In [442]:

```python
# Import statements
# https://matplotlib.org/3.1.1/gallery/style_sheets/ggplot.html
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from scipy.stats import skew
from scipy.special import boxcox1p
from sklearn.preprocessing import RobustScaler
from sklearn.linear_model import Lasso, LassoCV
from sklearn.model_selection import permutation_test_score
from sklearn.metrics import make_scorer
from sklearn import linear_model
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from lightgbm import LGBMRegressor
from xgboost import XGBClassifier
from scipy.cluster.hierarchy import linkage,dendrogram
plt.style.use('ggplot')
```

In [364]:

```python
# Reading the Train File
data_train = pd.read_csv(r'I:\Data Science Fundamentals\house-prices-advanced-regression-techniques\train.csv')
data_train_copies = pd.read_csv(r'I:\Data Science Fundamentals\house-prices-advanced-regression-techniques\train.csv')
```

In [365]:

```python
# Checking if it is loaded properly
data_train.head(20)
```

Out[365]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleC
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2008	WD	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	5	2007	WD	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2008	WD	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2006	WD	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12	2008	WD	
5	6	50	RL	85.0	14115	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	MnPrv	Shed	700	10	2009	WD	
6	7	20	RL	75.0	10084	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	8	2007	WD	
7	8	60	RL	NaN	10382	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	Shed	350	11	2009	WD	
8	9	50	RM	51.0	6120	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	4	2008	WD	
9	10	190	RL	50.0	7420	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	1	2008	WD	
10	11	20	RL	70.0	11200	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2008	WD	
11	12	60	RL	85.0	11924	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	7	2006	New	
12	13	20	RL	NaN	12968	Pave	NaN	IR2	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2008	WD	
13	14	20	RL	91.0	10652	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	8	2007	New	
14	15	20	RL	NaN	10920	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	GdWo	NaN	0	5	2008	WD	
15	16	45	RM	51.0	6120	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	GdPrv	NaN	0	7	2007	WD	
16	17	20	RL	NaN	11241	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	Shed	700	3	2010	WD	
17	18	90	RL	72.0	10791	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	Shed	500	10	2006	WD	
18	19	20	RL	66.0	13695	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	6	2008	WD	
19	20	20	RL	70.0	7560	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0	5	2009	COD	

20 rows × 81 columns

In [366]:

```python
# Reading the Test File
data_test = pd.read_csv(r'I:\Data Science Fundamentals\house-prices-advanced-regression-techniques\test.csv')
data_test_copy = pd.read_csv(r'I:\Data Science Fundamentals\house-prices-advanced-regression-techniques\test.csv')
```

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

In [367]:

```python
# Task 1
# Create a list of features to do a pairwise correlation
correlation_features = data_train[['SalePrice', 'OverallQual', 'YearBuilt', 'BedroomAbvGr', 'FullBath', 'GarageArea', 'GrLivArea', 'Fireplaces', 'TotRmsA
bvGrd', 'TotalBsmtSF', '1stFlrSF', 'LotArea']]
```

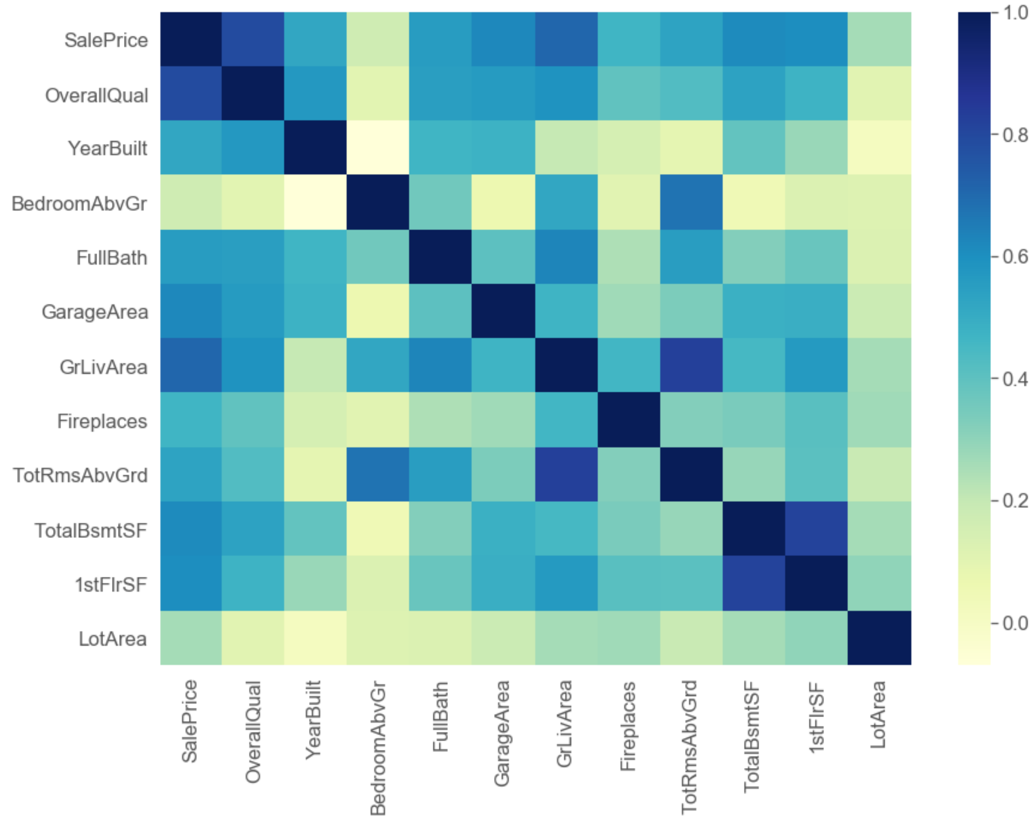
In [368]:
```
correlation_features.head(10)
```

Out[368]:

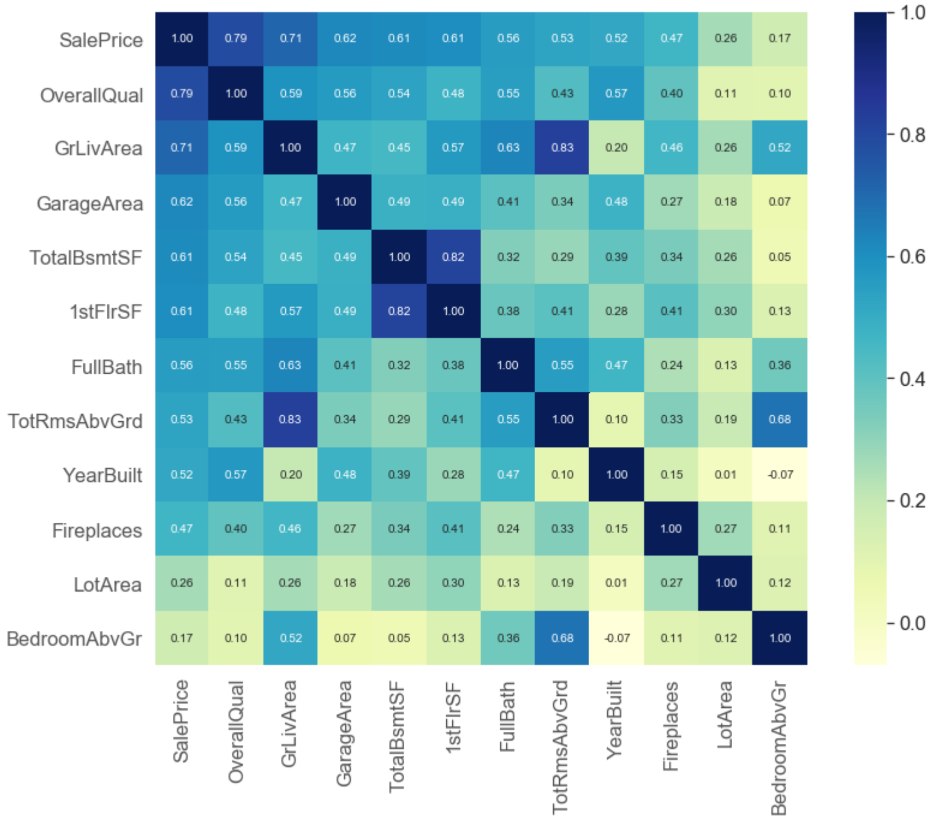
	SalePrice	OverallQual	YearBuilt	BedroomAbvGr	FullBath	GarageArea	GrLivArea	Fireplaces	TotRmsAbvGrd	TotalBsmtSF	1stFlrSF	LotArea
0	208500	7	2003	3	2	548	1710	0	8	856	856	8450
1	181500	6	1976	3	2	460	1262	1	6	1262	1262	9600
2	223500	7	2001	3	2	608	1786	1	6	920	920	11250
3	140000	7	1915	3	1	642	1717	1	7	756	961	9550
4	250000	8	2000	4	2	836	2198	1	9	1145	1145	14260
5	143000	5	1993	1	1	480	1362	0	5	796	796	14115
6	307000	8	2004	3	2	636	1694	1	7	1686	1694	10084
7	200000	7	1973	3	2	484	2090	2	7	1107	1107	10382
8	129900	7	1931	2	2	468	1774	2	8	952	1022	6120
9	118000	5	1939	2	1	205	1077	2	5	991	1077	7420

