CSE519_HW3_Aditya

October 21, 2019

1 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.

[0]:
```
import pandas as pd
import seaborn as sns
import numpy as np
from scipy.stats import skew
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
%matplotlib inline
```

[2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id
=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redire
ct_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googl
eapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%2
0https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2
Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

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

[0]:
```
DATA_DIR = '/content/drive/My Drive/dsf/Asg-3/'
```

[0]:
```
train = pd.read_csv(DATA_DIR+'housing_data/train.csv')
```

[0]:
```
test = pd.read_csv(DATA_DIR+'housing_data/test.csv')
```

[6]:
```
train.head()
```

[6]:
```
   Id  MSSubClass MSZoning  ...  SaleType  SaleCondition SalePrice
0   1          60       RL  ...        WD         Normal    208500
1   2          20       RL  ...        WD         Normal    181500
2   3          60       RL  ...        WD         Normal    223500
3   4          70       RL  ...        WD        Abnorml    140000
4   5          60       RL  ...        WD         Normal    250000

[5 rows x 81 columns]
```

[0]:
```

```

1.1 Part 1 - Pairwise Correlations

[0]:
```
# TODO: show visualization
```

[0]:
```
train.columns
```

[0]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

Finding out numeric columns first and then by looking at the description of the columns , slecting a few interesting ones

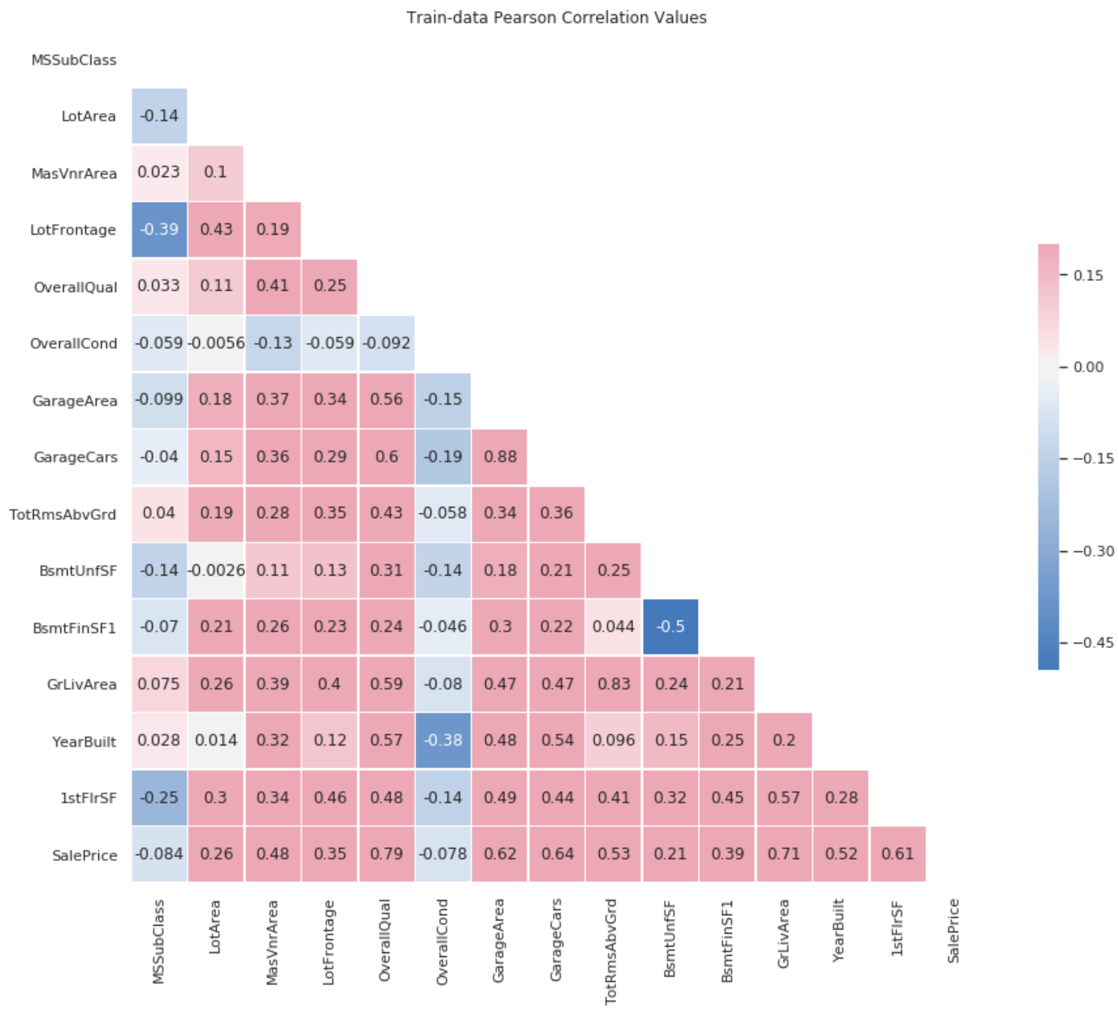
[0]:
```
len(train._get_numeric_data().columns),train._get_numeric_data().columns
```

[0]:
```
(38, Index(['Id', 'MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual',
        'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1',
        'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF',
        'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
        'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd',
        'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF',
        'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea',
        'MiscVal', 'MoSold', 'YrSold', 'SalePrice'],
       dtype='object'))
```

[0]:
```python
cols = ["MSSubClass", "LotArea", "MasVnrArea","LotFrontage", "OverallQual",
        "OverallCond", "GarageArea", "GarageCars",
"TotRmsAbvGrd","BsmtUnfSF", "BsmtFinSF1","GrLivArea",
"YearBuilt","1stFlrSF","SalePrice"]
```

[0]:
```python
corr = train[cols].corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
plt.figure(figsize=(15,12))
plt.title('Train-data Pearson Correlation Values')
cmap = sns.diverging_palette(250, 5,n=19, as_cmap=True)
sns.heatmap(corr, annot=True, mask=mask, cmap=cmap, vmax=.2, center=0,␣
 ↪square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2a732e7c50>



[0]:
```
# plt.figure(figsize=(12,10))
# plt.title('Train-data Pearson Correlation Values')

# sns.heatmap(train[cols].corr())
# plt.show()
```

[0]:
```
```

[0]:
```
# plt.figure(figsize=(12,10))
# plt.title('Test-data Pearson Correlation Values')
# sns.heatmap(test[cols].corr())
```

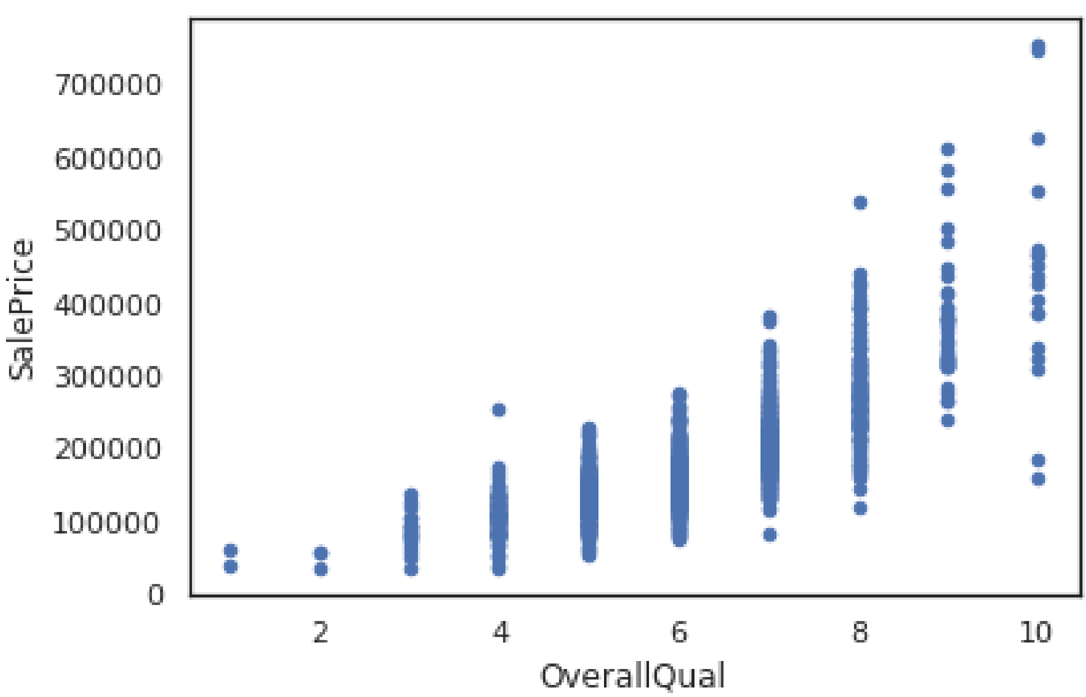
Discuss most positive and negative correlations.

- Most Positive
- From the correlation matrix we can see that most positive correlation is between 'SalePrice' and 'OverallQual'.

[0]:
```
train.plot.scatter(y='SalePrice',x='OverallQual')
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
```

[0]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2a72f32748>
```

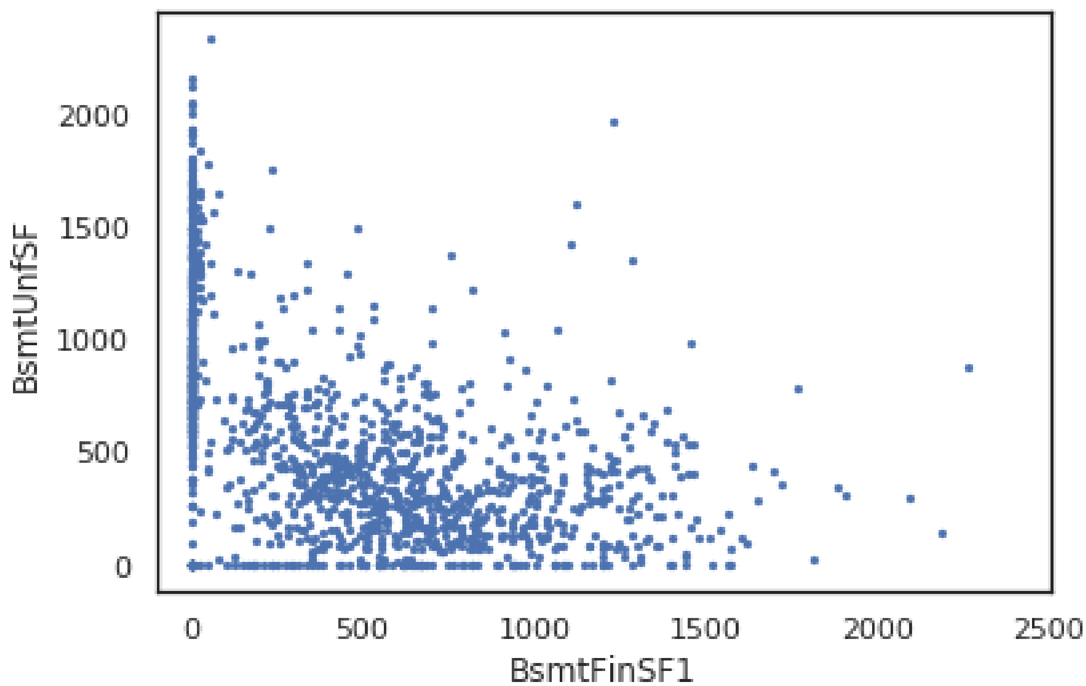


- Most Negative
- From the correlation matrix we can see that most negative correlation is between 'BsmtFinsSF1' and 'BsmtUnfSF'.

