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 bc to answer the questions. We also ask that code be commented to make it easier to follow.

Double-click (or enter) to edit

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import scipy as sc
import seaborn as sb
from scipy.spatial.distance import squareform
from scipy.spatial.distance import pdist
from sklearn import preprocessing
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?clier

Enter your authorization code:
··········
Mounted at /content/drive
```

```
train = pd.read_csv('/content/drive/My Drive/house-prices-advanced-regression-techniqu
test = pd.read_csv('/content/drive/My Drive/house-prices-advanced-regression-technique
train.shape
```

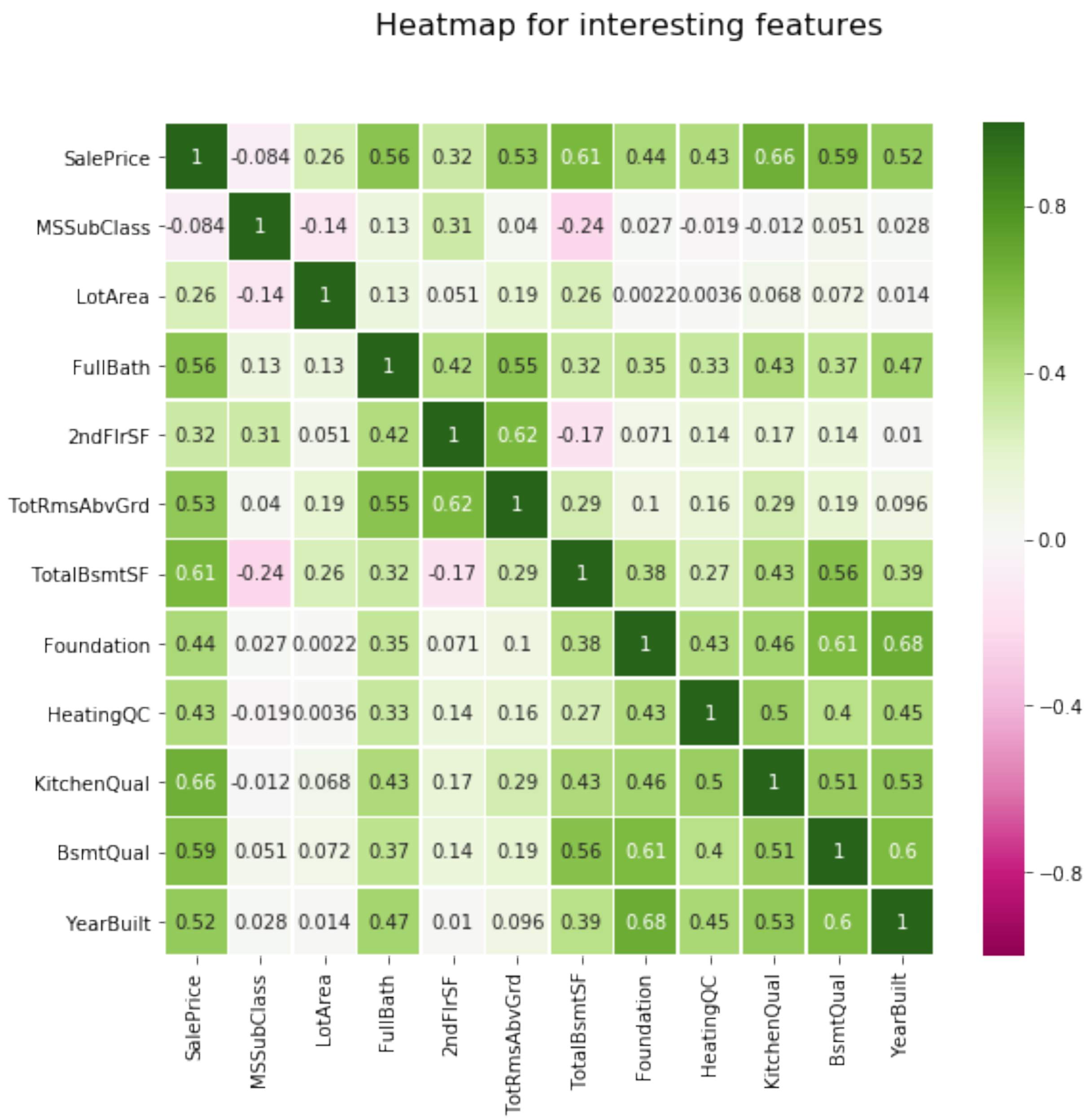
```
(1460, 81)
```

```
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
```

Part 1 - Pairwise Correlations

```
pd.set_option('mode.chained_assignment', None)

plt.figure(figsize=(9,8))
plt.suptitle('Heatmap for interesting features', fontsize=16)

Interestingcolumns = ['SalePrice', 'MSSubClass','LotArea', 'FullBath',
                      '2ndFlrSF','TotRmsAbvGrd', 'TotalBsmtSF', 'Foundation',
                      'HeatingQC', 'KitchenQual', 'BsmtQual','YearBuilt']
```

```python
QualityMapping = {'NA' : 0, 'Po' : 1, 'Fa' : 2,'TA' : 3,'Gd' : 4, 'Ex': 5}
FoundationMapping = {'Wood' : 0, 'Stone' : 1, 'Slab' : 2, 'BrkTil' : 3,
                     'CBlock' : 4, 'PConc': 5}

q1_train = train[Interestingcolumns]
q1_train['Foundation'].fillna('NA', inplace=True)
q1_train['Foundation']=q1_train['Foundation'].apply(lambda x: FoundationMapping[x])

for col in ['HeatingQC', 'KitchenQual', 'BsmtQual']:
    q1_train[col].fillna('NA', inplace=True)
    q1_train[col] = q1_train[col].apply(lambda x: QualityMapping[x])

corr = pd.DataFrame()
for a in Interestingcolumns:
    for b in Interestingcolumns:
        corr.loc[a, b] = q1_train.corr().loc[a, b]

heatmap = sb.heatmap(corr, cmap="PiYG", annot=True, linewidths=.8, vmin=-1, vmax=1)

plt.show()
```



The most positive correlations are between

a) YearBuilt and Foundation type : **0.68**

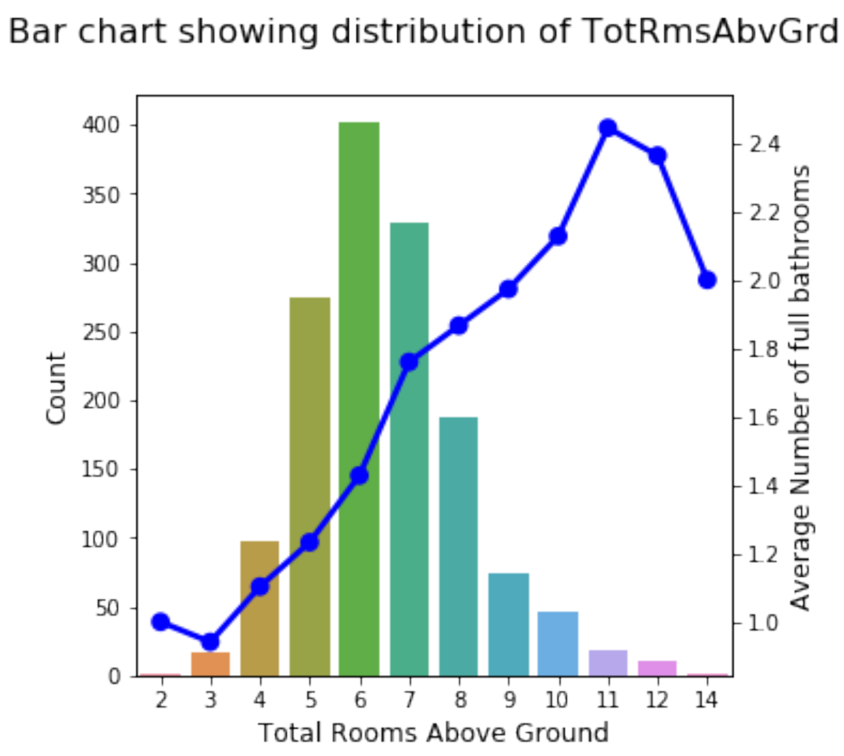
b) KitchenQuality and Sales Price : **0.66**

The least correlation is between MSSubClass (the type of dwelling)

and TotalBsmtSF (Total square feet of basement area) : **-0.24**

Part 2 - Informative Plots

```
plt.figure(figsize=(5,5))
plt.suptitle('Bar chart showing distribution of TotRmsAbvGrd', fontsize=16)
q2a = train.loc[:, ['TotRmsAbvGrd', 'SalePrice', 'FullBath']]
totalRooms = []
roomMean = []
bathsMean = []

for rooms in q2a.TotRmsAbvGrd.unique():
    q2a_ = q2a[q2a['TotRmsAbvGrd'] == rooms]
    totalRooms.append(rooms)
    roomMean.append(q2a_['SalePrice'].mean())
    bathsMean.append(q2a_['FullBath'].mean())

q2a_ = pd.DataFrame()
q2a_['rooms']  = totalRooms
q2a_['roomMean']  = roomMean
q2a_['bathMean'] = bathsMean

cp1 = sb.countplot(train.TotRmsAbvGrd)
lp1 = cp1.twinx()
lp1 = sb.pointplot(y="bathMean", x="rooms", data=q2a_, color="blue")
cp1.set_xlabel("Total Rooms Above Ground", fontsize=12)
cp1.set_ylabel("Count", fontsize=12)
lp1.set_ylabel("Average Number of full bathrooms", fontsize=12)
plt.show()
```



- Most of the houses have 3 bedrooms. Therefore, it is expected that most of them

will have at least 6 rooms counting the kitchen, living room and dining room.

Please note that the bathrooms were not taken in consideration for this variable.

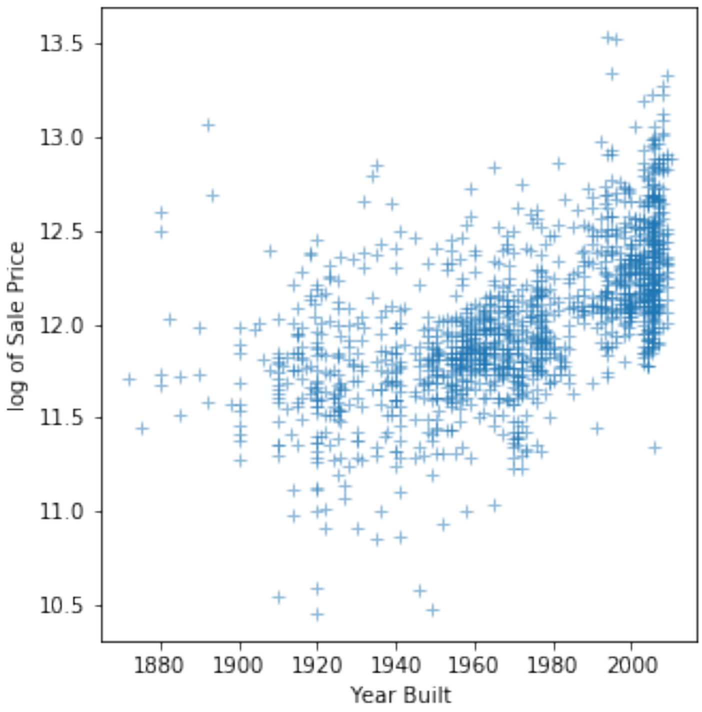
- The subplots above show the expected trend. There is a linear correlation

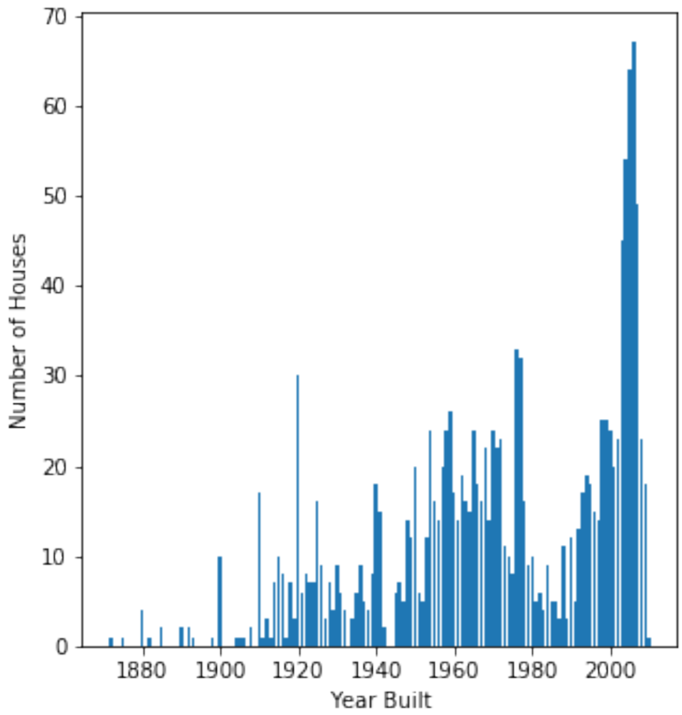
between No of Rooms and Sales Price (except few outliers), and no of rooms and

number of bathrooms in a house.

