

## 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 [653]: import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import math
import numpy as np
from sklearn.metrics.pairwise import euclidean_distances
from catboost import CatBoostRegressor
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import permutation_test_score
from sklearn.metrics import make_scorer
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.manifold import TSNE
import plotly.graph_objs as go
import plotly.offline as pyo
from plotly import tools
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import plotly_express as px
init_notebook_mode(connected=False)
from matplotlib import cm
```

```

In [451]: # Initial data reading

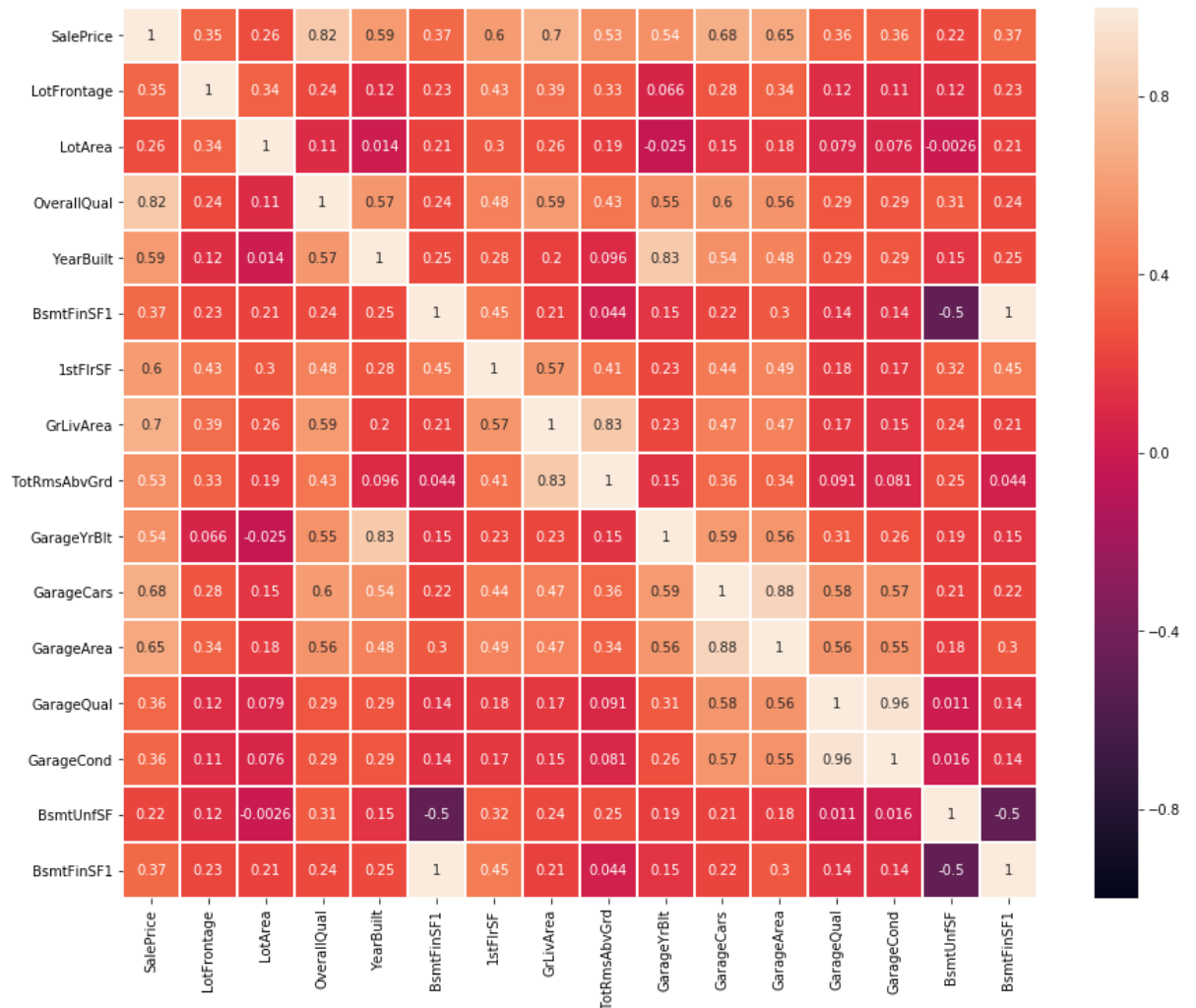
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
train_Id = train['Id']
test_Id = test['Id']
train.drop("Id", axis = 1, inplace = True)
test.drop("Id", axis = 1, inplace = True)

# Data Cleaning. Filling NA values and converting categorical columns to nominal.
qualMap = {'Ex':5, 'Gd':4, 'TA':3, 'Fa':2, 'Po':1, 'NA':0}
for c in ['HeatingQC', 'KitchenQual', 'BsmtQual', 'ExterQual', 'ExterCond', 'BsmtCond', 'Gar
    train[c]=train[c].fillna("NA")
    train[c]=train[c].replace(qualMap)
    train[c]=train[c].astype(int)
fMap = {'CBBlock':5, 'PConc':4, 'BrkTil':3, 'Slab':2, 'Stone':1, 'Wood':0}
train['Foundation']=train['Foundation'].fillna("NA")
train['Foundation']=train['Foundation'].replace(fMap)
train['Foundation']=train['Foundation'].astype(int)
salePrice=train.copy()
train['SalePrice'] = train['SalePrice'].apply(lambda x: math.log1p(x))
for col in ['MiscFeature', 'Alley', 'Fence', 'FireplaceQu', 'GarageType', 'GarageFinish', 'Mas
    train[col]=train[col].fillna('None')
for col in ['GarageArea', 'GarageCars', 'MasVnrArea']:
    train[col]=train[col].fillna(0)
for col in ['Electrical', 'MSZoning', 'KitchenQual', 'Exterior1st', 'Exterior2nd', 'SaleType']
    train[col]=train[col].fillna(train[col].mode()[0])
train['LotFrontage']=train.groupby('Neighborhood')['LotFrontage'].transform(lambda x: x.
train.drop(['Utilities'],axis=1,inplace=True)
for col in ['MoSold', 'MSSubClass', 'YrSold', 'OverallCond']:
    train[col]=train[col].astype(str)
train['BsmtGrade'] = train['BsmtQual'] * train['BsmtCond']
train['TotalArea'] = train['TotalBsmtSF'] + train['GrLivArea']
train['TotalFlrSF'] = train['1stFlrSF'] + train['2ndFlrSF']
train['TotalBath'] = train['BsmtFullBath'] + 0.5 * train['BsmtHalfBath'] + train['FullBa
train['OverallCond']=train['OverallCond'].astype(int)

```

## Part 1 - Pairwise Correlations

```
In [452]: # plotting the heatmap for only a few selected values which gave good correlation values
fields = ['SalePrice', 'LotFrontage', 'MSSubClass', 'LotArea', 'OverallQual', 'YearBuilt', 'BsmtFinSF1', '1stFlrSF', 'GrLivArea', 'TotRmsAbvGrd', 'GarageYrBlt', 'GarageCars', 'GarageArea', 'GarageQual', 'GarageCond', 'BsmtUnfSF', 'BsmtFinSF1']
plt.figure(figsize=(15,12))
sns.heatmap(train[fields].corr(method = "pearson"), vmin=-1, vmax=1, linewidths=1, annot
```



Discuss most positive and negative correlations.

The highest correlation seen in the above heatmap is between GarageQual and GarageCond with a value of 0.88.

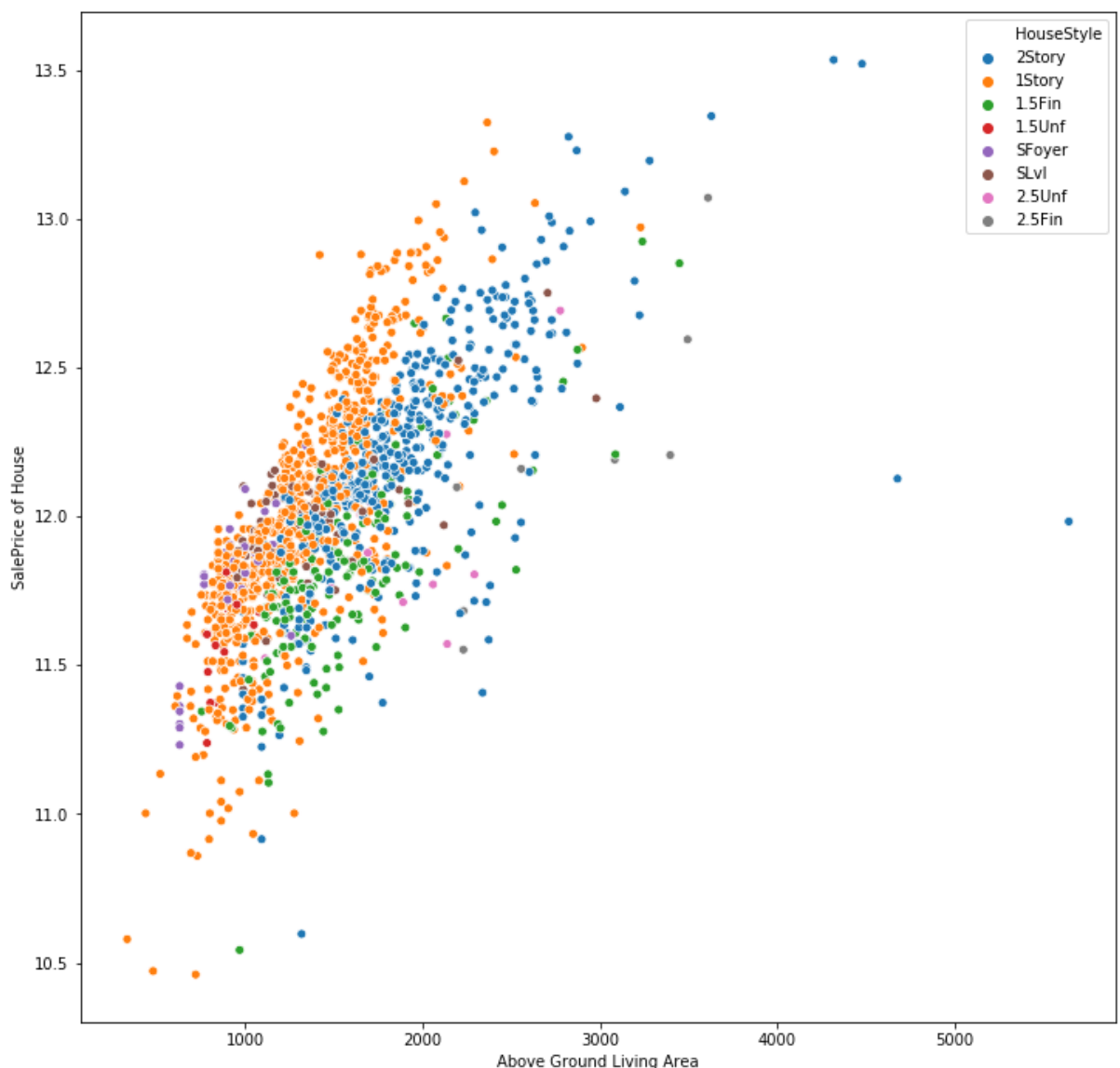
The highest correlation that helps with the modelling seen in the above heatmap is between OverallQual and SalePrice with a value of 0.82. The correlation is a genuine one because OverallQual is a feature that describes a house's quality evaluation which will eventually increase its price.

The least correlation seen in the above heatmap is between BsmtFinSF1 and BsmtUnfSF with a value of -0.5. The correlation is a genuine one since the unfinished surface area of basement is inversely proportional to the finished surface area of the basement.

## Part 2 - Informative Plots

