Homework 3 - Ames Housing Dataset

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

In [338]:
```
#Mount drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [0]:
```
#Organizing my imports
import pandas as pd
import numpy as np
import seaborn as sns
sns.set()
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from scipy.stats import zscore
from scipy.spatial.distance import pdist
from scipy.spatial.distance import squareform
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram
from sklearn.model_selection import permutation_test_score


import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [340]:
```
train_data = pd.read_csv("/content/drive/My Drive/train.csv")
test_data = pd.read_csv("/content/drive/My Drive/test.csv")

print("File reading successfull!!")
```

```
File reading successfull!!
```

In [0]:
```
#Replace NaN values with mode
train_data = train_data.fillna(train_data.mode().iloc[0])
test_data = test_data.fillna(test_data.mode().iloc[0])
```

In [0]:
```
#Mapping categorical data to numeric values
for col in train_data:
  if col != 'SalePrice':
    if train_data[col].dtype == 'object' or test_data[col].dtype == 'object':
      le = preprocessing.LabelEncoder()
      le.fit(list(train_data[col].values) + list(test_data[col].values))
      train_data[col] = le.transform(list(train_data[col].values))
      test_data[col] = le.transform(list(test_data[col].values))
```

Part 1 - Pairwise Correlations

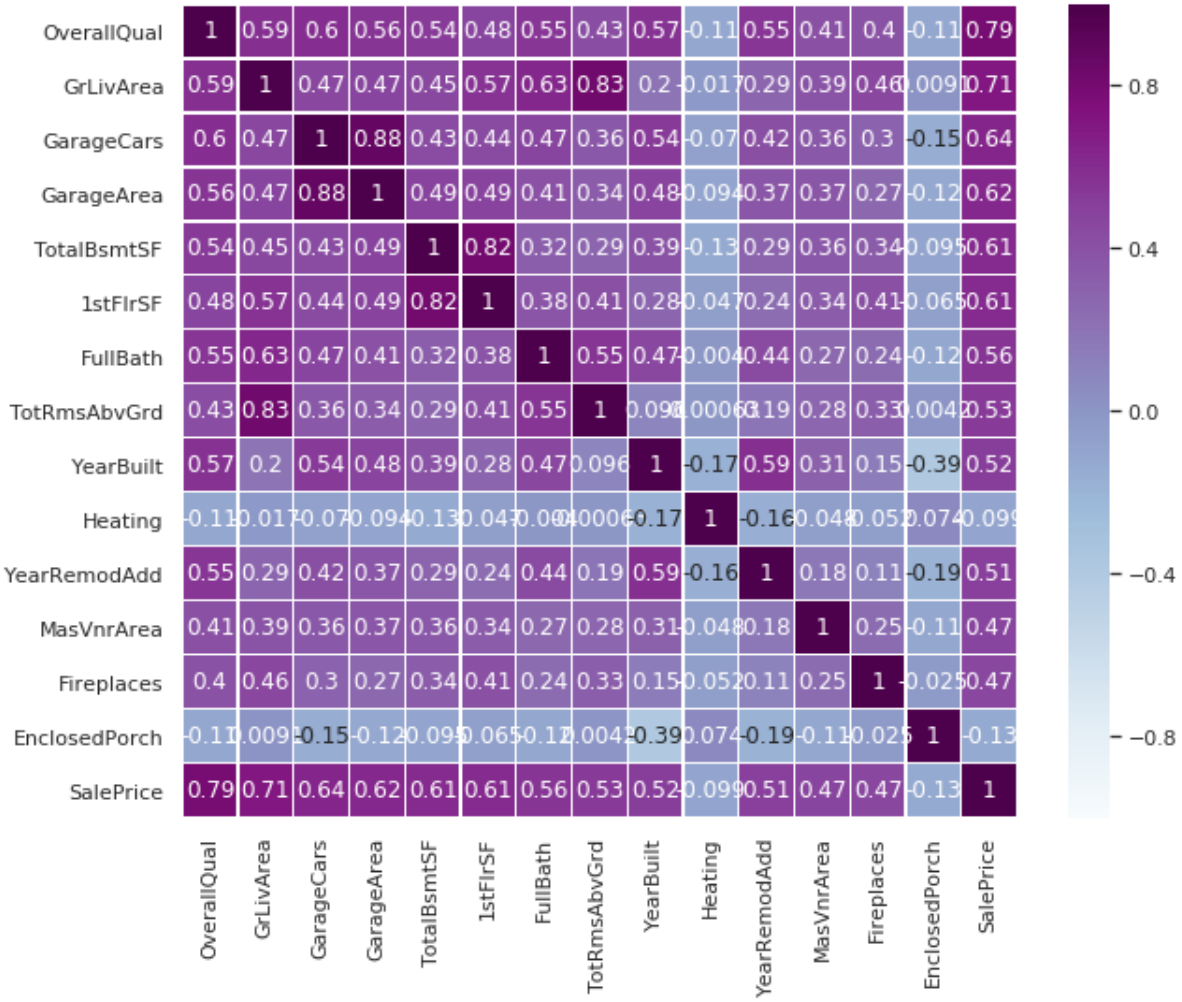
In [343]:
```
# TODO: show visualization

corr_matrix = train_data[train_data.columns[1:]].corr()['SalePrice'][:]

corr_matrix_cols = corr_matrix[corr_matrix > -0.02].sort_values(ascending = Fa
lse)[1:30]
corr_matrix_desc = corr_matrix[corr_matrix < -0.02].sort_values(ascending = Fa
lse)[1:20]

print("Most","\n",corr_matrix_cols,"\n")
print("Least","\n",corr_matrix_desc)
```

```
Most
 OverallQual      0.790982
GrLivArea        0.708624
GarageCars       0.640409
GarageArea       0.623431
TotalBsmtSF      0.613581
1stFlrSF         0.605852
FullBath         0.560664
TotRmsAbvGrd     0.533723
YearBuilt        0.522897
YearRemodAdd     0.507101
MasVnrArea       0.472614
Fireplaces       0.466929
GarageYrBlt      0.397778
BsmtFinSF1       0.386420
Foundation       0.382479
LotFrontage      0.329220
WoodDeckSF       0.324413
2ndFlrSF         0.319334
OpenPorchSF      0.315856
HalfBath         0.284108
LotArea          0.263843
CentralAir       0.251328
Electrical       0.234945
PavedDrive       0.231357
BsmtFullBath     0.227122
RoofStyle        0.222405
BsmtUnfSF        0.214479
SaleCondition    0.213092
Neighborhood     0.210851
Name: SalePrice, dtype: float64

Least
 LowQualFinSF    -0.025606
Alley           -0.027655
YrSold          -0.028923
SaleType        -0.054911
LotConfig       -0.067396
OverallCond     -0.077856
MSSubClass      -0.084284
BldgType        -0.085591
BsmtFinType1    -0.092106
Heating         -0.098812
EnclosedPorch   -0.128578
KitchenAbvGr    -0.135907
PoolQC          -0.136215
MSZoning        -0.166872
LotShape        -0.255580
BsmtExposure    -0.295407
GarageType      -0.302105
HeatingQC       -0.400178
GarageFinish    -0.537242
Name: SalePrice, dtype: float64
```