In [369]:
```
# Find pearson correlation matrix
correlation = correlation_features.corr(method ='pearson')
```

In [370]:
```
# Create a Heatmap from the features selected above
ax = plt.subplots(figsize=(14, 10))
sns.heatmap(correlation,cmap='YlGnBu')
plt.show()
```



In [371]:
```python
# Create a Heatmap with Correlation Values for better understanding
f  = 12
fig,ax=plt.subplots(figsize=(14,10))
cols = correlation.nlargest(f, 'SalePrice')['SalePrice'].index
cm = np.corrcoef(data_train[cols].values.T)
sns.set(font_scale=1.5)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10}, yticklabels=cols.values, xticklabels=cols.values, cmap='YlGnBu')
plt.show()
```



Analysis: As we can see from the heatmap above, features like OverallQual and SalePrice has a positive high correlation(0.79). Also, TotalBsmtSF and 1stFlrSF has a high pairwise correlation (0.82), along with GrLivArea and TotRmsAbvGrd, who also share positive correlation (0.83) amongst themselves. Coming to negative correlation, we see that YearBuilt and BedroomAbvGr show such a trend (-0.07). Low correlation can also be seen between YearBuilt and LotArea (0.01).

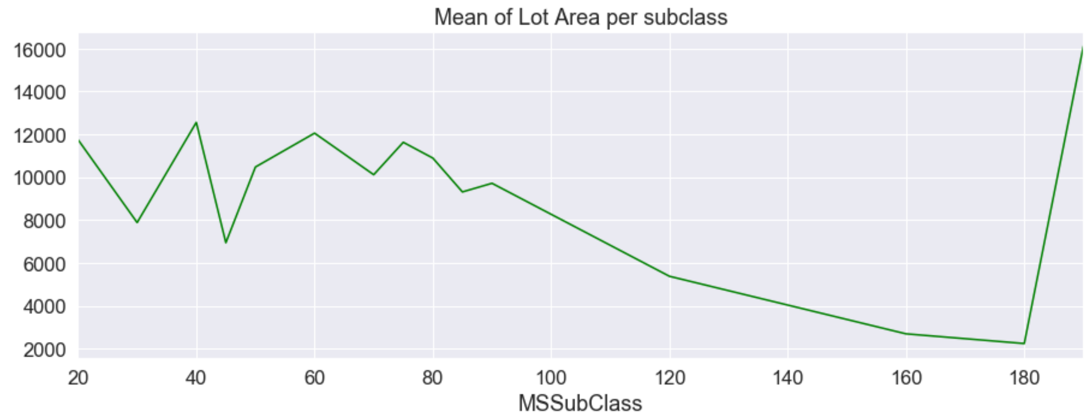
Part 2 - Informative Plots

In [372]:
```python
# Task 2
# Plot 1
# https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.scatter.html
# Scatter Plot showing sales price vs TotalBsmtSF
ax = plt.subplots(figsize=(15, 5))
xax = data_train['GrLivArea']
yax = data_train['SalePrice']
plt.scatter(xax, yax, c=yax, alpha=1, cmap='Spectral')
plt.xlabel('Grade (Ground) Living Area (in square feet)')
plt.ylabel('Sale Price of House')
plt.show()
```

Analysis of Plot 1: From the scatterplot, we can infer that Grade (Ground) Living Area plays a huge role in the sale price of the House. A low Grade Living Area leads to a low Sale Price. The houses with close to zero ground living area leads to sale price being zero as well.

/

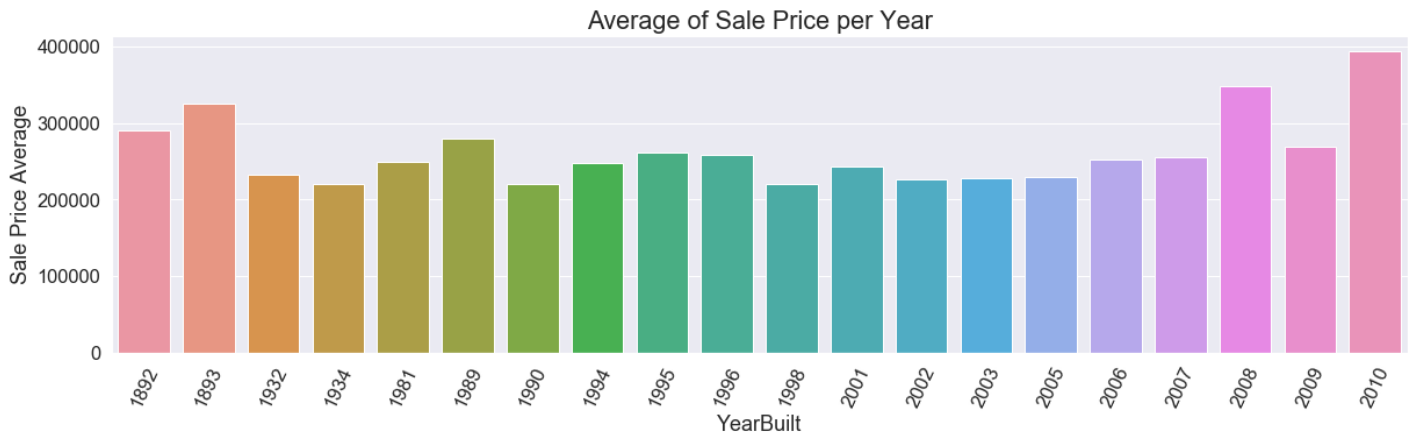
In [373]:
```
# Plot 2
# https://matplotlib.org/tutorials/introductory/pyplot.html
data_train.groupby('MSSubClass') \
    .mean()['LotArea'] \
    .plot(kind='line', title='Mean of Lot Area per subclass', figsize = (15,5), color = 'Green')
plt.show()
```



Analysis of Plot 2: MSSubClass is a field in the dataset that identifies the type of dwelling involved in the sale. For e.g.: Split Foyer, Duplex, etc. In this Line plot, I show how this MSSubClass gives us an idea of the Lot Area of the house. We see that MSSubClass 190 which refers to 2 FAMILY CONVERSION - ALL STYLES AND AGES, has the highest average lot area amongst all types of houses. On the other hand, MSSubClass 180 which refers to PUD - MULTILEVEL - INCL SPLIT LEV/FOYER has the lowest average lot area.

