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

simplefilter(action='ignore', category=FutureWarning)
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee649

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

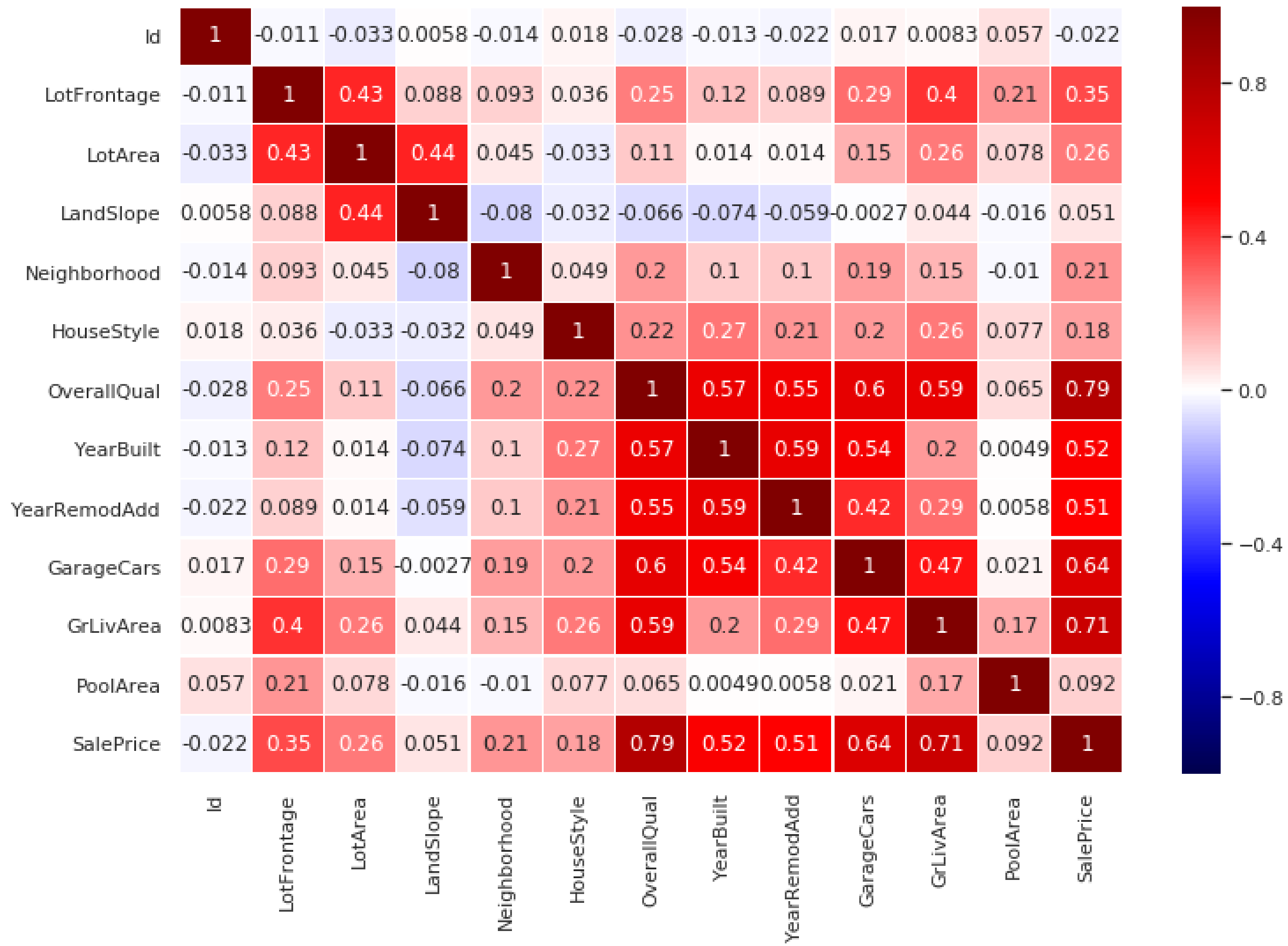
Homework 3 - Ames Housing Dataset

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.

Part 1 - Pairwise Correlations

```
# TODO: show visualization
import pandas as pd
import numpy as np
trans = pd.read_csv('/content/drive/My Drive/Colab Notebooks/house-prices (1)/train.csv')
#trans.head()

trans = trans[['Id','LotFrontage','MSZoning','LotArea','Street','LandContour','LandSlope','Neighborhood','HouseStyle','OverallQual

trans.Neighborhood = pd.Categorical(trans.Neighborhood)
trans.Neighborhood = trans.Neighborhood.cat.codes
trans.LandSlope = pd.Categorical(trans.LandSlope)
trans.LandSlope = trans.LandSlope.cat.codes
trans.HouseStyle = pd.Categorical(trans.HouseStyle)
trans.HouseStyle = trans.HouseStyle.cat.codes
```

```
import seaborn as sns
sns.set(rc={'figure.figsize':(12,8)})
sns.heatmap(trans.corr(), vmin=-1, vmax=1, cmap='seismic', linewidths=0.2, annot=True);
```



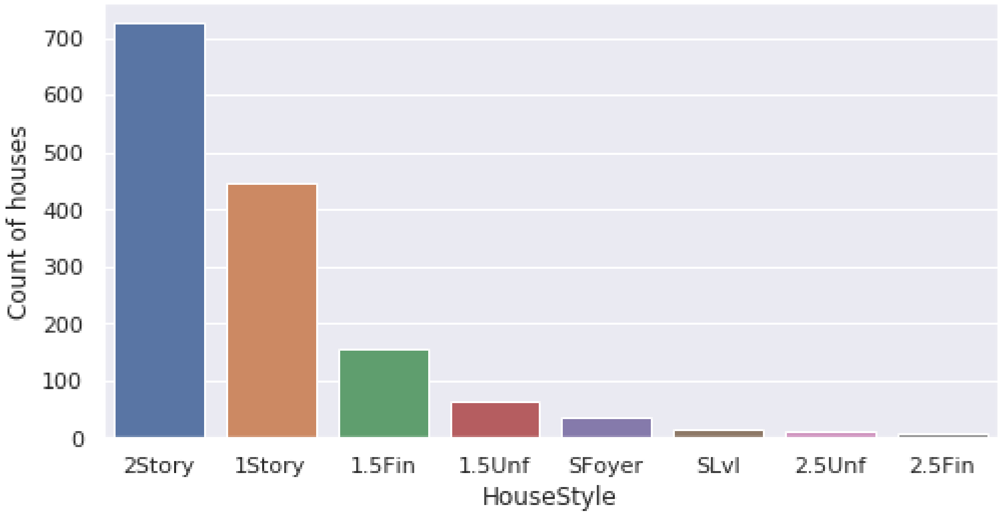
Discuss most positive and negative correlations.