In [0]:
```
train_data_filtered = train_data.filter(['OverallQual','GrLivArea','GarageCar
s','GarageArea','TotalBsmtSF',
                                         '1stFlrSF','FullBath','TotRmsAbvGrd',
'YearBuilt','Heating','YearRemodAdd','MasVnrArea','Fireplaces','EnclosedPorch'
,'SalePrice'])
```

In [345]:
```
plt.figure(figsize=(10,8))
corr = train_data_filtered.corr()
sns.heatmap(train_data_filtered.corr(), vmin=-1, vmax=1, cmap='BuPu', linewidt
hs=0.2, annot=True)
```

Out[345]: <matplotlib.axes._subplots.AxesSubplot at 0x7f116372be10>



Question: Discuss most positive and negative correlations.

Answer: Firstly to pick the most interesting variables, I took a correlation check of all with the Housing Price to find the ones which were positively and negatively correlated with the pricing.

I then took a pairwise correlation among these. (as shown in above heatmap)

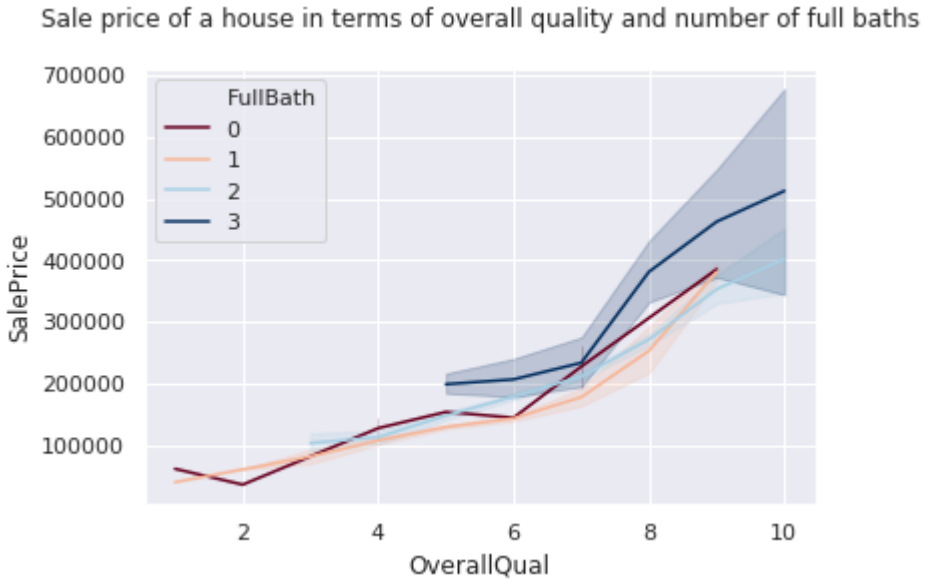
From the heatmap, we can see a high correlation value of **0.88** between GarageArea and GarageCars and a negative correlation value of **-0.11** between OverallQual and Heating.

This gives us an insight of how the data is, where having heating in the house doesn't necessarily mean Overall better quality of the house.

Whereas, a house with bigger garage area might most probably have more cars in it.

Part 2 - Informative Plots

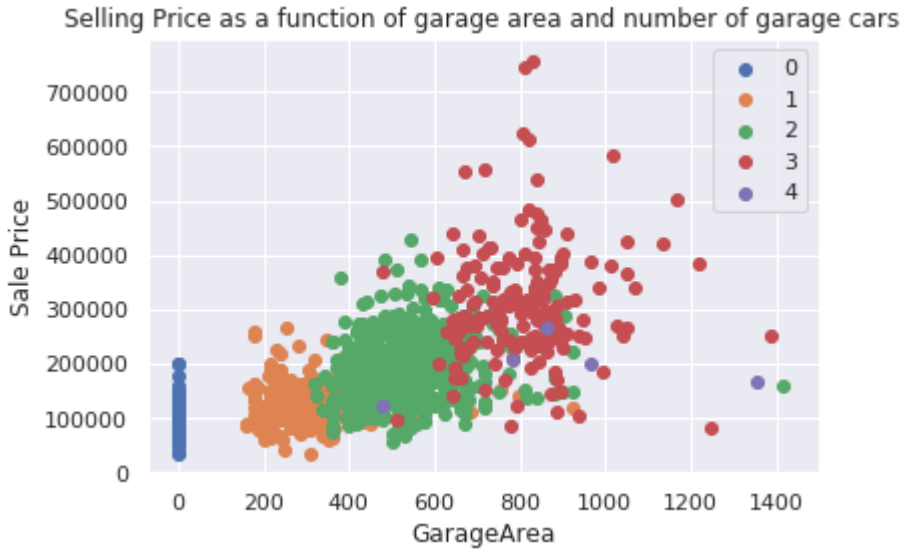
In [346]:
```
# TODO: code to generate Plot 1

ax = sns.lineplot(x=train_data_filtered['OverallQual'],
                  y=train_data_filtered['SalePrice'], hue = train_data_filtere
d['FullBath'],palette = "RdBu",
                  data=train_data_filtered)
plt.title("Sale price of a house in terms of overall quality and number of ful
l baths\n");
```



Question: What interesting properties does Plot 1 reveal?

Answer: Plot1 shows that the selling price of a house increases with the overall quality of the house or the better the house in terms of quality, the higher the selling price. Also, the colors denote the number of full baths in each house. Clearly, houses with more number of full baths cost higher.