```
In [453]: # TODO: code to generate Plot 1
plt.figure(figsize=(12,12))
plt.suptitle("Above Ground Living Area with respect to SalePrice")
splot=sns.scatterplot(x=train['GrLivArea'], y=train['SalePrice'],hue=train['HouseStyle'])
splot.set_xlabel("Above Ground Living Area")
splot.set_ylabel("SalePrice of House");
```

Above Ground Living Area with respect to SalePrice

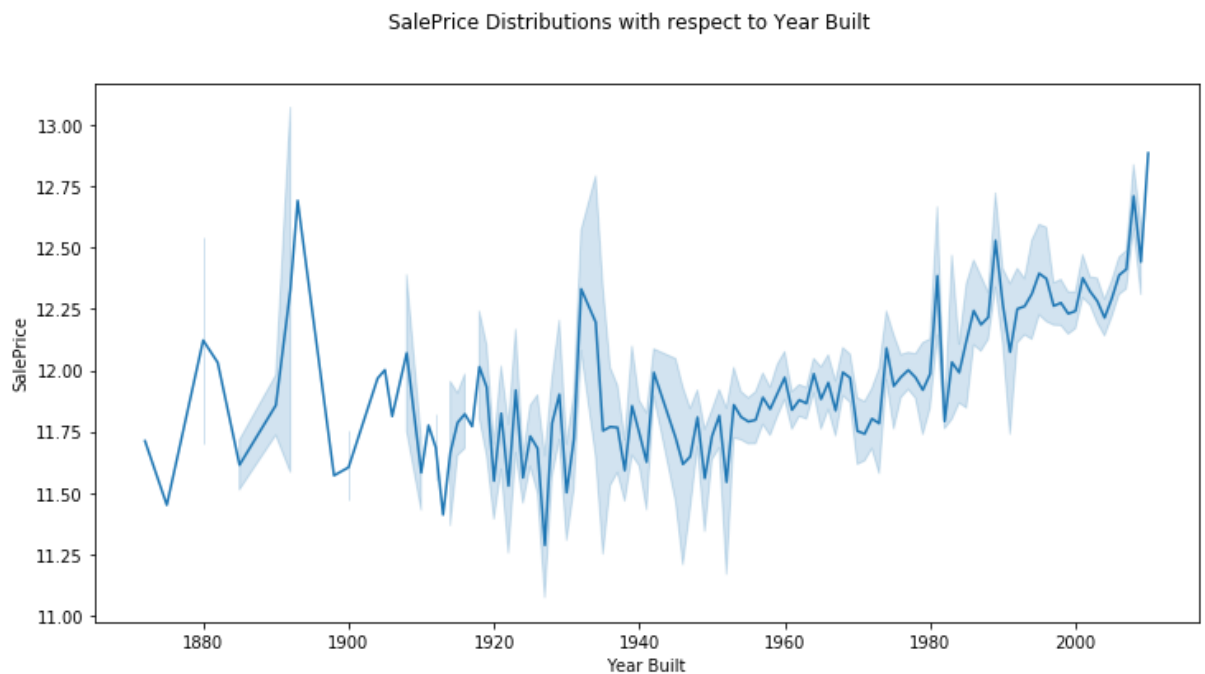


What interesting properties does Plot 1 reveal?

We see that as the GrLivArea increases, the SalePrice is increasing linearly. So we observe a good correlation between the data and GrLivArea could be a potential important feature in designing the prediction model.

```
In [454]: # TODO: code to generate Plot 2
plt.figure(figsize=(12,6))
plt.suptitle("SalePrice Distributions with respect to Year Built")
val=train[['SalePrice','YearBuilt']]
lplot=sns.lineplot(x=val['YearBuilt'], y=val['SalePrice'])
lplot.set_xlabel("Year Built")
lplot.set_ylabel("SalePrice")
```

Out[454]: Text(0, 0.5, 'SalePrice')



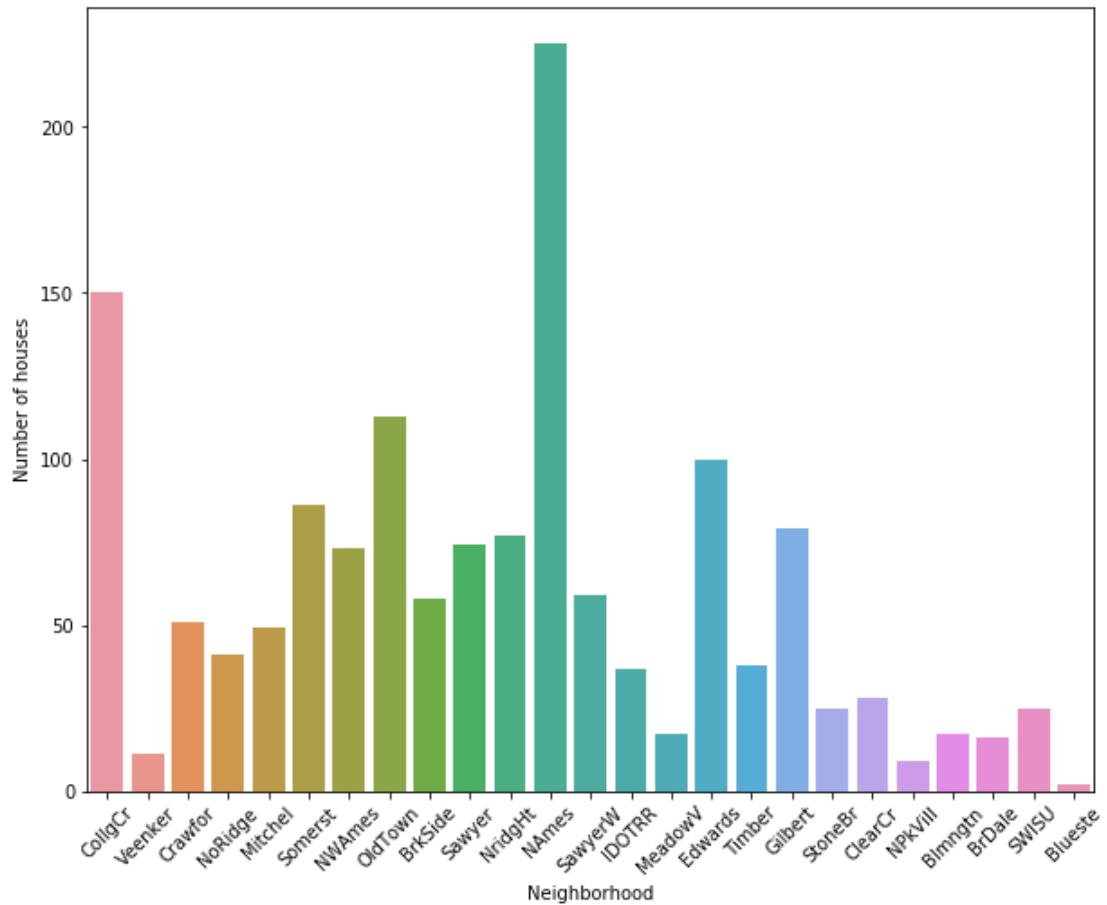
What interesting properties does Plot 2 reveal?

We see that a few houses which are quite old have been sold for a price greater than most of the new houses. These houses could be considered antique which attracts antique collectors.

Also, we see that houses built recently fetch a better price than old ones.