The most positive correlations with SalePrice are OverallQual, GrLivArea, GarageCars, YearBuilt. OverallQual correlates positively with GarageCars and GrLivArea. There are not many negatively correlated fields, housestyle correlates negatively with LotArea. Landslope correlates negatively with YearBuilt, OverallQual, YearRemodAdd.

Part 2 - Informative Plots

```
# TODO: code to generate Plot 1
import seaborn as sns
trans = pd.read_csv('/content/drive/My Drive/Colab Notebooks/house-prices (1)/train.csv')

sns.set(rc={'figure.figsize':(8,4)})
fig=sns.barplot(trans['HouseStyle'].unique()[:10],trans['HouseStyle'].value_counts()[:10])
fig.set(xlabel='HouseStyle', ylabel='Count of houses')
```

```
[Text(0, 0.5, 'Count of houses'), Text(0.5, 0, 'HouseStyle')]
```



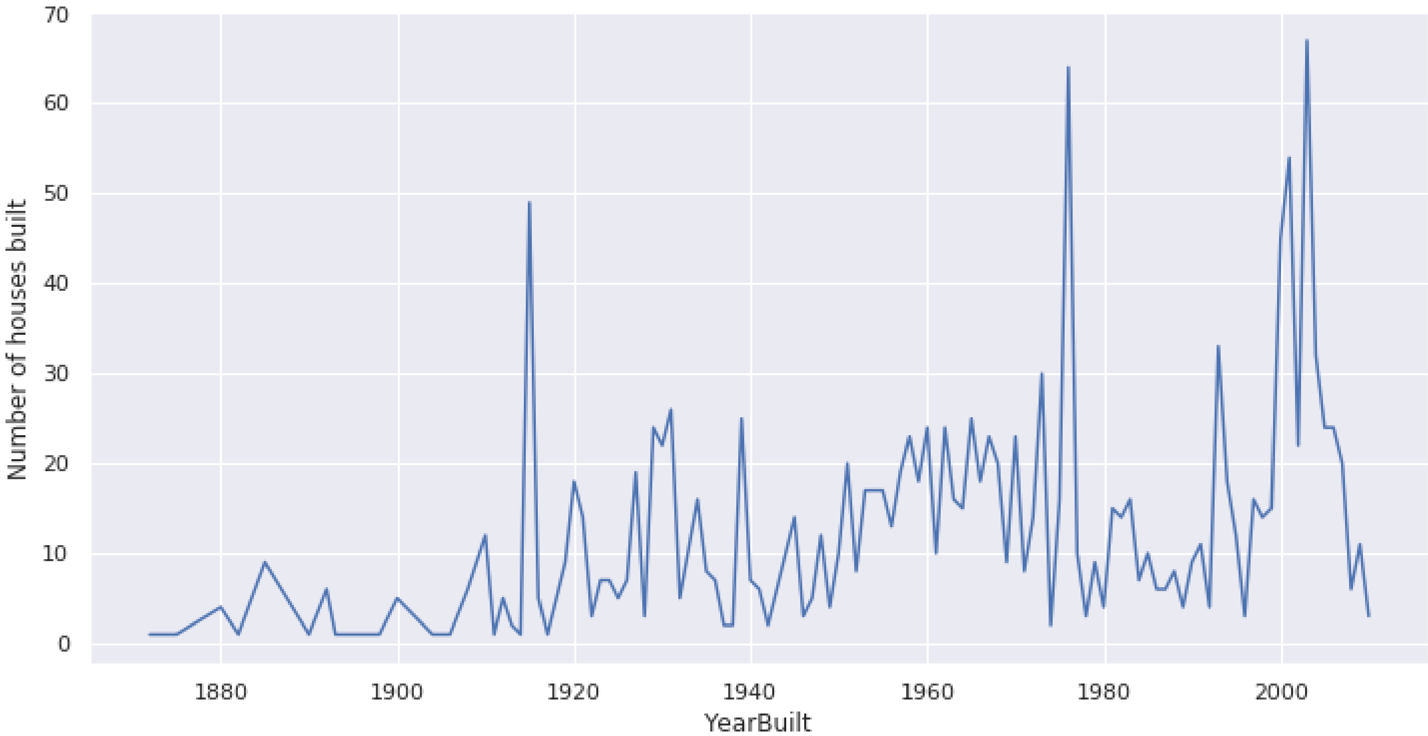
What interesting properties does Plot 1 reveal?

Most houses were 2 storey, followed by 1 storey.

```
sns.set(rc={'figure.figsize':(12,6)})
fig=sns.lineplot(trans['YearBuilt'].unique()[:1000],trans['YearBuilt'].value_counts()[:1000])
```

```
fig.set(xlabel='YearBuilt', ylabel='Number of houses built')
```

```
[Text(0, 0.5, 'Number of houses built'), Text(0.5, 0, 'YearBuilt')]
```