In [347]:
```
# TODO: code to generate Plot 2
#Scatter plot
for g in sorted(train_data['GarageCars'].unique()):
    plt.scatter(train_data[train_data['GarageCars']==g]['GarageArea'],
        train_data[train_data['GarageCars']==g]['SalePrice'],
                label = g)
plt.xlabel("GarageArea")
plt.ylabel("Sale Price")
plt.title("Selling Price as a function of garage area and number of garage car
s")
plt.legend()
plt.show();
```

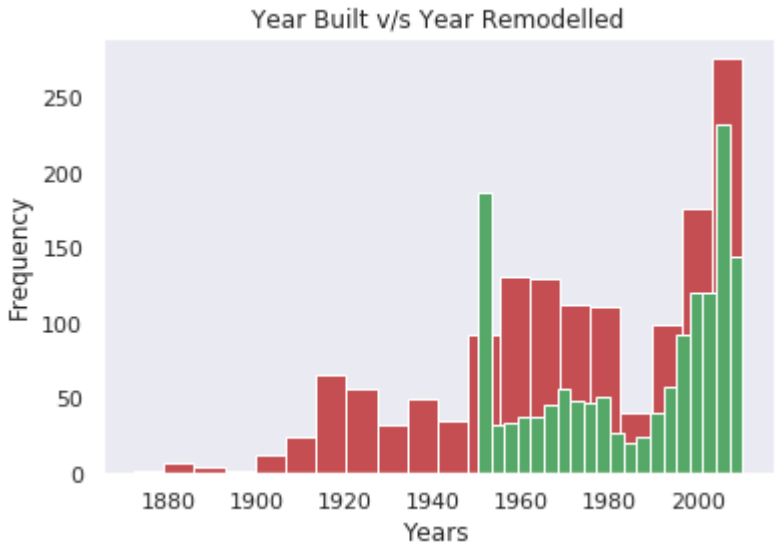


Question: What interesting properties does Plot 2 reveal?

Answer: The above scatter plot provides us two interesting insights:

1. It clearly shows that the area of a garage is mostly proportional to the number of cars in it, Garage area between 200-400sq units will be likely to have 1-2 cars in it, area between 400-800sq units will have 2-3 cars, however the data for larger garage areas has some outliers where there might be 2-4 cars in extremely huge garage spaces.
2. The sale price remains below 500000 even when the garage area continues increasing excluding some outliers. Maybe people don't pay higher amount for just a huge garage!!

In [348]:
```
# TODO: code to generate Plot 3

plt.hist(train_data['YearBuilt'],  bins = 20,color = "r")
plt.hist(train_data['YearRemodAdd'], bins = 20, color = "g")
plt.xlabel("Years")
plt.ylabel("Frequency")
plt.title("Year Built v/s Year Remodelled")
plt.grid()
plt.show()
```

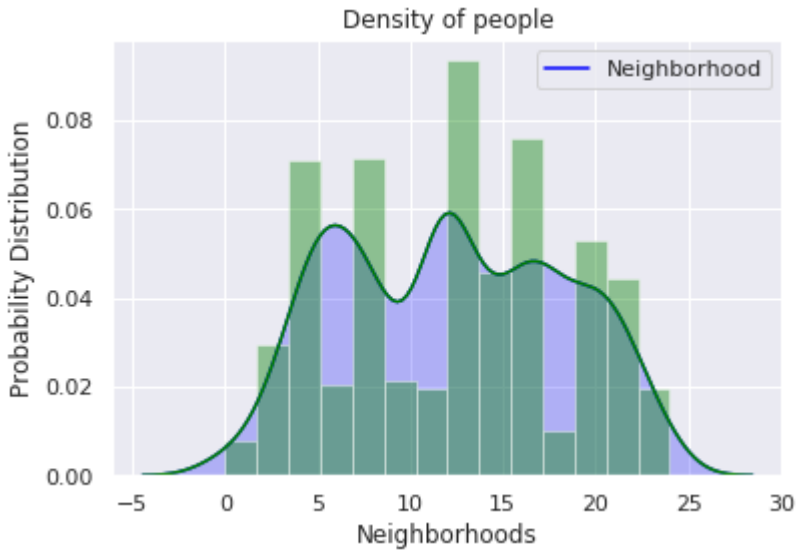


Question: What interesting properties does Plot 3 reveal?

Answer: The interesting insight this plot provides us with is that houses weren't remodelled that frequently between the time period of 1880 - 1950 even when they were being built at a decent frequency. After 1950, there was a surge in houses being remodelled.

Maybe there was an increase in the number of people opting for interior designing jobs and hence a rise in houses being remodelled.

In [349]:
```
# TODO: code to generate Plot 4

sns.kdeplot(train_data['Neighborhood'], shade = True, color = 'blue')
sns.distplot(train_data['Neighborhood'],color = 'green')
plt.xlabel("Neighborhoods")
plt.ylabel("Probability Distribution")
plt.title("Density of people");
```

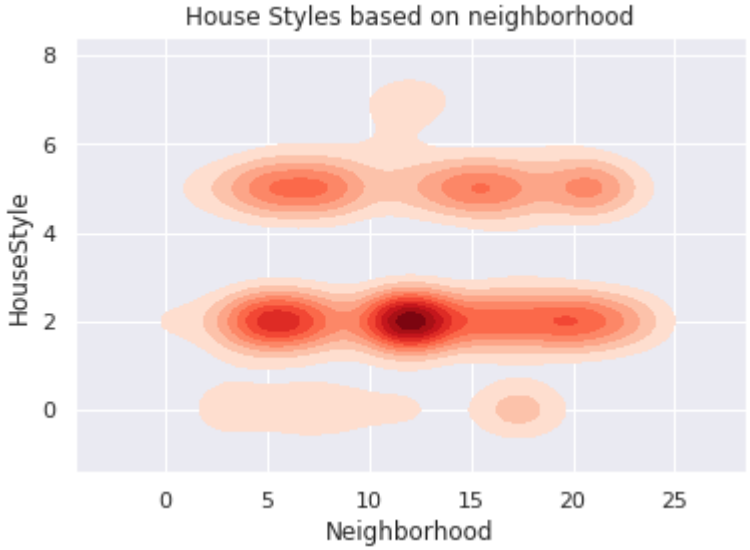


Question: What interesting properties does Plot 4 reveal?