```
plt.figure(figsize=(5,12))
grid = plt.GridSpec(2, 1, wspace=0.2, hspace=0.2)
plt.suptitle('Charts showing distribution across Year Built', fontsize=16)

plt.subplot(grid[0, 0])
plt.plot(train.YearBuilt, np.log(train.SalePrice) ,'+', alpha = 0.5)
plt.xlabel("Year Built")
plt.ylabel("log of Sale Price")

plt.subplot(grid[1, 0])
q2a_t1 = train['YearBuilt'].value_counts().reset_index(name='YearBuilt')
q2a_t1.columns = ['YearBuilt', 'Count']
plt.xlabel("Year Built")
plt.ylabel("Number of Houses")
plt.bar(q2a_t1["YearBuilt"], q2a_t1["Count"])

plt.show()
```

Charts showing distribution across Year Built

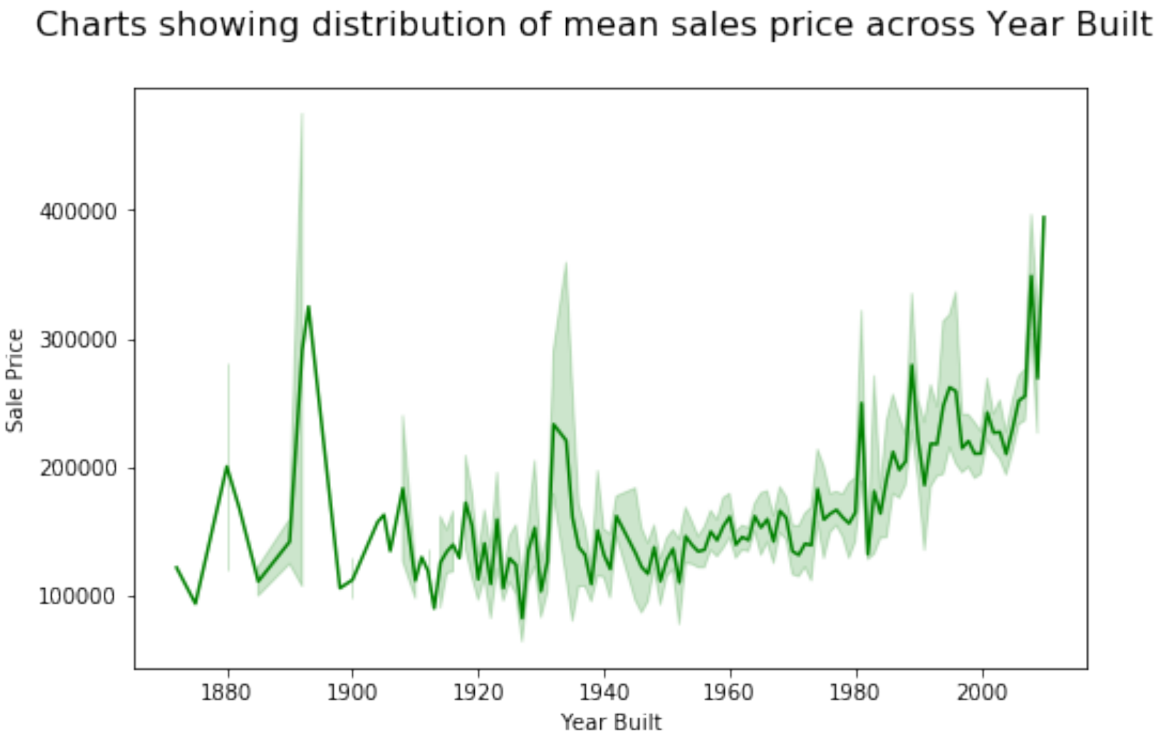




- The greater part of the houses in the dataset was built in the last 10 years.
- Older houses from the beginning of the last century was sold at high prices.

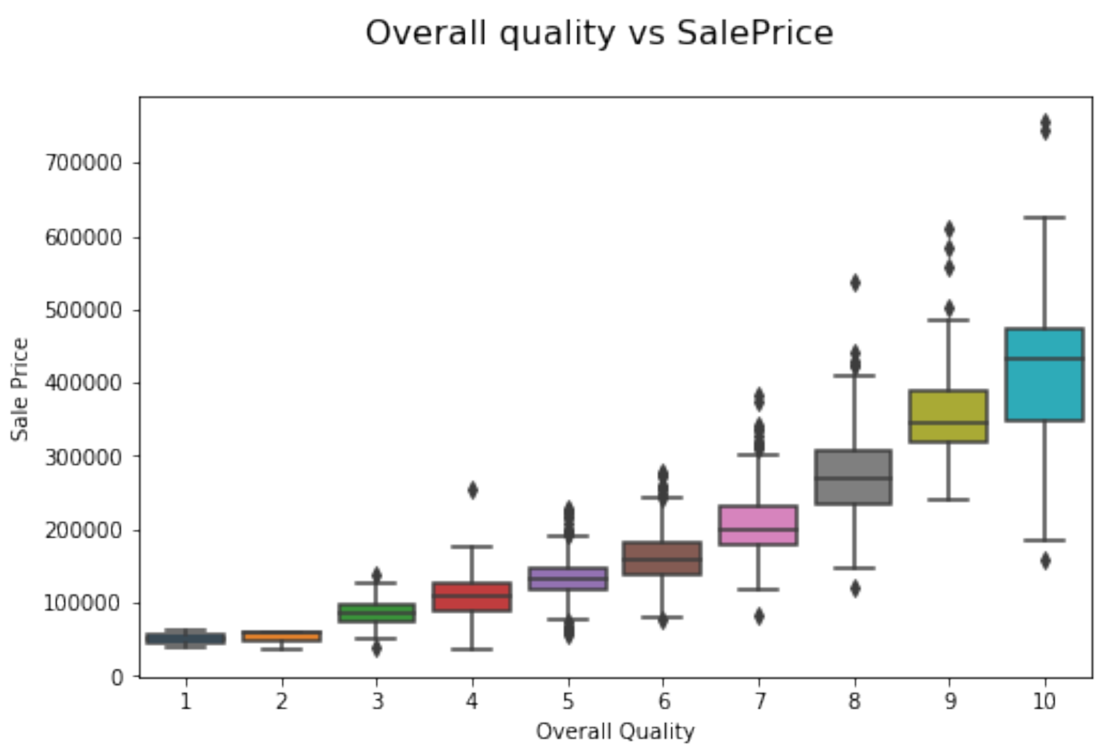
Double-click (or enter) to edit

```
plt.figure(figsize=(8,5))
plt.suptitle('Charts showing distribution of mean sales price across Year Built'
, fontsize=16)
q2b = train[['YearBuilt','SalePrice']]
q2b.groupby('YearBuilt', as_index=False)['SalePrice'].mean()
sb.lineplot(y='SalePrice', x='YearBuilt', data=q2b, color='green')
plt.xlabel("Year Built")
plt.ylabel("Sale Price")
plt.show()
```



There has been a noticable increase in the mean price of the houses (Even though the number of house have increased)through out the time period.

```
plt.figure(figsize=(8,5))
sb.boxplot(x='OverallQual', y='SalePrice', data=train[['SalePrice',
                                                      'OverallQual']])
plt.suptitle('Overall quality vs SalePrice', fontsize=16)
plt.xlabel("Overall Quality")
plt.ylabel("Sale Price")
plt.show()
```

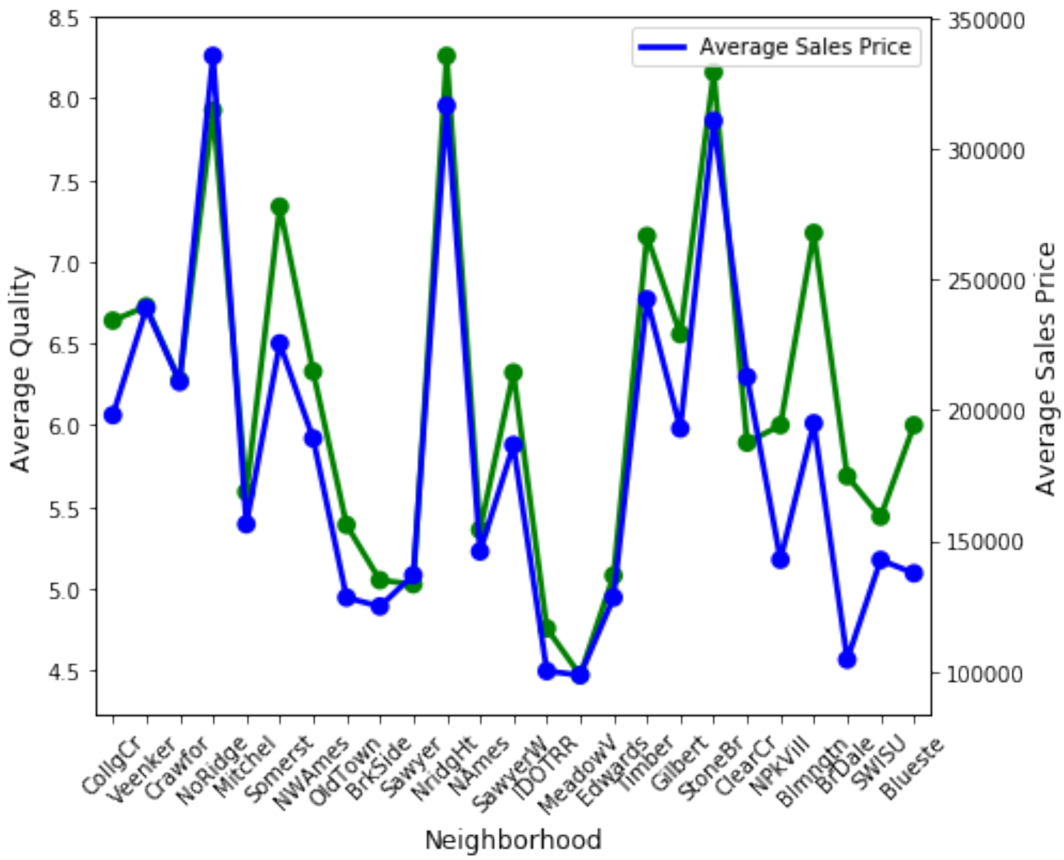


It is the overall finish of the house (including material and make) on a scale from 1 (very poor) to 10 (very excellent). There is a strong positive correlation as seen from the plot. There are a few outliers present as seen from the plot.

```
plt.figure(figsize=(7,6))
plt.suptitle('overall qaulity and average & sales price in a neighbourhood',
             fontsize=16)

q2a = train.loc[:, ['OverallQual', 'SalePrice', 'Neighborhood']]
no = []
avgSalePrice = []
neighborhood = []
avgQuality = []

for n in q2a.Neighborhood.unique():
    q2a_ = q2a[q2a['Neighborhood'] == n]
    neighborhood.append(n)
    no.append(len(q2a_.index))
    avgSalePrice.append(q2a_['SalePrice'].mean())
    avgQuality.append(q2a_['OverallQual'].mean())

q2a_ = pd.DataFrame()
q2a_['no']  = no
```

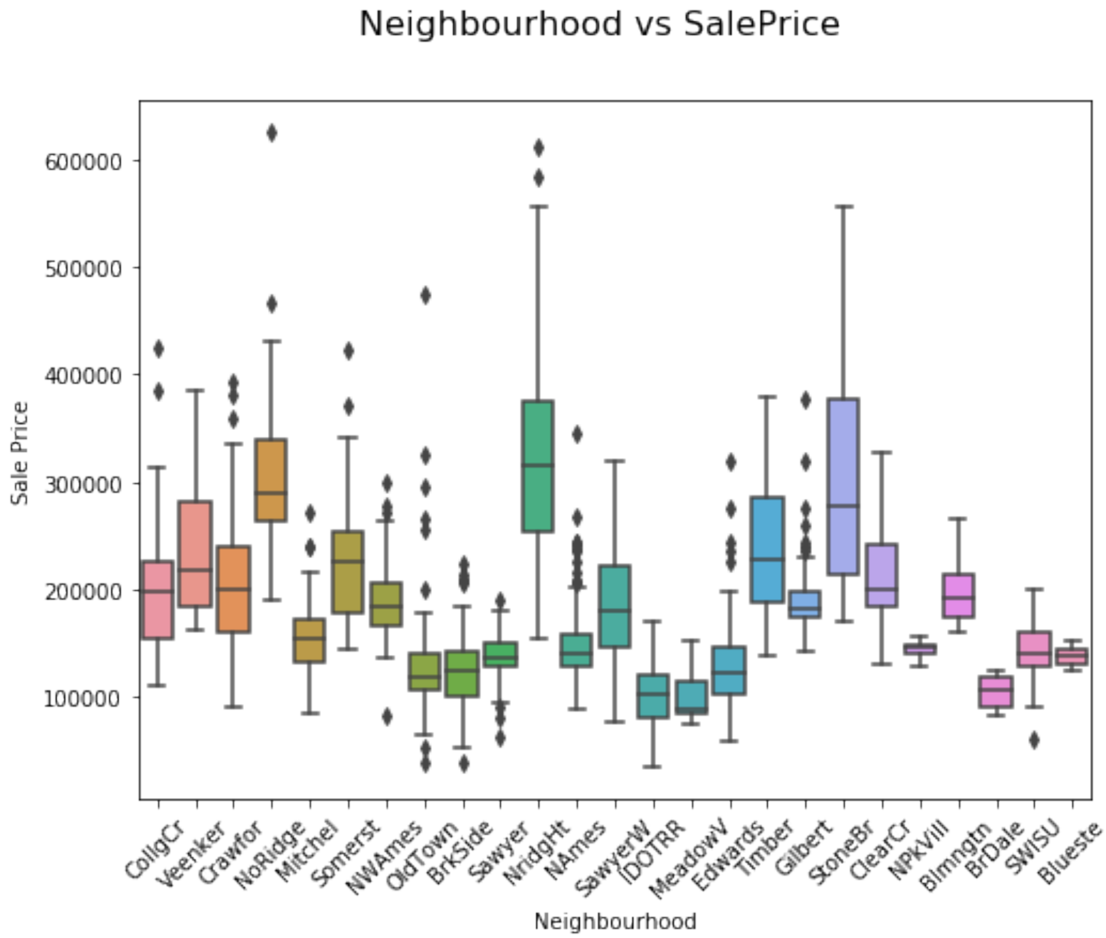
```
q2a_['avgSalePrice']  = avgSalePrice
q2a_['neighborhood'] = neighborhood
q2a_['avgQuality'] = avgQuality

cp1 = sb.pointplot(y="avgQuality", x="neighborhood", data=q2a_, color="green",
                   labels="Average Quality")
lp1 = cp1.twinx()
lp1 = sb.pointplot(y="avgSalePrice", x="neighborhood", data=q2a_, color="blue",
                   labels="Average Sales Price")

cp1.set_xticklabels(q2a_['neighborhood'], rotation=45)
cp1.set_xlabel("Total Rooms Above Ground", fontsize=12)
cp1.set_ylabel("Count", fontsize=12)

cp1.set_xlabel("Neighborhood", fontsize=12)
cp1.set_ylabel("Average Quality", fontsize=12)
lp1.set_ylabel("Average Sales Price", fontsize=12)

plt.legend(labels=['Average Sales Price'])
plt.show()
```

overall qaulity and average & sales price in a neighbourhood



The Average Sale price and average quality of Neighborhood goes hand in hand justifying that they have a ver