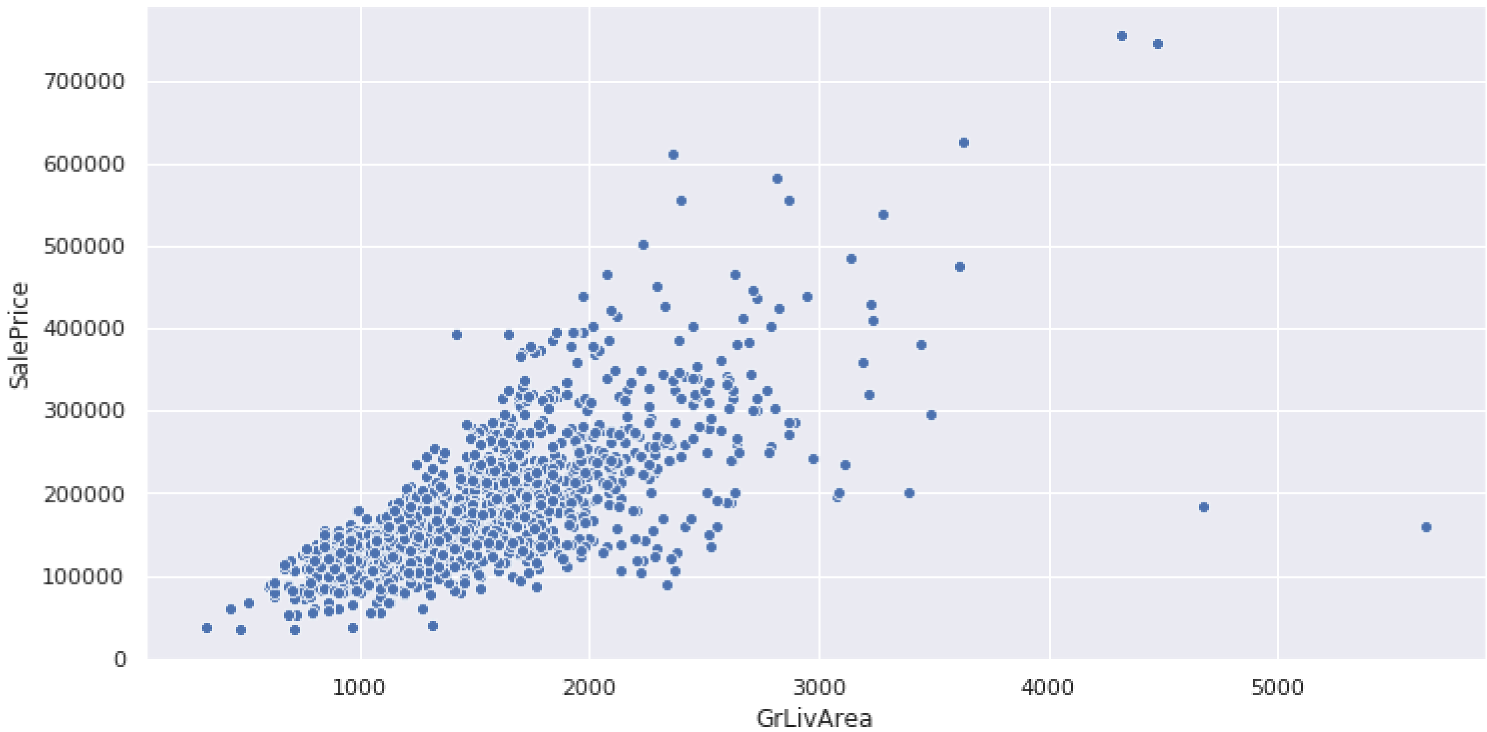


What interesting properties does Plot 2 reveal?

The highest number of houses were built in the year 2003 and 1976.

```
import numpy as np
fig=sns.scatterplot((trans['GrLivArea']),trans['SalePrice'])
fig.set(xlabel='GrLivArea', ylabel='SalePrice')
```

```
[Text(0, 0.5, 'SalePrice'), Text(0.5, 0, 'GrLivArea')]
```

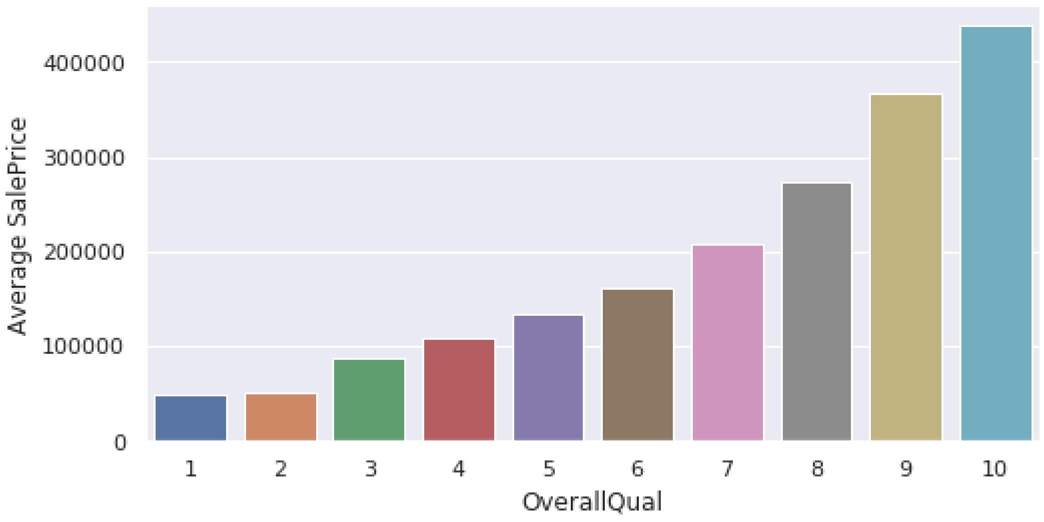


What interesting properties does Plot 3 reveal?

The SalePrice appears to increase with increase in GrLiving area for most cases. Most houses have GrLiving area between 1000 and 2000 units.

```
sns.set(rc={'figure.figsize':(8,4)})

group = trans.groupby(['OverallQual'])
group = group.mean()
group = group.reset_index();

fig=sns.barplot(group['OverallQual'].unique(),group['SalePrice'])
fig.set(xlabel='OverallQual', ylabel='Average SalePrice')
```

```
[Text(0, 0.5, 'Average SalePrice'), Text(0.5, 0, 'OverallQual')]
```