Answer: Since we have mapped neighborhoods to a numeric value, we can see that the number of people staying in neighborhoods mapped to 10-17 are the highest.

People might prefer these areas for living. This helps the real estate agents decide the prices based on popularity of a certain neighborhood.

In [350]:
```
#Plotting a kernel destination estimate graph in two dimensional space
sns.kdeplot(train_data['Neighborhood'],
            train_data['HouseStyle'],
            color='r', shade=True,
            cmap="Reds", shade_lowest=False)
plt.title("House Styles based on neighborhood");
```



Question: What interesting properties does Plot 5 reveal?

Answer: The above plot can be a good visual to those who want to understand what kind of houses are found in what neighborhood.

From above, we can say neighborhood values mapped to 10-14 most usually have house styles mapped to 2. The darker the shade of the kernel the higher the density in the position.

Part 3 - Handcrafted Scoring Function

In [351]:
```
# TODO: code for scoring function
#My scoring function
w1 = 3000
w2 = 1000
w3 = 2000
desirability_score = w1 * train_data['OverallQual'] + w2 * train_data['PoolAre
a'] + w3 * train_data['Utilities']  - 0.1 * train_data['SalePrice']
#desirability_score

train_data['DesirabilityScore'] = desirability_score
largest = pd.DataFrame(train_data.nlargest(10,'DesirabilityScore'))
smallest = pd.DataFrame(train_data.nsmallest(10,'DesirabilityScore'))
print("=========Below are the top 10 most desirable houses============")
print(largest)
print("\n")
print("=========Below are the top 10 least desirable houses===========")
print(smallest)
```

```
=========Below are the top 10 most desirable houses=============
        Id  MSSubClass  MSZoning  ...  SaleCondition  SalePrice  Desirability
Score
1423  1424          80         3  ...              2     274970           728
503.0
810    811          20         3  ...              4     181000           647
900.0
1170  1171          80         3  ...              4     171000           576
900.0
1386  1387          60         3  ...              4     250000           515
000.0
197    198          75         3  ...              0     235000           512
500.0
1182  1183          60         3  ...              0     745000           510
500.0
1298  1299          60         3  ...              5     160000           494
000.0
632    633          20         3  ...              3      82500            12
750.0
1349  1350          70         4  ...              4     122000            11
800.0
523    524          60         3  ...              5     184750            11
525.0

[10 rows x 82 columns]


=========Below are the top 10 least desirable houses============
        Id  MSSubClass  MSZoning  ...  SaleCondition  SalePrice  Desirability
Score
691    692          60         3  ...              4     755000           -45
500.0
898    899          20         3  ...              5     611657           -34
165.7
1169  1170          60         3  ...              4     625000           -32
500.0
803    804          60         3  ...              5     582933           -31
293.3
769    770          60         3  ...              4     538000           -29
800.0
1046  1047          60         3  ...              5     556581           -28
658.1
440    441          20         3  ...              4     555000           -25
500.0
178    179          20         3  ...              5     501837           -23
183.7
798    799          60         3  ...              5     485000           -21
500.0
473    474          20         3  ...              5     440000           -20
000.0

[10 rows x 82 columns]
```

Question: What are the ten most desirable houses?

Answer: The ten most desirable houses in decreasing order of desirability are the ones with house ids as: 1424, 811, 1171, 1387, 198, 1183, 1299, 633, 1350, 524.

Question: What are the ten least desirable houses?

Answer: The ten least desirable houses in increasing order of desirability are the ones with house ids as: 692, 899, 1170, 804, 770, 1047, 441, 179, 799, 474.

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

Answer: I computed the desirability of houses as a weighted sum of four features, namely - Overall quality, Pool Area, Utilities and Sale Price.

For me, personally the quality of a house is of utmost importance and hence I gave it a larger weight. The rest were given weights accordingly and because of my low budget, I would prefer a house with a lower selling price. I have then defined my scoring function as a function of above. I think it worked pretty well.

Part 4 - Pairwise Distance Function

In [352]:
```
# TODO: code for distance function

train_data_filtered = train_data.filter(['OverallQual','GrLivArea','GarageCar
s','GarageArea','TotalBsmtSF','1stFlrSF','FullBath','TotRmsAbvGrd','YearBuilt'
,'YearRemodAdd','MasVnrArea',
 'Fireplaces','GarageYrBlt','BsmtFinSF1','Foundation','LotFrontage','WoodDeckS
F','2ndFlrSF','OpenPorchSF','HalfBath','LotArea', 'CentralAir',
 'Electrical','PavedDrive','BsmtFullBath','RoofStyle','BsmtUnfSF','SaleConditi
on','Neighborhood'])

temp = train_data_filtered
temp = temp.apply(zscore)

#Defining a pairwise distance function
#writing my own pairwise function based on Euclidean distance formula


pairwise = pd.DataFrame(
  squareform(pdist(temp,metric = "euclidean")),
  columns = temp.index,
  index = temp.index
)
print(pairwise)

print("Minimum distance value:",pairwise.values.min())
print("Maximum distance value:",pairwise.values.max())

print("Distance between house 1001 and house 5:",pairwise[1000][5])
print("Distance between house 19 and house 34:",pairwise[20][35])
```