MSSubClass Key:
20 1-STORY 1946 & NEWER ALL STYLES
30 1-STORY 1945 & OLDER
40 1-STORY W/FINISHED ATTIC ALL AGES
45 1-1/2 STORY - UNFINISHED ALL AGES
50 1-1/2 STORY FINISHED ALL AGES
60 2-STORY 1946 & NEWER
70 2-STORY 1945 & OLDER
75 2-1/2 STORY ALL AGES
80 SPLIT OR MULTI-LEVEL
85 SPLIT FOYER
90 DUPLEX - ALL STYLES AND AGES
120 1-STORY PUD (Planned Unit Development) - 1946 & NEWER
150 1-1/2 STORY PUD - ALL AGES
160 2-STORY PUD - 1946 & NEWER
180 PUD - MULTILEVEL - INCL SPLIT LEV/FOYER
190 2 FAMILY CONVERSION - ALL STYLES AND AGES

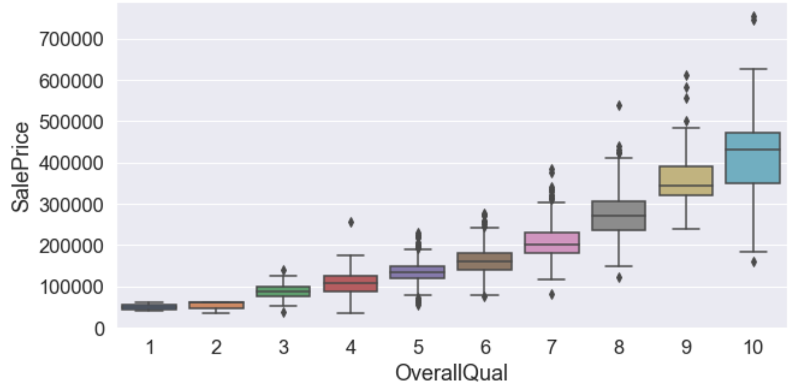
In [374]:
```
# Plot 3
# https://seaborn.pydata.org/generated/seaborn.barplot.html
# https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Python_Seaborn_Cheat_Sheet.pdf
salepriceperyear = pd.DataFrame()
salepriceperyear['Sale Price Average'] = data_train.groupby(['YearBuilt'])['SalePrice'].mean()
salepriceperyear['YearBuilt'] = salepriceperyear.index
group_top = salepriceperyear.sort_values(by='Sale Price Average',ascending=False).head(20)

plt.figure(figsize=(20, 5))
sns.set(color_codes=True)
sns.set(font_scale = 1.5)
ax = sns.barplot(x="YearBuilt", y="Sale Price Average", data=group_top)

font_size= {'size': 'large'}
ax.set_title("Average of Sale Price per Year", **font_size)
xt = plt.xticks(rotation=65)
```



Analysis of Plot 3: In this Bar Plot, what I tried to do is to see whether I can find the relation between the Average Sale Price by the Year Built. Unsurprisingly, the houses which are newly built have a higher sale price. However some antique houses built in the years of 1893 and 1892 also have a high value, and in some cases more than houses built during the years of 1996-2007.

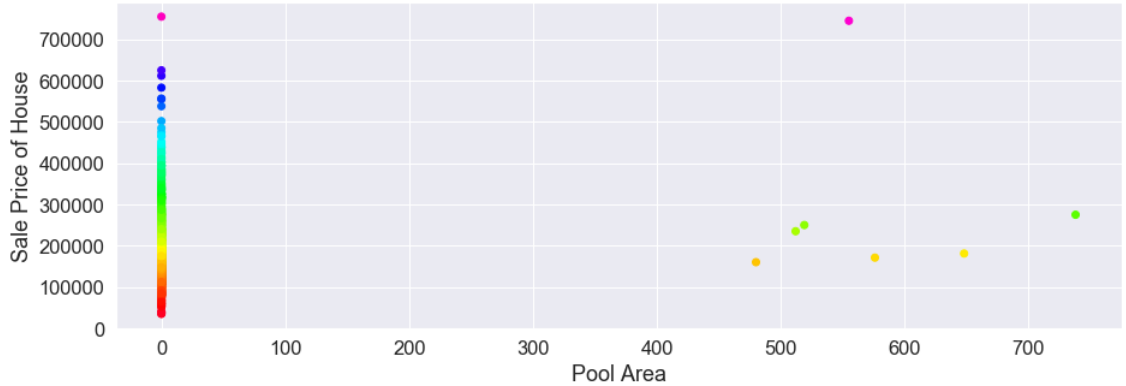
/

In [375]:
```
# Plot 4
# https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.boxplot.html
box_plot = pd.concat([data_train['SalePrice'], data_train['OverallQual']], axis=1)
f, ax = plt.subplots(figsize=(10,5))
fig = sns.boxplot(x='OverallQual', y='SalePrice', data=box_plot)
plt.show()
```



Analysis of Plot 4: We know that a boxplot is a standardized way of displaying the distribution of data based on a five number summary (“minimum”, first quartile (Q1), median, third quartile (Q3), and “maximum”). That's the same thing I am potraying in the boxplot. It shows us the above 5 measures of SalePrice with respect to OverallQual of the house. What I found interesting that although it shows a symmetrical plot, that all the 5 measures increase with respect to quality, there is some anomaly as well. We have some houses in the dataset which have an overall quality of 10, yet its price is less than many houses which have overall quality of 9.

In [376]:
```
# Plot 5
# https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.scatter.html
# Scatter Plot showing sales price vs TotalBsmtSF
ax = plt.subplots(figsize=(15, 5))
xax = data_train['PoolArea']
yax = data_train['SalePrice']
plt.scatter(xax, yax, c=yax, alpha=1, cmap='gist_rainbow')
plt.xlabel('Pool Area')
plt.ylabel('Sale Price of House')
plt.show()
```



Analysis of Plot 5: I find this scatterplot interesting because it tells us one thing for sure, and that is, pool area plays almost no role at all in the sale price of the houses in the dataset. It also shows us that most of the houses at Ames doesn't have a pool at all.

Part 3 - Handcrafted Scoring Function

Task 3

For Desirability, I have chosen five features which I think, will influence the desirability of the houses. The features are:

1. OverallQual: This feature rates the overall material and finish of the house. The higher the number the better it is in terms of desirability. For Eg.: 10 is Very Excellent, 9 is Excellent and so on.
2. YearBuilt: YearBuilt is the original construction date. A higher value of this means that the construction is newer. I am assuming that a newer construction is more desirable than an older construction.
3. BedroomAbvGr: This feature stands for Bedroom above Grade. In real estate, above grade means the portion of a house that is above the ground. It is quite obvious that a greater number of bedrooms will increase its desirability.
4. GrLivArea: GrLivArea is the above grade (ground) living area in square feet. More the GrLivArea, greater is the Sale Price.
5. FullBath: FullBath is the number of Full bathrooms above grade. FullBath is directly proportional to the desirability of the house.

I have ranked the above features in the following manner: OverallQual>BedroomAbvGr>YearBuilt>GrLivArea>FullBath

With the above ranking, I have used a weighing mechanism to compute the sum of scores in my scoring function.