```
In [455]: # TODO: code to generate Plot 3
plt.figure(figsize=(10,8))
plt.suptitle("Bar plot of the number of houses in a Neighborhood.")
bplot=sns.countplot(x=train['Neighborhood'])
bplot.set_xlabel("Neighborhood")
bplot.set_ylabel("Number of houses")
bplot.set_xticklabels(bplot.get_xticklabels(),rotation=45);
```

Bar plot of the number of houses in a Neighborhood.

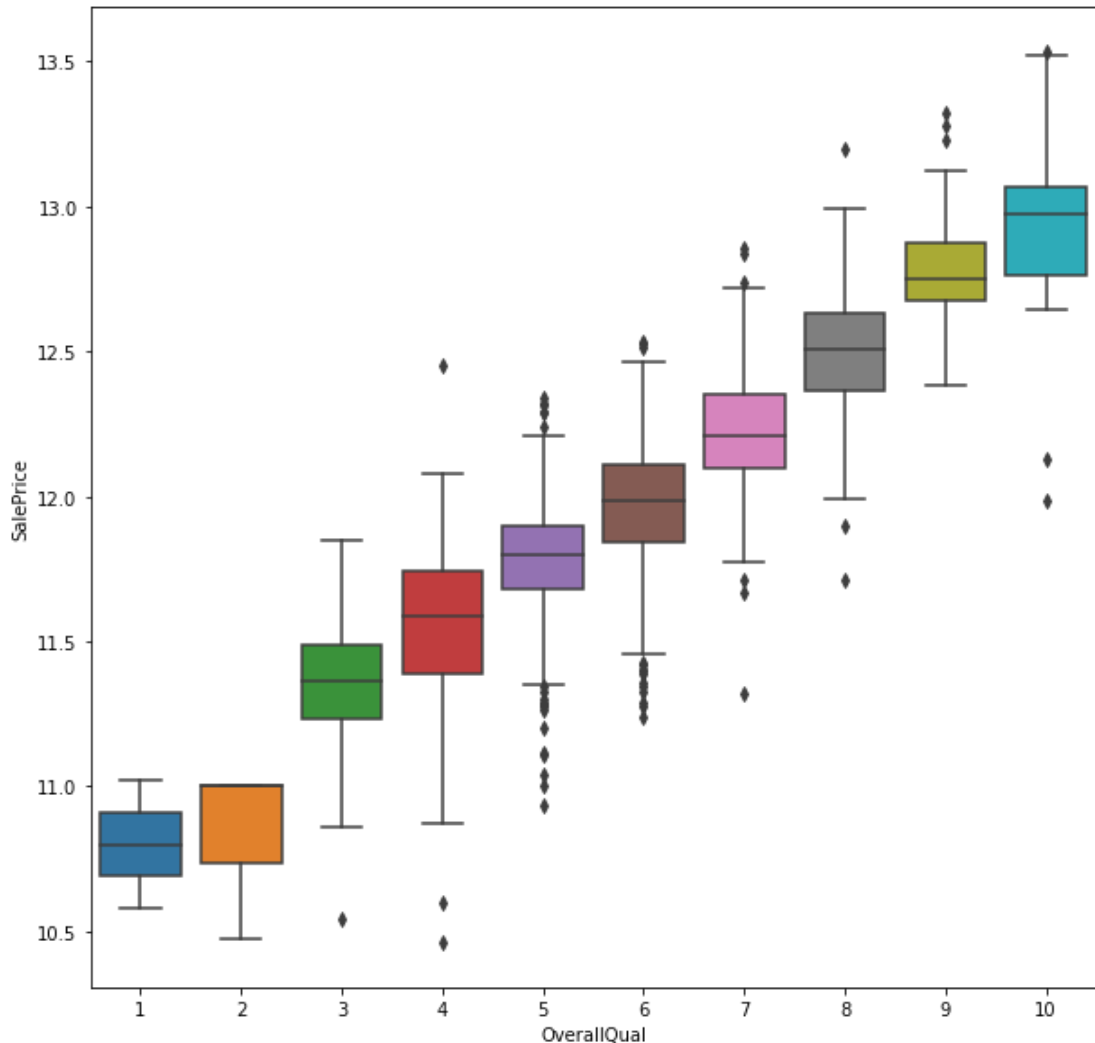


What interesting properties does Plot 3 reveal?

We see that most of the houses sold are from the neighborhoods NAmes and CollgCr. So these neighborhoods could be considered as the most preferred neighborhoods which can be further confirmed by analyzing the prices of the houses sold in these neighborhoods.

```
In [456]: # TODO: code to generate Plot 4
plt.figure(figsize=(10,10))
plt.suptitle("Distribution of Saleprice with respect to Overall Quality")
bxplot.set_xlabel("Overall Quality")
bxplot.set_ylabel("SalePrice")
bxplot=sns.boxplot(x=train['OverallQual'],y=train['SalePrice'])
```

Distribution of Saleprice with respect to Overall Quality

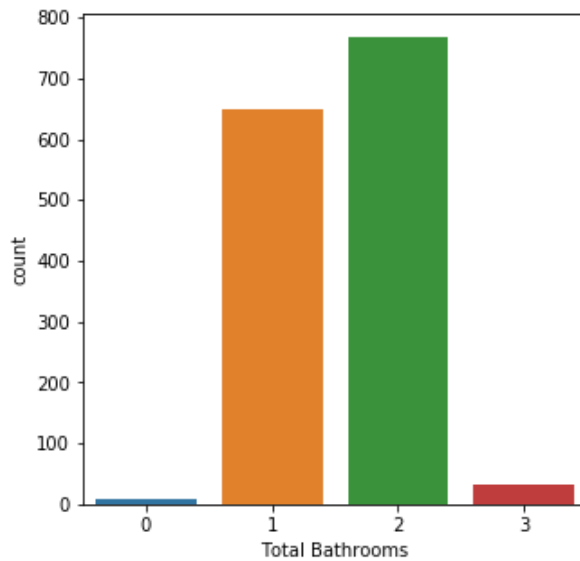


What interesting properties does Plot 4 reveal?

Since we see a high correlation between SalePrice and OverallQual, I plotted a boxplot between the 2 features on how the correlation can be improved by excluding the outliers. I see that the houses in the bins of OverallQual value 5 and 6 have a lot of outliers which might cause skewed model.

```
In [457]: # TODO: code to generate Plot 5
plt.figure(figsize=(5,5))
plt.suptitle("SalePrice Distributions with respect to Total Bathrooms")
cplot=sns.countplot(x=train['FullBath'])
cplot.set_xlabel("Total Bathrooms");
```

SalePrice Distributions with respect to Total Bathrooms



What interesting properties does Plot 5 reveal?

We see that most of the houses have only 1-2 baths. We can also see that houses with less than 1 bath are rarely purchased.

## Part 3 - Handcrafted Scoring Function

```
In [458]: # TODO: code for scoring function

# Chose certain columns which might have a considerable ontribution towards a big or a s
train['Desirability'] = (train['OverallQual']*train['OverallCond']*0.4)+(train['TotalAre
```

### Most Desirable Houses



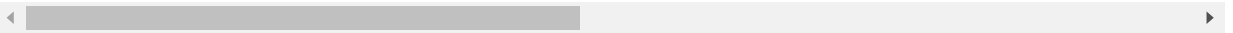
```
In [459]: desire=train.copy()

# Sorting in Descending order to get the houses with top scores.
desire.sort_values('Desirability',ascending=False).head(10)
```

Out[459]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	LotConfig	Lane
1298	60	RL	313.0	63887	Pave	None	IR3	Bnk	Corner	
523	60	RL	130.0	40094	Pave	None	IR1	Bnk	Inside	
1182	60	RL	160.0	15623	Pave	None	IR1	Lvl	Corner	
691	60	RL	104.0	21535	Pave	None	IR1	Lvl	Corner	
1169	60	RL	118.0	35760	Pave	None	IR1	Lvl	CulDSac	
496	20	RL	91.0	12692	Pave	None	IR1	Lvl	Inside	
185	75	RM	90.0	22950	Pave	None	IR2	Lvl	Inside	
440	20	RL	105.0	15431	Pave	None	Reg	Lvl	Inside	
1373	20	RL	91.0	11400	Pave	None	Reg	Lvl	Inside	
798	60	RL	104.0	13518	Pave	None	Reg	Lvl	Inside	

10 rows × 84 columns



### Least Desirable Houses