	0	1	2	...	1457	1458	1459
0	0.000000	6.558119	2.223984	...	6.891454	8.158297	7.883417
1	6.558119	0.000000	6.217629	...	7.943080	6.770880	6.211999
2	2.223984	6.217629	0.000000	...	6.330828	8.079341	7.980329
3	6.894359	7.418821	6.519514	...	8.867026	8.686144	9.065447
4	4.081255	6.430606	3.953747	...	7.313878	9.276918	8.411336
5	5.581848	7.798948	5.565848	...	7.389157	8.713799	8.621660
6	5.768866	4.326673	5.583883	...	7.895465	8.192171	7.413516
7	5.352589	5.835736	4.791153	...	7.137956	8.442796	7.080765
8	8.955130	7.242390	8.365937	...	8.418790	8.279090	9.848334
9	7.778098	6.435265	7.146365	...	8.492307	7.198242	7.500902
10	6.614804	5.106498	6.596116	...	8.793386	5.601526	7.119939
11	6.423956	7.547518	6.304592	...	7.821172	10.340438	10.583110
12	7.026379	5.067650	6.888704	...	9.103163	5.094332	6.440414
13	6.251776	6.389386	5.512103	...	7.601145	9.070597	8.944811
14	6.574697	6.571998	6.217807	...	8.363291	7.033552	7.206380
15	7.519908	7.565154	7.342588	...	9.524726	6.136933	8.752903
16	5.463454	4.577269	4.861669	...	7.728130	6.397546	7.037722
17	6.487702	5.696381	6.468860	...	7.496902	7.649335	8.569310
18	4.629892	5.492763	4.682312	...	8.290565	7.386208	7.361606
19	7.614100	6.352136	7.563378	...	8.544296	6.782294	8.046753
20	6.241553	7.676321	5.913384	...	7.831832	10.743435	9.981035
21	8.707521	7.777376	8.321757	...	8.042525	7.188435	9.249018
22	7.166498	7.298465	6.575767	...	7.440272	8.905083	9.088575
23	5.197559	4.908940	4.943953	...	7.204344	6.631685	6.475784
24	6.821230	4.419204	6.540942	...	8.069393	5.267117	4.852434
25	7.216292	7.245752	6.581170	...	8.563578	10.294066	10.594435
26	5.885327	4.528756	5.797108	...	8.461836	5.979621	6.288562
27	5.490549	5.530311	5.149599	...	8.001797	9.127104	9.061543
28	7.769333	6.155210	7.344721	...	8.063835	7.650822	6.509263
29	9.470456	8.387786	9.284711	...	10.437922	8.083023	8.839350
...	...	...	...	...	...	...	...
1430	4.246606	6.328505	3.570806	...	6.100033	8.201762	8.106680
1431	5.198744	3.935778	5.335502	...	7.982537	6.866895	7.255159
1432	7.692269	5.872435	7.497763	...	8.478396	6.631864	8.248763
1433	3.469653	6.540981	2.774114	...	6.245371	8.298146	8.369140
1434	6.051685	3.834807	5.576154	...	8.267295	6.445320	5.613056
1435	6.786584	7.044927	6.210130	...	8.045260	8.219207	8.449759
1436	5.826203	4.881472	5.752905	...	7.806646	6.642486	7.301405
1437	6.681183	6.817511	6.406673	...	8.849501	9.506371	9.893466
1438	7.003722	6.694189	6.996265	...	9.244773	6.089204	8.545110
1439	4.809160	5.431976	4.342899	...	7.242944	7.357259	8.094573
1440	7.245510	6.598531	7.181739	...	7.926267	8.467925	7.666953
1441	5.393569	5.711531	4.714446	...	8.219909	6.751991	7.027428
1442	4.762837	6.670549	4.908003	...	7.918643	9.795615	9.191028
1443	10.073380	8.910106	9.580113	...	9.987291	7.890671	9.679688
1444	5.424601	5.823059	5.167233	...	7.064963	7.915701	7.707385
1445	5.928120	5.022343	6.026319	...	8.435866	6.065148	7.033053
1446	6.924406	4.959862	6.859863	...	8.215142	6.327391	5.693486
1447	2.874898	6.666855	2.928436	...	6.598381	8.920830	8.332010
1448	7.505568	7.577576	7.398357	...	8.473059	7.500208	7.639306
1449	8.205016	8.114959	8.170802	...	10.357858	7.942329	8.692593
1450	6.028288	7.521066	6.179029	...	7.909188	8.632765	8.046859
1451	6.784047	6.126494	6.081061	...	7.849560	9.603022	10.087658
1452	4.684070	6.275088	4.807719	...	8.296739	6.997924	7.722300
1453	7.696805	7.534246	7.550123	...	8.327349	8.189413	8.719275
1454	4.879504	5.026913	4.851458	...	7.490617	7.519305	8.080414

```
1455   3.843308  5.824879  3.018404  ...   5.669232   8.062641   8.173574
1456   6.959946  4.274791  6.175447  ...   7.569113   7.600928   6.591150
1457   6.891454  7.943080  6.330828  ...   0.000000   9.494344   9.961000
1458   8.158297  6.770880  8.079341  ...   9.494344   0.000000   6.445733
1459   7.883417  6.211999  7.980329  ...   9.961000   6.445733   0.000000

[1460 rows x 1460 columns]
Minimum distance value: 0.0
Maximum distance value: 32.16594339108772
Distance between house 1001 and house 5: 7.86190062790858
Distance between house 19 and house 34: 2.8127790921693068
```

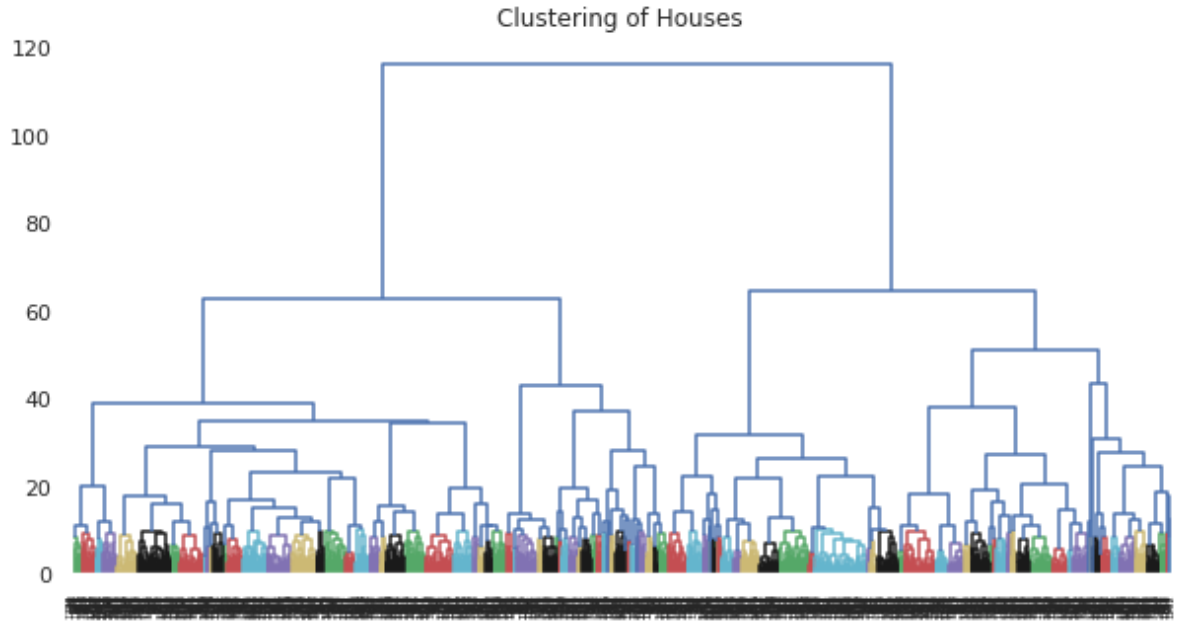
Question: How well does the distance function work? When does it do well/badly?