In [377]:
```
# Extract the features to be used in my scoring function
score_col_with_id = ['Id','OverallQual','YearBuilt','BedroomAbvGr','GrLivArea','FullBath']
score_col = ['OverallQual','YearBuilt','BedroomAbvGr','GrLivArea','FullBath']
desirability_index = pd.read_csv(r'I:\Data Science Fundamentals\house-prices-advanced-regression-techniques\train.csv', usecols=score_col_with_id)
normalized_desirability_index = pd.read_csv(r'I:\Data Science Fundamentals\house-prices-advanced-regression-techniques\train.csv', usecols=score_col_with
_id)
```

/

In [378]:
```
# Drop NaN values to clean the data
desirability_index = desirability_index.dropna(axis=0, how='any')
normalized_desirability_index = normalized_desirability_index.dropna(axis=0, how='any')
```

In [379]:
```
# Next we normalize our data
# https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
scale = MinMaxScaler()
normalized_desirability_index[score_col] = scale.fit_transform(normalized_desirability_index[score_col])
```

In [380]:
```
# Next, I will create my scoring function by giving weights to the individual features I had selected above. My scoring function
# works as follows: Σ(Weight of feature * Value of Feature)
feature_weight = pd.DataFrame(pd.Series([0.6, 0.3, 0.5, 0.2, 0.1],index=score_col, name='feature_weight'))
```

In [381]:
```
feature_weight
```

Out[381]:

	feature_weight
OverallQual	0.6
YearBuilt	0.3
BedroomAbvGr	0.5
GrLivArea	0.2
FullBath	0.1

In [382]:
```
# Add the score index with the corresponding scores generated by my scoring function
normalized_desirability_index['score'] = normalized_desirability_index[score_col].dot(feature_weight)
```

In [383]:
```
# Now I will replace my normalized desirability index with my actual desirability index
desirability_index = pd.merge(normalized_desirability_index[['Id','score']], desirability_index, on='Id', how='left')
```

In [384]:
```
# Once my desirability index is prepared, I am first sorting the values by their scores
desirability_index = desirability_index.sort_values(['score'], ascending=False)
```

In [385]:
```
# Now, I am getting the Top 10 desirable houses
desirability_index.head(10)
```

Out[385]:

	Id	score	OverallQual	YearBuilt	GrLivArea	FullBath	BedroomAbvGr
1182	1183	1.375632	10	1996	4476	3	4
691	692	1.365255	10	1994	4316	3	4
1298	1299	1.349819	10	2008	5642	2	3
523	524	1.344580	10	2007	4676	3	3
1169	1170	1.341468	10	1995	3627	3	4
798	799	1.284713	9	2008	3140	3	4
58	59	1.277184	10	2006	2945	3	3
803	804	1.272731	9	2008	2822	3	4
1046	1047	1.267942	9	2005	2868	3	4
320	321	1.259868	9	2006	2596	3	4

Analysis of Top 10 Desirable Houses: We can see that the overall quality index does play a huge role in the desirability of a house. The better the quality, the more the customers want to buy such a house. Also, most customers want more number of bedrooms above grade, which in a way leads to a greater number of Full Bathrooms in most cases, hence these two factors also influence heavily on the desirability of a house. Finally, we see that the most desirable houses are built after 1990.

In [386]:
```
# Now, I am getting the Least 10 desirable houses
desirability_index.tail(10)
```

Out[386]:

	Id	score	OverallQual	YearBuilt	GrLivArea	FullBath	BedroomAbvGr
29	30	0.422407	4	1927	520	1	1
106	107	0.413459	4	1885	1047	1	2
620	621	0.402941	3	1914	864	1	2
1380	1381	0.402941	3	1914	864	1	2
968	969	0.398164	3	1910	968	1	2
636	637	0.319189	2	1936	800	1	1
916	917	0.302059	2	1949	480	0	1
1100	1101	0.270766	2	1920	438	1	1
533	534	0.256703	1	1946	334	1	1
375	376	0.192673	1	1922	904	0	1

Analysis of Least 10 Desirable Houses: We can see that the least 10 desirable houses paint the exact opposite picture than what the top 10 desirable houses potrayed. The houses in this list has an Overall Quality Index less than 5. Also, the number of bedrooms above grade is less than 3. We also see that the number of full bathrooms is very less or in some cases there is no full bathroom at all. Finally, we see that the least desirable houses are built before 1950. This makes it clear that old houses have the least desirability.

Part 4 - Pairwise Distance Function

Task 4

https://en.wikipedia.org/wiki/Euclidean_distance (https://en.wikipedia.org/wiki/Euclidean_distance)

https://en.wikipedia.org/wiki/Taxicab_geometry (https://en.wikipedia.org/wiki/Taxicab_geometry)

https://en.wikipedia.org/wiki/Minkowski_distance (https://en.wikipedia.org/wiki/Minkowski_distance)

In this task, let's explore pairwise distance between data rows. While choosing distance function I explored the various distance measures available, which are as follows:

1. Euclidean Distance
2. Manhattan Distance
3. Minkowski Distance

I found Euclidean Distance to be the most convenient for this dataset as it has some advantages over other metrics: a. **Manhattan Distance** This distance metric calculates the distance between two points by taking absolute difference between the data points. But as the features in this dataset has sparse values, taking absolute difference in this case could lead to large and unintelligible results.

b. **Minkowski Distance** This distance metric calculates the distance between two points by combining the properties of both Manhattan and Euclidean Distance. However, this also leads to requires choosing some external parameters. The results of the function depends on other external factors, which can can result in some complex computations.

Euclidean Distance

The Euclidean distance between points p and q is the length of the line segment connecting them. The standard Euclidean distance can be squared in order to place progressively greater weight on objects that are farther apart. Squared Euclidean Distance is not a metric as it does not satisfy the triangle inequality, however, it is frequently used in optimization problems in which distances only have to be compared.

Advantages of Euclidean Distance:

a. When data is dense or continuous, this is the best proximity measure. b. Euclidean distance can be extended to any dimensions of the data. c. Euclidean distance is an implicit distance measure in many clustering implementation.

Pairwise distance on Ames Housing Dataset using Euclidean Distance Metric
Features to be used as input for my distance function I inspected several features in the dataset to select the most relevant features that can be used as inputs to my distance function. My aim was to select features which have a highest influence on the similarity between two properties. The features are as follows:

- **OverallQual**
- **YearBuilt**
- **BedroomAbvGr**
- **GrLivArea**
- **FullBath**
- **1stFlrSF**

Applying Euclidean Distance Function

I used scipy library's pdist method to apply the euclidean distance function on this dataset, whereby I passed 'euclidean' as the distance metric. This brought about a 1460 *1460 distance matrix, the observations of which are noted below. I also plotted a 20* 20 heatmap to better visualize the data.