```
plt.figure(figsize=(8,6))
sb1 = sb.boxplot(y='SalePrice', x='Neighborhood',
                 data=train[train.SalePrice < 700000])
plt.suptitle('Neighbourhood vs SalePrice', fontsize=16)
plt.xlabel("Neighbourhood")
plt.xticks(rotation=45)
plt.ylabel("Sale Price")
plt.show()
```



Boxplot depicting

Data Exploration

Feature extraction preparation will help in the next questions. (Q3-Q10)

Check for Missing values

```
# The missing values in the dataset (More than 40% data missing)
model = train.copy()
model_test = test.copy()
missing_values = train.isnull().sum()
missing_values_percent = 100 * missing_values / len(train)
missing_values_table = pd.concat([missing_values, missing_values_percent], axis=1)
missing_values_table = missing_values_table.rename(columns =
                                                   {0 : 'Missing Values', 1 : '%'})
missing_values_table = missing_values_table[
missing_values_table.iloc[:,1] > 40].sort_values('%', ascending=False).round(1)

model = model.drop(
    ['PoolQC', 'MiscFeature', 'Alley', 'Fence', 'Utilities', 'Street'], axis=1)
model_test = model_test.drop(
    ['PoolQC', 'MiscFeature', 'Alley', 'Fence', 'Utilities', 'Street'], axis=1)
missing_values_table
```

Dropping columns **PoolQC**, **MiscFeature**, **Alley** and **Fence** right away because 50% of the data has null va

Dropping **Utlities** and **Street** columns because they have 1 and 6 alternate values respectively.

Droping outliers

```
model = model[model.SalePrice < 700000]
model = model[model.LotArea < 100000]

plt.figure(figsize=(9,7))
grid = plt.GridSpec(2, 1, wspace=0.2, hspace=0.2)
plt.suptitle('Outliers wrt Sales Price and Lot Area', fontsize=16)

plt.subplot(grid[0, 0])

sb.boxplot(x='SalePrice', data=train[['SalePrice']])
plt.suptitle('SalePrice', fontsize=16)
plt.ylabel("Sale Price")

plt.subplot(grid[1, 0])
sb.boxplot(x='LotArea', data=train[['LotArea']])
plt.suptitle('LotArea', fontsize=16)
plt.ylabel("Lot Area")

plt.show()
```

Using the seaborn boxplot method we notice points outside the interquartile range as outliers. As is evidence
that sold for more than $700k.

Changing rating strings to numericals

```
qualityBasedColumnsOutof6 = ['ExterQual', 'ExterCond', 'BsmtQual', 'BsmtCond',
                             'HeatingQC','KitchenQual', 'GarageQual',
                             'GarageCond', 'FireplaceQu']

QualityMappingOutof6 = {'NA' : 0, 'Po' : 1, 'Fa' : 2,'TA' : 3,'Gd' : 4, 'Ex': 5}

for col in qualityBasedColumnsOutof6:
    model[col].fillna('NA', inplace=True)
    model[col] = model[col].apply(lambda x: QualityMappingOutof6[x])
    model_test[col].fillna('NA', inplace=True)
    model_test[col] = model_test[col].apply(lambda x: QualityMappingOutof6[x])
```

Handling the null values for numeric features by replacing null with the mean value

```
print('Train Table\n')
for col in model.select_dtypes(include=[np.number]).columns.tolist():
    nullCount = model[col].isna().sum()
    if nullCount > 0:
        colMean = model[col].mean()
        model[col].fillna(colMean, inplace=True)
        print (col  + ' has ' + str(nullCount) +
               ' null values. Replacing them with ' + str(colMean))

print('\nTest Table\n')
for col in model_test.select_dtypes(include=[np.number]).columns.tolist():
    nullCount = model_test[col].isna().sum()
    if nullCount > 0:
        colMean = model_test[col].mean()
        model_test[col].fillna(colMean, inplace=True)
        print (col  + ' has ' + str(nullCount) +
               ' null values. Replacing them with ' + str(colMean))
```

Handling the null values for Categorical features by replacing null with the mode

```
print('Train Table\n')
for col in model.select_dtypes(exclude=["number","bool_"]).columns.tolist():
    nullCount = model[col].isna().sum()
    if nullCount > 0:
        colMode = model[col].mode()[0]
        model[col].fillna(colMode, inplace=True)
        print (col  + ' has ' + str(nullCount)
        + ' null values. Replacing them with ' + str(colMode))

print('Test Table\n')
for col in model_test.select_dtypes(exclude=["number","bool_"]).columns.tolist():
    nullCount = model_test[col].isna().sum()
    if nullCount > 0:
        colMode = model_test[col].mode()[0]
        model_test[col].fillna(colMode, inplace=True)
        print (col  + ' has ' + str(nullCount)
        + ' null values. Replacing them with ' + str(colMode))
```

```
Train Table

MasVnrType has 8 null values. Replacing them with None
BsmtExposure has 38 null values. Replacing them with No
BsmtFinType1 has 37 null values. Replacing them with Unf
BsmtFinType2 has 38 null values. Replacing them with Unf
Electrical has 1 null values. Replacing them with SBrkr
GarageType has 81 null values. Replacing them with Attchd
GarageFinish has 81 null values. Replacing them with Unf
Test Table

MSZoning has 4 null values. Replacing them with RL
Exterior1st has 1 null values. Replacing them with VinylSd
Exterior2nd has 1 null values. Replacing them with VinylSd
MasVnrType has 16 null values. Replacing them with None
BsmtExposure has 44 null values. Replacing them with No
BsmtFinType1 has 42 null values. Replacing them with GLQ
BsmtFinType2 has 42 null values. Replacing them with Unf
Functional has 2 null values. Replacing them with Typ
GarageType has 76 null values. Replacing them with Attchd
GarageFinish has 78 null values. Replacing them with Unf
SaleType has 1 null values. Replacing them with WD
```

Neighbourhood values to numericals

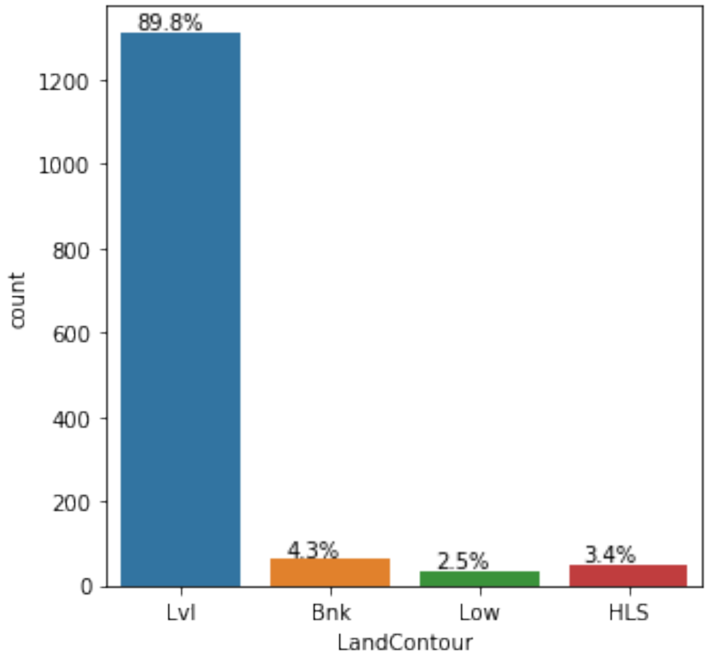
Count	Average Selling Price	Neighbourhood	Average Quality
17	98576.5	MeadowV	4.47
37	100123.8	IDOTRR	4.76
74	136793.1	Sawyer	5.03
58	124834.1	BrkSide	5.05
100	128219.7	Edwards	5.08
225	145847.1	NAmes	5.36
113	128225.3	OldTown	5.39
25	142591.4	SWISU	5.44
49	156270.1	Mitchel	5.59
16	104493.8	BrDale	5.69
28	212565.4	ClearCr	5.89
2	137500.0	Blueste	6.00
9	142694.4	NPkVill	6.00
51	210624.7	Crawfor	6.27
59	186555.8	SawyerW	6.32
73	189050.1	NWAmes	6.33
79	192854.5	Gilbert	6.56
150	197965.8	CollgCr	6.64
11	238772.7	Veenker	6.73
38	242247.4	Timber	7.16
17	194870.9	Blmngtn	7.18
86	225379.8	Somerst	7.34
41	335295.3	NoRidge	7.93
25	310499.0	StoneBr	8.16
77	316270.6	NridgHt	8.26

```
neighbourhoodMapping = {'MeadowV' : 0, 'IDOTRR' : 1, 'Sawyer' : 2,'BrkSide' : 2,
                        'Edwards' : 2, 'NAmes': 3,
                       'OldTown' : 3, 'SWISU' : 3, 'Mitchel' : 4,'BrDale' : 2,
                        'ClearCr' : 5, 'Blueste': 3,
                       'NPkVill' : 3, 'Crawfor' : 6, 'SawyerW' : 5,'NWAmes' : 5,
                        'Gilbert' : 5, 'CollgCr': 5,
                       'Veenker' : 6, 'Timber' : 7, 'Blmngtn' : 6,'Somerst' : 7,
                        'NoRidge' : 9, 'StoneBr': 8,
                       'NridgHt' : 8}
model['Neighborhood'] = model['Neighborhood'].apply(
    lambda x: neighbourhoodMapping[x])
model_test['Neighborhood'] = model_test['Neighborhood'].apply(
    lambda x: neighbourhoodMapping[x])
```

Data Analysis and Cleaning for some categorical features

LandContour has Lvl mostly. Hence adding a new column **LandLeveled** and dropping LandContour.

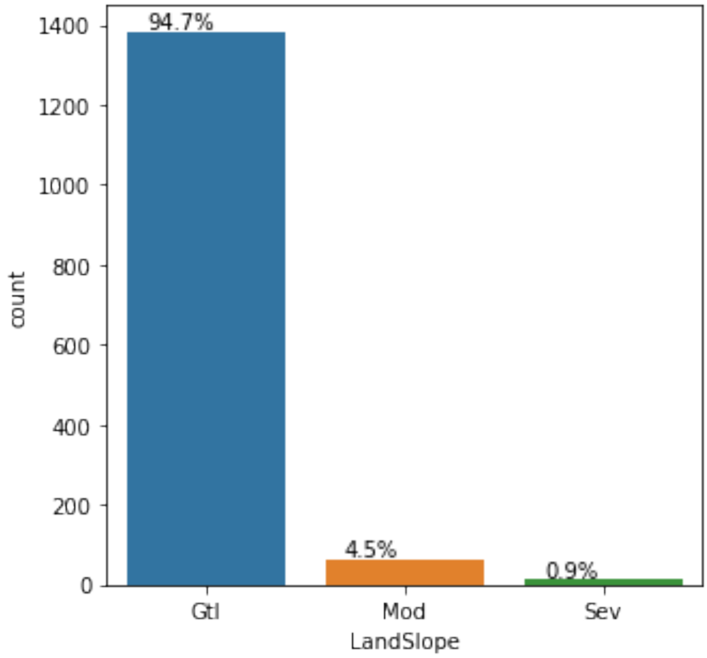
```
plt.figure(figsize=(5,5))
ax = sb.countplot(x='LandContour', data=train)
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_height()/1460), (p.get_x()+0.1,
                                                                p.get_height()+7))
plt.show()
```




```
model['LandLeveled'] = train['LandContour'].apply(lambda x: 1 if x=="Lvl"
else 0)
model = model.drop(['LandContour'], axis=1)
model_test['LandLeveled'] = test['LandContour'].apply(lambda x: 1 if x=="Lvl"
else 0)
model_test = model_test.drop(['LandContour'], axis=1)
```

LandSlope has Gtl mostly. Hence adding a new column **GentleSloped** and dropping LandSlope.

