in range(3,6
)],  'subsample':[i/10.0 for i in range(6,11)],
'colsample_bytree':[i/10.0 for i in range(6,11)], 'max_depth': [2,3,4]
}
```

In [ ]:
```
xgb = XGBRegressor(nthread=-1)

grid_search = GridSearchCV(xgb, params)
grid_search.fit(x_train_split, y_train_split)
```

In [ ]:
```
ytrain_xgb_pred = grid_search.predict(x_train_split)
ytest_xgb_pred = grid_search.predict(x_val_split)

mse_train = (np.mean((ytrain_xgb_pred - y_train_split)**2))
print("MSE on Training set after xgb : ", mse_train )


mse_test = (np.mean((ytest_xgb_pred - y_val_split)**2))

print("MSE on Test set after xgb : ", mse_test )
```

In [ ]:
```
y_pred_test_final = lasso.predict(test_final)
#y_pred_test_final = np.exp(y_pred_test_final)
y_predict_test_df = pd.DataFrame(y_pred_test_final, columns=['SalePric
e'])
final_df= pd.DataFrame(data={'Id':test_ID,'SalePrice':y_predict_test_d
f['SalePrice']})
print(final_df.head(5))
final_df.to_csv('submissionlas.csv', index=False)
```

Report the rank, score, number of entries, for your highest rank. Include a snapshot of your best score on the leaderboard as confirmation. Be sure to provide a link to your Kaggle profile. Make sure to include a screenshot of your ranking. Make sure your profile includes your face and affiliation with SBU.

Kaggle Link: [https://www.kaggle.com/meghanavemulapalli (https://www.kaggle.com/meghanavemulapalli)](https://www.kaggle.com/meghanavemulapalli)

Highest Rank: 914

Score: 0.11971

Number of entries: 7

914	**Meghana Vemulapalli**	0.11971	7	~10s