Evaluation

- **I observed that the inputs I selected brought about a very sparse matrix of distances between the points. While the white region represents a distance of 0 or close to 0 , darker purple regions depicts a large distance.**
- **The diagonal of the matrix represents the same points. Hence the distance is 0 and the color is white. This verifies that the distance function performs accurately.**

In [387]:

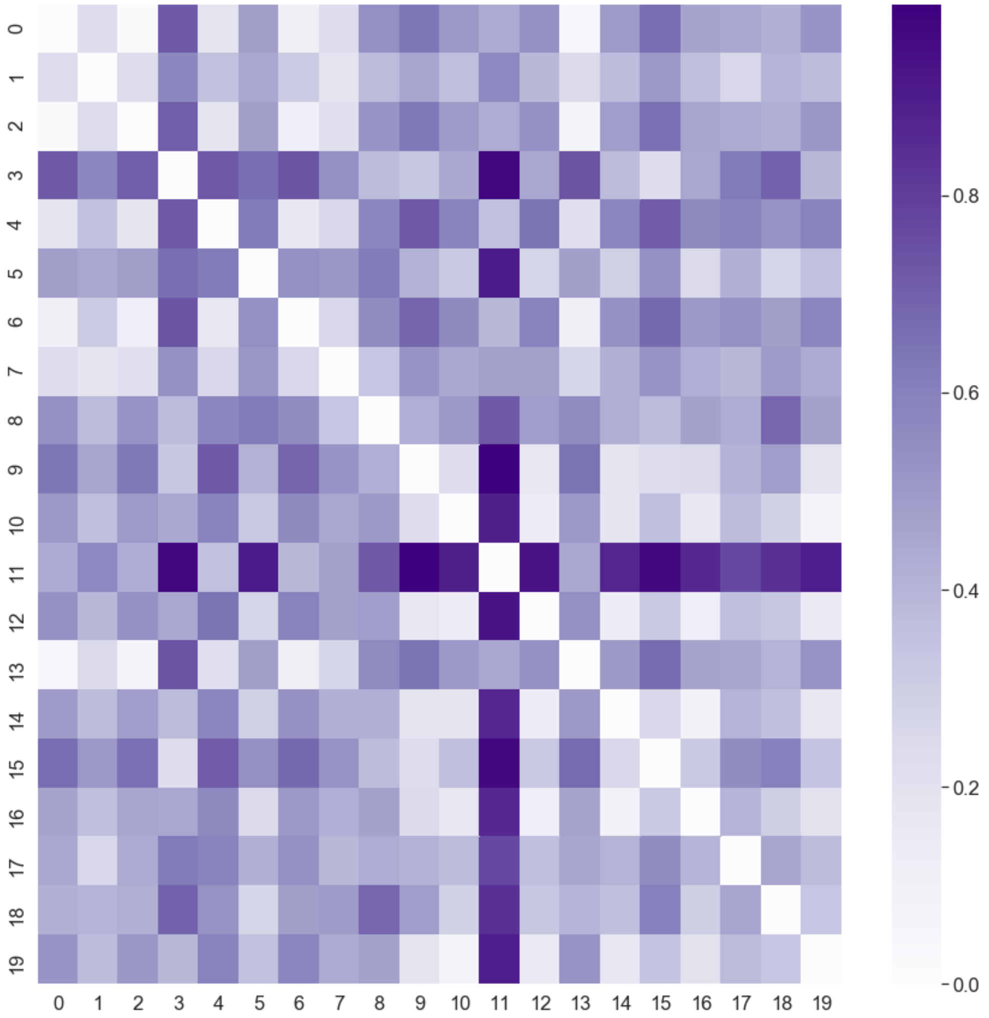
```
# Task 4
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.pdist.html
from scipy.spatial.distance import pdist, squareform
distance_df=data_train[score_col]
distance_df = (distance_df - distance_df.min()) / (distance_df.max() - distance_df.min())
distances = pdist(distance_df.values, metric='euclidean')
distances.shape
distance_matrix = pd.DataFrame(squareform(distances))
distance_matrix
```

Out[387]:

	0	1	2	3	4	5	6	7	8	9	...	1450	1451	1452	1453	1454	1455	1456	1457
0	0.000000	0.240310	0.020373	0.719549	0.192082	0.482227	0.111388	0.228876	0.536640	0.636727	...	0.330768	0.119486	0.436779	0.415329	0.155449	0.115441	0.223252	0.481207
1	0.240310	0.000000	0.234328	0.571134	0.355455	0.448873	0.311728	0.192747	0.378956	0.460633	...	0.195322	0.326645	0.429563	0.413816	0.263055	0.181765	0.153474	0.365432
2	0.020373	0.234328	0.000000	0.706855	0.184521	0.482426	0.114537	0.210827	0.522426	0.629133	...	0.321386	0.128274	0.441648	0.420260	0.165613	0.115072	0.207478	0.464278
3	0.719549	0.571134	0.706855	0.000000	0.725726	0.660171	0.734443	0.541011	0.374558	0.331348	...	0.599254	0.760461	0.710707	0.704297	0.742563	0.702952	0.579972	0.419534
4	0.192082	0.355455	0.184521	0.725726	0.000000	0.624681	0.159627	0.258195	0.575523	0.724371	...	0.390459	0.180635	0.575357	0.528646	0.331005	0.275383	0.301624	0.442548
5	0.482227	0.448873	0.482426	0.660171	0.624681	0.000000	0.543128	0.512648	0.619671	0.414279	...	0.526550	0.546070	0.161776	0.270413	0.427993	0.436727	0.464450	0.690691
6	0.111388	0.311728	0.114537	0.734443	0.159627	0.543128	0.000000	0.261483	0.554999	0.687905	...	0.417536	0.036301	0.501629	0.483038	0.189503	0.225331	0.299963	0.501193
7	0.228876	0.192747	0.210827	0.541011	0.258195	0.512648	0.261483	0.000000	0.334360	0.522725	...	0.261174	0.293214	0.516402	0.499709	0.304778	0.234111	0.116913	0.267607
8	0.536640	0.378956	0.522426	0.374558	0.575523	0.619671	0.554999	0.334360	0.000000	0.425555	...	0.457149	0.583666	0.682297	0.696960	0.539147	0.520912	0.383586	0.281285
9	0.636727	0.460633	0.629133	0.331348	0.724371	0.414279	0.687905	0.522725	0.425555	0.000000	...	0.506044	0.704809	0.478262	0.501481	0.618938	0.582794	0.504142	0.528981
10	0.502246	0.362712	0.498296	0.443762	0.591410	0.327640	0.563268	0.450539	0.505907	0.226209	...	0.388665	0.574096	0.315717	0.297698	0.507099	0.444112	0.412559	0.516088
11	0.435556	0.567484	0.432703	0.973581	0.354024	0.907198	0.391435	0.481490	0.721994	0.994183	...	0.607577	0.399132	0.871843	0.841076	0.515972	0.505969	0.527601	0.612849
12	0.535714	0.392073	0.532070	0.451666	0.647491	0.270692	0.593447	0.481378	0.487164	0.169541	...	0.456791	0.603905	0.313049	0.345151	0.506468	0.479731	0.447622	0.564382
13	0.046136	0.248022	0.065871	0.740075	0.217834	0.482168	0.118218	0.264180	0.560157	0.646535	...	0.349185	0.113164	0.427191	0.406130	0.135942	0.125497	0.255758	0.512720
14	0.493544	0.374408	0.487272	0.376756	0.581984	0.292533	0.533555	0.415717	0.421013	0.191316	...	0.454489	0.548499	0.346181	0.373544	0.474502	0.460558	0.409408	0.496840
15	0.663542	0.510866	0.655574	0.228803	0.717476	0.537818	0.677860	0.531618	0.375982	0.237484	...	0.600462	0.696707	0.595287	0.615827	0.641296	0.647070	0.563825	0.509458
16	0.462554	0.361924	0.459618	0.452611	0.566438	0.245555	0.503705	0.425928	0.473675	0.250992	...	0.456986	0.513456	0.277190	0.310934	0.431080	0.430780	0.413106	0.541727
17	0.448188	0.263253	0.442673	0.621390	0.588295	0.418011	0.539133	0.388590	0.432751	0.407832	...	0.293514	0.551586	0.448399	0.468842	0.428015	0.350928	0.304613	0.497788
18	0.416118	0.406696	0.420707	0.691534	0.529522	0.266527	0.483903	0.494738	0.686589	0.487369	...	0.436246	0.480316	0.125460	0.015298	0.420149	0.367223	0.437716	0.661724
19	0.521260	0.375074	0.514467	0.389288	0.597214	0.356151	0.581211	0.438550	0.470228	0.192399	...	0.384008	0.596259	0.366265	0.349841	0.536399	0.463781	0.404454	0.476296
20	0.393745	0.514126	0.390227	0.955594	0.336969	0.860369	0.378546	0.442442	0.697398	0.952197	...	0.533657	0.387054	0.823655	0.790857	0.483071	0.443702	0.466537	0.581879
21	0.635452	0.485190	0.626202	0.158045	0.662231	0.567965	0.650530	0.492370	0.377534	0.263240	...	0.542571	0.671392	0.600352	0.593900	0.643998	0.619490	0.526775	0.432390
22	0.112493	0.308160	0.111360	0.721887	0.146967	0.543719	0.023919	0.244122	0.541008	0.681584	...	0.409761	0.059680	0.506826	0.488149	0.199686	0.225017	0.287004	0.483635
23	0.462352	0.353419	0.460456	0.509993	0.560408	0.284451	0.526931	0.445669	0.548220	0.295840	...	0.382052	0.534337	0.244522	0.217913	0.467126	0.404307	0.400110	0.546438
24	0.489708	0.358142	0.486194	0.460656	0.581018	0.313936	0.551853	0.446610	0.515848	0.244533	...	0.384245	0.561927	0.295831	0.275774	0.495067	0.431449	0.406361	0.522063
25	0.116685	0.322335	0.124354	0.753915	0.175756	0.545000	0.028039	0.285603	0.576492	0.700258	...	0.430381	0.008348	0.497948	0.479359	0.183143	0.229830	0.318566	0.525490
26	0.570760	0.401157	0.565359	0.375668	0.650939	0.403365	0.626178	0.485973	0.473534	0.155879	...	0.427497	0.639643	0.412061	0.401107	0.572081	0.514048	0.458881	0.504943
27	0.114835	0.326770	0.120311	0.753596	0.163889	0.546960	0.021821	0.279885	0.575710	0.703283	...	0.429179	0.024819	0.502229	0.483285	0.191633	0.229910	0.313650	0.520635
28	0.536339	0.402607	0.528210	0.397645	0.628097	0.292726	0.595110	0.445065	0.443920	0.163467	...	0.435999	0.611917	0.361770	0.386279	0.530647	0.481443	0.412528	0.505838
29	0.798925	0.607134	0.793192	0.481692	0.910510	0.515987	0.855215	0.695212	0.542679	0.215740	...	0.661440	0.869131	0.598545	0.645139	0.752742	0.735041	0.667213	0.700503
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1430	0.256514	0.289667	0.256795	0.775867	0.342087	0.517049	0.357106	0.347896	0.632118	0.650307	...	0.224805	0.360011	0.440950	0.379580	0.354185	0.176510	0.261171	0.522884
1431	0.293806	0.137496	0.291758	0.595750	0.443511	0.400063	0.354121	0.271890	0.377242	0.442544	...	0.315821	0.363897	0.409974	0.433032	0.236577	0.245463	0.244868	0.454927
1432	0.670503	0.440627	0.661838	0.515092	0.728734	0.705921	0.737018	0.531534	0.444419	0.440386	...	0.390434	0.755566	0.711204	0.683993	0.698008	0.594628	0.494898	0.433836
1433	0.114267	0.200538	0.111353	0.709254	0.266192	0.441693	0.224864	0.231899	0.527240	0.593815	...	0.251927	0.233171	0.398490	0.374747	0.200955	0.028262	0.167980	0.470549
1434	0.311423	0.114257	0.308365	0.612159	0.441929	0.434776	0.401051	0.288455	0.437061	0.450162	...	0.178439	0.410883	0.409887	0.394055	0.321882	0.217703	0.210305	0.430546
1435	0.461290	0.352260	0.453349	0.359847	0.517156	0.355520	0.503980	0.375053	0.437650	0.251513	...	0.385942	0.521215	0.364330	0.345831	0.485029	0.428283	0.367084	0.430261
1436	0.548995	0.409181	0.547406	0.549188	0.656954	0.330247	0.624721	0.525148	0.592667	0.288706	...	0.412456	0.631365	0.300346	0.281734	0.547315	0.472675	0.463622	0.602021
1437	0.176161	0.367029	0.176919	0.771285	0.261480	0.511071	0.135925	0.305256	0.569705	0.705810	...	0.484777	0.141678	0.498944	0.510216	0.176432	0.268596	0.336114	0.562558
1438	0.508102	0.381728	0.501475	0.358901	0.594080	0.310785	0.547319	0.422386	0.411372	0.173943	...	0.460548	0.562893	0.366447	0.392905	0.489346	0.474720	0.418041	0.492604
1439	0.173936	0.142922	0.159834	0.571134	0.243040	0.487814	0.212599	0.081356	0.369720	0.524441	...	0.257837	0.239319	0.476183	0.459260	0.239707	0.183209	0.129215	0.323842
1440	0.618228	0.460931	0.600874	0.388528	0.623700	0.707937	0.654806	0.395725	0.232106	0.481446	...	0.436601	0.686750	0.760843	0.751686	0.666490	0.583604	0.415833	0.220380
1441	0.460849	0.469960	0.466538	0.719430	0.605512	0.167560	0.498394	0.539600	0.670329	0.501684	...	0.585636	0.492695	0.172639	0.279431	0.379499	0.444502	0.512220	0.742632
1442	0.339933	0.520577	0.339732	0.824237	0.263938	0.713326	0.231733	0.419142	0.663321	0.846213	...	0.621779	0.236463	0.683286	0.668313	0.386657	0.454295	0.494919	0.605301
1443	0.746272	0.564963	0.736989	0.220894	0.805361	0.587596	0.776078	0.596365	0.399066	0.201688	...	0.622611	0.796537	0.654819	0.674208	0.729837	0.711070	0.610889	0.535867
1444	0.054739	0.233285	0.071939	0.728101	0.224017	0.479036	0.122358	0.257487	0.547584	0.634191	...	0.342236	0.118531	0.424875	0.404384	0.130610	0.124319	0.250767	0.503932
1445	0.466785	0.347398	0.463499	0.414569	0.543532	0.346815	0.507905	0.418567	0.479289	0.259239	...	0.412147	0.518476	0.329751	0.313354	0.467197	0.432916	0.407960	0.494141
1446	0.508364	0.365982	0.503045	0.418700	0.591505	0.337693	0.569155	0.442408	0.488640	0.209380	...	0.383726	0.582007	0.336442	0.318969	0.518445	0.450356	0.405830	0.496365
1447	0.144331	0.304421	0.132348	0.681512	0.131726	0.551129	0.099091	0.194321	0.496584	0.662525	...	0.391210	0.134827	0.529037	0.510498	0.242954	0.239141	0.254103	0.428145
1448	0.834691	0.636477	0.824350	0.364601	0.906773	0.624276	0.890249	0.682573	0.501877	0.243053	...	0.638825	0.911308	0.699223	0.716525	0.828707	0.771535	0.661577	0.608484
1449	0.567077	0.449471	0.566468	0.559162	0.705244	0.216323	0.620971	0.546921	0.549943	0.270516	...	0.548179	0.626454	0.294760	0.373878	0.499217	0.516553	0.513052	0.670122
1450	0.330768	0.195322	0.321386	0.599254	0.390459	0.526550	0.417536	0.261174	0.457149	0.506044	...	0.000000	0.434814	0.492415	0.442261	0.413175	0.248064	0.177801	0.342381
1451	0.119486	0.326645	0.128274	0.760461	0.180635	0.546070	0.036301	0.293214	0.583666	0.704809	...	0.434814	0.000000	0.497400	0.478792	0.182577	0.231959	0.324559	0.533195
1452	0.436779	0.429563	0.441648	0.710707	0.575357	0.161776	0.501629	0.516402	0.682297	0.478262	...	0.492415	0.497400	0.000000	0.125864	0.401664	0.390777	0.461438	0.703662
1453	0.415329	0.413816	0.420260	0.704297	0.528646	0.270413	0.483038	0.499709	0.696960	0.501481	...	0.442261	0.478792	0.125864	0.000000	0.420192	0.367632	0.442177	0.670136
1454	0.155449	0.263055	0.165613	0.742563	0.331005	0.427993	0.189503	0.304778	0.539147	0.618938	...	0.413175	0.182577	0.401664	0.420192	0.000000	0.189009	0.298717	0.561564
1455	0.115441	0.181765	0.115072	0.702952	0.275383	0.436727	0.225331	0.234111	0.520912	0.582794	...	0.248064	0.231959	0.390777	0.367632	0.189009	0.000000	0.172041	0.470807
1456	0.223252	0.153474	0.207478	0.579972	0.301624	0.464450	0.299963	0.116913	0.383586	0.504142	...	0.177801	0.324559	0.461438	0.442177	0.298717	0.172041	0.000000	0.319980
1457	0.481207	0.365432	0.464278	0.419534	0.442548	0.690691	0.501193	0.267607	0.281285	0.528981	...	0.342381	0.533195	0.703662	0.670136	0.561564	0.470807	0.319980	0.000000
1458	0.581202	0.419261	0.574881	0.379242	0.678616	0.339969	0.635951	0.490149	0.443445	0.079710	...	0.471117	0.650664	0.398552	0.424774	0.560659	0.525975	0.464099	0.532705
1459	0.493593	0.360294	0.488382	0.433821	0.577639	0.322594	0.555787	0.434214	0.496330	0.228602	...	0.375748	0.568325	0.317557	0.297904	0.505994	0.435413	0.394996	0.498067