[0]:
```
train.plot.scatter(y='BsmtUnfSF',x='BsmtFinSF1',s=5,xlim=(-100,2500))
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
```

[0]: `<matplotlib.axes._subplots.AxesSubplot at 0x7f2a72c42b00>`



[0]:
```
```

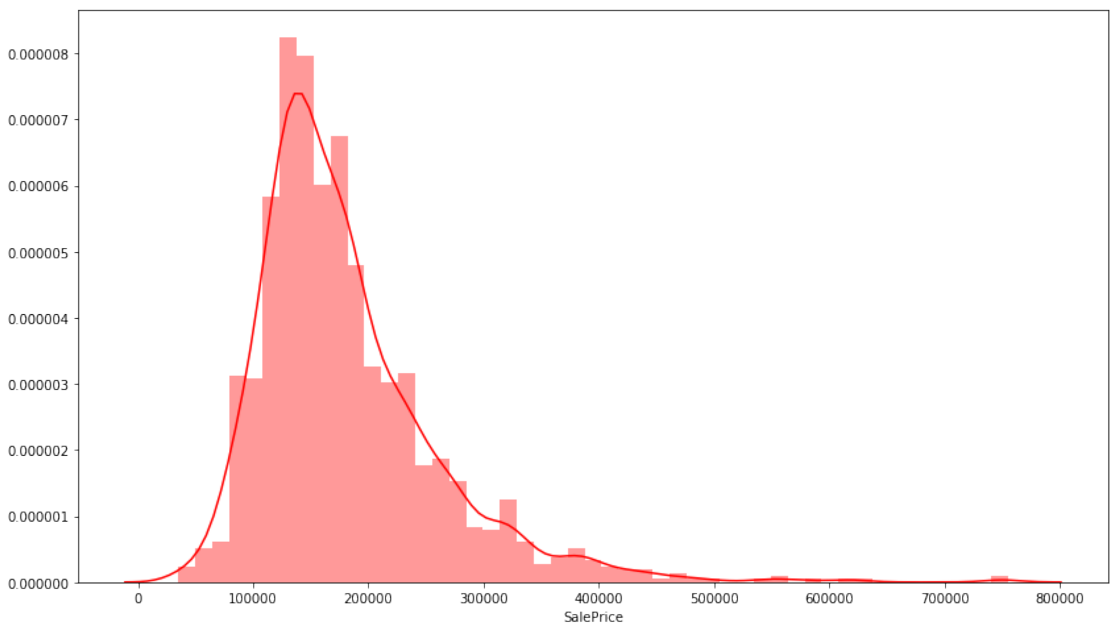
1.2 Part 2 - Informative Plots

1.2.1 1. SalesPrice Distribution

[0]:
```
train.head()
```

[0]:
```
# TODO: code to generate Plot 1
plt.figure(figsize=(14, 8))
sns.distplot(y,color='red')
```

[0]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcaba1cf978>
```



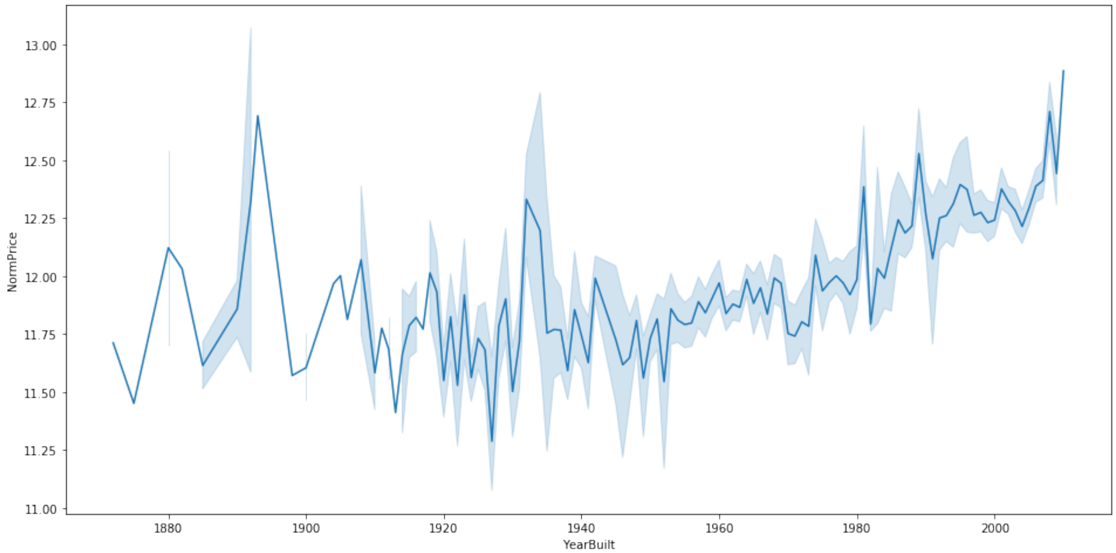
What interesting properties does Plot 1 reveal?

This plot shows that the SalePrice is a Right-skewed distribution. Majority of values are b/w 0 to 300000 but there are some outliers with high values, causing the mean to shift towards higer value than median.

1.2.2 2. Normalized SalePrice v/s YearBuilt

[0]:
```
# TODO: code to generate Plot 2
plt.figure(figsize=(16,8))
sns.lineplot(x="YearBuilt", y=norm_y, data=train)
```

[0]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcaba1bacc0>
```



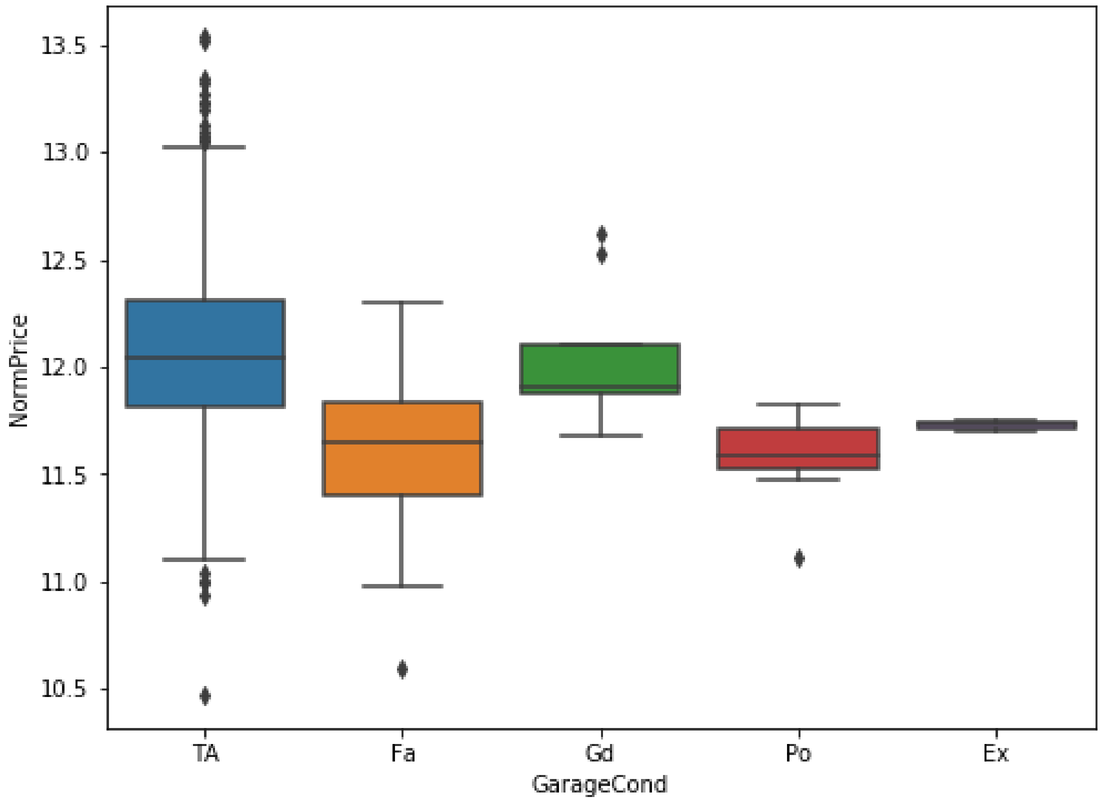
What interesting properties does Plot 2 reveal?