Answer: The distance function works pretty well as we can see that for houses with ids 1001 and 5, it gives a higher distance of 7.86 which makes sense as these houses have varied properties and have very less features in common.

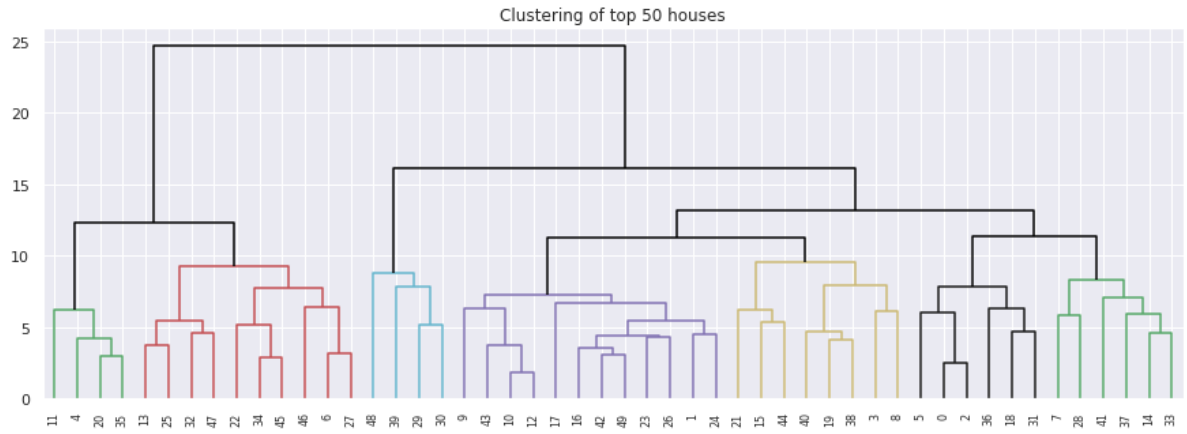
As opposed to above, houses with ids 19 and 34 have more features in common and hence have a lesser distance of 2.812.

Part 5 - Clustering

In [353]:
```
# TODO: code for clustering and visualization
train_data_filtered = train_data.filter(['OverallQual','GrLivArea','GarageCar
s','GarageArea','TotalBsmtSF','1stFlrSF','FullBath','TotRmsAbvGrd','YearBuilt'
,'YearRemodAdd','MasVnrArea',
 'Fireplaces','GarageYrBlt','BsmtFinSF1','Foundation','LotFrontage',
 'Electrical','PavedDrive','SalePrice'])


#Clustering - we feed our condensed distance matrix to it
plt.figure(figsize=(10,5))
linkage_matrix = linkage(squareform(pairwise), "ward")
dendrogram(linkage_matrix, color_threshold=10, labels=pairwise.index,show_leaf
_counts=True)
plt.title("Clustering of Houses")
plt.show()
```



In [354]:
```
#Now we are not able to visualize anything in above dendrogram, so we plot the
dendrogram for first 50 houses.

temp50 = temp.head(n = 50)
temp50 = temp50.apply(zscore)
#Defining a pairwise distance function
#writing my own pairwise function based on Euclidean distance formula


pairwise50 = pd.DataFrame(
  squareform(pdist(temp50,metric = "euclidean")),
  columns = temp50.index,
  index = temp50.index
)

#pairwise

#Now we calculate our clusters by feeding our condensed matrix to it
plt.figure(figsize=(15,5))
linkage_matrix = linkage(squareform(pairwise50), "ward")
dendrogram(linkage_matrix, color_threshold=10, labels=pairwise50.index,show_le
af_counts=True, above_threshold_color='black')
plt.title("Clustering of top 50 houses")
plt.show()
```



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

Answer: From the clusters above, the different colors help us understand which houses belong in the same clusters based on the distance matrix we calculated earlier. From the first 50 houses, looking at the first group, we can very well see that houses 11,4,20,35 belong in one neighborhood. When we check them with our original data, they however appear in different neighborhoods but they do have many features in common. Since our dendrogram is built on our distance matrix, we see them in the same cluster because of the relative distance being less between them.