```
plt.figure(figsize=(5,5))
ax = sb.countplot(x='LandSlope', data=train)
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_height()/1460), (p.get_x()+0.1,
                                                                  p.get_height()+7))
plt.show()
```

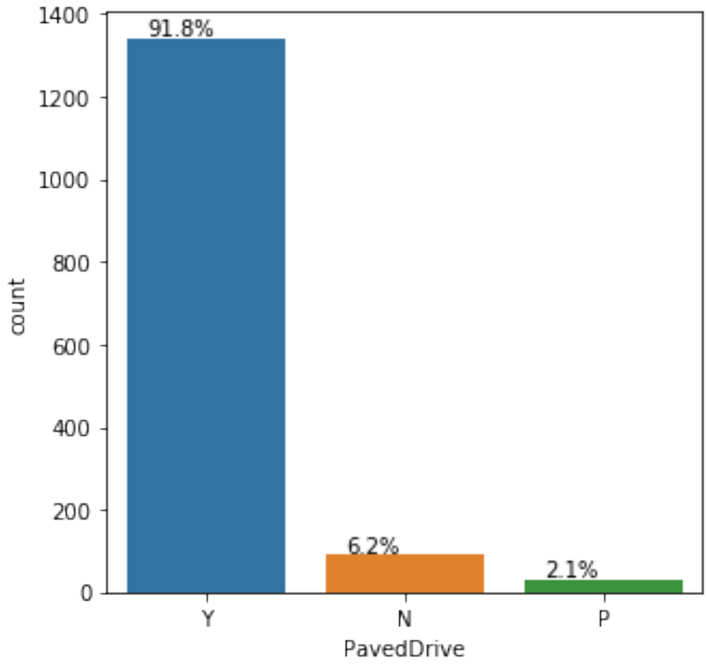



```
model['GentleSloped'] = train['LandSlope'].apply(lambda x: 1 if x=="Gtl" else 0)
model = model.drop(['LandSlope'], axis=1)
model_test['GentleSloped'] = test['LandSlope'].apply(lambda x: 1 if x=="Lvl" else 0)
model_test = model_test.drop(['LandSlope'], axis=1)
```

PavedDrive has Y mostly. Hence adding a new column **hasPavedDrive** and

dropping LandSlope.

```
plt.figure(figsize=(5,5))
ax = sb.countplot(x='PavedDrive', data=train)
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_height()/1460),
                    (p.get_x()+0.1, p.get_height()+7))
plt.show()
```




```
model['hasPavedDrive'] = train['PavedDrive'].apply(lambda x: 1 if x=="Y" else 0)
model = model.drop(['PavedDrive'], axis=1)
model_test['hasPavedDrive'] = test['PavedDrive'].apply(
    lambda x: 1 if x=="Y" else 0)
model_test = model_test.drop(['PavedDrive'], axis=1)
```

GarageType's Attchd and Biltin can be grouped together as inHouse hasAttachedGarage. Hence adding a new column **hasAttachedGarage** and dropping GarageType.

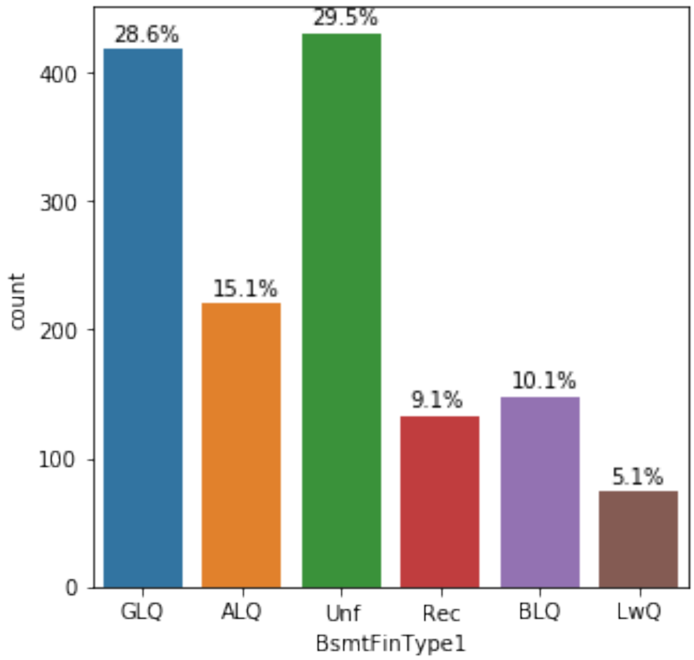
```
plt.figure(figsize=(5,5))
ax = sb.countplot(x='GarageType', data=train)
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_height()/1460),
                    (p.get_x()+0.1, p.get_height()+7))
plt.show()
```

```
model['hasAttachedGarage'] = train['GarageType'].apply(
    lambda x: 1if (x=="Attchd" or x=="BuiltIn") else 0)
model['hasDetachedGarage'] = train['GarageType'].apply(
    lambda x: 1 if (x!="Attchd" and x!="BuiltIn") else 0)
model = model.drop(['GarageType'], axis=1)
```

```
model_test['hasAttachedGarage'] = test['GarageType'].apply(
    lambda x: 1 if (x=="Attchd" or x=="BuiltIn") else 0)
model_test['hasDetachedGarage'] = train['GarageType'].apply(
    lambda x: 1 if (x!="Attchd" and x!="BuiltIn") else 0)
model_test = model_test.drop(['GarageType'], axis=1)
```

BsmtFinType1 and **BsmtFinType2** values are replaced with numericals.

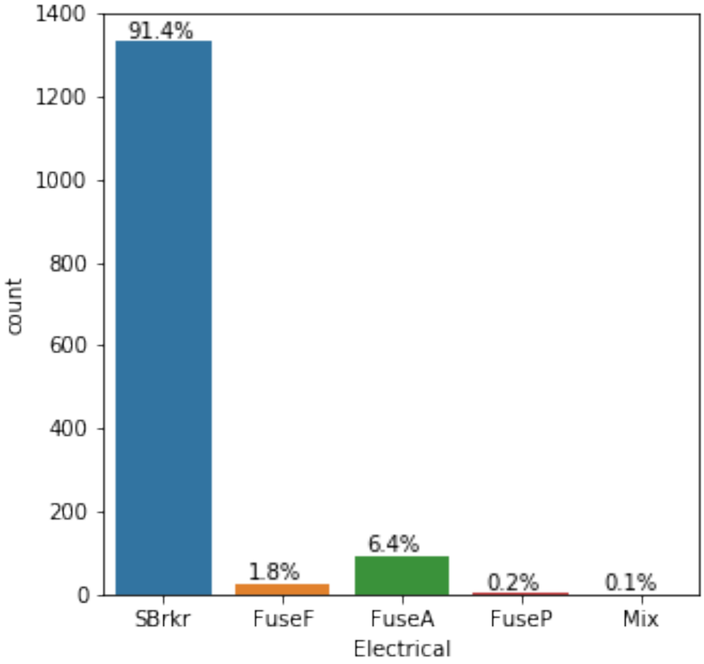
```
plt.figure(figsize=(5,5))
ax = sb.countplot(x='BsmtFinType1', data=train)
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_height()/1460),
                    (p.get_x()+0.1, p.get_height()+7))
plt.show()
```




```
BsmtFinType = {'Unf' : 1, 'LwQ' : 2, 'BLQ' : 3,'Rec' : 4,'ALQ' : 5, 'GLQ': 6}
model['BsmtFinType1'] = model['BsmtFinType1'].apply(lambda x: BsmtFinType[x])
model_test['BsmtFinType1'] = model_test['BsmtFinType1'].apply(
    lambda x: BsmtFinType[x])
model['BsmtFinType2'] = model['BsmtFinType2'].apply(lambda x: BsmtFinType[x])
model_test['BsmtFinType2'] = model_test['BsmtFinType2'].apply(
    lambda x: BsmtFinType[x])
```

Electrical values are replaced with numericals.

```
plt.figure(figsize=(5,5))
ax = sb.countplot(x='Electrical', data=train)
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_height()/1460),
                    (p.get_x()+0.1, p.get_height()+7))
plt.show()
```




```
ElectricalType = {'Mix' : 1, 'FuseP' : 2, 'FuseF' : 2,'FuseA' : 2,'SBrkr' : 3}
model['Electrical'] = model['Electrical'].apply(lambda x: ElectricalType[x])
model_test['Electrical'] = model_test['Electrical'].apply(
    lambda x: ElectricalType[x])
```

BsmtExposure values are replaced with numericals.

```
plt.figure(figsize=(5,5))
ax = sb.countplot(x='BsmtExposure', data=train)
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_height()/1460),
                    (p.get_x()+0.1, p.get_height()+7))
plt.show()
```

```
BsmtExposureType = {'No' : 1, 'Mn' : 2, 'Av' : 3,'Gd' : 4}
model['BsmtExposure'] = model['BsmtExposure'].apply(
    lambda x: BsmtExposureType[x])
model_test['BsmtExposure'] = model_test['BsmtExposure'].apply(
    lambda x: BsmtExposureType[x])
```

Functional values are replaced with numericals.

```
plt.figure(figsize=(5,5))
ax = sb.countplot(x='Functional', data=train)
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_height()/1460),
                    (p.get_x()+0.1, p.get_height()+7))
plt.show()
```

```
FunctionalType = {'Maj2' : 1, 'Sev' : 2, 'Min2' : 3,'Min1' : 4,
                  'Maj1' : 5, 'Mod' : 6, 'Typ' : 7}
model['Functional'] = model['Functional'].apply(lambda x: FunctionalType[x])
model_test['Functional'] = model_test['Functional'].apply(
    lambda x: FunctionalType[x])
```

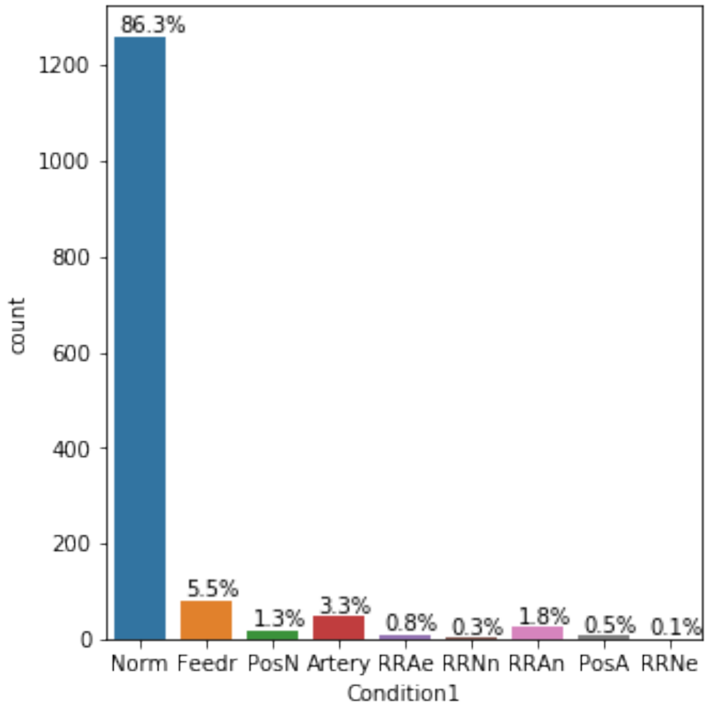
GarageFinish values are replaced with numericals.

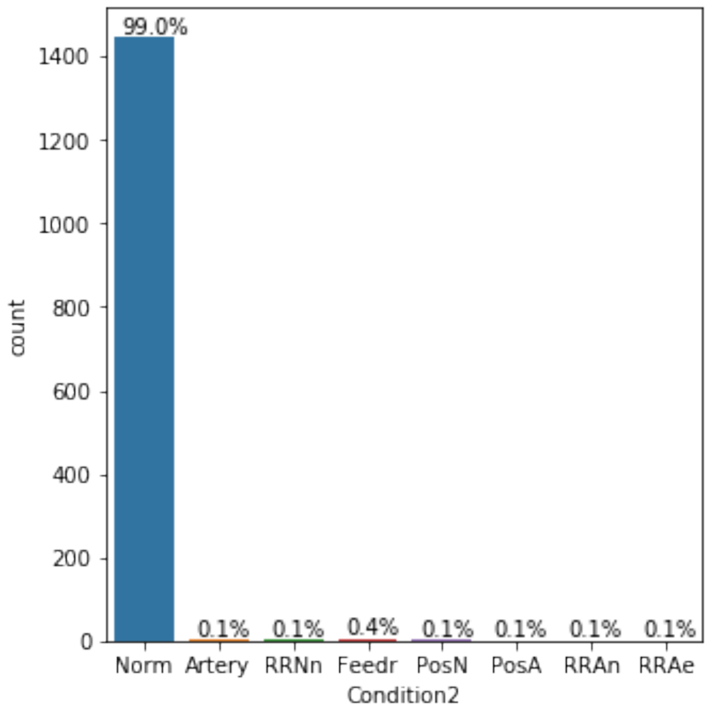
```
plt.figure(figsize=(5,5))
ax = sb.countplot(x='GarageFinish', data=train)
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_height()/1460),
                    (p.get_x()+0.1, p.get_height()+7))
plt.show()
```

```
GarageFinishType = {'Unf' : 1, 'RFn' : 2, 'Fin' : 3}
model['GarageFinish'] = model['GarageFinish'].apply(lambda x: GarageFinishType[x])
model_test['GarageFinish'] = model_test['GarageFinish'].apply(
    lambda x: GarageFinishType[x])
```

Condition1 and **condition2** values are replaced with numericals.