We can see that there are certain peaks in the distribution which decreases from 1890 to 1970 and then again increases from there. There is a general trend that after 1950 SalePrice are increasing, this may be due to the rising inflation and world-war-2 ending. Though we can't conclude that relation b/w SalePrice and YearBuilt is linear

1.2.3 3. Normalized SalePrice vs GarageCond

[0]:
```
# TODO: code to generate Plot 3
plt.figure(figsize=(8,6))
sns.boxplot(x="GarageCond", y=norm_y, data=train)
```

[0]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcab9d09748>
```



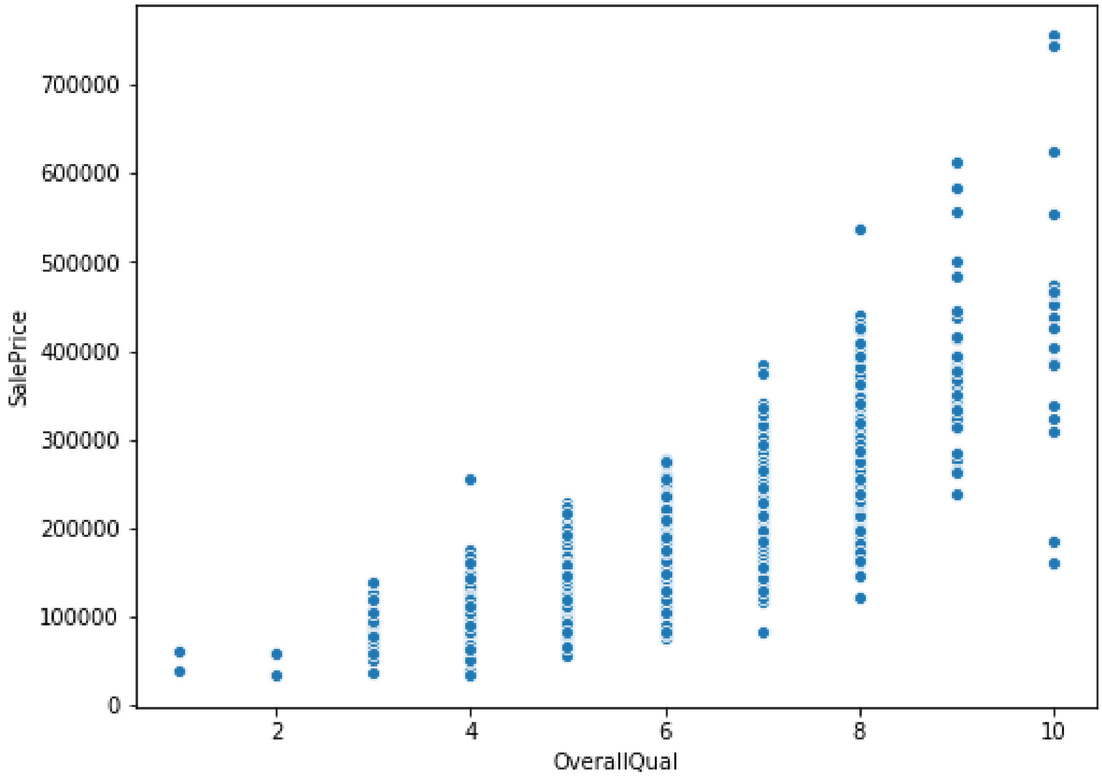
What interesting properties does Plot 3 reveal?

- This shows that each Garage Condition has well-defined range of SalePrice and and corresponding medians are also distinct. Thus this feature can help predicting the prices better.

1.2.4 4. SalesPrice v/s OverallQual

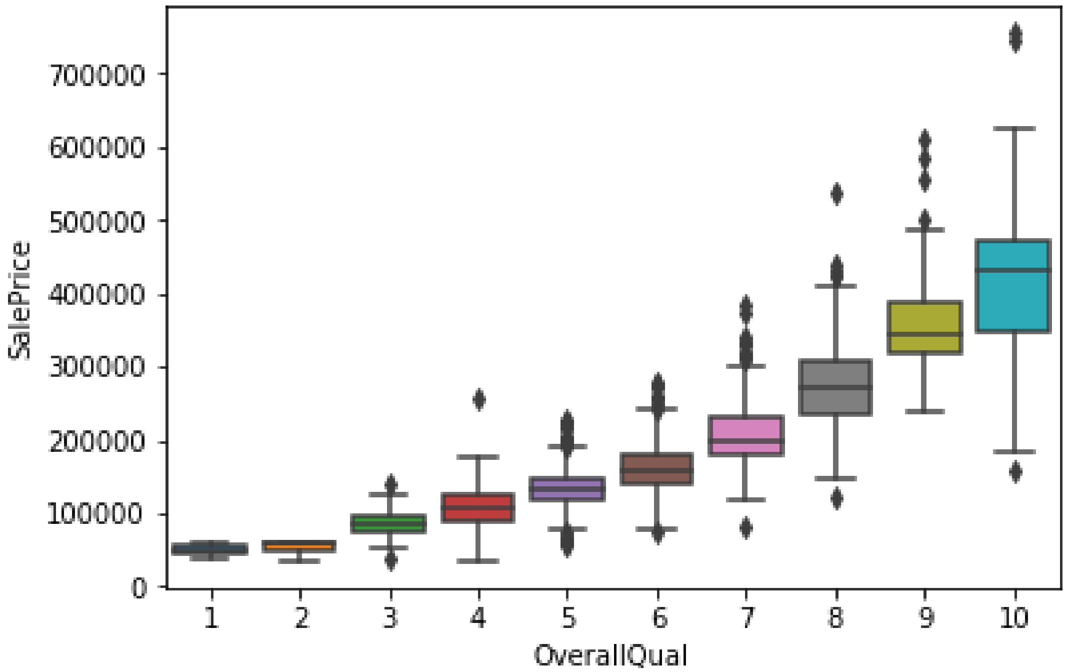
[0]:
```
# TODO: code to generate Plot 4
plt.figure(figsize=(8,6))
sns.scatterplot(x='OverallQual',y=y,data=train)
# sns.boxplot(x="OverallQual", y=y, data=train)
```

[0]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcab9729b70>
```



[0]:
```
sns.boxplot(x="OverallQual", y=y, data=train)
```

[0]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcab9661cf8>
```

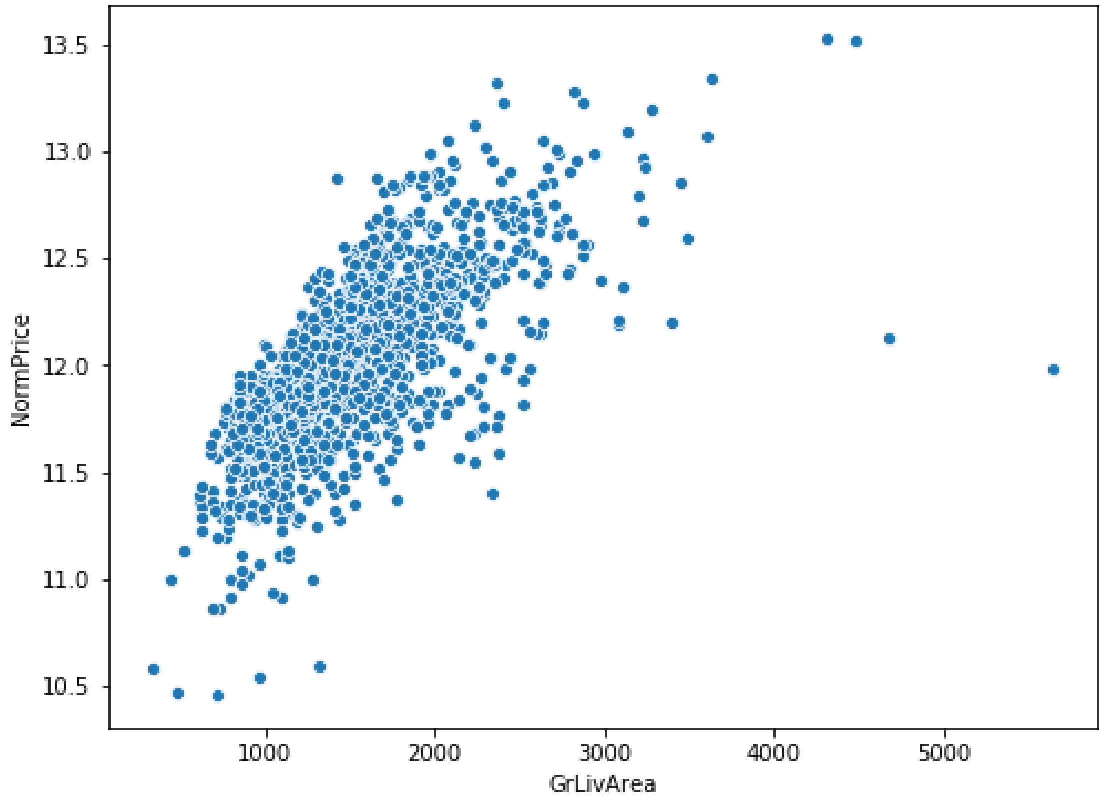


What interesting properties does Plot 4 reveal? * OverallQual and SaleProce are positively co-related. Higher the OverallQual results in higher SalePrice

1.2.5 5. Normalized SalePrice v/s GrLivArea

[0]:
```
# TODO: code to generate Plot 5
plt.figure(figsize=(8,6))
sns.scatterplot(x='GrLivArea',y=norm_y,data=train,ci=85)
```

[0]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcab92365f8>
```



What interesting properties does Plot 5 reveal?

This plot shows us a positive co-relation b/w 'GrLivArea' and 'Normalized SalePrice'. SalePrice is increasing as the Area increases But we also see variation amongst SalePrice for the same GrLivArea, which depicts that saleprice also depends other factors.

1.3 Part 3 - Handcrafted Scoring Function

[0]:
```
# TODO: code for scoring function
```

- Numerical skewed features are already normalized in Part-6 , and taking important features from Xgboost model and Correlation values with 'SalePrice'

[0]:
```
score_cols =␣
↪['GarageCars','OverallQual','BsmtQual_Ex','GarageType_Attchd','KitchenQual_Ex','GrLivArea','F
score_train = X_train[score_cols]
score_train['Id'] = X_train['Id']
score_train['NormalizedPrice'] = norm_y
score_train['SalePrice'] = y
```

[0]:
```
score_train['score'] = (score_train['GarageCars'] *0.7 +␣
↪score_train['OverallQual']*0.28 + score_train['BsmtQual_Ex']*0.24
```

```
+ score_train['GarageType_Attchd']*0.25 + score_train['KitchenQual_Ex']*0.21 +␣
↪score_train['GrLivArea']*0.18 + score_train['Fireplaces']*0.18)/norm_y
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2:
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
```

What is the ten most desirable houses?