1460 rows × 1460 columns

In [388]:
```
# Representing the above with a Heatmap
heatmap_distance_matrix = distance_matrix.iloc[:20, :20]
plt.figure(figsize=(15,15))
sns.heatmap(heatmap_distance_matrix, square=False, cmap='Purples')
plt.show()
```



Part 5 - Clustering

/

Task 5

Clustering

In this task, I will cluster the data. By this, I hope to bring out the underlying structural similarities within the data.

https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1 (https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1)

https://en.wikipedia.org/wiki/K-means_clustering (https://en.wikipedia.org/wiki/K-means_clustering)

K Means Clustering Algorithm

K-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.his results in a partitioning of the data space into Voronoi cells. k-Means minimizes within-cluster variances (squared Euclidean distances), but not regular Euclidean distances, which would be the more difficult Weber problem: the mean optimizes squared errors, whereas only the geometric median minimizes Euclidean distances. Better Euclidean solutions can for example be found using k-medians and k-medoids.

The most common algorithm uses an iterative refinement technique. Due to its ubiquity it is often called the k-means algorithm; it is also referred to as Lloyd's algorithm, particularly in the computer science community. Given an initial set of k means the algorithm proceeds by alternating between two steps:

1. **Assignment Step : Assign each observation to the cluster whose euclidean distance with the mean of cluster is minimum.**
2. **Update Step : After assigning each points to the cluster, calculate the new means for each cluster.**

   Initialization methods: There are couple of methods to initialize the clustering algorithm: a. Forgy Partition - Randomly choose k observations and choose them as means for each cluster. b. Random Partition - Randomly assign each obervation a cluster and then update means of clusters.

Applying K-Means on this Dataset

1. **Before we proceed, the first step is to reduce the dimensions of the data to visualize it better using Principal Component Analysis (PCA).**

https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60 (https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60)

https://en.wikipedia.org/wiki/Principal_component_analysis (https://en.wikipedia.org/wiki/Principal_component_analysis)

Principle Component Analysis (PCA)

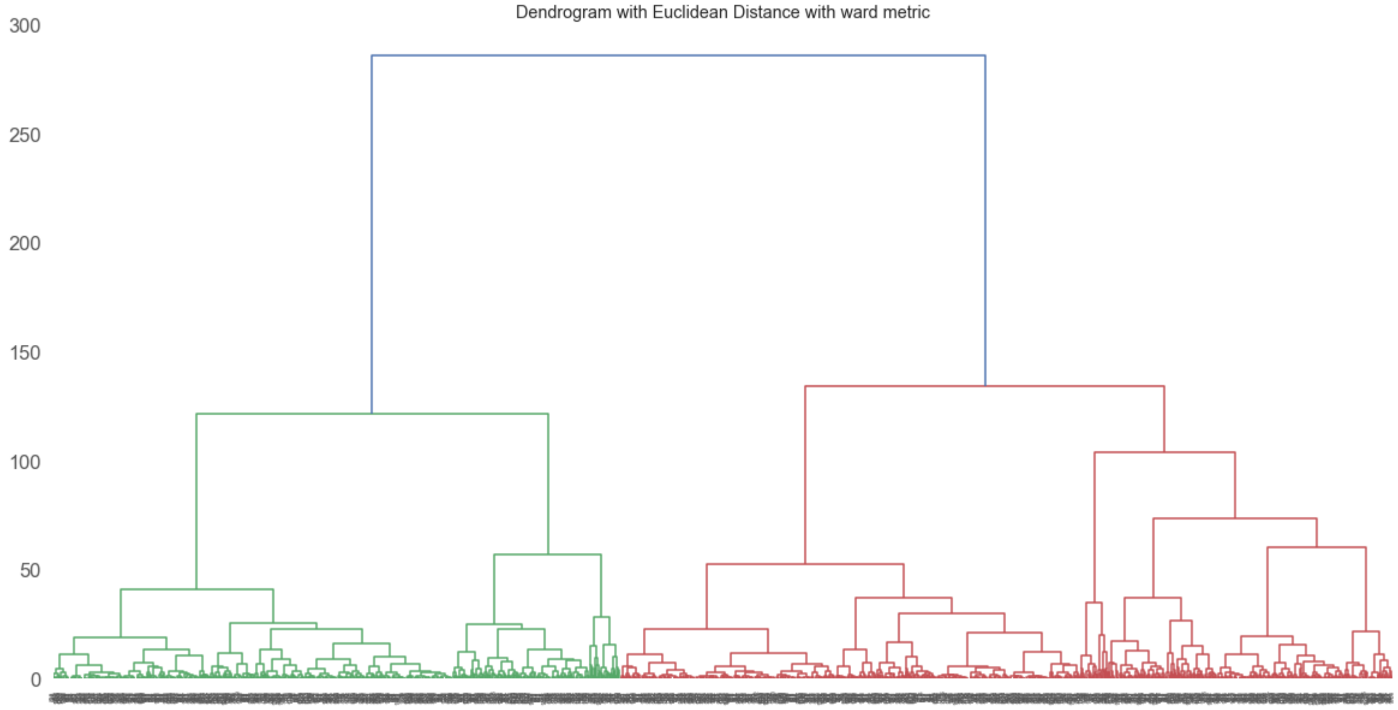
Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components.

A very high-level description of PCA is that it serves as a dimensionality reduction method on the features of our original dataset by projecting these features onto a lower dimension.