Part 6 - Linear Regression

In [355]:
```
# TODO: code for linear regression
train_data_filtered = train_data.filter(['OverallQual','GrLivArea','GarageCar
s','GarageArea','TotalBsmtSF','1stFlrSF','FullBath','TotRmsAbvGrd','YearBuilt'
,'YearRemodAdd','MasVnrArea',
 'Fireplaces','GarageYrBlt','BsmtFinSF1','Foundation','LotFrontage','WoodDeckS
F','2ndFlrSF','OpenPorchSF','HalfBath','LotArea', 'CentralAir',
 'Electrical','PavedDrive','BsmtFullBath','RoofStyle','BsmtUnfSF','SaleConditi
on','Neighborhood'])

x_train = train_data[['GrLivArea','OverallQual','GarageCars','Heating']]
y_train = np.log(train_data[['SalePrice']])
data_train, data_test, target_train, target_test = train_test_split(x_train,y_
train,test_size = 0.4, random_state = 1)
linear_reg = LinearRegression().fit(data_train,target_train)
preds = linear_reg.predict(data_test)
error1 = np.sqrt(metrics.mean_squared_error(target_test,preds))
print("Root mean square error with 4 variables:",error1)


x_train = train_data[['GrLivArea','GarageCars','Heating']]
y_train = np.log(train_data[['SalePrice']])
data_train, data_test, target_train, target_test = train_test_split(x_train,y_
train,test_size = 0.4, random_state = 1)
linear_reg = LinearRegression().fit(data_train,target_train)
#test_data_final = test_data[['OverallQual']]
preds = linear_reg.predict(data_test)
error2 = np.sqrt(metrics.mean_squared_error(target_test,preds))
print("Root mean square error without OverallQual variable:",error2)


x_train = train_data[['GrLivArea','OverallQual','Heating']]
y_train = np.log(train_data[['SalePrice']])
data_train, data_test, target_train, target_test = train_test_split(x_train,y_
train,test_size = 0.4, random_state = 1)
linear_reg = LinearRegression().fit(data_train,target_train)
#test_data_final = test_data[['OverallQual']]
preds = linear_reg.predict(data_test)
error3 = np.sqrt(metrics.mean_squared_error(target_test,preds))
print("Root mean square error without GarageCars variable:",error3)

x_train = train_data[['GarageCars','OverallQual','Heating']]
y_train = np.log(train_data[['SalePrice']])
data_train, data_test, target_train, target_test = train_test_split(x_train,y_
train,test_size = 0.4, random_state = 1)
linear_reg = LinearRegression().fit(data_train,target_train)
#test_data_final = test_data[['OverallQual']]
preds = linear_reg.predict(data_test)
error4 = np.sqrt(metrics.mean_squared_error(target_test,preds))
print("Root mean square error without GrLivArea variable:",error4)


x_train = train_data[['GarageCars','OverallQual','GrLivArea']]
y_train = np.log(train_data[['SalePrice']])
data_train, data_test, target_train, target_test = train_test_split(x_train,y_
train,test_size = 0.4, random_state = 1)
linear_reg = LinearRegression().fit(data_train,target_train)
```

```
#test_data_final = test_data[['OverallQual']]
preds = linear_reg.predict(data_test)
error5 = np.sqrt(metrics.mean_squared_error(target_test,preds))
print("Root mean square error without Heating variable:",error5)
```

```
Root mean square error with 4 variables: 0.20213711239662535
Root mean square error without OverallQual variable: 0.2531612158875837
Root mean square error without GarageCars variable: 0.21790163779641303
Root mean square error without GrLivArea variable: 0.2204949193233532
Root mean square error without Heating variable: 0.20352190826019811
```

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

Answer: I built the regression model with 4 variables, predicting the pricing. The model worked best and had the least error when the following four features were used: OverallQual, GrLivArea, GarageCars and Heating.

By playing around with the above four features, I noticed that the **OverallQual** was the most important variable as when I built a model without it (case 2 above), the error rose by 20%. Whereas out of the four, the model definitely wasn't impacted with or without the **Heating** variable much.

Part 7 - External Dataset

In [356]:
```
# TODO: code to import external dataset and test
external_data = pd.read_csv("/content/drive/My Drive/external_dataset.csv")

print("File reading successfull!!")

external_data = external_data.filter(['YearBuilt','UseCd','WallHt','HeatedArea','ACArea','WdJst_sf'])
merged_data = train_data.join(external_data)

#Fill NaN with mode
merged_data = merged_data.fillna(merged_data.mode().iloc[0])

print("All null values replaced by mode of the column!")

#Mapping categorical data to numeric values
for col in merged_data:
  if merged_data[col].dtype == 'object':
    le = preprocessing.LabelEncoder()
    le.fit(list(merged_data[col].values))
    merged_data[col] = le.transform(list(merged_data[col].values))

print("Categorical values mapped to numeric!")
```

```
File reading successfull!!
All null values replaced by mode of the column!
Categorical values mapped to numeric!
```

In [364]:
```
#Original dataset
train_data_filtered = train_data.filter(['OverallQual','GrLivArea','GarageCar
s','GarageArea','TotalBsmtSF','1stFlrSF','FullBath','TotRmsAbvGrd','YearBuilt'
,'YearRemodAdd','MasVnrArea',
 'Fireplaces','GarageYrBlt','BsmtFinSF1','Foundation','LotFrontage','WoodDeckS
F','2ndFlrSF','OpenPorchSF','HalfBath','LotArea', 'CentralAir',
 'Electrical','PavedDrive','SalePrice'])
features_train = train_data_filtered.drop('SalePrice',axis = 1)
label_train = np.log(train_data[['SalePrice']])
x_train = features_train
y_train = label_train
data_train, data_test, target_train, target_test = train_test_split(x_train,y_
train, test_size = 0.4, random_state = 1)
linear_reg = LinearRegression().fit(data_train,target_train)
preds = linear_reg.predict(data_test)
#Calcuating root mean square error locally
error = np.sqrt(metrics.mean_squared_error(target_test,preds))
print("RMSE of original dataset:",error)
```

```
RMSE of original dataset: 0.18368250633921354
```

In [365]:
```
#Checking if my model works better with the external dataset

merged_data_filtered = merged_data.filter(['OverallQual','GrLivArea','GarageCa
rs','GarageArea','TotalBsmtSF','1stFlrSF','FullBath','TotRmsAbvGrd','YearBuil
t','YearRemodAdd','MasVnrArea',
 'Fireplaces','GarageYrBlt','BsmtFinSF1','Foundation','LotFrontage','WoodDeckS
F','2ndFlrSF','OpenPorchSF','HalfBath','LotArea', 'CentralAir',
 'Electrical','PavedDrive','SalePrice','UseCd','WallHt','HeatedArea','ACArea',
'WdJst_sf'])

x_train = merged_data.drop('SalePrice',axis = 1)
y_train = np.log(merged_data[['SalePrice']])

data_train, data_test, target_train, target_test = train_test_split(x_train,y_
train,test_size = 0.4, random_state = 1)

linear_reg = LinearRegression().fit(data_train,target_train)
preds = linear_reg.predict(data_test)

error = np.sqrt(metrics.mean_squared_error(target_test,preds))
print("RMSE of merged dataset:",error)
```

```
RMSE of merged dataset: 0.11007500190365282
```

Question: Describe the dataset and whether this data helps with prediction.

Answer: This dataset is taken from the government city assessor website of City of Ames. The data has characteristics for non-residential and condominium apartments and I have used five features/variables from the data and merged it on the "YearBuilt" column with my original dataset.

The columns used were:

UseCd - Used by whether florists, apartment residents - whether owned by a shop.

WallHt - Height of walls of the houses.

HeatedArea - Heating area.

ACArea - Air conditioning area.

WdJst_SF - Surface area of width adjustment.