```
In [460]: # Sorting in Ascending order to get the houses with least scores.
desire.sort_values('Desirability',ascending=True).head(10)
```

Out[460]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	LotConfig	Lane
533	20	RL	50.0	5000	Pave	None	Reg	Low	Inside	
1218	50	RM	52.0	6240	Pave	None	Reg	Lvl	Inside	
1179	20	RL	77.0	8335	Pave	None	Reg	Lvl	Corner	
705	190	RM	70.0	5600	Pave	None	Reg	Lvl	Inside	
39	90	RL	65.0	6040	Pave	None	Reg	Lvl	Inside	
1321	20	RL	52.0	6627	Pave	None	IR1	Lvl	Corner	
375	30	RL	65.5	10020	Pave	None	IR1	Low	Inside	
636	30	RM	51.0	6120	Pave	None	Reg	Lvl	Inside	
1000	20	RL	74.0	10206	Pave	None	Reg	Lvl	Corner	
520	190	RL	60.0	10800	Pave	Grvl	Reg	Lvl	Inside	

10 rows × 84 columns



### Scoring mechanism

```
In [461]: # Printing the correlation of our handcrafted score with SalePrice.
print(train['Desirability'].corr(train['SalePrice']))
```

0.8777196210348981

I have considered the below columns to design the handcrafted scoring function, each contributing a

specific weight towards the final score.

Desirability

OverallQual

OverallCond

TotalArea

KitchenQual

KitchenAbvGr

TotalFlrSF

BsmtCond

BsmtQual

GarageQual

GarageCond

ExterCond

ExterQual

BsmtFullBath

BsmtHalfBath

FullBath

HalfBath

I see that the desirability score has worked really well from the fact that it is highly correlated with the SalePrice.

## Part 4 - Pairwise Distance Function

```
In [462]: # TODO: code for distance function
fie=[]

# Getting all the nominal columns in the List fie.
for i in train.columns:
    if train[i].dtype in ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']:
        if i != 'GarageYrBlt':
            fie.append(i)
result = train[fie]
# Normalizing all the values in result dataframe
for feature_name in fie:
    max_value = train[feature_name].max()
    min_value = train[feature_name].min()
    result[feature_name] = (result[feature_name]) / (max_value);
# Defining the pairwise distance function on the normalized data.
x=pd.DataFrame(euclidean_distances(result[fie]))
x=x.sort_values(0)
x.head(5)
```

C:\Users\abhin\Anaconda3\lib\site-packages\ipykernel\_launcher.py:14: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.  
Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy> (<http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy>)

Out[462]:

	0	1	2	3	4	5	6	7	8	9
0	0.000000	1.169521	0.390461	1.063859	0.608332	1.211198	0.888441	0.962524	1.258356	1.293519
1240	0.093875	1.180294	0.375153	1.058699	0.583711	1.224249	0.895021	0.959883	1.241548	1.307403
1366	0.109196	1.178395	0.363093	1.056381	0.596557	1.210036	0.895705	0.945665	1.255507	1.304849
1365	0.189511	1.172285	0.380542	1.048716	0.621436	1.206878	0.900362	0.965737	1.253003	1.305868
1410	0.216777	1.163302	0.358803	1.050292	0.632626	1.198753	0.894405	0.978452	1.251157	1.296838

5 rows × 1460 columns

```
In [463]: # Printing the top values with least pairwise distance values to compare SalePrice
idList =[0,1240,1366,1365,1410]
saleP ={}
for idx in idList:
    sp = salePrice['SalePrice'].iloc[idx]
    saleP[idx]=sp
print(saleP)
```

```
{0: 208500, 1240: 224900, 1366: 193000, 1365: 216000, 1410: 230000}
```

How well does the distance function work? When does it do well/badly?

We see that the sale price of the top 5 houses with least pairwise distances are very close to each other.  
The prices of these houses are very close to each other.

```
In [464]: x=x.sort_values(0,ascending=False)
x.head(5)
```

Out[464]:

	0	1	2	3	4	5	6	7	8	9
<b>1298</b>	2.844171	3.049677	2.748983	3.008864	2.548135	3.226953	2.634346	2.634045	3.044550	3.150012
<b>533</b>	2.329332	2.281517	2.349162	2.291114	2.600159	2.169484	2.470954	2.533139	2.097214	1.998404
<b>705</b>	2.111934	2.152924	2.153858	2.163415	2.341247	2.129105	2.328278	2.311242	1.843594	1.941259
<b>1179</b>	2.069543	2.004601	2.049785	1.955888	2.262288	2.037524	2.145889	2.138778	1.718853	1.709148
<b>1218</b>	2.040463	2.037988	2.065156	2.061929	2.305469	1.971886	2.201620	2.283581	1.870541	1.781986

5 rows × 1460 columns

```
In [465]: # Printing the top values with highest pairwise distance values to compare SalePrice
idList =[0,1298,533,705,1179,1218]
saleP ={}
for idx in idList:
    sp = salePrice['SalePrice'].iloc[idx]
    saleP[idx]=sp
print(saleP)
```

```
{0: 208500, 1298: 160000, 533: 39300, 705: 55000, 1179: 93000, 1218: 80500}
```

How well does the distance function work? When does it do well/badly?

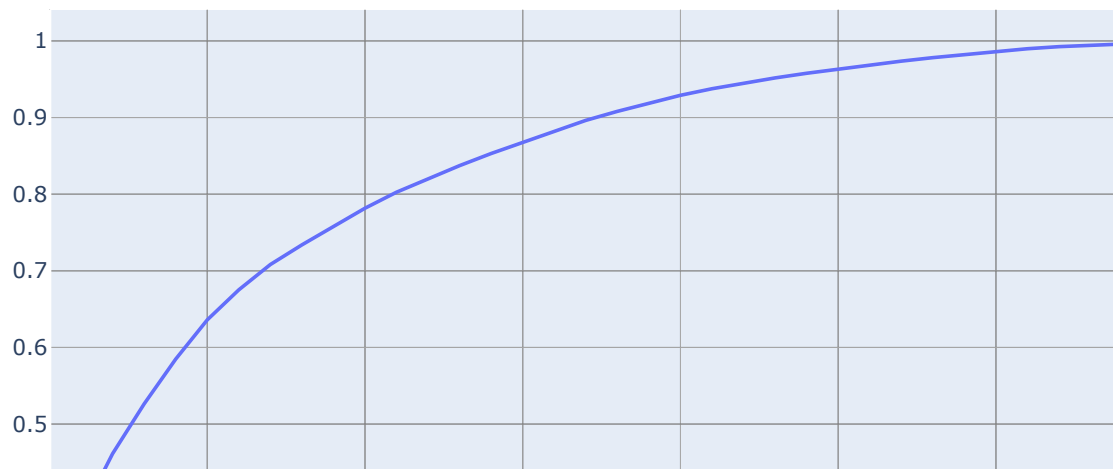
We see that the sale price of the top 5 houses with highest pairwise distances are very different to each other. There is a huge difference between their prices.

After making the above observations, I feel that the handcrafted pairwise distance works really well.

## Part 5 - Clustering

```
In [466]: # TODO: code for clustering and visualization
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
def configure_plotly_browser_state():
    import IPython
    display(IPython.core.display.HTML('''
        <script src="/static/components/requirejs/require.js"></script>
        <script>
            requirejs.config({
                paths: {
                    base: '/static/base',
                    plotly: 'https://cdn.plot.ly/plotly-1.5.1.min.js?noext',
                },
            });
        </script>
        '''))
pca = PCA(n_components=48).fit(result)
expvar=np.cumsum(pca.explained_variance_ratio_)
configure_plotly_browser_state()
init_notebook_mode(connected=True)
data = [go.Scatter(y=expvar)]
layout = {'title': 'Review PCA Explained Variance to determine number of components'}
iplot({'data':data,'layout':layout})
pca = PCA(n_components=30)
XPCA = pca.fit_transform(result)
numberClusters=8
kmeans = KMeans(n_clusters=numberClusters, random_state=69)
Xkmeans = kmeans.fit_predict(XPCA)
XTSNE = TSNE(n_components=2).fit_transform(XPCA)
y = result.SalePrice.reset_index(drop=True)
```

### Review PCA Explained Variance to determine number of components



```
In [652]: (NE),pd.DataFrame(Xkmeans),pd.DataFrame(y),pd.DataFrame(np.exp1(y)),pd.DataFrame()),axis=
', 'logprice', 'price' ]

n='cluster',color_continuous_scale=px.colors.qualitative.Plotly,title="TSNE visualization
```

### TSNE visualization of House Clusters



#### Analysis of clusters

We have clustered houses based on the our distance scoring function and now we can compare the houses in the same cluster based on their Neighborhood. House # 4,11,20,35,58 and others in the list belongs to same cluster (number 2) and have Neighborhood NridgHt.

House # 13,22,25,32,45 and others in the list belongs to same cluster (number 4) and have Neighborhood CollgCr.

## Part 6 - Linear Regression



Everytime I run the above model, which takes 2 of the nominal features and runs the regression model, it prints the combination of features that gives the best RMSE score.

After running it multiple times, I see that the most repeated features are as below.

TotalArea

GrLivArea

TotalFlrSF

OverallQual

YearRemodAdd

KitchenQual

I would categorize the above features as the most important once.

But when we discuss about the model itself, using the simple linear regression model on one or few columns gives a very bad prediction since:

- 1) The dataset is small. Only 1460 rows.
- 2) The number of features used to predict are less(2 or less)

## Part 7 - External Dataset

**External Dataset:** <https://www.cityofames.org/government/departments-divisions-a-h/city-assessor/reports>  
(<https://www.cityofames.org/government/departments-divisions-a-h/city-assessor/reports>)

### About the data:

About Dataset: This dataset contains house data IOWA state with features similar to our data, but these houses are built in recent years(2016-2019) with the latest year being 2019. Now with the help of this dataset we can see the trend of the Sale Price in recent years.

The data was extensive with around 22000 records but since most of the records did not have the saleprice, we were only able to include around 3000 records. I have combined the external dataset with the original one and trying to see how a prediction on this set varies.