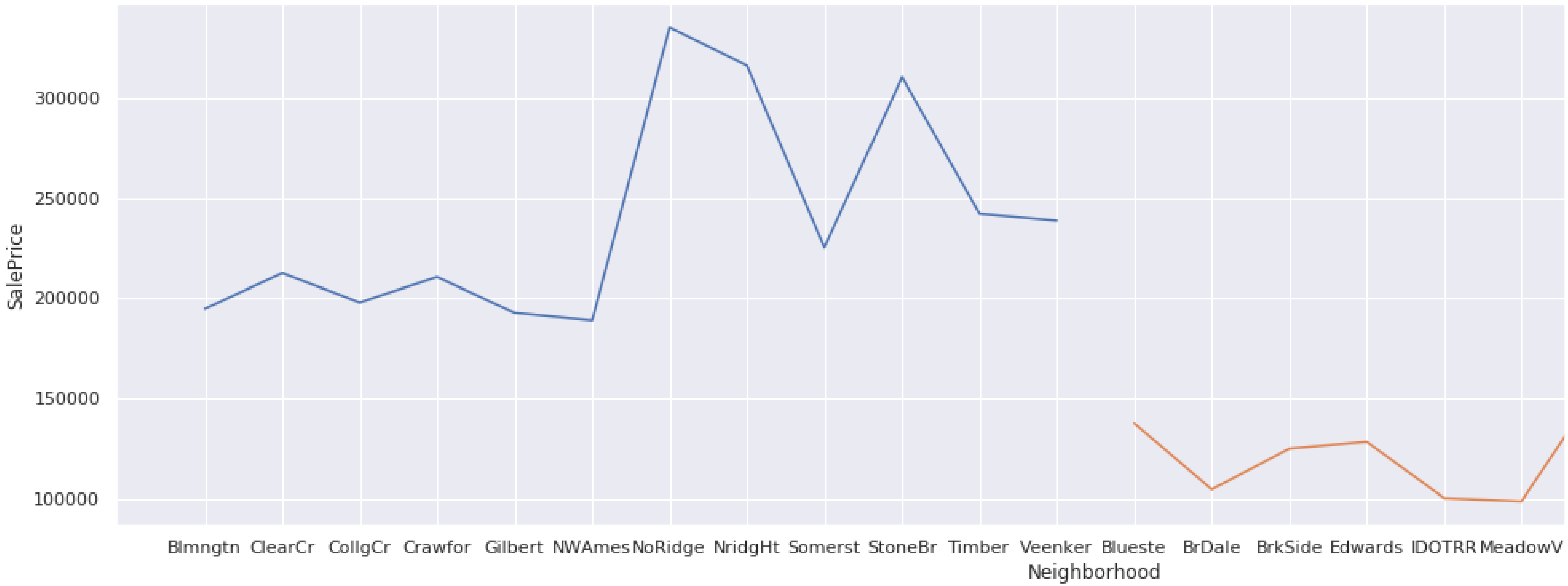


What interesting properties does Plot 4 reveal?

Average Saleprice increases exponentially with Overall quality.

```
# trans['Neighborhood'].value_counts()
sns.set(rc={'figure.figsize':(22,6)})

group = trans.groupby(['Neighborhood'])
group = group.mean()
group = group.reset_index();
group = group.sort_values(by=['SalePrice'],ascending=False)
# temp5 = group['Neighborhood']

fig.set(xlabel='Neighborhood', ylabel='Avg SalePrice')
print("Most Expensive Neighborhoods: NoRidge, NRidgHt,StonBr")
print("Least Expensive Neighborhoods: MeadowV, IDOTRR, BrDale")

fig=sns.lineplot(group['Neighborhood'][:12], group['SalePrice'][:12])      #Most expensive
print('\n')
fig1=sns.lineplot(group['Neighborhood'][13:], group['SalePrice'][13:])     #Least expensive
```

```
Most Expensive Neighborhoods: NoRidge, NRidgHt,StonBr
Least Expensive Neighborhoods: MeadowV, IDOTRR, BrDale
```



What interesting properties does Plot 5 reveal?

Most Expensive Neighborhoods: NoRidge, NRidgHt,StonBr

Least Expensive Neighborhoods: MeadowV, IDOTRR, BrDale

Part 3 - Handcrafted Scoring Function

```
# TODO: code for scoring function
```

```
dep_vars = trans[['Id','Neighborhood', 'OverallQual','YearBuilt','SalePrice']]
dep_vars = dep_vars.reset_index();

dep_vars['score'] = ((dep_vars['OverallQual']*1000 + dep_vars['YearBuilt']*10 +dep_vars['SalePrice']/10)/10000)    #score
maxscore = max(dep_vars['score'])                                          # normalize scores
dep_vars['score'] = dep_vars['score']/maxscore                             # score out of 1

print('MOST DESIRABLE')
print(dep_vars.sort_values(by=['score'],ascending=False).head(10))
print('\n')
print('LEAST DESIRABLE')
print(dep_vars.sort_values(by=['score']).head(10))
```

```
MOST DESIRABLE
      index    Id Neighborhood  OverallQual  YearBuilt  SalePrice     score
691     691   692      NoRidge           10       1994     755000  1.000000
1182   1182  1183      NoRidge           10       1996     745000  0.990706
1169   1169  1170      NoRidge           10       1995     625000  0.876802
898     898   899      NridgHt            9       2009     611657  0.855991
803     803   804      NridgHt            9       2008     582933  0.828654
440     440   441      NridgHt           10       2008     555000  0.811646
1046   1046  1047      StoneBr            9       2005     556581  0.803377
769     769   770      StoneBr            8       2003     538000  0.776081
178     178   179      StoneBr            9       2008     501837  0.751742
798     798   799      NridgHt            9       2008     485000  0.735774


LEAST DESIRABLE
      index    Id Neighborhood  OverallQual  YearBuilt  SalePrice     score
533     533   534      BrkSide            1       1946      39300  0.231316
916     916   917       IDOTRR            2       1949      35311  0.237302
968     968   969      OldTown            3       1910      37900  0.245542
375     375   376      Edwards            1       1922      61000  0.249621
495     495   496       IDOTRR            4       1920      34900  0.253130
1100   1100  1101        SWISU            2       1920      60000  0.257967
30       30    31       IDOTRR            4       1920      40000  0.257967
636     636   637      BrkSide            2       1936      60000  0.259484
710     710   711      BrkSide            3       1935      52000  0.261286
1380   1380  1381      Edwards            3       1914      58500  0.265459
```

What is the ten most desirable houses?

(Listed Above)

What is the ten least desirable houses?

(Listed Above)

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

Thr score is dependent on 3 variables - Overall quality, Year built and SalePrice. All of these are positively correlated to desirability. Higher the quality, newer the house construction and higher the price - more desirable the house is.