The model definitely helped in prediction, the additional attributes helped in better predicting the pricing of the houses and reduced the overall root mean square error by 63%(earlier - 0.18, now - 0.11)

Part 8 - Permutation Test

In [358]:
```
# TODO: code for all permutation tests

train_data_filtered = train_data.filter(['OverallQual','GrLivArea','YrSold','U
tilities','HeatingQC','GarageFinish','MSZoning','GarageType',
                                         'EnclosedPorch','PoolArea'])


y = np.log(train_data[['SalePrice']])


for col in train_data_filtered:
  x = train_data_filtered[[col]]
  data_train, data_test, target_train, target_test = train_test_split(x,y,test
_size = 0.2, random_state = 1)
  linear_reg = LinearRegression().fit(data_train,target_train)
  preds = linear_reg.predict(data_test)
  score, permutation_scores, pvalue = permutation_test_score(linear_reg, x, y,
n_permutations=100, n_jobs=1)
  print("p-value between",col,"and Sale Price:",pvalue)
```

```
p-value between OverallQual and Sale Price: 0.009900990099009901
p-value between GrLivArea and Sale Price: 0.009900990099009901
p-value between YrSold and Sale Price: 0.27722772277227725
p-value between Utilities and Sale Price: 0.4158415841584158
p-value between HeatingQC and Sale Price: 0.009900990099009901
p-value between GarageFinish and Sale Price: 0.009900990099009901
p-value between MSZoning and Sale Price: 0.009900990099009901
p-value between GarageType and Sale Price: 0.009900990099009901
p-value between EnclosedPorch and Sale Price: 0.009900990099009901
p-value between PoolArea and Sale Price: 0.039603960396039604
```

Question: Describe the results.

Answer: A p-value less than 0.05 is statistically significant. It indicates strong evidence against the hypothesis that there is no relationship between the two variables. If p-value is lesser than 0.05, then the two variables are indeed strongly correlated and we can safely say the results are not random.

In this case, the p-value for variables "OverallQual", "GrLivArea", "HeatingQC","GarageFinish","MSZoning","GarageType","EnclosedPorch" and "PoolArea" with "SalePrice" are lower than 0.05 and hence show greater correlation as opposed to randomness.

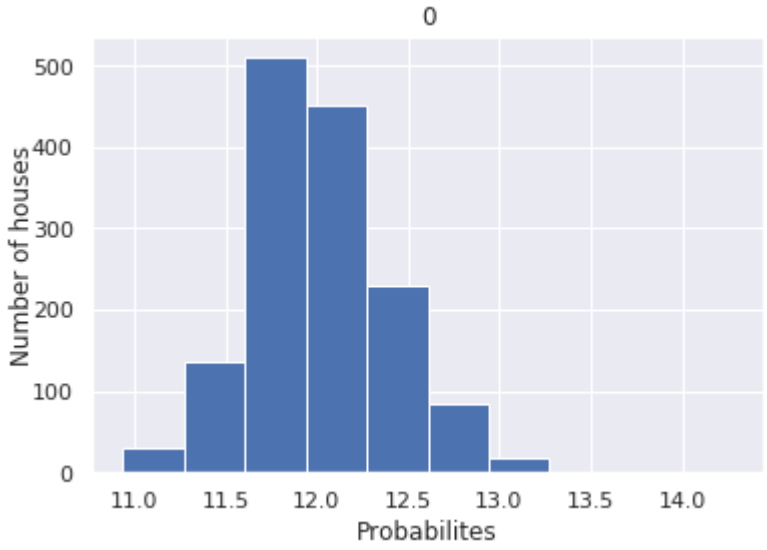
Whereas, the p-values for "YrSold" and "Utilities" with "SalePrice" are greater than 0.05 and signify randomness.

Part 9 - Final Result

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.

In [376]:
```
train_data_filtered = train_data.filter(['OverallQual','GrLivArea','GarageCar
s','GarageArea','TotalBsmtSF','1stFlrSF','FullBath','TotRmsAbvGrd','YearBuilt'
,
'YearRemodAdd','MasVnrArea','Fireplaces','GarageYrBlt','BsmtFinSF1','Foundatio
n','LotFrontage','WoodDeckSF','2ndFlrSF','OpenPorchSF','HalfBath','LotArea',
'CentralAir','Electrical','PavedDrive','BsmtFullBath','RoofStyle','BsmtUnfSF',
'SaleCondition','Neighborhood','SalePrice'])
features_train = train_data_filtered.drop('SalePrice',axis = 1)
label_train = np.log(train_data[['SalePrice']])


x_train = features_train
y_train = label_train
data_train, data_test, target_train, target_test = train_test_split(x_train,y_
train, test_size = 0.4, random_state = 1)
linear_reg = LinearRegression().fit(data_train,target_train)
preds = linear_reg.predict(data_test)

test_data_final = test_data[['OverallQual','GrLivArea','GarageCars','GarageAre
a','TotalBsmtSF','1stFlrSF','FullBath','TotRmsAbvGrd','YearBuilt',
'YearRemodAdd','MasVnrArea','Fireplaces','GarageYrBlt','BsmtFinSF1','Foundatio
n','LotFrontage','WoodDeckSF','2ndFlrSF','OpenPorchSF','HalfBath','LotArea',
'CentralAir','Electrical','PavedDrive','BsmtFullBath','RoofStyle','BsmtUnfSF',
'SaleCondition','Neighborhood']]

#Calcuating root mean square error locally
error = np.sqrt(metrics.mean_squared_error(target_test,preds))
print("RMSE of final model locally:",error)

#test_data_final.shape
preds_final = linear_reg.predict(test_data_final)


preds_final = np.exp(preds_final)
#Plotting the probabilities
pd.DataFrame(np.log(preds_final)).hist()
plt.xlabel("Probabilites", ha = 'center')
plt.ylabel("Number of houses", va = 'center')

sub = pd.read_csv('/content/drive/My Drive/sample_submission.csv')
result = pd.DataFrame()
result['Id'] = test_data['Id']
result['SalePrice'] = preds_final
result.to_csv('myPrediction11.csv',index = False)
```

```
RMSE of final model locally: 0.180276712080289
```



Kaggle Link: https://www.kaggle.com/payal95 (https://www.kaggle.com/payal95)

Highest Rank: 2726

Score: 0.143

Number of entries: 10

In [1]:
```
%%html
<iframe src = "https://drive.google.com/uc?id=1vTLO6M-Nlu7TPjCX1FzzKO60ox_QUceI" width = "840" height = "480"></iframe>
```