[0]:
```
score_train = score_train.set_index('Id').reset_index()
```

[0]:
```
score_train.sort_values('score', ascending=False).head(10)
```

[0]:

	Id	GarageCars	OverallQual	...	NormalizedPrice	SalePrice	score
1298	1299	2.0	10	...	11.982935	160000	0.583736
523	524	3.0	10	...	12.126764	184750	0.581447
1442	1443	3.0	10	...	12.644331	310000	0.565382
1373	1374	3.0	10	...	13.053015	466500	0.565212
309	310	3.0	9	...	12.793862	360000	0.564579
994	995	3.0	10	...	12.729324	337500	0.559410
224	225	3.0	10	...	12.864243	386250	0.558172
610	611	3.0	9	...	12.653962	313000	0.558048
440	441	3.0	10	...	13.226725	555000	0.556540
825	826	3.0	10	...	12.861001	385000	0.556384

[10 rows x 11 columns]

What is the ten least desirable houses?

[0]:
```
score_train.sort_values('score').head(10)
```

[0]:

	Id	GarageCars	OverallQual	...	NormalizedPrice	SalePrice	score
533	534	0.0	1	...	10.579005	39300	0.125394
375	376	0.0	1	...	11.018646	61000	0.136626
636	637	0.0	2	...	11.002117	60000	0.176644
1326	1327	0.0	3	...	11.277216	79000	0.180675
620	621	0.0	3	...	11.112463	67000	0.185134
710	711	0.0	3	...	10.859018	52000	0.186642
250	251	0.0	3	...	11.245059	76500	0.189558
88	89	0.0	3	...	11.350418	85000	0.190265
968	969	0.0	3	...	10.542733	37900	0.197077
528	529	0.0	4	...	11.362114	86000	0.200072

[10 rows x 11 columns]

Describe your scoring function and how well you think it worked.

- The formulated scoring function take following Variables in account - ['GarageCars', 'OverallQual','BsmtQual_Ex', 'GarageType_Attchd', 'KitchenQual_Ex', 'GrLivArea', 'Fireplaces', 'Normalized_SalePrice']
- These columns are already normalized and I have considered desirability as an optimum price range - not too low and not too high.
- The columns are selected according to feature importances from a good performing XgBoost model(Part-9) and the Pearson correlation of different varaibles with SalePrice from 'Part-1' and weights are assigned accordingly. Finally the score is divided by Normalized_SalePrice so that 'cost' factor comes into play. I have used "Normalized" Sale price so that score does not reduces to too low values, making it hard to distinguish between different scores.
- **Interesting fact** - The scoring function has performed well considering both value and cost.If you see the highest desirable and lowest desirable house, both of them have lowest prices compared to other top-10 desirable and least-10 desirable houses respectively. Also the 3rd most desirable house has price comparable to 'least' desirable house, stating the importances of other variables in desirability.

1.4 Part 4 - Pairwise Distance Function

[0]:
```
from scipy.spatial.distance import pdist
from scipy.spatial.distance import squareform
from scipy.cluster.hierarchy import ward
from scipy.cluster.hierarchy import fcluster
from sklearn.manifold import TSNE
```

[0]:
```
dist_cols = ['OverallQual', 'GarageCars', 'FullBath','Fireplaces',
             'GrLivArea', 'GarageArea', '1stFlrSF', 'Id']
dist_train = pd.DataFrame(X_train, columns = dist_cols)
dist_train['NormalizedPrice'] = norm_y
dist_train = dist_train.set_index('Id').reset_index()
dist_train.head()
```

[0]:
```
   Id  OverallQual  GarageCars  ...  GarageArea  1stFlrSF  NormalizedPrice
0   1            7         2.0  ...       548.0  6.753438        12.247699
1   2            6         2.0  ...       460.0  7.141245        12.109016
2   3            7         2.0  ...       608.0  6.825460        12.317171
3   4            7         3.0  ...       642.0  6.869014        11.849405
4   5            8         3.0  ...       836.0  7.044033        12.429220

[5 rows x 9 columns]
```

[0]:
```
len(pdist(dist_train, metric = 'cosine')), (dist_train.shape[0])
```

[0]:
```
(1065070, 1460)
```

[0]:
```
pair_dist = pdist(dist_train, metric = 'cosine')
```

[0]:
```
len(squareform(pair_dist)),len(squareform(pair_dist)[0])
```

[0]:
```
(1460, 1460)
```

[0]:
```
squareform(pair_dist)[0]
```

[0]:
```
array([0.00000000e+00, 2.09251094e-05, 1.06923259e-05, ...,
       8.27602267e-01, 8.35629222e-01, 8.12212515e-01])
```

[0]:
```
pairwise = pd.DataFrame(squareform(pair_dist), columns = dist_train.index,
             index = dist_train.index).unstack()
```

[0]:
```
# pairwise
```

[0]:
```
pairwise.index.rename(["H-1#", "H-2#"],inplace=True)
```

[0]:
```
pairwise.shape
```

[0]:
```
(2131600,)
```

[0]:
```
pairwise = pairwise.to_frame('Cosine_distance').reset_index()
```

[0]:
```
pairwise.head()
```

[0]:
```
   H-1#  H-2#  Cosine_distance
0     0     0         0.000000
1     0     1         0.000021
2     0     2         0.000011
3     0     3         0.000026
4     0     4         0.000060
```

[0]:
```
pairwise[(pairwise['Cosine_distance'] < 0.02) & (pairwise['H-1#'] !=␣
 ↪pairwise['H-2#'])]\
.sort_values('Cosine_distance').head(10)
```

[0]:
```
         H-1#  H-2#  Cosine_distance
1661151  1137  1131     2.726037e-08
1652397  1131  1137     2.726037e-08
1935837  1325  1337     2.836413e-08
1953345  1337  1325     2.836413e-08
1669911  1143  1131     3.173942e-08
1652403  1131  1143     3.173942e-08
1669917  1143  1137     3.833399e-08
1661163  1137  1143     3.833399e-08
1474170  1009  1030     4.981400e-08
1504809  1030  1009     4.981400e-08
```

[0]:
```
dist_train[(dist_train.Id == 1138 ) | (dist_train.Id == 1132 )]
```

[0]:
```
        Id  OverallQual  GarageCars  ...  GarageArea  1stFlrSF  NormalizedPrice
1131  1132            5         0.0  ...         0.0  6.882437        11.445727
1137  1138            5         0.0  ...         0.0  6.660575        11.451061

[2 rows x 9 columns]
```

[0]:
```
dist_train[(dist_train.Id == 1326 ) | (dist_train.Id == 1338 )]
```

[0]:
```
        Id  OverallQual  GarageCars  ...  GarageArea  1stFlrSF  NormalizedPrice
1325  1326            4         0.0  ...         0.0  6.680855        10.915107
1337  1338            4         0.0  ...         0.0  6.542472        10.868587

[2 rows x 9 columns]
```

[0]:
```
pairwise[(pairwise['Cosine_distance'] > 0.5) & (pairwise['H-1#'] !=␣
 ↪pairwise['H-2#'])]\
.sort_values('Cosine_distance', ascending=False).head(10)
```

[0]:
```
         H-1#  H-2#  Cosine_distance
1449        0  1449         0.997838
2115540  1449     0         0.997838
1453        0  1453         0.997830
2121380  1453     0         0.997830
1337        0  1337         0.997827
1952020  1337     0         0.997827
1935960  1326     0         0.997825
1326        0  1326         0.997825
1325        0  1325         0.997821
1934500  1325     0         0.997821
```

[0]:
```
dist_train[(dist_train.Id == 1 ) | (dist_train.Id == 1450 )]
```

[0]:
```
        Id  OverallQual  GarageCars  ...  GarageArea  1stFlrSF  NormalizedPrice
0        1            7         2.0  ...       548.0  6.753438        12.247699
1449  1450            5         0.0  ...         0.0  6.447306        11.429555

[2 rows x 9 columns]
```

[0]:
```
dist_train[(dist_train.Id == 1 ) | (dist_train.Id == 1454 )]
```

[0]:
```
        Id  OverallQual  GarageCars  ...  GarageArea  1stFlrSF  NormalizedPrice
0        1            7         2.0  ...       548.0  6.753438        12.247699
1453  1454            5         0.0  ...         0.0  7.039660        11.344519

[2 rows x 9 columns]
```

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

- The distance function is working quite well , as it can be seen that I have taken 2 pairs each of nearest houses and furthest houses. Let us see the nearest houses first - House-Id# 1138:1132 and 1326:1338 , Both the pairs have almost similar values for the differnet variables considered. Now lets see 2 pairs from furthest houses - 1:1450 and 1:1453, it can be seen that these house-ids have different values for the variables, where GarageArea is the most distinguishing factor.
- Thus we can argue that the distance function has captured the similarities and dissimilarities quite well.
- Since value ranges in different variables are not normalized in same range , some variables are influencing the distance function more than others.

- I have also tried with original 'SalePrice' and eucledian distance formula but that seems to degrade the results. The reason was the 'SalePrice' magnitude, it was heavily dominating the distance values.