Part 4 - Pairwise Distance Function

```
# TODO: code for distance function
from sklearn.metrics.pairwise import euclidean_distances
trans.head()
pairwise = trans[['OverallQual','YearBuilt','SalePrice','GarageCars']]
# pairwise.head()
distances = euclidean_distances(pairwise, pairwise)
#pd.DataFrame(distances).head(10).hist()
max_ele = max(map(max, distances))
min_ele = np.amin(distances[distances != np.amin(distances)])
# print(min_ele)

print('Max distance is ' ,max(map(max, distances)))

list=[]
for i in range(0,1459):
  for j in range(1,1459):
    if distances[i][j] == max_ele:
      if i>=j:
        print("Max distance between house numbers ", "(", i+1,", ", j+1,')')

print('Min distance is ', np.amin(distances[distances != np.amin(distances)]))

for i in range(0, 1459):
  for j in range(0, 1459):
    if distances[i][j] == 1.0:
      if i>=j:

        list.append('('+str(i+1)+', '+str(j+1)+')')
```

```
print('Min distance between house numbers ', list)
```

```
Max distance is  720100.0038334953
Max distance between house numbers  ( 692 ,  496 )
Min distance is  1.0
Min distance between house numbers  ['(194, 146)', '(258, 163)', '(274, 20)', '(332, 274)', '(365, 110)', '(367, 287)
```

```
print('\n','Eg. for max dist:')
print(trans[(trans.Id == 692) | (trans.Id == 496)])
print('\n','Eg. for min dist:')
print(trans[(trans.Id == 194) | (trans.Id == 146)])
```

```
 Eg. for max dist:
      Id  MSSubClass MSZoning  ...  SaleType  SaleCondition SalePrice
495  496          30  C (all)  ...        WD        Abnorml     34900
691  692          60       RL  ...        WD         Normal    755000

[2 rows x 81 columns]

 Eg. for min dist:
      Id  MSSubClass MSZoning  ...  SaleType  SaleCondition SalePrice
145  146         160       RM  ...        WD         Normal    130000
193  194         160       RM  ...        WD         Normal    130000

[2 rows x 81 columns]
```

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

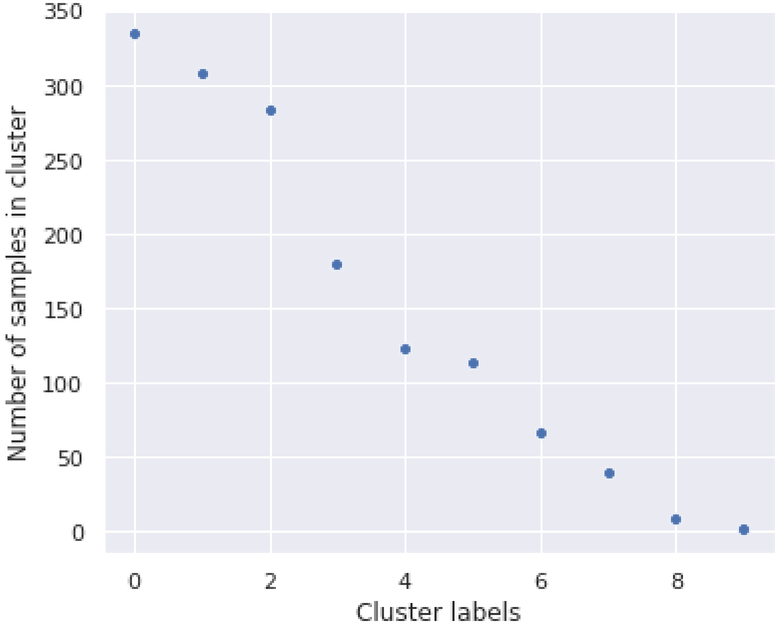
The distance vector is max for properties 496 and 692. Looking at fields 'OverallQual','YearBuilt','SalePrice','GarageCars' we observe that they are extremely different in values.

The distance vector is min for a number of properties. For eg 194 and 146. Looking at fields 'OverallQual','YearBuilt','SalePrice','GarageCars' we observe that they are quite similar/same in values.

Part 5 - Clustering

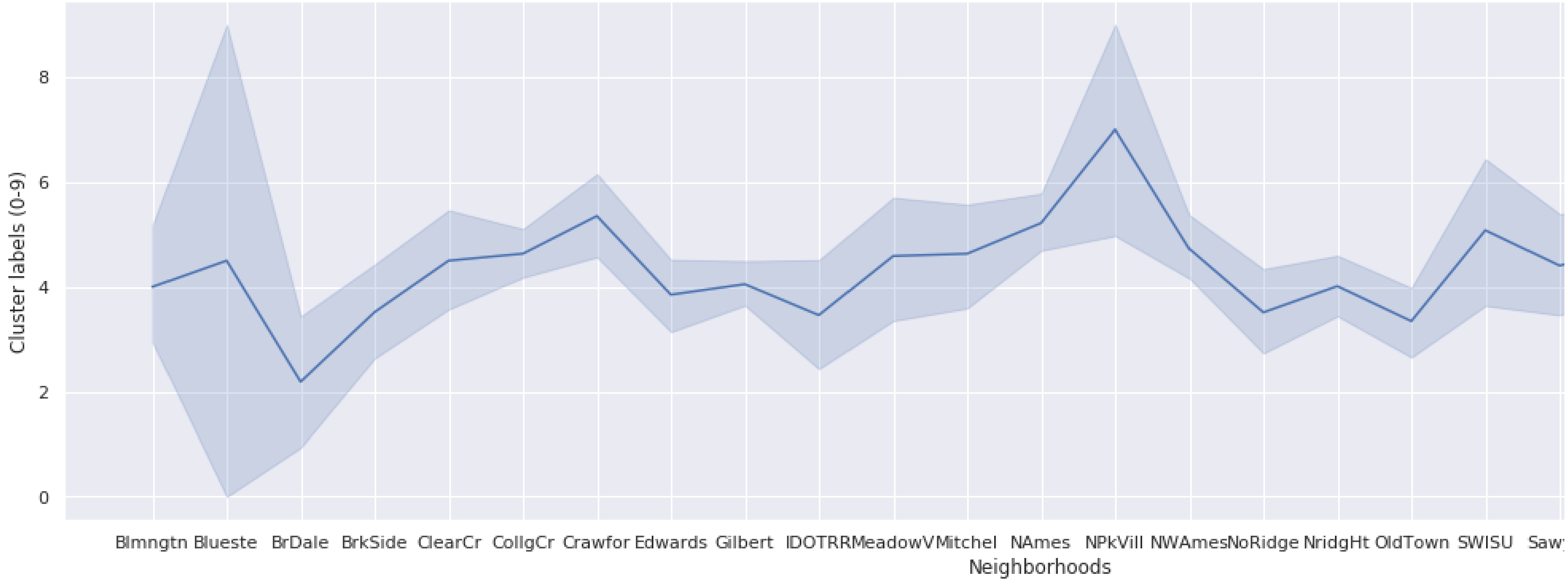
```
# TODO: code for clustering and visualization
# agglo
from sklearn.cluster import KMeans
import numpy as np

kmeans = KMeans(n_clusters=10, random_state=0).fit(distances)
kmeans.labels_
kmeans.predict(distances)
arr = kmeans.labels_
df=trans[['Id','Neighborhood']].copy()
df['labels'] = arr

fig=sns.scatterplot(np.unique(arr),pd.Series(arr).value_counts())
sns.set(rc={'figure.figsize':(6,5)})

fig.set(xlabel='Cluster labels', ylabel='Number of samples in cluster')
print('Cluster labels:')
arr
```

```
Cluster labels:
array([6, 3, 6, ..., 1, 9, 9], dtype=int32)
```



```
sns.set(rc={'figure.figsize':(22,6)})

fig=sns.lineplot((df['Neighborhood']),df['labels'])
fig.set(xlabel='Neighborhoods', ylabel='Cluster labels (0-9)')
```

```
[Text(0, 0.5, 'Cluster labels (0-9)'), Text(0.5, 0, 'Neighborhoods')]
```