```
plt.figure(figsize=(5,12))
grid = plt.GridSpec(2, 1, wspace=0.5, hspace=0.2)
plt.subplot(grid[0,0])
ax = sb.countplot(x='Condition1', data=train)
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_height()/1460),
                    (p.get_x()+0.1, p.get_height()+7))
plt.subplot(grid[1,0])
ax1 = sb.countplot(x='Condition2', data=train)
for p in ax1.patches:
        ax1.annotate('{:.1f}%'.format(100*p.get_height()/1460),
                     (p.get_x()+0.1, p.get_height()+7))
plt.show()
```



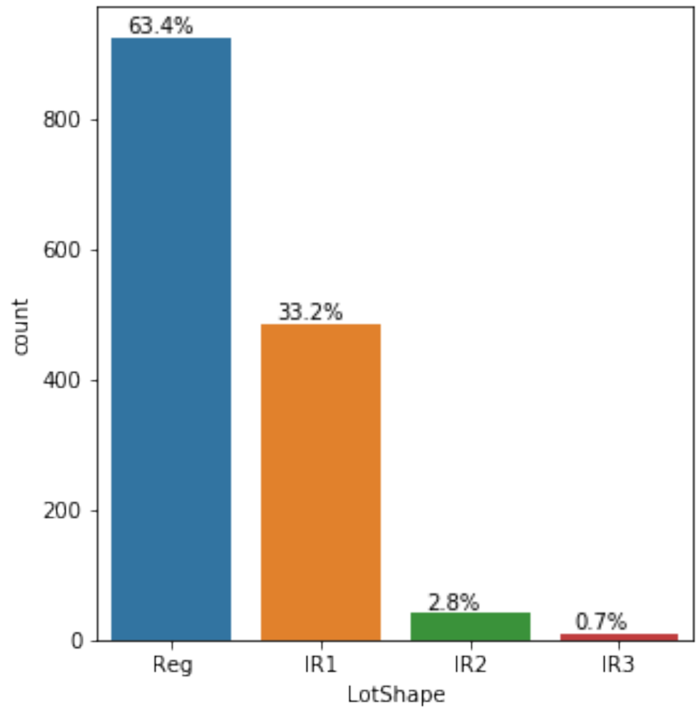


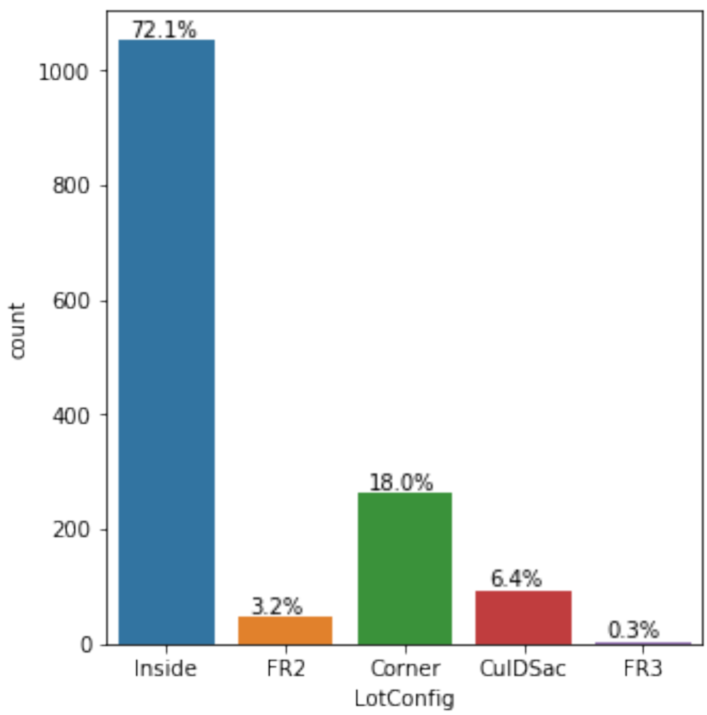

```
ConditionType = {'Feedr' : 1, 'Artery' : 1,'RRAe': 2, 'RRAn' : 2, 'RRNe' : 2, 'RRNn' :
                        'Norm' : 3, 'PosA': 4, 'PosN' : 4}
model['Condition1'] = model['Condition1'].apply(lambda x: ConditionType[x])
model_test['Condition1'] = model_test['Condition1'].apply(lambda x: ConditionType[x])
model['Condition2'] = model['Condition2'].apply(lambda x: ConditionType[x])
model_test['Condition2'] = model_test['Condition2'].apply(lambda x: ConditionType[x])
```

LotShape & **LotConfig** values are replaced with numericals.

```
plt.figure(figsize=(5,12))
```

```
grid = plt.GridSpec(2, 1, wspace=0.5, hspace=0.2)
plt.subplot(grid[0,0])
ax = sb.countplot(x='LotShape', data=train)
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_height()/1460),
                    (p.get_x()+0.1, p.get_height()+7))
plt.subplot(grid[1,0])
ax1 = sb.countplot(x='LotConfig', data=train)
for p in ax1.patches:
        ax1.annotate('{:.1f}%'.format(100*p.get_height()/1460),
                     (p.get_x()+0.1, p.get_height()+7))
plt.show()
```

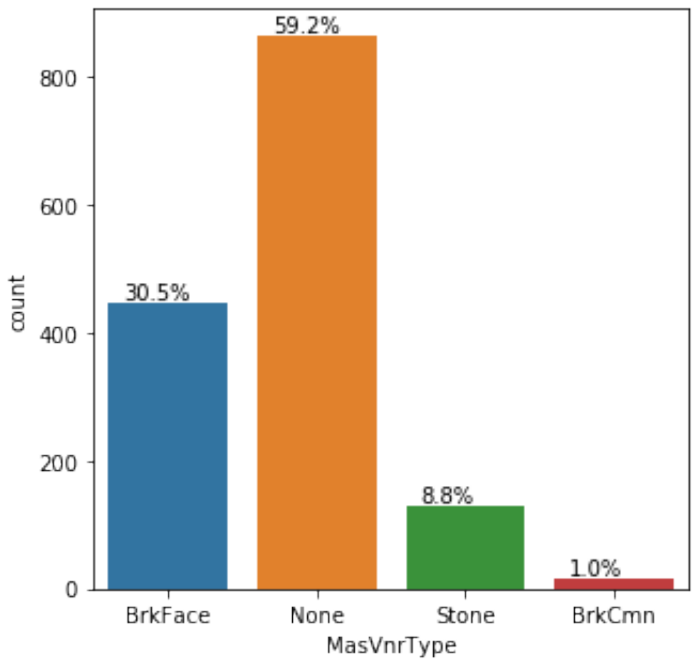




```
LotShapeType = {'Reg' : 1,'IR1' : 2, 'IR2' : 3, 'IR3' : 4}
LotConfigType = {'Inside' : 1,'FR3' : 2, 'FR2' : 2, 'Corner' : 3, 'CulDSac' : 4}

model['LotShape'] = model['LotShape'].apply(lambda x: LotShapeType[x])
model_test['LotShape'] = model_test['LotShape'].apply(lambda x: LotShapeType[x])

model['LotConfig'] = model['LotConfig'].apply(lambda x: LotConfigType[x])
model_test['LotConfig'] = model_test['LotConfig'].apply(lambda x: LotConfigType[x])
```

MasVnrType values are replaced with numericals.

```
plt.figure(figsize=(5,5))
ax = sb.countplot(x='MasVnrType', data=train)
for p in ax.patches:
        ax.annotate('{:.1f}%'.format(100*p.get_height()/1460),
                    (p.get_x()+0.1, p.get_height()+7))
plt.show()
```




```
MasVnrTypeType = {'None' : 0,'BrkCmn' : 1, 'Stone' : 2, 'BrkFace' : 3}
model['MasVnrType'] = model['MasVnrType'].apply(lambda x: MasVnrTypeType[x])
model_test['MasVnrType'] = model_test['MasVnrType'].apply(lambda x: MasVnrTypeType[x])
```

MSZoning values are replaced with numericals.

MSZoning	Average Selling Price
RL	189662.7
RM	126316.8
C	74528.0
FV	214014.0
RH	131558.3

```
MSZoningType = {'C (all)' : 1,'RM' : 2, 'RH' : 3, 'RL' : 4, 'FV': 5 }
model['MSZoning'] = model['MSZoning'].apply(lambda x: MSZoningType[x])
model_test['MSZoning'] = model_test['MSZoning'].apply(lambda x: MSZoningType[x])
```

BldgType values are replaced with numericals.

```
plt.figure(figsize=(7,5))

a = []
b = []
for x in train.BldgType.unique():
    q6a_ = train[train['BldgType'] == x]
    a.append(x)
    b.append(q6a_['SalePrice'].mean())

q6a_ = pd.DataFrame()
q6a_['a']  = a
q6a_['Average SalePrice']  = b
cp1 = sb.countplot(train.BldgType)
lp1 = cp1.twinx()
lp1 = sb.pointplot(y="Average SalePrice", x="a", data=q6a_, color="blue")
plt.show()
```

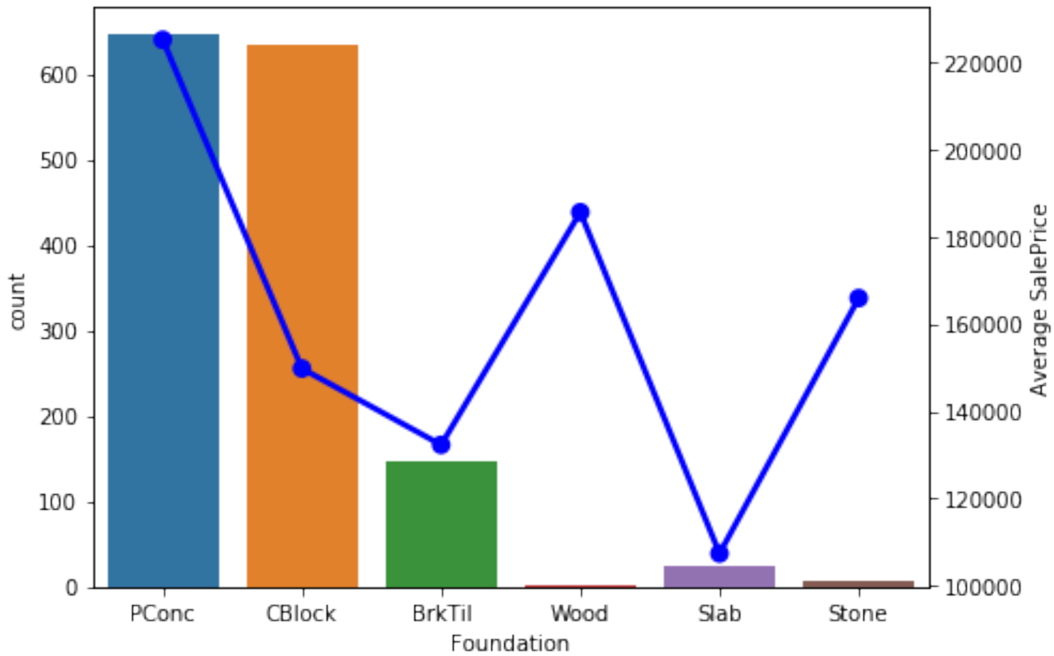
```
BldgTypeType = {'2fmCon' : 1,'Duplex' : 2, 'Twnhs' : 3, 'TwnhsE' : 4, '1Fam': 5 }
model['BldgType'] = model['BldgType'].apply(lambda x: BldgTypeType[x])
model_test['BldgType'] = model_test['BldgType'].apply(lambda x: BldgTypeType[x])
```

HouseStyle values are replaced with numericals.

```
HouseStyleType = {'1.5Unf' : 1,'SFoyer' : 1, '1.5Fin' : 1, 'SLvl' : 2,
                  '2.5Unf': 2, '1Story' : 2, '2.5Fin':3, '2Story':3}
model['HouseStyle'] = model['HouseStyle'].apply(lambda x: HouseStyleType[x])
model_test['HouseStyle'] = model_test['HouseStyle'].apply(
    lambda x: HouseStyleType[x])
```

Foundation values are replaced with numericals.