[0]:
```
# TODO: code for distance function
```

1.5 Part 5 - Clustering

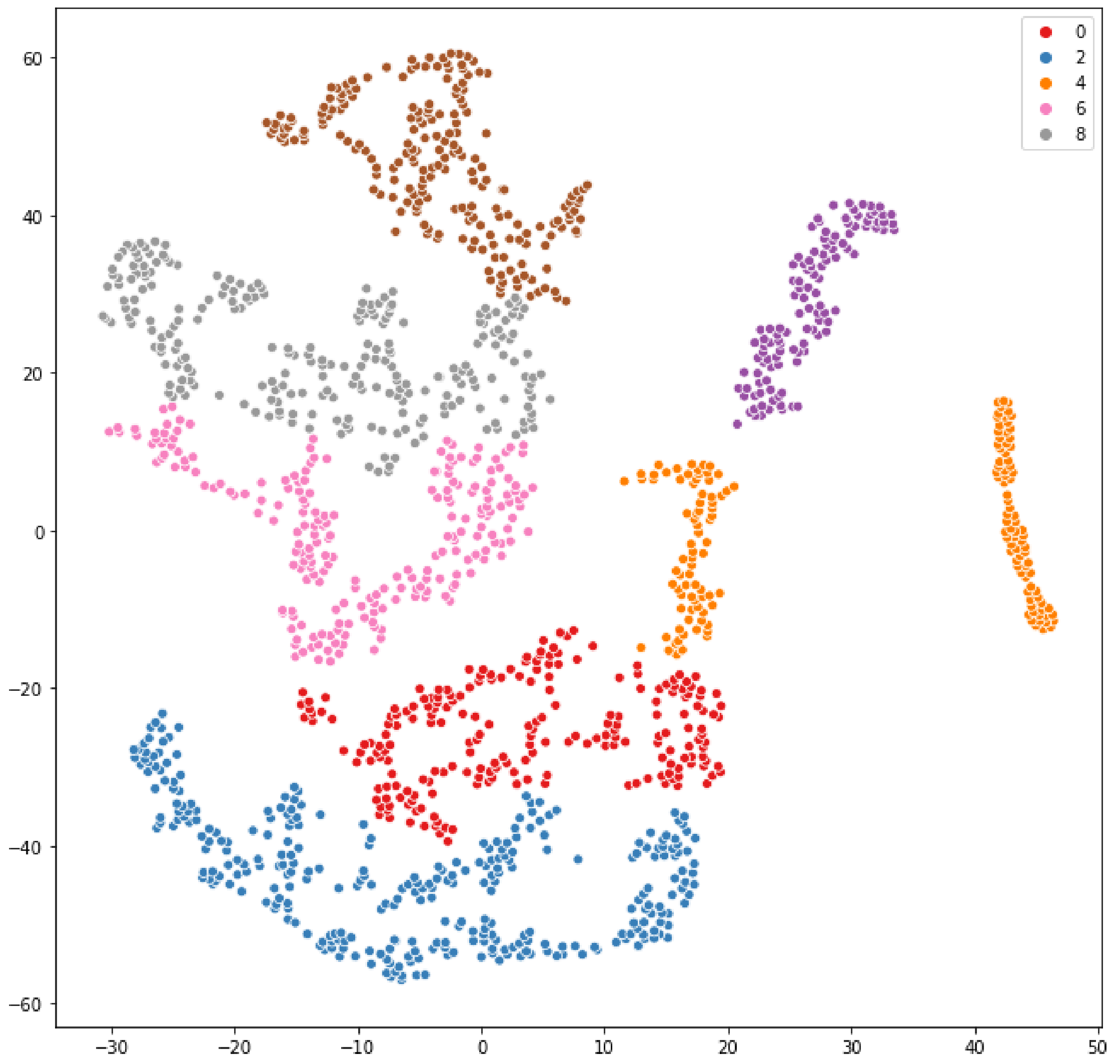
[0]:
```
from sklearn.preprocessing import Normalizer
```

[0]:
```
# TODO: code for clustering and visualization
norm = Normalizer()

norm_train = norm.fit_transform(dist_train)
```

[0]:
```
tsne = TSNE(random_state = 13)
tsne_data = tsne.fit_transform(dist_train)
plt.figure(figsize=(11,11))
vb = fcluster(ward(tsne_data), t=300, criterion='distance')
sns.scatterplot(x=tsne_data[:,0], y=tsne_data[:,1], hue=vb, palette="Set1")
```

[0]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcaba8fb0f0>
```




```
[0]: drop_train = dist_train.drop(columns=['Id'],axis=1)
```

```
[0]: drop_train.head()
```

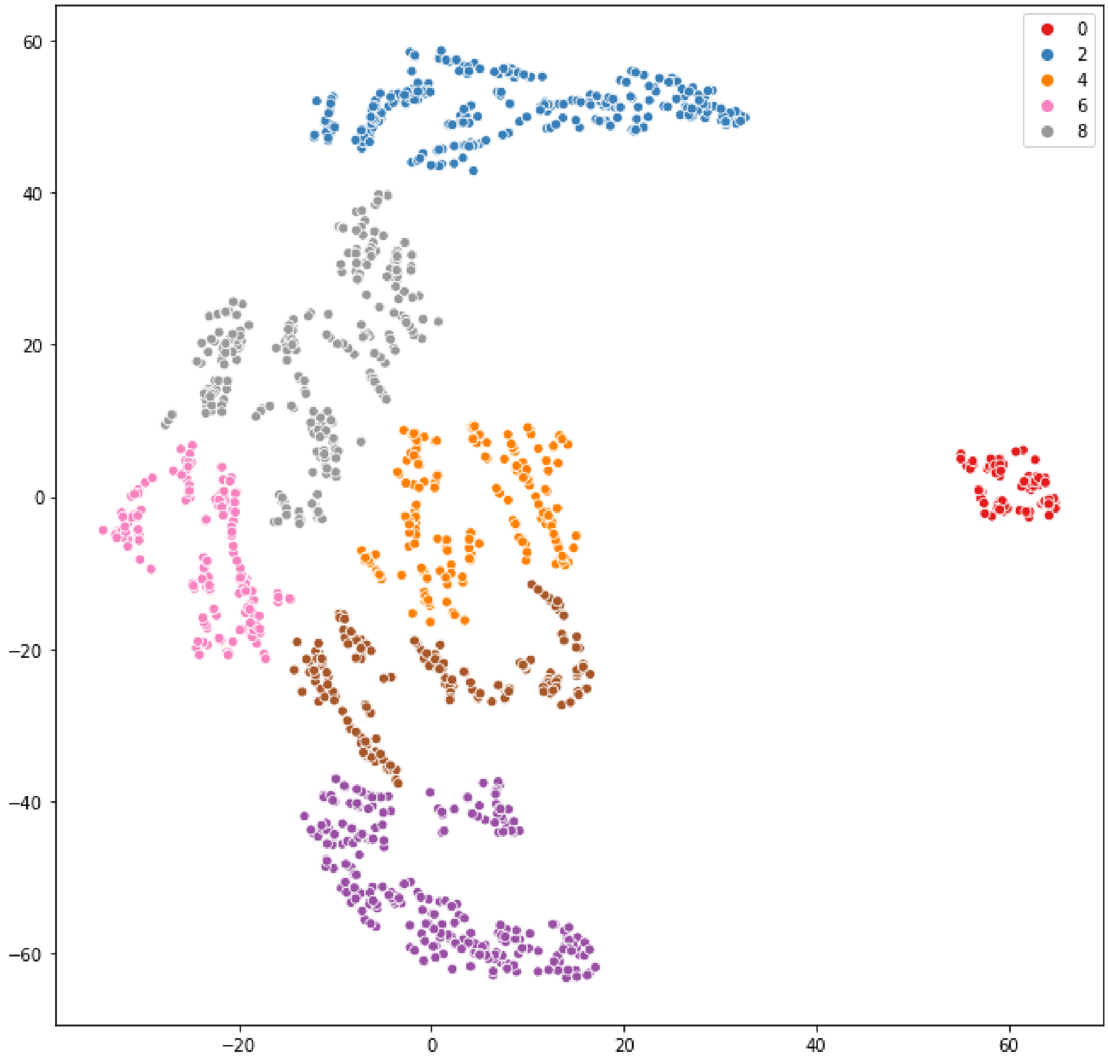
```
[0]:    OverallQual  GarageCars  FullBath  ...  GarageArea  1stFlrSF  NormalizedPrice
     0            7         2.0         2  ...       548.0  6.753438        12.247699
     1            6         2.0         2  ...       460.0  7.141245        12.109016
     2            7         2.0         2  ...       608.0  6.825460        12.317171
     3            7         3.0         1  ...       642.0  6.869014        11.849405
     4            8         3.0         2  ...       836.0  7.044033        12.429220

     [5 rows x 8 columns]
```

```
[0]: tsne = TSNE(random_state = 13, metric="cosine")
     tsne_data = tsne.fit_transform(drop_train)
     plt.figure(figsize=(11,11))
```

```
vb = fcluster(ward(tsne_data), t=300, criterion='distance')
sns.scatterplot(x=tsne_data[:,0], y=tsne_data[:,1], hue=vb, palette="Set1")
```

[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7fcaba823748>



How well do the clusters reflect neighborhood boundaries? Write a discussion on what your clusters capture and how well they work.

- I have plotted two cluster plots , one with Id column and second without Id column. The t-SNE algorithm used here is helpful in visualizination of high dimensional data. It uses dimenionality reduction technique to visualize high dimensional data into smaller dimensions. Both of the plots have resulted in 7 clusters. Following variables have been used 'OverallQual', 'GarageCars', 'FullBath','Fireplaces','GrLivArea', 'GarageArea', '1stFlrSF', 'Id' to generate clusters.Data was already normalized in Part-6'

1.6 Part 6 - Linear Regression

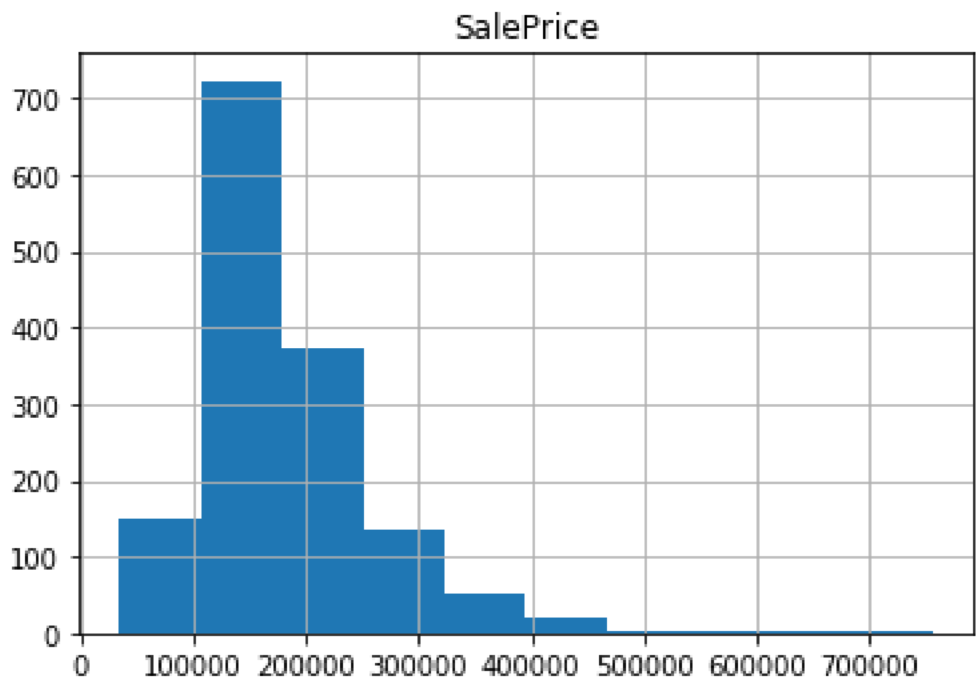
[0]:
```
# TODO: code for linear regression
```

[0]:
```
from sklearn.impute import SimpleImputer

my_imputer = SimpleImputer()
```

[0]:
```
train.hist(column='SalePrice')
```

[0]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fcac2233a20>]],
      dtype=object)
```