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

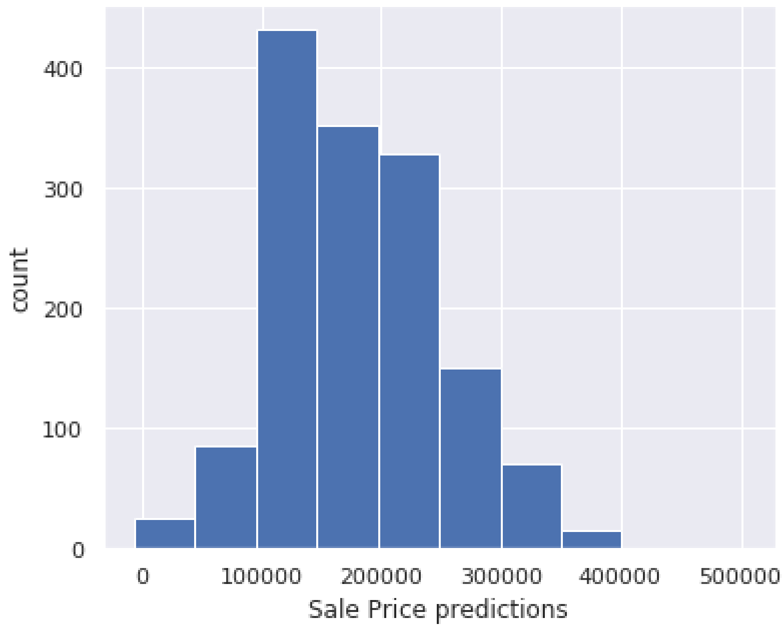
The distance matrix measures eucildean distances of variables 'OverallQual','YearBuilt','SalePrice','GarageCars'. The clusters reflect neighborhood boundaries to a good extent for almost all neighborhoods(with exceptions of Blueste, NPkVill, Veenkar as shown in the graph).

Part 6 - Linear Regression

```
# TODO: code for linear regression
import pandas as pd
```

```
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_log_error
from math import sqrt
from sklearn import preprocessing
import matplotlib.pyplot as plt


trans = pd.read_csv('/content/drive/My Drive/Colab Notebooks/house-prices (1)/train.csv')

trans_X = trans[['OverallQual','YearBuilt','SalePrice','GarageCars','GrLivArea']]
trans_X = trans_X.fillna(trans_X.mode().iloc[0])

trans_y = pd.DataFrame(trans_X['SalePrice'])

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

X, y = trans_X, trans_y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=500)

clf = LinearRegression()
clf.fit(X_train, y_train)

preds = clf.predict(X_test)
print("MEAN SQUARED LOG ERROR: ", np.sqrt(metrics.mean_squared_log_error(preds, y_test)))
#-----------------------------------------------------------------------------------------------------------


test = pd.read_csv('/content/drive/My Drive/Colab Notebooks/house-prices (1)/test.csv')
test = test[['OverallQual','YearBuilt', 'GarageCars','GrLivArea']]
test = test.fillna(test.mode().iloc[0])

preds = clf.predict(test)
plt.hist(preds)
plt.xlabel('Sale Price predictions')
plt.ylabel('count')

plt.show()
```

```
MEAN SQUARED LOG ERROR:  0.24335448037635204
```



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

It worked well - RMSLE = 0.24. The most imporatnt variables were 'OverallQual','YearBuilt', 'GarageCars','GrLivArea'. These are the ones with a high correlation with SalePrice.

Part 7 - External Dataset

```
# TODO: code to import external dataset and test

inflation = pd.read_csv('/content/inflation.csv')   #Average CPI for years 1913-2018
inflation['YrSold'] = inflation['Year']
inflation=inflation.drop(['Year'],axis=1)

trans = pd.read_csv('/content/drive/My Drive/Colab Notebooks/house-prices (1)/train.csv')
# trans_X=trans[['Id','LotFrontage','LotArea','OverallQual','OverallCond','YearBuilt','YearRemodAdd','GarageCars','PoolArea','YrSo

trans_X = trans[['OverallQual','YearBuilt','SalePrice','GarageCars','GrLivArea','YrSold']]
trans_X = pd.merge(trans_X, inflation, on='YrSold', how='left', left_index=True)
```

```
trans_X

trans_X = trans_X.fillna(trans_X.mode().iloc[0])

trans_y = pd.DataFrame(trans_X['SalePrice'])

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

X, y = trans_X, trans_y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=500)

clf = LinearRegression()
clf.fit(X_train, y_train)

preds = clf.predict(X_test)
print("MEAN SQUARED LOG ERROR: ", np.sqrt(metrics.mean_squared_log_error(preds, y_test)))
```

```
MEAN SQUARED LOG ERROR:  0.24638999666977993
```

Describe the dataset and whether this data helps with prediction.