```

In [506]: # TODO: code to import external dataset and test
extData = pd.read_csv("extData.csv")
extCols = list(extData.columns)
def mapCols(columns,df,mapDict):
    df[columns] = df[columns].map(mapDict)
    return df
buildTypeMap = {}
buildTypeMap['1FamDet'] = '1Fam'
buildTypeMap['2FmConv'] = '2FmCon'
buildTypeMap['Duplex'] = 'Duplx'
buildTypeMap['Twnhs-E'] = 'TwnhsE'
buildTypeMap['Twnhs-I'] = 'TwnhsI'

houseStyleMap = {}
houseStyleMap['1-Story'] = '1Story'
houseStyleMap['1.5 Fin'] = '1.5Fin'
houseStyleMap['1.5 Unf'] = '1.5Unf'
houseStyleMap['2-Story'] = '2Story'
houseStyleMap['2.5 Fin'] = '2.5Fin'
houseStyleMap['2.5 Unf'] = '2.5Unf'
houseStyleMap['S/Foyer'] = 'SFoyer'
houseStyleMap['S/Level'] = 'SLvl'

garageTypeMap = {}
garageTypeMap['2 Types'] = '2Types'
garageTypeMap['Attachd'] = 'Attchd'
garageTypeMap['Basment'] = 'Basment'
garageTypeMap['Detached'] = 'Detchd'
garageTypeMap['None'] = 'NA'

saleTypeMap = {}
saleTypeMap['WRDCash'] = 'CWD'
saleTypeMap['WRDConv'] = 'VWD'
saleTypeMap['NewSold'] = 'New'
saleTypeMap['COD/Est'] = 'COD'
saleTypeMap['CtrLwDn'] = 'ConLw'
saleTypeMap['CtrLInt'] = 'ConLI'
saleTypeMap['CtrLD&I'] = 'ConLD'
saleTypeMap['Other'] = 'Oth'

extData = mapCols('BldgType',extData,buildTypeMap)
extData = mapCols('HouseStyle',extData,houseStyleMap)
extData = mapCols('GarageType',extData,garageTypeMap)
extData = mapCols('SaleType',extData,saleTypeMap)

extData['GarageType'] = extData['GarageType'].fillna('None')
extData['HouseStyle'] = extData['HouseStyle'].fillna('None')
extData['SaleType'] = extData['SaleType'].fillna('None')
extData['BldgType'] = extData['BldgType'].fillna('None')
extData['SalePrice'] = np.log(extData['SalePrice'])
extData = extData.fillna(0)

```

In [507]:

extData.head(10)

Out[507]:

	LotArea	BldgType	YearBuilt	HouseStyle	RoofMatl	Exterior1st	Exterior2nd	MasVnrType	Heating	Cer
0	10015	1Fam	2016	2Story	CompShg	VinylSd	VinylSd	BrkFace	GasA	
1	10015	1Fam	2016	2Story	CompShg	VinylSd	VinylSd	BrkFace	GasA	
2	14904	1Fam	2018	2Story	CompShg	VinylSd	VinylSd	Stone	GasA	
3	14904	1Fam	2018	2Story	CompShg	VinylSd	VinylSd	Stone	GasA	
4	13915	1Fam	2017	1Story	CompShg	CemntBd	CmentBd	Stone	GasFWA	
5	13915	1Fam	2017	1Story	CompShg	CemntBd	CmentBd	Stone	GasFWA	
6	9200	1Fam	2018	1Story	CompShg	VinylSd	VinylSd	Stone	GasFWA	
7	9200	1Fam	2018	1Story	CompShg	VinylSd	VinylSd	Stone	GasFWA	
8	9200	1Fam	2018	1Story	CompShg	VinylSd	VinylSd	Stone	GasFWA	
9	7153	TwnhsE	2017	1Story	CompShg	VinylSd	VinylSd	Stone	GasFWA	

10 rows × 24 columns

```

In [650]: originaldf = pd.read_csv("train.csv")
originaldf.drop("Id", axis = 1, inplace = True)
qualMap = {'Ex':5, 'Gd':4, 'TA':3, 'Fa':2, 'Po':1, 'NA':0}
for c in ['HeatingQC', 'KitchenQual', 'BsmtQual', 'ExterQual', 'ExterCond', 'BsmtCond', 'Gar
originaldf[c]=originaldf[c].fillna("NA")
originaldf[c]=originaldf[c].replace(qualMap)
originaldf[c]=originaldf[c].astype(int)
fMap = {'CBLOCK':5, 'PConc':4, 'BrkTil':3, 'Slab':2, 'Stone':1, 'Wood':0}
originaldf['Foundation']=originaldf['Foundation'].fillna("NA")
originaldf['Foundation']=originaldf['Foundation'].replace(fMap)
originaldf['Foundation']=originaldf['Foundation'].astype(int)
for col in ['MiscFeature', 'Alley', 'Fence', 'FireplaceQu', 'GarageType', 'GarageFinish', 'Mas
originaldf[col]=originaldf[col].fillna('None')
for col in ['GarageArea', 'GarageCars', 'MasVnrArea']:
originaldf[col]=originaldf[col].fillna(0)
for col in ['Electrical', 'MSZoning', 'KitchenQual', 'Exterior1st', 'Exterior2nd', 'SaleType'
originaldf[col]=originaldf[col].fillna(train[col].mode()[0])
originaldf['LotFrontage']=originaldf.groupby('Neighborhood')['LotFrontage'].transform(la
originaldf.drop(['Utilities'], axis=1, inplace=True)
for col in ['MoSold', 'MSSubClass', 'YrSold', 'OverallCond']:
originaldf[col]=originaldf[col].astype(str)
originaldf['BsmtGrade'] = originaldf['BsmtQual'] * originaldf['BsmtCond']
originaldf['TotalArea'] = originaldf['TotalBsmtSF'] + originaldf['GrLivArea']
originaldf['TotalFlrSF'] = originaldf['1stFlrSF'] + originaldf['2ndFlrSF']
originaldf['OverallCond']=originaldf['OverallCond'].astype(int)

totalData = pd.concat([originaldf, extData], axis=0)
nonNumericExtList = totalData.select_dtypes([object]).columns.tolist()
fie=[]
for i in extCols:
    if i not in nonNumericExtList and i != 'GarageYrBlt':
        fie.append(i)

for i in fie:
    totalData[i].fillna('0')
totalData=totalData.drop(['GarageYrBlt'], axis=1)

```

C:\Users\abhin\Anaconda3\lib\site-packages\ipykernel\_launcher.py:27: FutureWarning:

Sorting because non-concatenation axis is not aligned. A future version of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

```

In [649]: trainingSet, testSet = train_test_split(totalData[fie], test_size=0.2)
XLabel = (np.log(trainingSet['SalePrice']))
Xtrain = trainingSet.drop('SalePrice', axis =1)
Ytest = testSet.drop('SalePrice', axis =1)
lr = LinearRegression()
lr.fit(Xtrain, XLabel)
#predict on test data split
pred = lr.predict(Ytest)
score = mean_squared_error(np.log(testSet['SalePrice']), (pred))
score = np.sqrt(score)
print(score)

```

4.550419373988922

Describe the dataset and whether this data helps with prediction.

I have run the linear regression model on nearly 4000 records and we have achieved an RMSE score of 4.55 which is not that great but can be improved by improving the data cleaning.

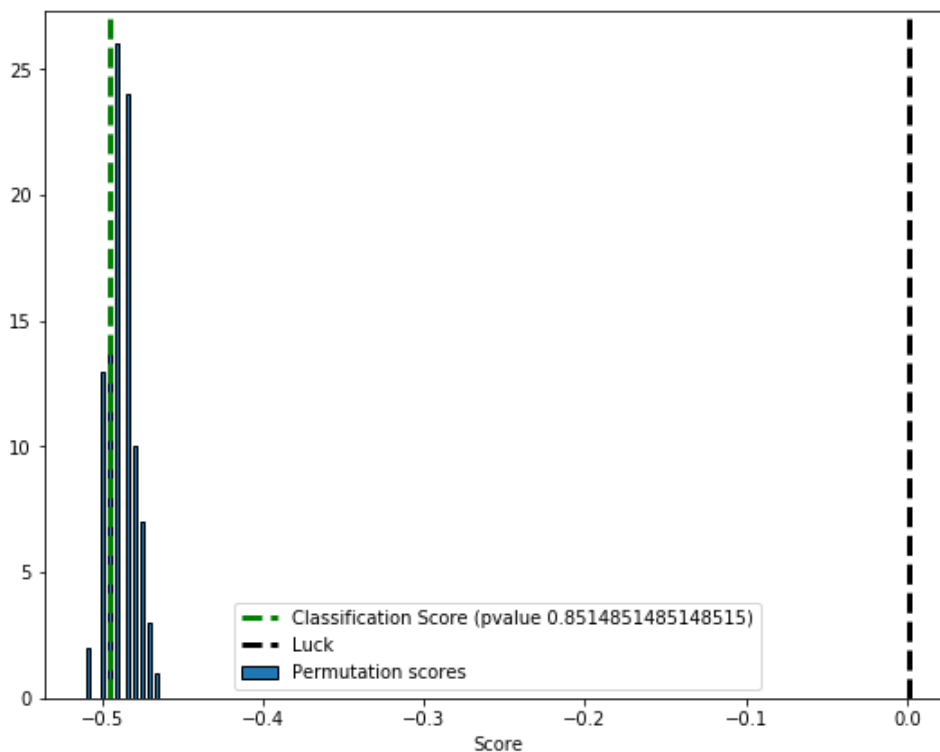
## Part 8 - Permutation Test