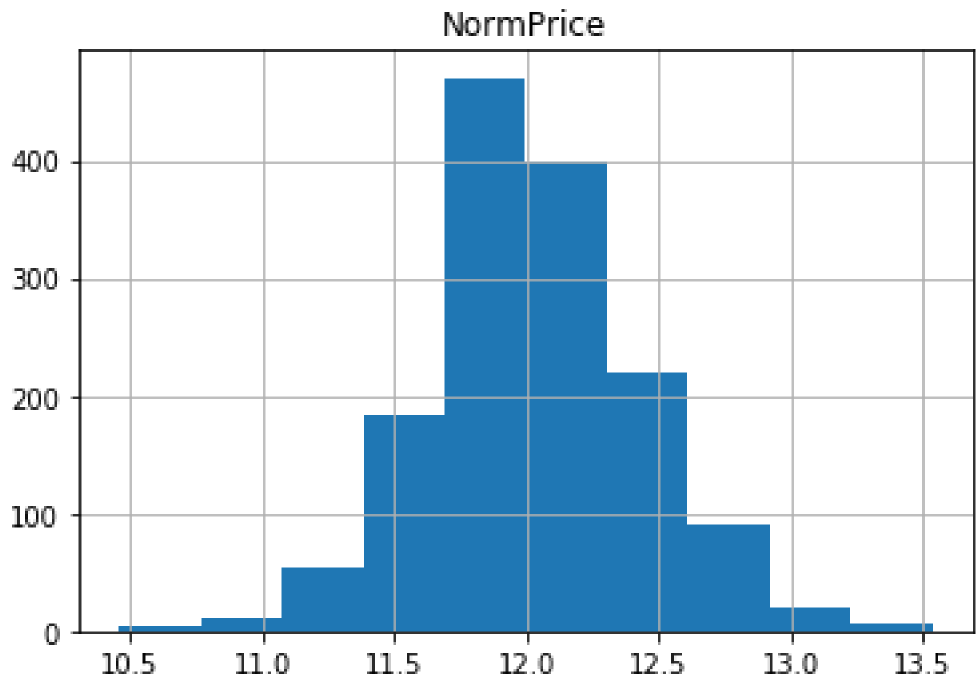


- Since the prediction column 'SalePrice' is skewed , taking log to normalize it.

[0]:
```
train['NormPrice'] = np.log1p(train['SalePrice'])
```

[0]:
```
train.hist(column='NormPrice')
```

[0]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fcac19a8358>]],
      dtype=object)
```



- Checking and normalizing other skewed features now

[0]:
```
norm_y = train['NormPrice']
y = train['SalePrice']
train.drop(labels=['NormPrice','SalePrice'],axis=1,inplace=True)
```

[0]:
```
combined_data = pd.concat([train,test])
```

[0]:
```
numeric_feats = combined_data.dtypes[combined_data.dtypes != "object"].index
 #compute skewness
skewed_feats = train[numeric_feats].apply(lambda x: skew(x.dropna()))
skewed_feats = skewed_feats[skewed_feats > 0.76]
skewed_feats = skewed_feats.index
skewed_feats
```

[0]:
```
Index(['MSSubClass', 'LotFrontage', 'LotArea', 'MasVnrArea', 'BsmtFinSF1',
       'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtHalfBath', 'KitchenAbvGr',
       'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
       'ScreenPorch', 'PoolArea', 'MiscVal'],
      dtype='object')
```

[0]:
```
combined_data[skewed_feats] = np.log1p(combined_data[skewed_feats])
```

- Doing One-Hot Encoding of Categorical variables
- Filling Nan values with Mean

[0]:
```
combined_data = pd.get_dummies(combined_data)
combined_data = combined_data.fillna(combined_data.mean())
```

[0]:
```
combined_data.head()
```

[0]:
```
   Id  MSSubClass  ...  SaleCondition_Normal  SaleCondition_Partial
0   1    4.110874  ...                     1                      0
1   2    3.044522  ...                     1                      0
2   3    4.110874  ...                     1                      0
3   4    4.262680  ...                     0                      0
4   5    4.110874  ...                     1                      0

[5 rows x 289 columns]
```

[0]:
```
X_train = combined_data[:train.shape[0]]
X_test = combined_data[train.shape[0]:]
```

- Evaluating Linear Regresion model

[0]:
```
X_train, X_test, y_train, y_test = train_test_split(X_train, norm_y,
 random_state=17, test_size=.32)
```

[0]:
```
# with default params
lr = LinearRegression()
```

[0]:
```
model = lr.fit(X_train, y_train)
print("R-square : ", model.score(X_test, y_test))
```

```
R-square :  0.8949958211932214
```

[0]:
```
preds = model.predict(X_test)
```

[0]:
```
print("RMSE on normalized y : %f"% mean_squared_error(y_test, preds))
# print("RMSE on original y : %f"% mean_squared_error(np.expm1(y_test), np.
 ↪expm1(preds)))
```

```
RMSE on normalized y : 0.016401
RMSE on original y : 551479094.576008
```

[0]:
```

```

[0]:
```
model = lr.fit(X_train, norm_y)
preds = model.predict(X_test)
```

[0]:
```
# final_preds = np.expm1(preds)
```

[0]:
```
final_preds = np.exp(preds)-1
```

[0]:
```
final_preds
```

[0]: array([121199.00928362, 163670.84544991, 187603.11991847, ...,
       175885.50865188, 120905.80171867, 217019.8413718])

[0]:
```
submission = pd.DataFrame()
submission['Id'] = X_test.Id
```

[0]:
```
submission['SalePrice'] = final_preds
```

[0]:
```
submission.to_csv('./lr_submission.csv',index=False)
```

How well/badly does it work? Which are the most important variables?

[0]:
```
np.argsort(np.abs(lr.coef_))
```

[0]:
```
indexes = np.argsort(np.abs(lr.coef_))[::-1]
indexes
```

[0]:
```
inf = lr.coef_[indexes]

for i in range(0 , 15):
  print(X_train.columns[indexes[i]],"\t",inf[i])
```

```
PoolQC_Fa        -8.583167877978802
PoolQC_Gd        -8.386971390935196
PoolQC_Ex        -8.134720917790235
PoolArea         1.3301460616321892
RoofMatl_ClyTile         -1.2701931597865928
Condition2_PosN          -0.5658302079087277
GrLivArea        0.5459231517337
MiscFeature_Gar2         0.37445633329598427
MSZoning_C (all)         -0.36905893807392254
RoofMatl_Membran         0.2957441293452841
BsmtCond_Fa      -0.29342176770069667
GarageQual_Ex    0.29164398673105124
BsmtQual_TA      -0.28882871120254705
BsmtQual_Gd      -0.28752024138849636
Functional_Sev   -0.2870315233894054
```

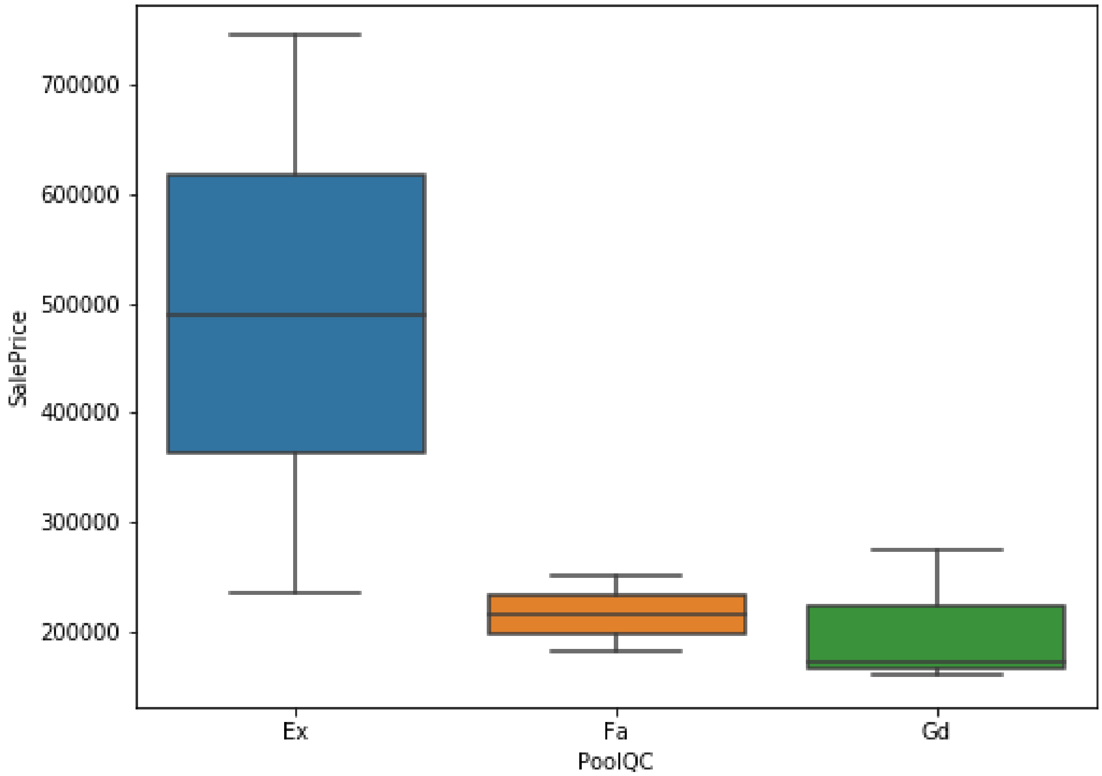
[0]:
```
train['PoolQC'].unique()
```

[0]: array([nan, 'Ex', 'Fa', 'Gd'], dtype=object)

[0]:
```
orig_train = train.copy()
orig_train['SalePrice'] = y
```

[0]:
```
plt.figure(figsize=(8,6))
sns.boxplot(x="PoolQC", y="SalePrice", data=orig_train)
```

[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa4d33f92b0>



We can see that PoolQC values is affecting prices, where Excellent condition is significantly distinguished itself from other two. Hence from above analysis on 68:32 Train-Test split -

- R-square score = 0.89499582119
- RMSE = 0.016401
- PoolQC is the most important variable, the top 10 most important ones are =

  PoolQC_Fa
  PoolQC_Gd
  PoolQC_Ex
  PoolArea
  RoofMatl_ClyTile
  Condition2_PosN
  GrLivArea
  MiscFeature_Gar2
  MSZoning_C (all)
  RoofMatl_Membran

[0]:
```
# train['PoolQC'].unique(),test['PoolQC'].unique()
```

[0]:
```
# for col in X_train.columns:
#   if "ool" in col:
#     print (col)
```

[0]:
```python
# for i in range(0 , 10):
#   print(X_train.columns[indexes[i]],"\t")
```

1.7 Part 7 - External Dataset

[0]:
```python
# TODO: code to import external dataset and test
```

[0]:
```python
## Data Reference - https://www.cityofames.org/government/
↪departments-divisions-a-h/city-assessor/reports
```

[0]:
```python
extra_train = pd.read_excel("./Ames Real Estate Data.xlsx")
```

[0]:
```python
extra_train.head()
```

[0]:
```
    MapRefNo   GeoRefNo  Tier  ...                 Date               Source  
NmbrBRs
0  520400410  520400410     0  ... 2019-06-25 15:13:38  Ames City Assessor
NaN
1  521200150  521200150     0  ... 2019-06-25 15:13:38  Ames City Assessor
3.0
2  521400005  521400005     0  ... 2019-06-25 15:13:38  Ames City Assessor
NaN
3  522100003  522100003     0  ... 2019-06-25 15:13:38  Ames City Assessor
NaN
4  522100004  522100004     0  ... 2019-06-25 15:13:38  Ames City Assessor
NaN