```
plt.figure(figsize=(7,5))
a = []
b = []
for x in train.Foundation.unique():
    q6a_ = train[train['Foundation'] == x]
    a.append(x)
    b.append(q6a_['SalePrice'].mean())

q6a_ = pd.DataFrame()
q6a_['a']  = a
q6a_['Average SalePrice']  = b
cp1 = sb.countplot(train.Foundation)
lp1 = cp1.twinx()
lp1 = sb.pointplot(y="Average SalePrice", x="a", data=q6a_, color="blue")
plt.show()
```




```
FoundationType = {'Wood': 1, 'Stone' : 1,'Slab' : 1, 'BrkTil' : 2,
                  'CBlock' : 2, 'PConc': 3}
model['Foundation'] = model['Foundation'].apply(lambda x: FoundationType[x])
model_test['Foundation'] = model_test['Foundation'].apply(
    lambda x: FoundationType[x])
```

Part 3 - Handcrafted Scoring Function

```
train_copy = model.copy()
qualityBasedColumnsOutof10 = ['OverallQual', 'OverallCond']
qualityBasedColumnsOutof6 = ['ExterQual', 'ExterCond', 'BsmtQual', 'BsmtCond',
                             'HeatingQC','KitchenQual', 'GarageQual', 'GarageCond']

amenitiesCols = ['GarageCars', 'BsmtFinType1',
                 'BsmtExposure', 'BldgType', 'HouseStyle']
otherFactors = ['YearBuilt', 'Neighborhood', 'MSZoning',
                'YearRemodAdd', 'YearBuilt', 'SalePrice']


# 8 * 5 (qualityBasedColumnsOutof6) + 20 (qualityBasedColumnsOutof10) +
# 8 (Neighbourhood) 5 MSZoning = 73 -> Maximum score you can get
def qualityScoringFunction(row):
    qualitySum = 0
    for col in qualityBasedColumnsOutof6:
        qualitySum += row[col]
    qualitySum += row['OverallQual'] * 0.5
    qualitySum += row['OverallCond'] * 1.5
    qualitySum += row['Neighborhood'] * 1.5
    qualitySum += row['MSZoning']
    return qualitySum/77.0 * 100

# TotRmsAbvGrd 2-14, GarageCars 0-4, BsmtFinType1 1-6, BsmtExposure 1-4,
# BldgType 1-5, HouseStyle 1-3
def amenitiesScoringFunction(row):
    amenitiesSum = 0
    for col in amenitiesCols:
        amenitiesSum += row[col]
    return amenitiesSum/36 * 100

# if Saleprice/LotArea is less, the house is more desirable (For a person with budget)
def costScoringFunction(row):
    return 100 - row['SalePrice']/row['LotArea']

# 1950 - Latest year when ant of the house was remodeled
#
def builtYearScoringFunction(yr,yb):
    return (((yr - 1949)/60.0) * 100) * 0.5 + (((yb - 1872)/138.0) * 100) * 0.5

# Factors that can further be tuned as per requirements
qualityFactor = 0.5
builtYearFactor = 0.1
# costFactor = 0.2
amenitiesFactor = 0.2

qualityScore = []
builtYearScore = []
priceScore = []
amenitiesScore = []
overallScore = []

for index, row in model.iterrows():
    qs = qualityScoringFunction(row)
    # ps = costScoringFunction(row)
    ys = builtYearScoringFunction(row['YearRemodAdd'], row['YearBuilt'])
    ass = amenitiesScoringFunction(row)
    qualityScore.append(qs)
    #priceScore.append(ps)
    builtYearScore.append(ys)
    amenitiesScore.append(ass)
    overallScore.append(qualityFactor * qs + builtYearFactor * ys
                        + amenitiesFactor * ass)

train_copy = train_copy.assign(qualityScore=qualityScore)
```

```
#train_copy = train_copy.assign(priceScore=priceScore)
train_copy = train_copy.assign(builtYearScore=builtYearScore)
train_copy = train_copy.assign(amenitiesScore=amenitiesScore)
train_copy = train_copy.assign(overallScore=overallScore)

reqdCols = ['Id','qualityScore','builtYearScore',
        'amenitiesScore', 'overallScore']
reqdCols = reqdCols + qualityBasedColumnsOutof6 + qualityBasedColumnsOutof10 +\
amenitiesCols + otherFactors
```

Ten Most Desirable Houses are:

```
train_copy.loc[:, train_copy.columns.isin(reqdCols)]\
.sort_values('overallScore',ascending=False).head(10)
```

	Id	MSZoning	Neighborhood	BldgType	HouseStyle	OverallQual	Overall
591	592	4	8	5	3	10	
1243	1244	4	8	5	2	10	
1373	1374	4	9	5	2	10	
440	441	4	8	5	2	10	
389	390	4	8	5	3	10	
898	899	4	8	5	2	9	
994	995	4	8	5	2	10	
1442	1443	5	7	5	3	10	
1058	1059	4	8	5	3	9	
527	528	4	8	5	3	9	

Ten least desirable houses as per the scoring function

```
train_copy.loc[:, train_copy.columns.isin(reqdCols)]\
.sort_values('overallScore', ascending=True).head(10)
```

	Id	MSZoning	Neighborhood	BldgType	HouseStyle	OverallQual	Overall
705	706	2	1	1	3	4	
533	534	4	2	5	2	1	
88	89	1	1	5	1	3	
636	637	2	2	5	2	2	
375	376	4	2	5	2	1	
398	399	2	1	5	2	5	
39	40	4	2	2	2	4	
1325	1326	2	1	5	2	4	
1218	1219	2	2	5	1	4	
1011	1012	4	2	2	2	5	

Scoring function below uses Quality, Price and Year Built as factors to compute a score out of 100.

Quality: OverallQual, OverallCond, ExterQual, ExterCond, BsmtQual, BsmtCond, HeatingQC, KitchenQual, GarageQual, GarageCond and **Neighborhood**

Year Remodeled (Year Built) : Older the house, less desirable it is to live in it.

Ameneties Total Living Rooms, Bathrooms , GarageCars, BsmtFinType1, BsmtExposure, BldgType, HouseSt

Part 4 - Pairwise Distance Function

Drop a few columns

```
columnsToDrop = ['Id','MasVnrType','MasVnrArea', '3SsnPorch', 'RoofStyle', 'RoofMatl',
                 'Exterior1st', 'Exterior2nd', 'Heating', 'CentralAir', 'SaleType', 'S
                'MiscVal', 'MoSold', 'YrSold']
similarityModel = model.copy()
similarityModel = similarityModel.drop(columnsToDrop, axis=1)
```

Change Sale Price to Sale price per SF using Lot Area

```
SalePricePerSF = []
for index, row in similarityModel.iterrows():
    SalePricePerSF.append(row['SalePrice']/row['LotArea'])
similarityModel = similarityModel.assign(SalePricePerSF=SalePricePerSF)

similarityModel = similarityModel.drop(['SalePrice'], axis=1)
```

Binning LotArea, 1stFlrSF, 2ndFlrSF, GrLivArea equally into 60 bins

```
colsWithCommonFactor60 = ['LotArea', '1stFlrSF', '2ndFlrSF', 'GrLivArea']
for col in colsWithCommonFactor60:
    similarityModel[col + 'Rank'] = similarityModel[col].rank(method='first')
    similarityModel[col + 'Bin'] = pd.qcut(similarityModel[col + 'Rank'].values, 60).c
    similarityModel = similarityModel.drop([col + 'Rank', col], axis=1)
```

Binning LotArea, 1stFlrSF, 2ndFlrSF, GrLivArea equally into 60 bins

```
colsWithCommonFactor20 = ['BsmtFinSF1', 'TotalBsmtSF', 'BsmtUnfSF', 'GarageArea']
for col in colsWithCommonFactor20:
    similarityModel[col + 'Rank'] = similarityModel[col].rank(method='first')
    similarityModel[col + 'Bin'] = pd.qcut(similarityModel[col + 'Rank'].values, 20).c
    similarityModel = similarityModel.drop([col + 'Rank', col], axis=1)
```

Binning WoodDeckS, OpenPorchSF, EnclosedPorch, LotFrontage into 10 bins

```
colsWithCommonFactor10 = ['WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', 'LotFrontage']
for col in colsWithCommonFactor10:
    similarityModel[col + 'Rank'] = similarityModel[col].rank(method='first')
    similarityModel[col + 'Bin'] = pd.qcut(similarityModel[col + 'Rank'].values, 10).c
    similarityModel = similarityModel.drop([col + 'Rank', col], axis=1)
```

```
similarityModel.head(3)
```

	MSSubClass	MSZoning	LotShape	LotConfig	Neighborhood	Condition1	Condi
0	60	4	1	1	5	3	
1	20	4	1	2	6	1	
2	60	4	2	1	5	3	

Creating a matrix consisting of Euclidean distances

between the different rows of the similarityModel dataset. Most of the parameters are reduced to smaller numbers which will return a good metric.

```
eucMatrix = sc.spatial.distance.cdist(similarityModel, similarityModel,
                                      metric='euclidean')
eucMatrixDF = pd.DataFrame(eucMatrix)
```

```
eucMatrixDF.head(3)
```

	0	1	2	3	4	5	6	
0	0.000000	88.894732	26.002227	97.407893	46.054559	47.047987	80.126710	68.91
1	88.894732	0.000000	86.411356	100.782108	90.669123	70.894315	57.611855	82.71
2	26.002227	86.411356	0.000000	94.319206	25.425435	36.246130	76.641279	62.97

3 rows × 1454 columns

Defining a method that inputs 2 numbers and prints out the similarity as percentage

The method divides the columns of house 1 into 20 bins and finds the value of

house 2 in one of them to print out similarity number as a factor of 5.

Most similar houses will have larger similarity.

```
def compareTwoHouses(id1,id2):
    if (id1-1 == id2-1):
        print('100% match! Duh!')
        return
    compareHouses = pd.DataFrame()
    compareHouses['houseRanks'] = eucMatrixDF[id1-1].rank(method='first')
    compareHouses['HouseBin'] = pd.qcut(compareHouses['houseRanks'].values, 20)
    i = 100
    for row in compareHouses['HouseBin'].value_counts(sort=False).index:
        i = i-5
        if (eucMatrixDF[id1-1][id2-1] in row):
            print ('House ID#' + str(id1) + ' and #ID' + str(id2) + ' have '
            + str(i) + '% similarity!')
```

Examples

Consider the following examples. The function seems to work fine.

```
compareTwoHouses(1,3)
```

```
House ID#1 and #ID3 have 95% similarity!
```

```
compareTwoHouses(1,1453)
```

```
House ID#1 and #ID1453 have 25% similarity!
```

```
train[train['Id'] == 1]
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	La
0	1	60	RL	65.0	8450	Pave	NaN	Reg	

```
train[train['Id'] == 3]
```

```
train[train['Id'] == 1453]
```

Taking a few examples has led to a conclusion that the scoring function gives good results.

Part 5 - Clustering

```
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
```

```
# Dropping Neighborhood
similarityModel = similarityModel.drop('Neighborhood', axis=1)
similarityModel.head(2)
```

```
pca = PCA(n_components=58).fit(similarityModel)
evr=np.cumsum(pca.explained_variance_ratio_)

plt.figure(figsize=(5,5))
plt.suptitle('PCA Explained Variance to determine number of components required for cl
plt.plot(evr,alpha = 1)
plt.show()
```

The explained variance saturates quickly, passing 99% with only 9 components.

So we'll reduce the dimensionality into 7 variables using PCA

```
pca = PCA(n_components=9).fit(similarityModel)
_pca = pca.fit_transform(similarityModel)
```

```
# Lets calculate score for number of clusters as 20
clusters = range(1,20)
kmeans = [KMeans(i) for i in clusters]
score = [kmeans[i].fit(
    similarityModel).score(similarityModel) for i in range(len(kmeans))]
```

```
plt.figure(figsize=(5,5))
plt.suptitle('Review Elbow Curve to determine number of clusters for KMeans',
             fontsize=16)
plt.plot(list(clusters), score,alpha = 1)
plt.show()
```

Lets make **9** clusters out of the data!

```
n_clusters=9
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
Xkmeans = kmeans.fit_predict(_pca)
```

```
sp = model.SalePrice.reset_index(drop=True)
neigh = model.Neighborhood.reset_index(drop=True)
```

```
_TSNE = TSNE(n_components=2).fit_transform(_pca)
```

```
clusterdf = pd.concat([pd.DataFrame(_TSNE),pd.DataFrame(Xkmeans),
                       pd.DataFrame(sp), pd.DataFrame(neigh)],axis=1)
clusterdf.columns = ['x1','x2','Cluster#','Sale Price','Neighborhood']
clusterdf.head()
```

```
plt.figure(figsize=(7,7))
sb.scatterplot(x="x1", y="x2", hue="Cluster#", palette="cubehelix", data=clusterdf)
plt.show()
```

```
clusterdf['Neighborhood'].value_counts(normalize=True) * 100
```

Looping through the clusters I have listed the top neighbourhoods the houses in each cluster belong to

Cluster	Topmost Neighborhood
0	Neighborhood 3 (43%)
1	Neighborhood 5 (53%)
2	Neighborhood 5 (43.75%)
3	Neighborhood 3 (62.50%)
4	Neighborhood 3 (45.76%)
5	Neighborhood 3 (26.96%)
6	Neighborhood 3 (47.05%)
7	Neighborhood 8 (17.75%)
8	Neighborhood 5 (33.75%)

Clustering algorithm vizualizations have come out decently.

However Neigborhood's 3 (OldTown,SWISU) and 5 (ClearCr,Crawfor,SawyerW,NWAmes)

have dominated most of the clusters as they have more than 50% share in the

entire dataset.

Part 6 - Linear Regression

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import sklearn.metrics as metrics
import math
```

Columns with higher correlation with Sale Price are chosen to participate in

the prediction process.