```

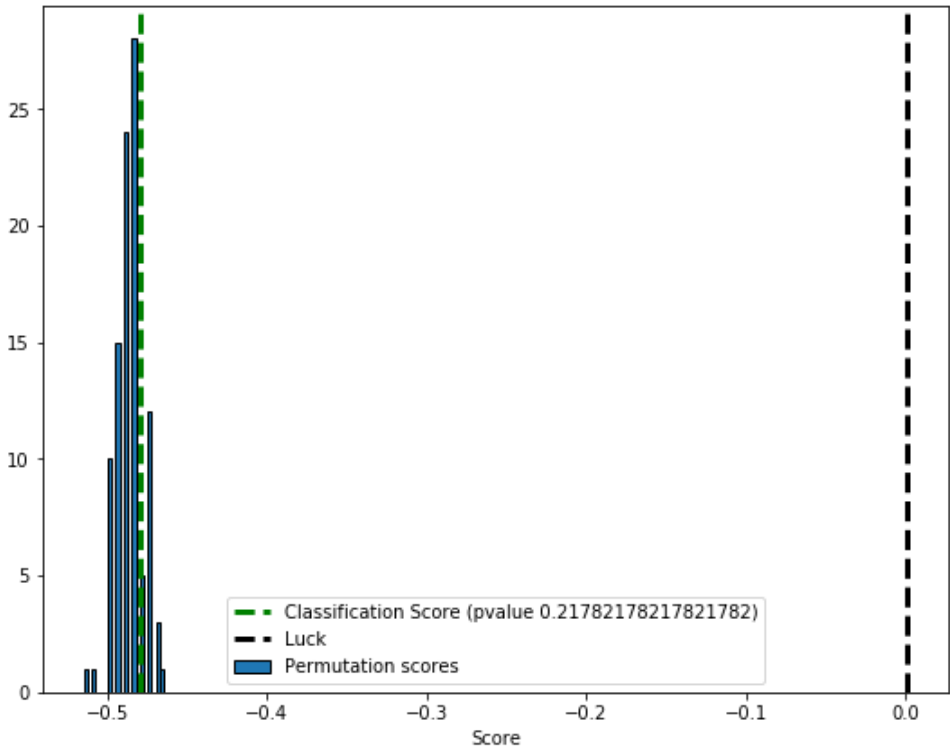
In [471]: # TODO: code for all permutation tests
def permutationList(col):
    X = train[col]
    y = train['SalePrice']
    n_classes = np.unique(y).size
    random = np.random.RandomState(seed=77)
    E = random.normal(size=(len(X), 2200))
    X = np.c_[X, E]
    rgr = LinearRegression()
    cv = KFold(2)
    def rmse_self(y_true, y_pred):
        e = sqrt(mean_squared_error(y_true, y_pred))
        rmse = round(e, 2)
        return rmse
    score = make_scorer(rmse_self, greater_is_better=False)
    score, permutation_scores, pvalue = permutation_test_score(rgr, X, y, scoring=score)
    plt.figure(figsize=(9,7))
    plt.hist(permutation_scores, 20, label='Permutation scores', edgecolor='black')
    ylim = plt.ylim()
    plt.plot(2 * [score], ylim, '--g', linewidth=3, label='Classification Score (pvalue)')
    plt.plot(2 * [1. / n_classes], ylim, '--k', linewidth=3, label='Luck')
    plt.ylim(ylim)
    plt.legend()
    plt.xlabel('Score')
    plt.suptitle(f"{col} Vs SalePrice Permutation score")
    plt.show();
    fiel=['LotArea', 'BsmtUnfSF', 'LotFrontage', 'BsmtUnfSF', 'BsmtFinSF1', 'OverallQual', 'Garage']
    for cl in fiel:
        permutationList(cl)

```

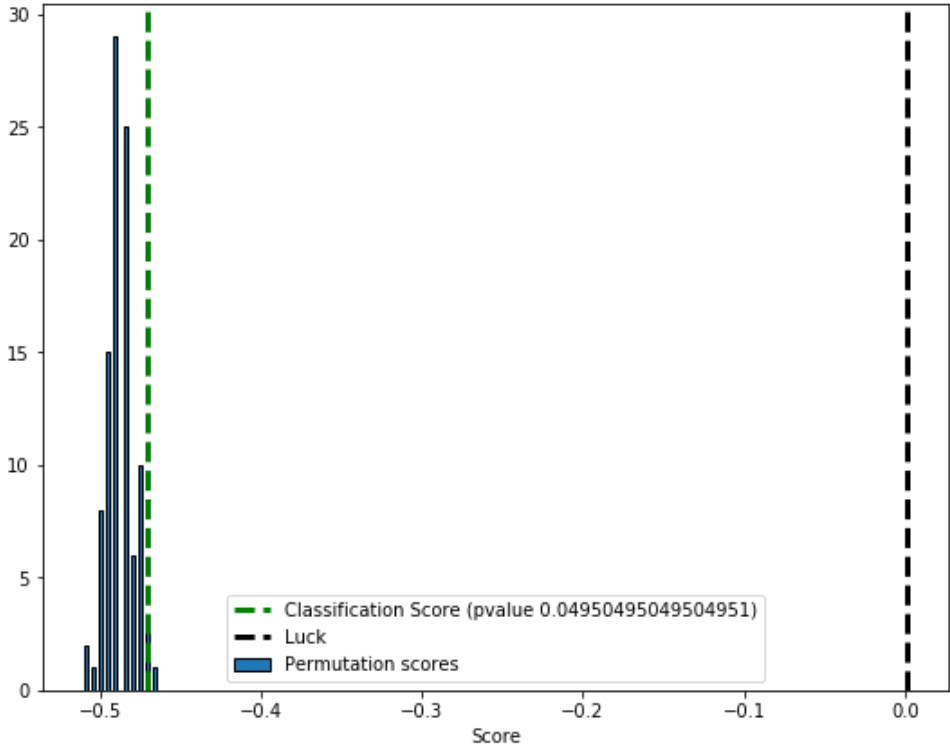
LotArea Vs SalePrice Permutation score