The dataset used is called 'Historical Consumer Price Index (CPI-U) Data' from https://inflationdata.com. The Consumer Price Index (CPI) is a measure of the average change over time in the prices paid by urban consumers for a market basket of consumer goods and services. A Consumer Price Index of 158 indicates 58% inflation since 1982, while a CPI index of 239 would indicate 139% inflation since 1982. CPI has a high correlation with property rates, hence it should improvise our model accuracy.

Part 8 - Permutation Test

```
# TODO: code for all permutation tests
from mlxtend.evaluate import permutation_test
from sklearn.model_selection import permutation_test_score

import pandas as pd
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.ensemble import AdaBoostRegressor, RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_log_error

from math import sqrt
from sklearn import preprocessing
```

```
trans = pd.read_csv('/content/drive/My Drive/Colab Notebooks/house-prices (1)/train.csv')
# trans_X=trans[['Id','LotFrontage','LotArea','OverallQual','OverallCond','YearBuilt','YearRemodAdd','GarageCars','PoolArea','YrSo

trans_X=trans[['OverallQual','SalePrice']]
trans_X = trans_X.fillna(trans_X.mode().iloc[0])


trans_y = (trans_X['SalePrice'])

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

X, y = trans_X, trans_y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=500)

clf = RandomForestRegressor(n_estimators=100, random_state=20)
model = clf.fit(X_train, y_train)
preds = clf.predict(X_test)


score, permutation_scores, pvalue = permutation_test_score(model, trans_X, trans_y, n_permutations=100, n_jobs=1)
print('Pvalue for OverallQual', pvalue)
print('RMSLE',np.sqrt(metrics.mean_squared_log_error(preds, y_test)))
print('\n')
#----------------------------------------------------------------------------------------------------------------------------------
trans_X=trans[['YearBuilt','SalePrice']]
trans_X = trans_X.fillna(trans_X.mode().iloc[0])

trans_y = (trans_X['SalePrice'])

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

X, y = trans_X, trans_y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=500)

clf = RandomForestRegressor(n_estimators=100, random_state=20)
model = clf.fit(X_train, y_train)
preds = clf.predict(X_test)

score, permutation_scores, pvalue = permutation_test_score(model, trans_X, trans_y, n_permutations=100, n_jobs=1)
print('Pvalue for YearBuilt', pvalue)
print('RMSLE',np.sqrt(metrics.mean_squared_log_error(preds, y_test)))
print('\n')

# #--------------------------------------------------------------------------------------------------------------------------------

trans_X=trans[['GarageCars','SalePrice']]
trans_X = trans_X.fillna(trans_X.mode().iloc[0])

trans_y = (trans_X['SalePrice'])

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

```
X, y = trans_X, trans_y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=500)

clf = RandomForestRegressor(n_estimators=100, random_state=20)
model = clf.fit(X_train, y_train)
preds = clf.predict(X_test)

score, permutation_scores, pvalue = permutation_test_score(model, trans_X, trans_y, n_permutations=100, n_jobs=1)
print('Pvalue for GarageCars', pvalue)
print('RMSLE',np.sqrt(metrics.mean_squared_log_error(preds, y_test)))
print('\n')

# #----------------------------------------------------------------------------------------------------------------------------------


trans_X=trans[['GrLivArea','SalePrice']]
trans_X = trans_X.fillna(trans_X.mode().iloc[0])

trans_y = (trans_X['SalePrice'])

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

X, y = trans_X, trans_y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=500)

clf = RandomForestRegressor(n_estimators=100, random_state=20)
model = clf.fit(X_train, y_train)
preds = clf.predict(X_test)

score, permutation_scores, pvalue = permutation_test_score(model, trans_X, trans_y, n_permutations=100, n_jobs=1)
print('Pvalue for GrLivArea', pvalue)
print('RMSLE',np.sqrt(metrics.mean_squared_log_error(preds, y_test)))
print('\n')

# #----------------------------------------------------------------------------------------------------------------------------------

trans_X=trans[['Id','SalePrice']]
trans_X = trans_X.fillna(trans_X.mode().iloc[0])

trans_y = (trans_X['SalePrice'])

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

X, y = trans_X, trans_y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=500)

clf = RandomForestRegressor(n_estimators=100, random_state=20)
model = clf.fit(X_train, y_train)
preds = clf.predict(X_test)
```

```
score, permutation_scores, pvalue = permutation_test_score(model, trans_X, trans_y, n_permutations=100, n_jobs=1)
print('Pvalue for Id', pvalue)
print('RMSLE',np.sqrt(metrics.mean_squared_log_error(preds, y_test)))
print('\n')

# #-----------------------------------------------------------------------------------------------------------------

trans_X=trans[['BsmtFinSF2','SalePrice']]
trans_X = trans_X.fillna(trans_X.mode().iloc[0])

trans_y = (trans_X['SalePrice'])

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

X, y = trans_X, trans_y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=500)

clf = RandomForestRegressor(n_estimators=100, random_state=20)
model = clf.fit(X_train, y_train)

score, permutation_scores, pvalue = permutation_test_score(model, trans_X, trans_y, n_permutations=100, n_jobs=1)
print('Pvalue for BsmtFinSF2', pvalue)
print('RMSLE',np.sqrt(metrics.mean_squared_log_error(preds, y_test)))
print('\n')

# #-----------------------------------------------------------------------------------------------------------------

trans_X=trans[['PoolArea','SalePrice']]
trans_X = trans_X.fillna(trans_X.mode().iloc[0])

trans_y = (trans_X['SalePrice'])