[5 rows x 91 columns]
```

[0]:
```python
extra_train.columns
```

[0]:
```
Index(['MapRefNo', 'GeoRefNo', 'Tier', 'Range', 'Prop_Addr', 'ZngCdPr',
       'ZngCdSc', 'ZngOLPr', 'ZngOLSc', 'ClassPr_S', 'ClassSc_S', 'Legal_Pr',
       'SchD_S', 'TxD_S', 'MA_Ownr1', 'MA_Ownr2', 'MA_Line1', 'MA_Line2',
       'MA_City', 'MA_State', 'MA_Zip1', 'MA_Zip2', 'Rcrd_Yr', 'Rcrd_Mo',
       'Inst1_No', 'Inst1_Yr', 'Inst1_Mo', 'Inst1TPr', 'LndAc_S', 'ImpAc_S',
       'OthAc_S', 'TtlVal_AsrYr', 'ValType', 'X1TPr_D', 'X1TSc_D', 'X2TPr_D',
       'X2TSc_D', 'X1TPr_S', 'X1TSc_S', 'X2TPr_S', 'X2TSc_S', 'LndAcX1S',
       'ImpAcX1S', 'ImpAcX2S', 'HSTtl_D', 'MilVal_D', 'HSTtl_S', 'MilVal_S',
       'AcreX_S1', 'AcreGr', 'AcreNt_S', 'Neighborhood', 'LotArea', 'ParType',
       'BldgNo_S', 'DwlgNo_S', 'BldgType', 'YrBuilt', 'HouseStyle',
       'Foundation', 'RoofMatl', 'Ext1', 'Ext2', 'MasVnrType', 'Heating',
       'Central Air', 'GLA', 'TtlBsmtSF', 'TotRmsAbvGrd', 'Fireplaces',
       'PoolArea', 'GarageType', 'GarYrBlt', 'Cars', 'GarageArea',
       'YrSold_YYYY', 'MoSold_MM', 'SalePrice', 'SaleType', 'SaleCond',
       'ParclRel', 'PA-Nmbr', 'PA-PreD', 'PA-Strt', 'PA-StSfx', 'PA-PostD',
       'PA-UnTyp', 'PA-UntNo', 'Date', 'Source', 'NmbrBRs'],
      dtype='object')
```

[0]:
```python
train.columns
```

[0]: Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')

[0]:
```
set(extra_train.columns).intersection(set(train.columns))
```

[0]:
```
{'BldgType',
 'Fireplaces',
 'Foundation',
 'GarageArea',
 'GarageType',
 'Heating',
 'HouseStyle',
 'LotArea',
 'MasVnrType',
 'Neighborhood',
 'PoolArea',
 'RoofMatl',
 'SalePrice',
 'SaleType',
 'TotRmsAbvGrd'}
```

- There seems to be some diffenrent naming for columns in two datasets. Therefore renaming some columns in the new dataset

[0]:
```
extra_train.rename(columns={'SaleCond':'SaleCondition','TtlBsmtSF':'TotalBsmtSF',
                            'MoSold_MM' : 'MoSold', 'YrSold_YYYY': 'YrSold',
                            'YrBuilt':'YearBuilt','Cars':'GarageCars'},␣
 ↪inplace=True)
```

[0]:
```
set(extra_train.columns).intersection(set(train.columns))
```

[0]:
```
{'BldgType',
 'Fireplaces',
 'Foundation',
```

```
'GarageArea',
'GarageCars',
'GarageType',
'Heating',
'HouseStyle',
'LotArea',
'MasVnrType',
'MoSold',
'Neighborhood',
'PoolArea',
'RoofMatl',
'SaleCondition',
'SalePrice',
'SaleType',
'TotRmsAbvGrd',
'TotalBsmtSF',
'YearBuilt',
'YrSold'}
```

[0]:
```
common_cols = list(set(extra_train.columns).intersection(set(train.columns)))
```

[0]:
```
extra_train = extra_train[common_cols]
```

[0]:
```
extra_train.shape,extra_train.head()
```

[0]:
```
((22232, 21),
   GarageArea  YearBuilt  Fireplaces  ... Neighborhood PoolArea  HouseStyle
0         NaN        NaN         NaN  ...          NaN      NaN         NaN
1         0.0     1900.0         0.0  ...      Gilbert      0.0     2-Story
2         NaN        NaN         NaN  ...          NaN      NaN         NaN
3         NaN        NaN         NaN  ...          NaN      NaN         NaN
4         NaN        NaN         NaN  ...          NaN      NaN         NaN

[5 rows x 21 columns])
```

[0]:
```
extra_train.SalePrice.isna().sum()
```

[0]:
```
19280
```

[0]:
```
# therefore filtering only populated columns
extra_train = extra_train[~extra_train.SalePrice.isnull()]
```

[0]:
```
train.shape
```

[0]:
```
(1460, 81)
```

[0]:
```
merged_df = pd.concat([train[common_cols], extra_train])
merged_df.shape
```

[0]:
```
(4412, 21)
```

[0]:
```
## doing pre-processing
```

```
numeric_feats = merged_df.dtypes[merged_df.dtypes != "object"].index
 #compute skewness
skewed_feats = merged_df[numeric_feats].apply(lambda x: skew(x.dropna()))
skewed_feats = skewed_feats[skewed_feats > 0.76]
skewed_feats = skewed_feats.index
skewed_feats
```

[0]:
```
Index(['Fireplaces', 'SalePrice', 'LotArea', 'TotRmsAbvGrd', 'PoolArea'],
dtype='object')
```

[0]:
```
merged_df[skewed_feats] = np.log1p(merged_df[skewed_feats])
merged_df = pd.get_dummies(merged_df)
merged_df = merged_df.fillna(merged_df.mean())
```

[0]:
```
merged_y = merged_df['SalePrice']
merged_df.drop(labels=['SalePrice'],axis=1,inplace=True)
```

[0]:
```
X_train, X_test, y_train, y_test = train_test_split(merged_df, merged_y,
 random_state=17, test_size=.32)
```

[0]:
```
lr = LinearRegression()
```

[0]:
```
model = lr.fit(X_train, y_train)
print("R-square : ", model.score(X_test, y_test))

preds = model.predict(X_test)
print("RMSE on normalized y : %f"% mean_squared_error(y_test, preds))
```

```
R-square :  0.5194325757525672
RMSE on normalized y : 0.294281
```

Describe the dataset and whether this data helps with prediction.

I have taken Ames Housing data for the external dataset. I did the exact pre-processing, normalizing skewed features, One-hot encoding, filling Nan values with mean. Based on the RMSE score and R-square we can see the new dataset has decreased the performance significantly. RMSE score on the combined dataset is now 0.294 as compared to 0.016 on the original training set. Therefore this new dataset doesn't help in prediction

[0]:
```

```

1.8 Part 8 - Permutation Test

Permutation test for 10 columns

[7]:
```
# TODO: code for all permutation tests

cols = ['GrLivArea', 'FullBath', 'LowQualFinSF', 'OpenPorchSF', 'OverallQual',␣
 ↪'TotRmsAbvGrd',  'BsmtFullBath', 'BsmtFinSF2', 'PoolArea', 'BsmtUnfSF']
y = np.log1p(train['SalePrice'])

for col in cols:
  train_col = pd.DataFrame(train[col])
```

```
  X_train, X_test, y_train, y_test = train_test_split(train_col, y,␣
↪random_state=17, test_size=.32)
  lr = LinearRegression()
  model = lr.fit(X_train, y_train)
  preds = model.predict(X_test)
  print("RMSE for ",col, "\t- ", mean_squared_error(y_test, preds))
```

```
RMSE for  GrLivArea     -  0.0750297448249356
RMSE for  FullBath      -  0.10025346153673487
RMSE for  LowQualFinSF  -  0.1561218197966818
RMSE for  OpenPorchSF   -  0.1373086975587337
RMSE for  OverallQual   -  0.053849225905258674
RMSE for  TotRmsAbvGrd  -  0.11214933153042124
RMSE for  BsmtFullBath  -  0.1475059269195753
RMSE for  BsmtFinSF2    -  0.15633264181516662
RMSE for  PoolArea      -  0.1560455155761854
RMSE for  BsmtUnfSF     -  0.14985788062620067
```

[0]:
```
# TODO: code for all permutation tests

rmse_vals = []
output = {}
y = np.log1p(train['SalePrice'])
for col in cols:
  rmse_vals = []
  for i in range(0, 200):
    train_col = pd.DataFrame(train[col])
    train_col[col] = np.random.permutation(train_col[col])