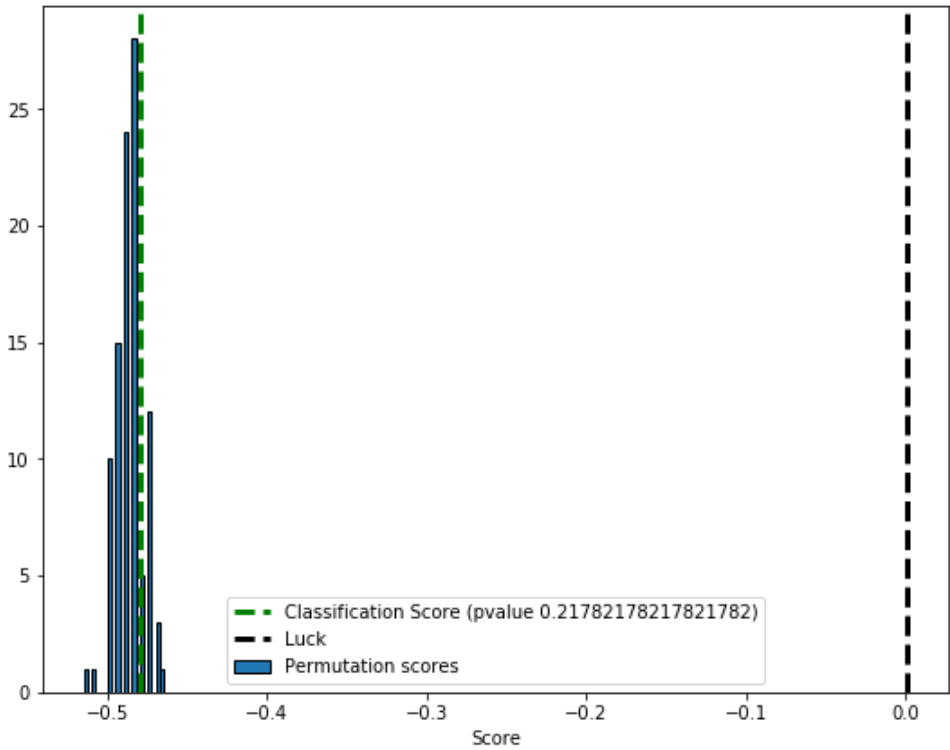
BsmtUnfSF Vs SalePrice Permutation score



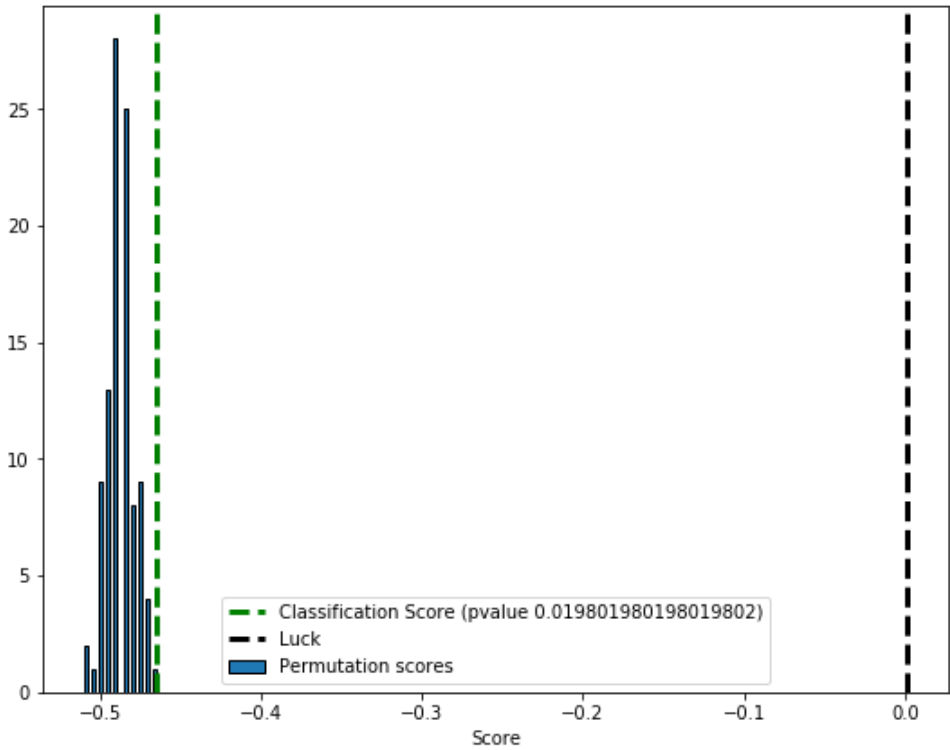
LotFrontage Vs SalePrice Permutation score



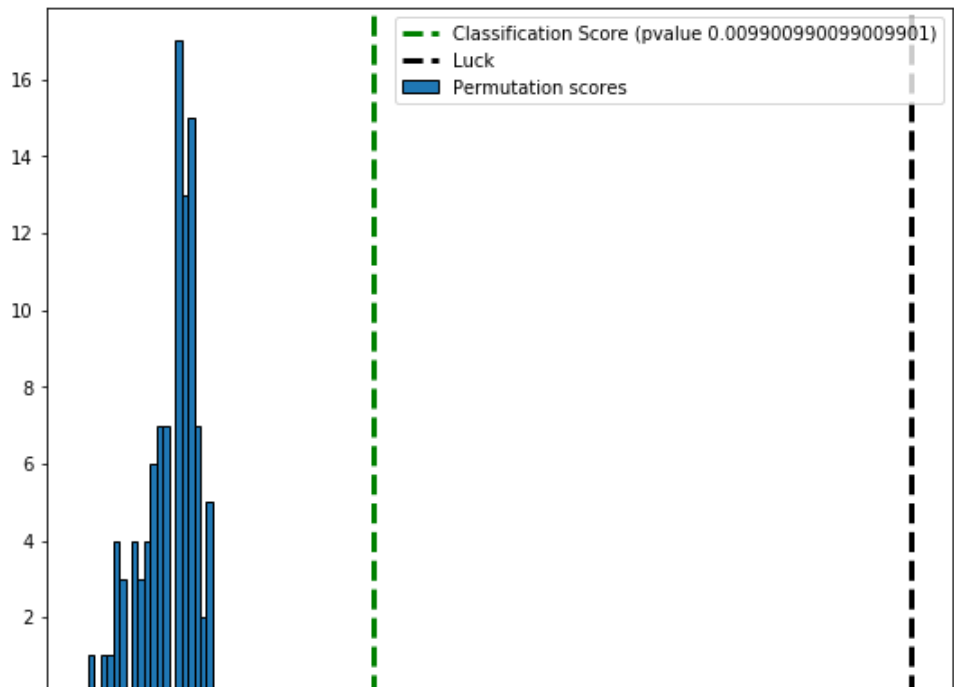
BsmtUnfSF Vs SalePrice Permutation score



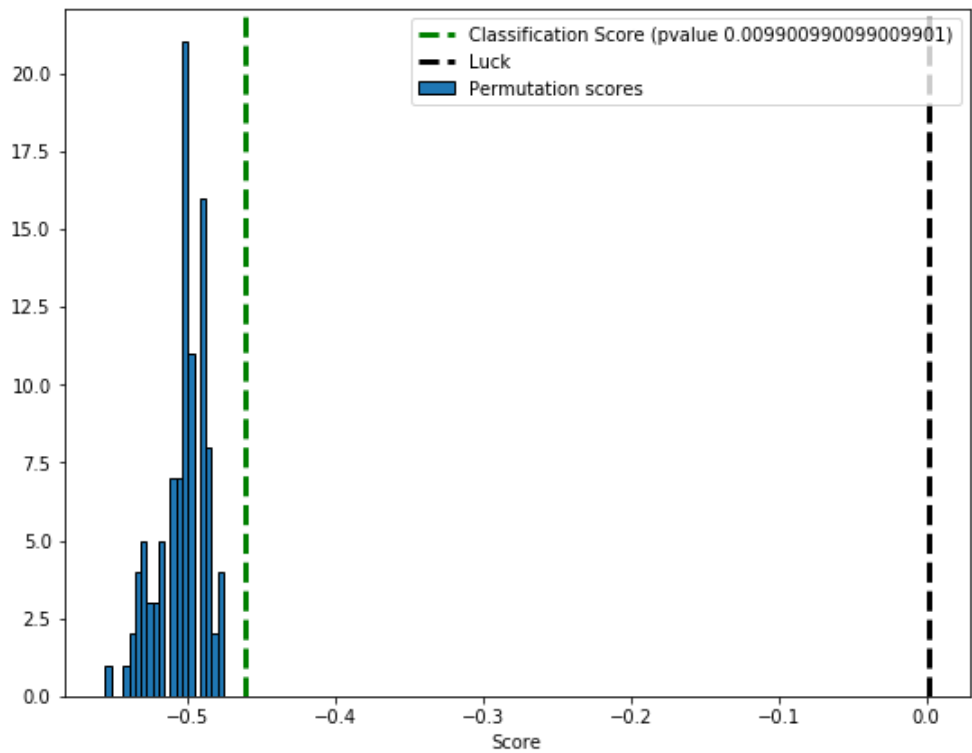
BsmtFinSF1 Vs SalePrice Permutation score



OverallQual Vs SalePrice Permutation score

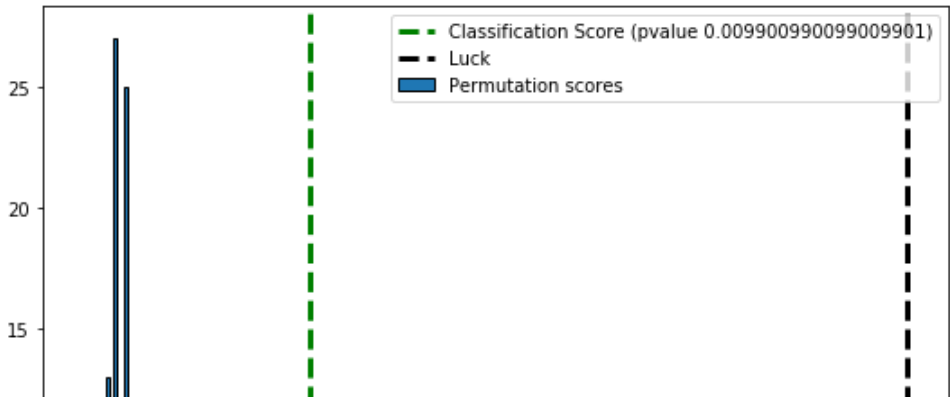


GarageCars Vs SalePrice Permutation score

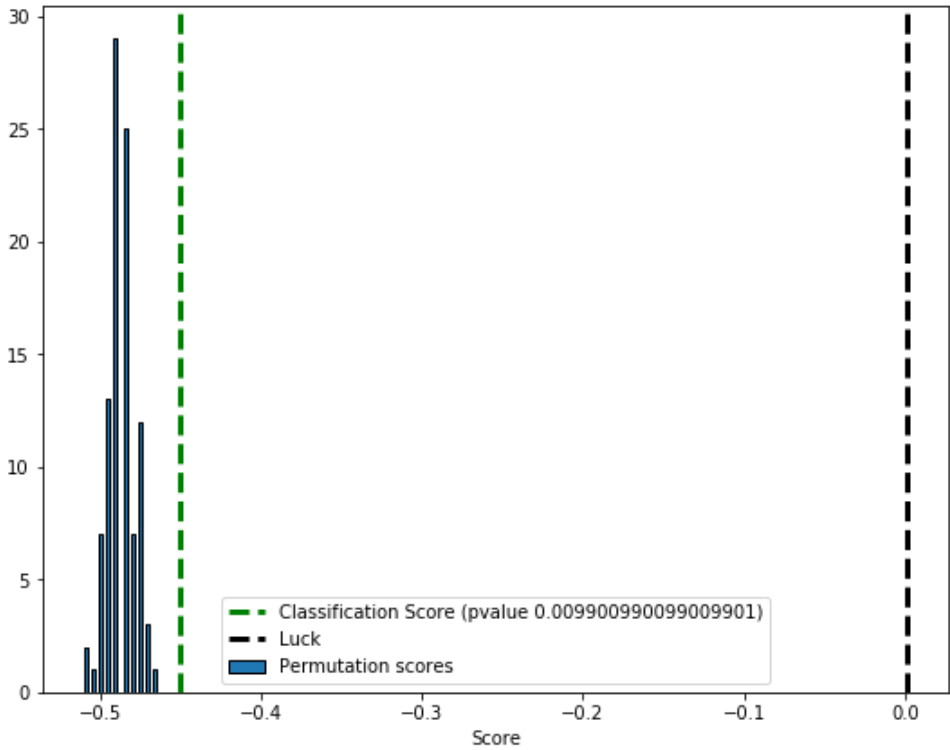




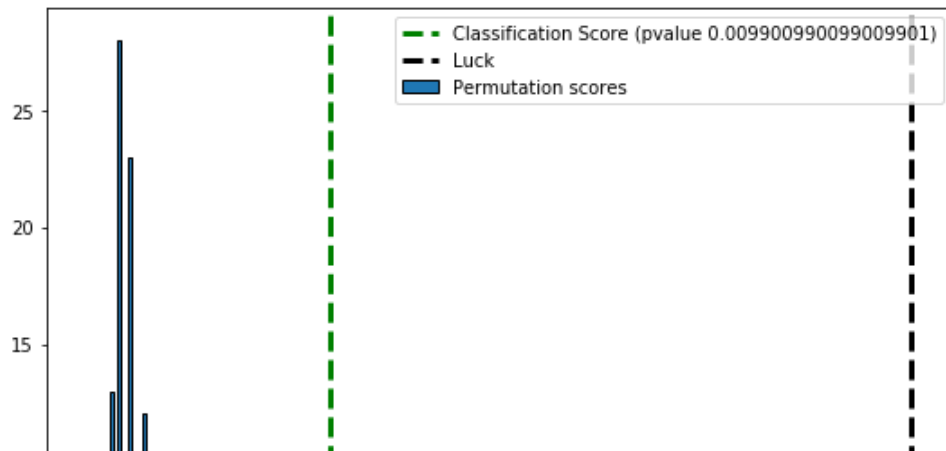
GarageArea Vs SalePrice Permutation score



MasVnrArea Vs SalePrice Permutation score



## GrLivArea Vs SalePrice Permutation score



Describe the results.

The below features give a very bad permutation test scores, which means that even if they gave a good correlation value in the heatmap in part-1, that was a coincidence and does not describe a genuine relationship between the feature and SalePrice.

['LotArea', 'BsmtUnfSF', 'LotFrontage', 'BsmtUnfSF', 'BsmtFinSF1']

The below columns give a good permutation test score which means that they are correlated well with SalePrice and their correlation values in Part-1 describe a genuine relationship between the feature and SalePrice.

['OverallQual', 'GarageCars', 'GarageArea', 'MasVnrArea', 'GrLivArea']

## Part 9 - Final Result