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

X, y = trans_X, trans_y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=500)

clf = RandomForestRegressor(n_estimators=100, random_state=20)
model = clf.fit(X_train, y_train)
preds = clf.predict(X_test)

score, permutation_scores, pvalue = permutation_test_score(model, trans_X, trans_y, n_permutations=100, n_jobs=1)
print('Pvalue for PoolArea', pvalue)
print('RMSLE',np.sqrt(metrics.mean_squared_log_error(preds, y_test)))
print('\n')

# #-----------------------------------------------------------------------------------------------------------------

trans_X=trans[['OverallCond','SalePrice']]
trans_X = trans_X.fillna(trans_X.mode().iloc[0])
```

```
trans_y = (trans_X['SalePrice'])

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

X, y = trans_X, trans_y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=500)

clf = RandomForestRegressor(n_estimators=100, random_state=20)
model = clf.fit(X_train, y_train)
preds = clf.predict(X_test)

score, permutation_scores, pvalue = permutation_test_score(model, trans_X, trans_y, n_permutations=100, n_jobs=1)
print('Pvalue for OverallCond', pvalue)
print('RMSLE',np.sqrt(metrics.mean_squared_log_error(preds, y_test)))
print('\n')

# #-----------------------------------------------------------------------------------------------------------------------------

trans_X=trans[['TotalBsmtSF','SalePrice']]
trans_X = trans_X.fillna(trans_X.mode().iloc[0])

trans_y = (trans_X['SalePrice'])

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

X, y = trans_X, trans_y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=500)

clf = RandomForestRegressor(n_estimators=100, random_state=20)
model = clf.fit(X_train, y_train)
preds = clf.predict(X_test)

score, permutation_scores, pvalue = permutation_test_score(model, trans_X, trans_y, n_permutations=100, n_jobs=1)
print('Pvalue for TotalBsmtSF', pvalue)
print('RMSLE',np.sqrt(metrics.mean_squared_log_error(preds, y_test)))
print('\n')


# #-----------------------------------------------------------------------------------------------------------------------------

trans_X=trans[['KitchenAbvGr','SalePrice']]
trans_X = trans_X.fillna(trans_X.mode().iloc[0])

trans_y = (trans_X['SalePrice'])

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

X, y = trans_X, trans_y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=500)

clf = RandomForestRegressor(n_estimators=100, random_state=20)
```

```
model = clf.fit(X_train, y_train)
preds = clf.predict(X_test)

score, permutation_scores, pvalue = permutation_test_score(model, trans_X, trans_y, n_permutations=100, n_jobs=1)
print('Pvalue for KitchenAbvGr', pvalue)
print('RMSLE',np.sqrt(metrics.mean_squared_log_error(preds, y_test)))
print('\n')

#-----------------------------------------------------------------------------------------------------------------------------------
```

```
Pvalue for OverallQual 0.009900990099009901
RMSLE 0.22348276760110886


Pvalue for YearBuilt 0.009900990099009901
RMSLE 0.2863191565877305


Pvalue for GarageCars 0.009900990099009901
RMSLE 0.26854153922440266


Pvalue for GrLivArea 0.009900990099009901
RMSLE 0.2974631532054856


Pvalue for Id 0.09900990099009901
RMSLE 0.44984043989531625


Pvalue for BsmtFinSF2 0.019801980198019802
RMSLE 0.44984043989531625


Pvalue for PoolArea 0.019801980198019802
RMSLE 0.3675701104872097


Pvalue for OverallCond 0.009900990099009901
RMSLE 0.3347009508097016


Pvalue for TotalBsmtSF 0.009900990099009901
RMSLE 0.3274647248042461


Pvalue for KitchenAbvGr 0.009900990099009901
RMSLE 0.3656845581892145
```

Describe the results.

The p-value is used in the context of null hypothesis testing in order to quantify the idea of statistical significance of evidence. A small p-value (typically ≤ 0.05) indicates strong evidence against the null hypothesis, so you reject the null hypothesis. The p-values here are less than 5%(statistically significant) indicating that the Random Forest Regression model is a good choice.

Part 9 - Final Result

```
# TODO: code for RandomForestRegressor
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import metrics
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn import preprocessing
import xgboost as xgb

trans = pd.read_csv('/content/drive/My Drive/Colab Notebooks/house-prices (1)/train.csv')
# trans_X=trans[['Id','LotFrontage','LotArea','OverallQual','OverallCond','YearBuilt','YearRemodAdd','GarageCars','PoolArea','YrSo

trans_X=trans[['OverallQual','YearBuilt','SalePrice','GarageCars','GrLivArea']]
trans_X = trans_X.fillna(trans_X.mode().iloc[0])


trans_y = pd.DataFrame(trans_X['SalePrice'])

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

X, y = trans_X, trans_y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=500)

clf = xgb.XGBRegressor(learning_rate=0.01,n_estimators=3460,
                                     max_depth=3, min_child_weight=0,
                                     gamma=0, subsample=0.7,
                                     colsample_bytree=0.7,
                                     objective='reg:linear', nthread=-1,
                                     scale_pos_weight=1, seed=27,
                                     reg_alpha=0.00006)
```

```
# XGBoost stands for "Extreme Gradient Boosting" is a supervised machine learning model, rmsle obtained = 0.1542

var = clf.fit(X_train, y_train)


preds = clf.predict(X_test)
print('RMSLE: ',np.sqrt(metrics.mean_squared_log_error(preds, y_test)))

test = pd.read_csv('/content/drive/My Drive/Colab Notebooks/house-prices (1)/test.csv')
test = test[['OverallQual','YearBuilt', 'GarageCars','GrLivArea']]
test = test.fillna(test.mode().iloc[0])

preds = clf.predict(test)
# pd.DataFrame(preds).hist()
plt.hist(preds)
plt.xlabel('Sale Price predictions')
plt.ylabel('count')

plt.show()
```

```
[10:46:24] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squa
RMSLE:  0.1542517704280983
```


```
pd.DataFrame(preds).to_csv("output03.csv")
```

Part 10 - Kaggle Submission

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/saylik

Highest Rank: 3781

Score: 0.17506

Number of entries: 5

INCLUDE IMAGE OF YOUR KAGGLE RANKING: https://drive.google.com/file/d/13ZGrRwrGMExtcKK10wgBnGvYO6E9mS28/view?usp=sharing