```
model_q6 = model.copy()
model_q6_test = model_test.copy()
labelEncoder = preprocessing.LabelEncoder()

columns = model_q6.select_dtypes(exclude=["number","bool_"]).columns.tolist()
print (columns)
for col in columns:
    value = list(model_q6[col].values.astype(str))+\
    list(model_q6_test[col].values.astype(str))
    labelEncoder.fit(value)
    model_q6[col] = labelEncoder.transform(model_q6[col].astype(str))
    model_q6_test[col] = labelEncoder.transform(model_q6_test[col].astype(str))
```

```
final_model_q6 = model_q6.drop(['Id', 'SalePrice'], axis=1)
final_model_q6_test = model_q6_test.drop(['Id'], axis=1)
colsWithHighCorrelation = []
for col in model_q6.columns:
    cor = model_q6[col].corr(model_q6['SalePrice'])
    if (cor > 0.50 and col != 'SalePrice'):
        colsWithHighCorrelation.append(col)
        print(col + '  ::  ' + str(cor))
```

```
for col in colsWithHighCorrelation:
    print('\n\nBuilding Model with ' + col + '\n')
    print('----------------------------------')

    train_q, test_q, train_a, test_a =train_test_split(
        final_model_q6[[col]],model_q6['SalePrice'], test_size=0.3)
    reg = LinearRegression()
    reg.fit(train_q,train_a)
    regTest = reg.predict(test_q)
    print("Training Dataset Accuracy = ", reg.score(train_q, train_a))
    print("Testing Accuracy = ", reg.score(test_q, test_a))
    mae = metrics.mean_absolute_error(test_a,regTest)
    mse = metrics.mean_squared_error(test_a,regTest)
    print ("MAE:                ", round(mae))
    print ("RMSE:               ", round(math.sqrt(mse)))
```

```
Building Model with Neighborhood

----------------------------------
Training Dataset Accuracy =  0.5161443678234168
Testing Accuracy =  0.49435575388978203
MAE:                     36948.0
RMSE:                    50628


Building Model with OverallQual

----------------------------------
Training Dataset Accuracy =  0.64182331717611
Testing Accuracy =  0.6233187483695706
MAE:                     32779.0
RMSE:                    45607


Building Model with YearBuilt

----------------------------------
Training Dataset Accuracy =  0.3008143478327716
Testing Accuracy =  0.2499631852252271
MAE:                     43882.0
RMSE:                    61565


Building Model with YearRemodAdd

----------------------------------
Training Dataset Accuracy =  0.2598728657562981
Testing Accuracy =  0.2897026557260728
MAE:                     47942.0
RMSE:                    69522


Building Model with ExterQual

----------------------------------
Training Dataset Accuracy =  0.4843263699988871
Testing Accuracy =  0.48005040927243503
MAE:                     39203.0
RMSE:                    54418


Building Model with Foundation

----------------------------------
Training Dataset Accuracy =  0.242095929840896
Testing Accuracy =  0.2769177167145952
MAE:                     47736.0
```

```
RMSE:                 64489


Building Model with BsmtQual

----------------------------------
Training Dataset Accuracy =  0.3668685453295358
Testing Accuracy =  0.2895376603075023
MAE:                  43716.0
RMSE:                 59904


Building Model with TotalBsmtSF

----------------------------------
Training Dataset Accuracy =  0.3645993469574984
Testing Accuracy =  0.37202338169333715
MAE:                  45856.0
RMSE:                 61998


Building Model with 1stFlrSF

----------------------------------
Training Dataset Accuracy =  0.375203847477804
Testing Accuracy =  0.3002285158167495
MAE:                  44104.0
RMSE:                 61055


Building Model with GrLivArea

----------------------------------
Training Dataset Accuracy =  0.4702188263100295
Testing Accuracy =  0.4999169581072456
MAE:                  36757.0
RMSE:                 55099


Building Model with FullBath

----------------------------------
Training Dataset Accuracy =  0.3137496710620441
Testing Accuracy =  0.30021993544683934
MAE:                  43278.0
RMSE:                 61629


Building Model with KitchenQual

----------------------------------
Training Dataset Accuracy =  0.44956568669766456
Testing Accuracy =  0.43216875739306826
MAE:                  40303.0
```

```
RMSE:                    57841


Building Model with TotRmsAbvGrd

----------------------------------
Training Dataset Accuracy =  0.2790493451672551
Testing Accuracy =  0.3004811518898418
MAE:                     46336.0
RMSE:                    64796


Building Model with FireplaceQu

----------------------------------
Training Dataset Accuracy =  0.2928703192737856
Testing Accuracy =  0.24208393109074544
MAE:                     49566.0
RMSE:                    68322


Building Model with GarageFinish

----------------------------------
Training Dataset Accuracy =  0.298002439509644
Testing Accuracy =  0.29228156468542743
MAE:                     46437.0
RMSE:                    67265


Building Model with GarageCars

----------------------------------
Training Dataset Accuracy =  0.4143654747602864
Testing Accuracy =  0.4282146156464839
MAE:                     45095.0
RMSE:                    65279


Building Model with GarageArea

----------------------------------
Training Dataset Accuracy =  0.38884466710275045
Testing Accuracy =  0.4203696101348636
MAE:                     42064.0
RMSE:                    61367
```

Even though the columns themselves are highly correlated, the prediction models built are weak and do not have good accuracies. **OverallQual** is an exception and obtains an RMSE of $42772. (Which is way too much)

```
train_q, test_q, train_a, test_a = train_test_split(
    final_model_q6[colsWithHighCorrelation],
    model_q6['SalePrice'], test_size=0.3)
reg = LinearRegression()
reg.fit(train_q,train_a)
regTest = reg.predict(test_q)
print("Training Dataset Accuracy = ", reg.score(train_q, train_a))
print("Testing Accuracy = ", reg.score(test_q, test_a))
mae = metrics.mean_absolute_error(test_a,regTest)
mse = metrics.mean_squared_error(test_a,regTest)
print ("MAE:                 ", mae)
print ("RMSE:                ", math.sqrt(mse))
```

Combination of the top columns from above gives a better model with a RMSE of ~$27K which is a slight improvement only.

Part 7 - External Dataset

Looked up for data from https://www.cityofames.org/home

Found a xlsx document at https://www.cityofames.org/government/departments-divisions-a-h/city-assessor that contains over 22000 records of housing data in AMES.

```
amesDataSet = pd.read_excel('/content/drive/My Drive/house-prices-advanced-regression-
```

```
amesDataSet.shape
```

```
(22232, 91)
```

The data from the sheet can be used to build a model on which the test dataset can be applied and prediction performance can be improved.

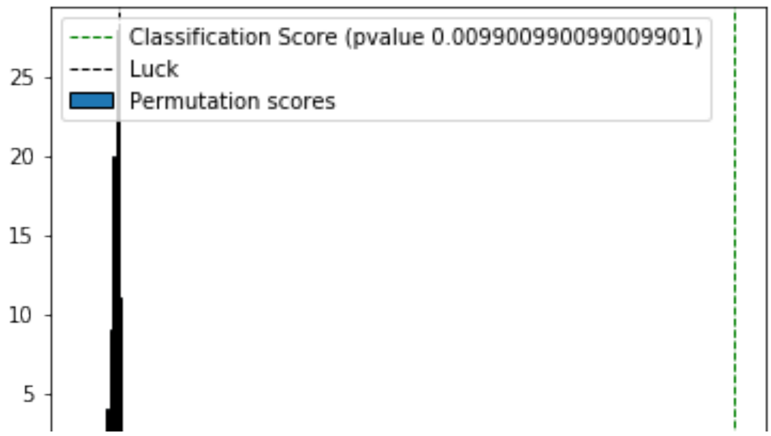
https://locationinc.com/data-catalog/ is a real estate analytics solution that performs analysis on FireRisk™, WaterRisk™, HailRisk™, Crime & CrimeRisk™, Real Estate , Economics and Employment ,Demographics ,Schools

Part 8 - Permutation Test

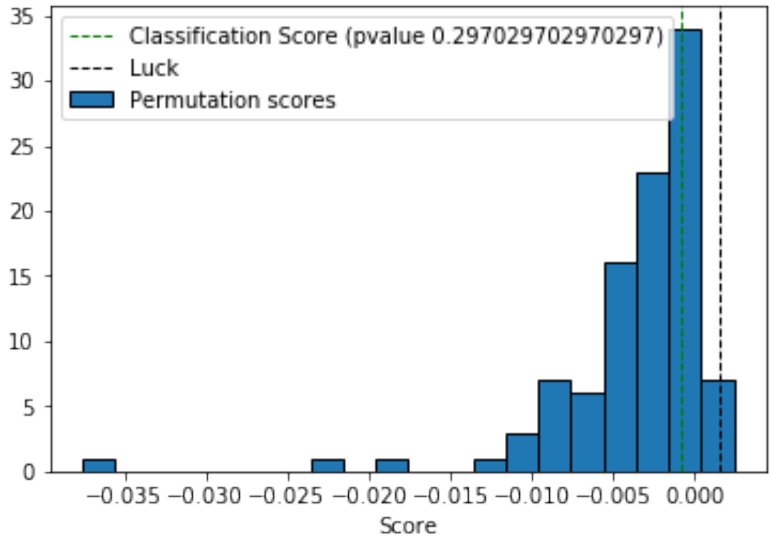
The p-value is given by the percentage of runs (randomized) for which the score obtained is greater than the classification score obtained in the first place.

```
from sklearn.model_selection import permutation_test_score
cols = ['OverallQual','BsmtFinSF2','MSSubClass','HouseStyle','YearBuilt',
        'OverallCond','ExterCond','3SsnPorch','YrSold']
for col in cols:
  x = pd.DataFrame({'col': final_model_q6[col]})
  y = np.log(model_q6['SalePrice'])
  n_classes =  np.unique(y).size

  train_q, test_q, train_a, test_a = train_test_split(x, y, test_size=0.2,
                                                      random_state=42)

  regressor = LinearRegression()
  regressor.fit(train_q, train_a)
  regTest = regressor.predict(test_q)

  prediction_df = pd.DataFrame({'Actual': test_a, 'Predicted': regTest})

  score, permutation_score, p_value = permutation_test_score(
      regressor, x, y, cv=2, n_permutations=100)
  pred = regressor.predict(x)
  print('### ', col, ' ###')
  print('--------------------------------------\n')
  print('Log Root Mean Squared Error :', np.sqrt(
      metrics.mean_squared_log_error(y,pred)))
  print('p-value :', p_value)
  plt.hist(permutation_score, 20, label='Permutation scores',
           edgecolor='black')
  ylim = plt.ylim()
  plt.plot(2 * [score], ylim, '--g', linewidth=1,
           label='Classification Score'
           ' (pvalue %s)' % p_value)
  plt.plot(2 * [1. / n_classes], ylim, '--k', linewidth=1, label='Luck')

  plt.ylim(ylim)
  plt.legend()
  plt.xlabel('Score')
  plt.show()
```

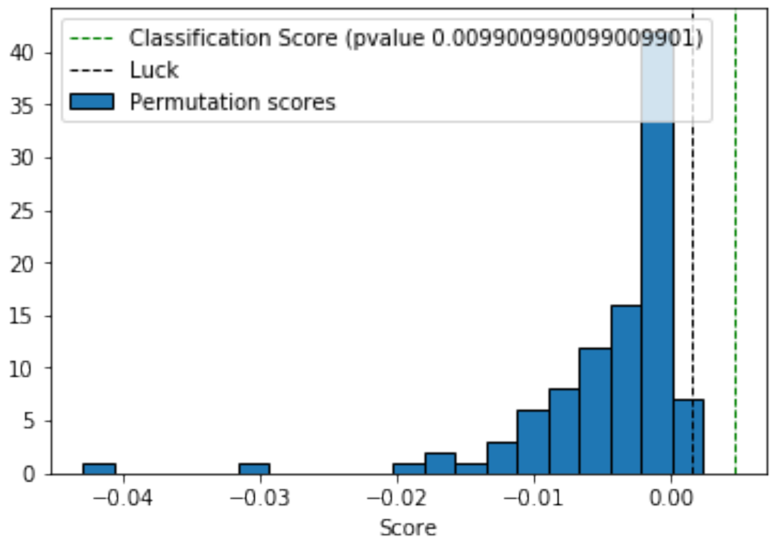
```
###  OverallQual  ###
--------------------------------------

Log Root Mean Squared Error : 0.017629321272943 81
p-value : 0.009900990099009901
```



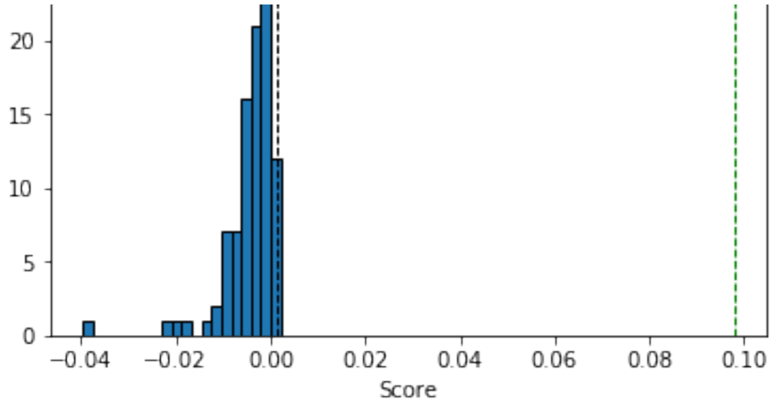