```

In [472]: train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
train_Id = train['Id']
test_Id = test['Id']
train.drop("Id", axis = 1, inplace = True)
test.drop("Id", axis = 1, inplace = True)
qualMap = {'Ex':5,'Gd':4,'TA':3,'Fa':2,'Po':1,'NA':0}
for c in ['HeatingQC', 'KitchenQual', 'BsmtQual', 'ExterQual', 'ExterCond', 'BsmtCond', 'Gar
    train[c]=train[c].fillna("NA")
    train[c]=train[c].replace(qualMap)
    train[c]=train[c].astype(int)
fMap = {'CBlock':5,'PConc':4,'BrkTil':3,'Slab':2,'Stone':1,'Wood':0}
train['Foundation']=train['Foundation'].fillna("NA")
train['Foundation']=train['Foundation'].replace(fMap)
train['Foundation']=train['Foundation'].astype(int)
for col in ['MiscFeature', 'Alley', 'Fence', 'FireplaceQu', 'GarageType', 'GarageFinish', 'Mas
    train[col]=train[col].fillna('None')
for col in ['GarageArea', 'GarageCars', 'MasVnrArea']:
    train[col]=train[col].fillna(0)
for col in ['Electrical', 'MSZoning', 'KitchenQual', 'Exterior1st', 'Exterior2nd', 'SaleType'
    train[col]=train[col].fillna(train[col].mode()[0])
train['LotFrontage']=train.groupby('Neighborhood')['LotFrontage'].transform(lambda x: x.
train.drop(['Utilities'],axis=1,inplace=True)
for col in ['MoSold', 'MSSubClass', 'YrSold', 'OverallCond']:
    train[col]=train[col].astype(str)
train['BsmtGrade'] = train['BsmtQual'] * train['BsmtCond']
train['TotalArea'] = train['TotalBsmtSF'] + train['GrLivArea']
train['TotalFlrSF'] = train['1stFlrSF'] + train['2ndFlrSF']
train['TotalBath'] = train['BsmtFullBath'] + 0.5 * train['BsmtHalfBath'] + train['FullBa
train['OverallCond']=train['OverallCond'].astype(int)

qualMap = {'Ex':5,'Gd':4,'TA':3,'Fa':2,'Po':1,'NA':0}
for c in ['HeatingQC', 'KitchenQual', 'BsmtQual', 'ExterQual', 'ExterCond', 'BsmtCond', 'Gar
    test[c]=test[c].fillna("NA")
    test[c]=test[c].replace(qualMap)
    test[c]=test[c].astype(int)
fMap = {'CBlock':5,'PConc':4,'BrkTil':3,'Slab':2,'Stone':1,'Wood':0}
test['Foundation']=test['Foundation'].fillna("NA")
test['Foundation']=test['Foundation'].replace(fMap)
test['Foundation']=test['Foundation'].astype(int)
for col in ['MiscFeature', 'Alley', 'Fence', 'FireplaceQu', 'GarageType', 'GarageFinish', 'Mas
    test[col]=test[col].fillna('None')
for col in ['GarageArea', 'GarageCars', 'MasVnrArea']:
    test[col]=test[col].fillna(0)
for col in ['Electrical', 'MSZoning', 'KitchenQual', 'Exterior1st', 'Exterior2nd', 'SaleType'
    test[col]=test[col].fillna(test[col].mode()[0])
test['LotFrontage']=test.groupby('Neighborhood')['LotFrontage'].transform(lambda x: x.fi
test.drop(['Utilities'],axis=1,inplace=True)
for col in ['MoSold', 'MSSubClass', 'YrSold', 'OverallCond']:
    test[col]=test[col].astype(str)
test['BsmtGrade'] = test['BsmtQual'] * test['BsmtCond']
test['TotalArea'] = test['TotalBsmtSF'] + test['GrLivArea']
test['TotalFlrSF'] = test['1stFlrSF'] + test['2ndFlrSF']
test['TotalBath'] = test['BsmtFullBath'] + 0.5 * test['BsmtHalfBath'] + test['FullBath']

```

```
In [473]: f12=[]
for i in test.columns:
    if test[i].dtype in ['object']:
        f12.append(i)
X_train, X_test, y_train, y_test = train_test_split(train.drop(["SalePrice"], axis = 1),
clf = CatBoostRegressor(iterations=12000,task_type="GPU",devices='0:1',cat_features=f12)
clf.fit(X_train,y_train,verbose=False)
preds=clf.predict(X_test)
new_preds=clf.predict(test)
output=pd.DataFrame({'Id': test_Id, 'SalePrice': new_preds[:]}))
output.to_csv('submission5.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/abhinaymadunanthu> (<https://www.kaggle.com/abhinaymadunanthu>)

Highest Rank: 1362

Score: 0.12423

Number of entries: 15

1357	Super Stars		0.12411	14	2d
1358	Harsh Chandnani		0.12415	7	6h
1359	Ekaterina Chunosova		0.12417	11	9d
1360	wyhhyw		0.12418	3	2mo
1361	Jaysen Shi		0.12419	2	6d
1362	Abhinay Reddy Madunanthu		0.12423	15	-10s
<b>Your Best Entry</b>  <p>You advanced 1,098 places on the leaderboard!</p> <p>Your submission scored 0.12423, which is an improvement of your previous score of 0.13846. Great job!</p>  <a href="#">Tweet this!</a>					
1363	R Shimazaki		0.12424	3	1mo