    X_train, X_test, y_train, y_test = train_test_split(train_col, y,␣
↪random_state=17, test_size=.32)
    lr = LinearRegression()
    model = lr.fit(X_train, y_train)
    preds = model.predict(X_test)
    rmse_vals.append(mean_squared_error(y_test, preds))
  if col not in output:
    output[col] = rmse_vals
```

[0]:
```
output_df = pd.DataFrame.from_dict(output)
```

[0]:
```
output_df.shape, output_df.head()
```

[0]:
```
from scipy.stats import zscore
from scipy.stats import norm
```

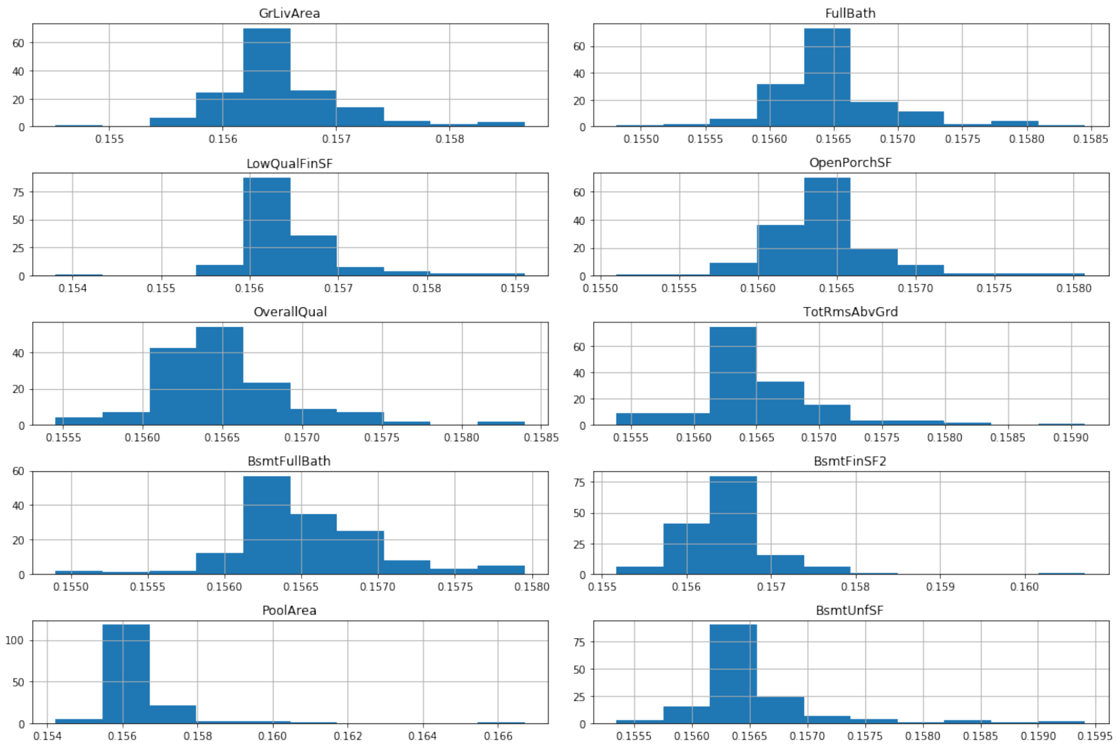
[14]:
```
for col in cols:
  p_values = norm.sf(abs(zscore(output_df[col])))
```

```
  sum = 0
  for i in range(0, len(p_values)):
    sum += p_values[i]
  print("p-value for ",col, "\t- ", sum/200)
```

```
p-value for  GrLivArea  -  0.2889011616904685
p-value for  FullBath   -  0.2911604406852294
p-value for  LowQualFinSF       -  0.3266064702558487
p-value for  OpenPorchSF        -  0.2979465965172188
p-value for  OverallQual        -  0.2836863256440624
p-value for  TotRmsAbvGrd       -  0.2955675633843144
p-value for  BsmtFullBath       -  0.28044158823964915
p-value for  BsmtFinSF2         -  0.3157575107589846
p-value for  PoolArea   -  0.36751770579771514
p-value for  BsmtUnfSF  -  0.32432542873533543
```

[13]:
```
fig, axes = plt.subplots(nrows = 5, ncols = 2, figsize = (15, 10))
cnt = 0
for col in cols:
  r = int(cnt/2)
  c = int(cnt %2)
  output_df[col].hist(ax=axes[r,c]);
  axes[r,c].set_title(col);
  cnt+=1

plt.tight_layout();
plt.show()
```



Describe the results. * The graphs are showing permutation test plots for the following columns - ‘GrLivArea’, ‘FullBath’, ‘LowQualFinSF’, ‘OpenPorchSF’, ‘OverallQual’, ‘TotRmsAbvGrd’, ‘BsmtFullBath’, ‘BsmtFinSF2’, ‘PoolArea’, ‘BsmtUnfSF’. I have built single variable regression model for each of these columns and regression has been run for 200 random permutations.

- Then comparing the RMSE of actual dataset v/s the RMSEs of permuted distribution we observe that -

    For TotRmsAbvGrd, FullBath, OverallQual, GrLivArea - the error rate increases to considerably high on shuffling the data, which is justified by their high correlation with SalePrice

    Whereas, for the Features - OpenPorchSF, LowQualFinSF, PoolArea, BsmtFullBath, BsmtFinSF2, BsmtUnfSF - the error rate doesn’t increases much and remains more or less near the RMSE value of the dataset of values without shuffling.

P-values for the 10 diffrent variables p-value for GrLivArea - 0.2889011616904685 p-value for FullBath - 0.2911604406852294 p-value for LowQualFinSF - 0.3266064702558487 p-value for OpenPorchSF - 0.2979465965172188 p-value for OverallQual - 0.2836863256440624 p-value for TotRmsAbvGrd - 0.2955675633843144 p-value for BsmtFullBath - 0.28044158823964915 p-value for BsmtFinSF2 - 0.3157575107589846 p-value for PoolArea - 0.36751770579771514 p-value for BsmtUnfSF - 0.32432542873533543

1.9 Part 9 - Final Result

Pre-processing has already been done in Part-6

[0]:
```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
```

[0]:
```
X_train = combined_data[:train.shape[0]]
X_test = combined_data[train.shape[0]:]
gbm = xgb.XGBRegressor(n_jobs=-1)
cv = GridSearchCV(gbm, {"colsample_bytree":[1.0],"min_child_weight":[1.0,1.3],
                        'max_depth': [3,5,7,9],
                        'n_estimators': [700,1100]})
cv.fit(X_train, norm_y)
```

- Best Params from Grid-Search CV

[0]:
```
cv.best_params_
"""{'colsample_bytree': 1.0,
 'max_depth': 5,
 'min_child_weight': 1.1,
 'n_estimators': 700}"""
```

[0]:
```
"{'colsample_bytree': 1.0,\n 'max_depth': 5,\n 'min_child_weight': 1.1,\n
'n_estimators': 700}"
```

[0]:
```
gbm = xgb.XGBRegressor(colsample_bytree = 1.0,
                       max_depth= 5,
                       min_child_weight = 1.1,
                       n_estimators= 700)
gbm.fit(X_train, norm_y)
```

```
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning:
Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning:
Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \

[22:11:57] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
```

[0]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1.0, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=5, min_child_weight=1.1, missing=None, n_estimators=700,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
```

[0]:
```
feat_import = gbm.get_booster().get_score(importance_type="gain")
```

[0]:
```
type(feat_import)
```

[0]:
```
dict
```

[0]:
```
for k,v in feat_import.items():
  if v >= 0.1:
    print (k,v)
```

```
OverallQual 0.2865572794132688
GrLivArea 0.18319810482205268
GarageCars 0.7278867730299138
TotalBsmtSF 0.13858593099322059
CentralAir_N 0.19846687472129526
Fireplaces 0.18878376804577807
GarageType_Attchd 0.24795886886861568
GarageCond_TA 0.43541222004157143
MSZoning_RM 0.1430064855559286
BsmtQual_Ex 0.24775981650863926
ExterQual_Fa 0.153884888
KitchenQual_Ex 0.21163646144974446
Functional_Min2 0.1019413200196
```

[0]:
```
submission = pd.DataFrame()
submission['Id'] = test.Id
preds = gbm.predict(X_test)
```

[0]:
```
final_preds = np.exp(preds) -1
submission['SalePrice'] = final_preds
submission.to_csv('xgb_submission.csv', index = False)
```

[0]:
```
final_preds
```

[0]:
```
array([126707.84, 169565.64, 195830.14, ..., 150910.52, 109449.39,
       221449.6 ], dtype=float32)
```

[0]:
```

```

- Submitted two models on Kaggle, there RMSE scores are -

1. Simple Linear Regression with default parameters - 0.58899
2. XgBoost with parameter tuning - 0.13664

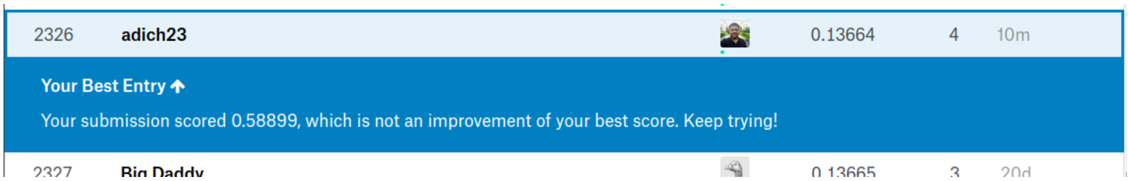
Report the rank, score, number of entries, for your highest rank. Include a snapshot of your best score on the leaderboard as confirmation. Be sure to provide a link to your Kaggle profile. Make sure to include a screenshot of your ranking. Make sure your profile includes your face and affiliation with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

Kaggle Link: https://www.kaggle.com/adich23
Highest Rank: 2326
Score: 0.13664

Number of entries: 5
INCLUDE IMAGE OF YOUR KAGGLE RANKING

[0]:
```
from IPython.display import Image
Image(filename='kaggle_rank.png')
```

[0]:



References * pandas-https://pandas.pydata.org/pandas-docs/stable/ * scikit-learn - https://scikit-learn.org/stable/ * xgboost - https://xgboost.readthedocs.io/en/latest/python/index.html * Others - https://www.cityofames.org/government/departments-divisions-a-h/city-assessor/reports https://www.kaggle.com/apapiu/regularized-linear-models

[0]:
```
```