```
###  BsmtFinSF2  ###
-------------------------------------

Log Root Mean Squared Error : 0.030375597255619093
p-value : 0.297029702970297
```



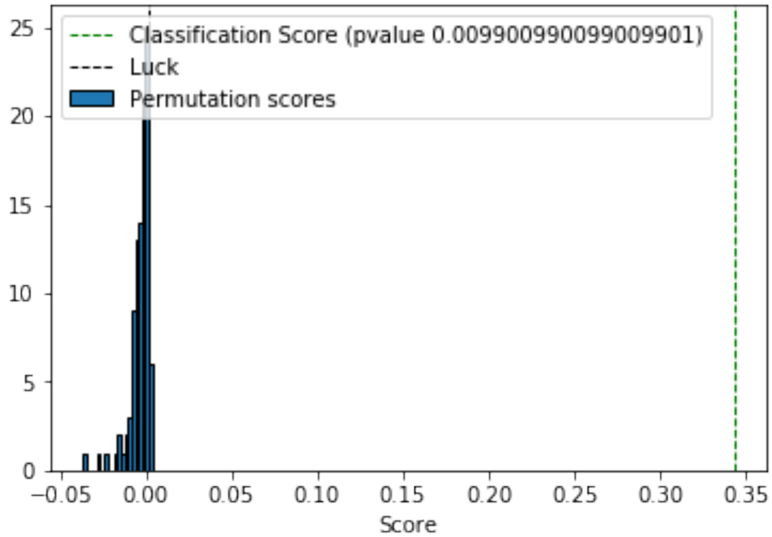

```
###  MSSubClass  ###
-------------------------------------

Log Root Mean Squared Error : 0.030300141342260684
p-value : 0.009900990099009901
```



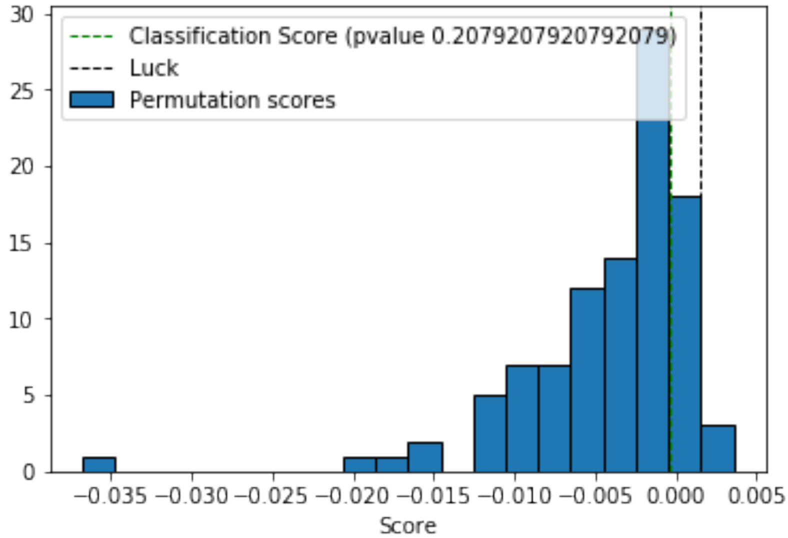

```
###  HouseStyle  ###
-------------------------------------

Log Root Mean Squared Error : 0.028812891150919874
p-value : 0.009900990099009901
```




```
###  YearBuilt  ###
-------------------------------------

Log Root Mean Squared Error : 0.024521866488150326
p-value : 0.009900990099009901
```

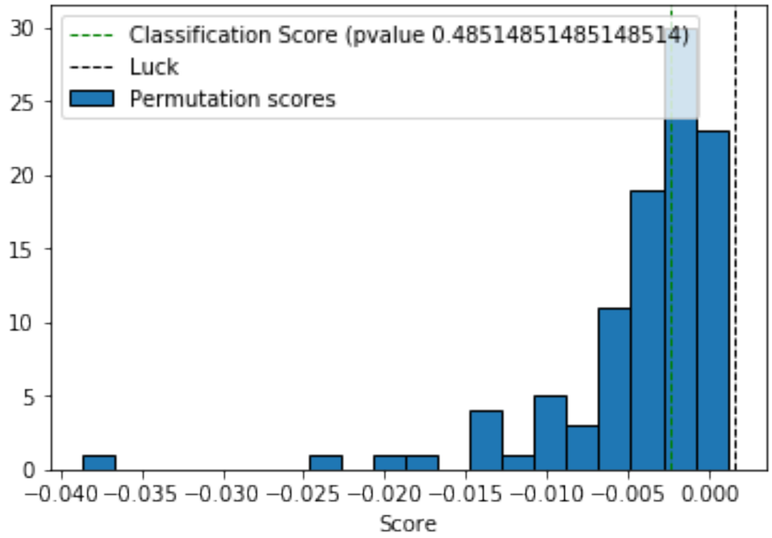



```
###  OverallCond  ###
-------------------------------------

Log Root Mean Squared Error : 0.030355926790238668
p-value : 0.2079207920792079
```

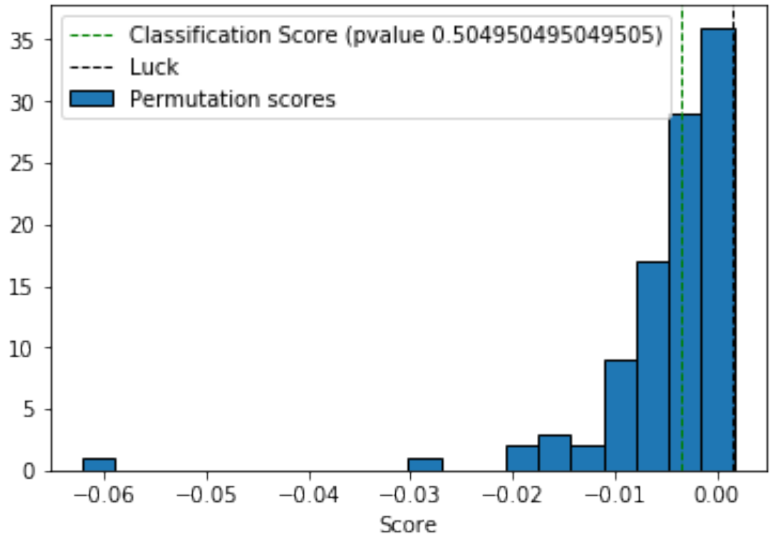



```
###  ExterCond  ###
```

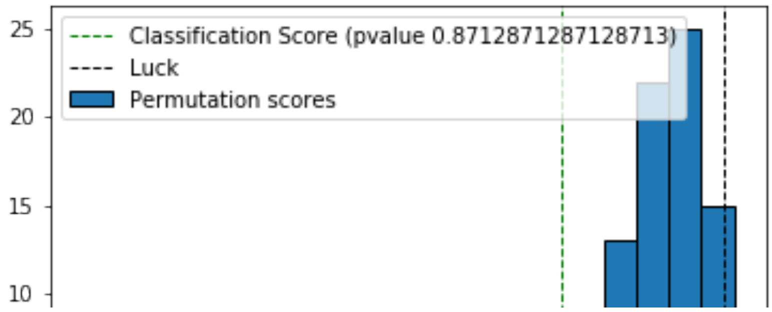
```
-------------------------------------

Log Root Mean Squared Error : 0.030327844932471083
p-value : 0.48514851485148514
```

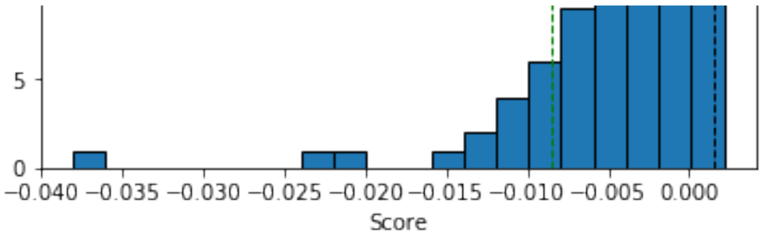



```
###  3SsnPorch  ###
-------------------------------------

Log Root Mean Squared Error : 0.030321019792423538
p-value : 0.504950495049505
```




```
###  YrSold  ###
-------------------------------------

Log Root Mean Squared Error : 0.03035488919659838
p-value : 0.8712871287128713
```





From the above plots it is clear that columns **OverallQual, MSSubClass,HouseStyle,YearBuilt** have good predictive powers and

columns **BsmtFinSF2, OverallCond, ExterCond, 3SsnPorch, YrSold, EnclosedPorchBin** do not possess such powers and will not be able to help in predicition.

Part 9 - Final Result

```
model_q6 = model.copy()
model_q6_test = model_test.copy()

highCorrCols = ['Exterior1st', 'Exterior2nd', 'BsmtFinType2',
 'BsmtFinType2', 'TotalBsmtSF', '1stFlrSF',
 'GarageCars','GarageArea', 'GarageQual',
 'GarageCond', 'GrLivArea', 'TotRmsAbvGrd',
 'FireplaceQu','Fireplaces']
model_q6 = model_q6.drop(highCorrCols, axis=1)
model_q6_test = model_q6_test.drop(highCorrCols, axis=1)

labelEncoder = preprocessing.LabelEncoder()

columns = model_q6.select_dtypes(exclude=["number","bool_"]).columns.tolist()
print (columns)
for col in columns:
    value = list(model_q6[col].values.astype(str)) + list(
        model_q6_test[col].values.astype(str))
    labelEncoder.fit(value)
    model_q6[col] = labelEncoder.transform(model_q6[col].astype(str))
    model_q6_test[col] = labelEncoder.transform(model_q6_test[col].astype(str))
```

```
['RoofStyle', 'RoofMatl', 'Heating', 'CentralAir', 'SaleType', 'SaleConditic
```

```
final_model_q6 = model_q6
final_model_q6_test = model_q6_test
```

```
final_model_q6 = model_q6.drop(['Id', 'SalePrice'], axis=1)
final_model_q6_test = model_q6_test.drop(['Id'], axis=1)
```

```
from sklearn.model_selection import train_test_split
train_q, test_q, train_a, test_a = train_test_split(
    final_model_q6,model_q6['SalePrice'], test_size=0.3)
```

```
#!pip install catboost
```

```
#from catboost import CatBoostClassifier
```

```
#CatBoostmodel = CatBoostClassifier(iterations=300,
#                            task_type="GPU",
#                            devices='0:1')
#CatBoostmodel.fit(train_q,
#          train_a,
#          verbose=False)
```

```
#CatBoostmodel.fit(train_q,train_a)
```

```
#regTest = CatBoostmodel.predict(test_q)
```

```
#final_model_q6_test.shape
```

```
#test.shape
```

```
#finalRegTest = CatBoostmodel.predict(final_model_q6_test)
```

```
#submission = pd.DataFrame({'Id':test['Id'], 'SalePrice': finalRegTest[:,0]})
```

```
#from google.colab import files
#submission.to_csv('submission.csv')
#files.download('submission.csv')
```

Kaggle Link: https://www.kaggle.com/rajat994/competitions

Highest Rank: **2083/4844**

Score: RMSE of **0.13317**

Number of entries: **12**

Proof of submission

Kaggle profile link: https://drive.google.com/open?id=1hvbN89zmmGuYfX4WzZihWyPpj0begKqL

Kaggle Submission proof: https://drive.google.com/open?id=1-qdnkU4cAASJrcuNHyBQqqB8-QTOXs0k

References

https://datascience.stackexchange.com/questions/31746/how-to-include-labels-in-sns-heatmap

https://seaborn.pydata.org/examples/heatmap_annotation.html

https://stackoverflow.com/questions/33779748/set-max-value-for-color-bar-on-seaborn-heatmap

https://chrisalbon.com/python/data_wrangling/pandas_list_unique_values_in_column/

https://stackoverflow.com/questions/26097916/convert-pandas-series-to-dataframe

https://stackoverflow.com/questions/41509936/append-pandas-series-to-dataframe-as-a-column

https://stackoverflow.com/questions/30482071/how-to-calculate-mean-values-grouped-on-another-column-i

https://stackoverflow.com/questions/31069191/simple-line-plots-using-seaborn

https://stackoverflow.com/questions/10202570/find-row-where-values-for-column-is-maximal-in-a-pandas-d

https://cmdlinetips.com/2018/04/how-to-drop-one-or-more-columns-in-pandas-dataframe/

https://stackoverflow.com/questions/13851535/delete-rows-from-a-pandas-dataframe-based-on-a-conditior

https://stackoverflow.com/questions/25039626/how-do-i-find-numeric-columns-in-pandas

https://dzone.com/articles/pandas-find-rows-where-columnfield-is-null

https://stackoverflow.com/questions/23748995/pandas-dataframe-column-to-list

https://stackoverflow.com/questions/41969986/how-to-compare-two-values-in-series-not-the-series-objects

https://stackoverflow.com/questions/11707586/how-do-i-expand-the-output-display-to-see-more-columns

https://stackoverflow.com/questions/26540035/rotate-label-text-in-seaborn-factorplot

https://stackoverflow.com/questions/31460146/plotting-value-counts-in-seaborn-barplot

https://stackoverflow.com/questions/22470690/get-list-of-pandas-dataframe-columns-based-on-data-type

http://varianceexplained.org/statistics/interpreting-pvalue-histogram/

https://www.machinelearningplus.com/plots/matplotlib-histogram-python-examples/

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

https://catboost.ai/docs/concepts/python-usages-examples.html

https://towardsdatascience.com/machine-learning-algorithms-part-9-k-means-example-in-python-f2ad05ed5