1. **The next step is to cluster the reduced data using K-means to 10 classes and assigned labels from 0 to 9 to each of our property data points.**
2. **The final step is to visualize the clustering. For this, I am going to use a dendogram and a scatter plot. Cluster data points are represented in plot with different colors.**

In [443]:
```
# Task 5
PCA_reduced_data = PCA(n_components=2).fit_transform(distance_df)
kmeanscluster = KMeans(n_clusters=10, random_state=0).fit(PCA_reduced_data)
cluster_labels = kmeanscluster.predict(PCA_reduced_data)
distance_df.loc[:, 'cluster label'] = cluster_labels
```

In [444]:
```
plt.figure(figsize=(20,10))
linkage_matrix_ward = linkage(distance_matrix, "ward")
dendrogram(linkage_matrix_ward)
plt.title("Dendrogram with Euclidean Distance with Ward metric")
plt.show()
```



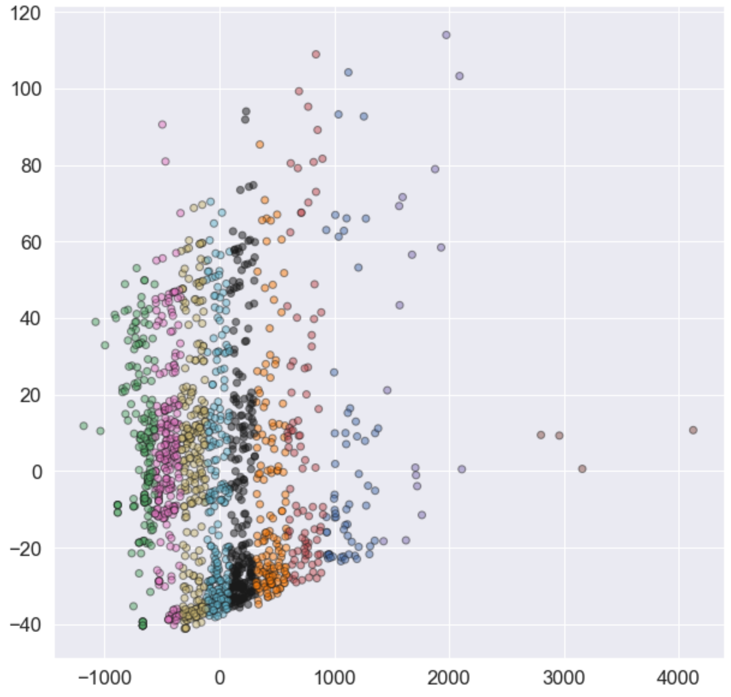
In [304]: distance_func_data

Out[304]:

	OverallQual	YearBuilt	BedroomAbvGr	GrLivArea	FullBath	cluster label
0	7	2003	3	1710	2	6
1	6	1976	3	1262	2	5
2	7	2001	3	1786	2	6
3	7	1915	3	1717	1	6
4	8	2000	4	2198	2	3
5	5	1993	1	1362	1	5
6	8	2004	3	1694	2	6
7	7	1973	3	2090	2	7
8	7	1931	2	1774	2	6
9	5	1939	2	1077	1	9
10	5	1965	3	1040	1	9
11	9	2005	4	2324	3	3
12	5	1962	2	912	1	2
13	7	2006	3	1494	2	0
14	6	1960	2	1253	1	5
15	7	1929	2	854	1	2
16	6	1970	2	1004	1	9
17	4	1967	2	1296	2	5
18	5	2004	3	1114	1	9
19	5	1958	3	1339	1	5
20	8	2005	4	2376	3	3
21	7	1930	3	1108	1	9
22	8	2002	3	1795	2	6
23	5	1976	3	1060	1	9
24	5	1968	3	1060	1	9
25	8	2007	3	1600	2	0
26	5	1951	3	900	1	2
27	8	2007	3	1704	2	6
28	5	1957	2	1600	1	0
29	4	1927	1	520	1	2
...	...	...	...	...	...	...
1430	5	2005	4	1838	2	7
1431	6	1976	2	958	2	9
1432	4	1927	4	968	2	9
1433	6	2000	3	1792	2	6
1434	5	1977	3	1126	2	9
1435	6	1962	3	1537	1	0
1436	4	1971	3	864	1	2
1437	8	2008	2	1932	2	7
1438	6	1957	2	1236	1	5
1439	7	1979	3	1725	2	6
1440	6	1922	3	2555	2	1
1441	6	2004	1	848	1	2
1442	10	2008	3	2007	2	7
1443	6	1916	2	952	1	2
1444	7	2004	3	1422	2	0
1445	6	1966	3	913	1	2
1446	5	1962	3	1188	1	5
1447	8	1995	3	2090	2	7
1448	4	1910	2	1346	1	5
1449	5	1970	1	630	1	2
1450	5	1974	4	1792	2	6
1451	8	2008	3	1578	2	0
1452	5	2005	2	1072	1	9
1453	5	2006	3	1140	1	9
1454	7	2004	2	1221	2	5
1455	6	1999	3	1647	2	6
1456	6	1978	3	2073	2	7
1457	7	1941	4	2340	2	3
1458	5	1950	2	1078	1	9
1459	5	1965	3	1256	1	5

1460 rows × 6 columns

In [305]:
```
figure = plt.figure(figsize=(10, 10))
colorkey = {1: 'c', 2: 'b', 3: 'g', 4: 'r', 5: 'm', 6: 'y',7:'k',8:'tab:orange',9:'tab:brown',10:'tab:pink'}

# colors = map(lambda x: colmap[x+1], labels)
colors = [colorkey[x+1] for x in cluster_labels]
plt.scatter(PCA_reduced_data[:,0], PCA_reduced_data[:,1], c=colors, alpha=0.5, edgecolor='k')
#for idx, centroid in enumerate(centroids):
    #plt.scatter(*centroid, c='w', marker='x',zorder=10, s=170)
plt.show()
```



Analysis: I have used two types of visualization schemes to potray the clustering of the houses. First I used a dendrogram. A dendrogram is a diagram that shows the hierarchical relationship between objects. It is most commonly created as an output from hierarchical clustering. The main use of a dendrogram is to work out the best way to allocate objects to clusters. Structurally, in a dendrogram, the clade is the branch. Each clade has one or more leaves.In simple terms, the clades are arranged according to how similar (or dissimilar) they are. Clades that are close to the same height are similar to each other; clades with different heights are dissimilar — the greater the difference in height, the more dissimilarity.

Secondly, I used a scatter plot. We can see from the scatterplot that the houses with similar neighbourhoods are clustered together and share the same color scheme in the plot.

Preprocessing Techniques:

In [389]:
```
# Before I move on to building my first prediction model, I would start off by preprocessing the data.
# First, I take a look at the data carefully, specifically its types.
train_test_combi=pd.concat([data_train,data_test], sort=False)
train_test_combi.select_dtypes(include='object').head()
```

Out[389]:

	MSZoning	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	...	GarageType	GarageFinish	GarageQual	GarageCond	PavedDrive	PoolQC
0	RL	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	...	Attchd	RFn	TA	TA	Y	NaN
1	RL	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	Veenker	Feedr	...	Attchd	RFn	TA	TA	Y	NaN
2	RL	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	...	Attchd	RFn	TA	TA	Y	NaN
3	RL	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	Norm	...	Detchd	Unf	TA	TA	Y	NaN
4	RL	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	Norm	...	Attchd	RFn	TA	TA	Y	NaN

5 rows × 43 columns

In [390]:
```
train_test_combi.select_dtypes(include=['float','int']).head()
```

Out[390]:

	LotFrontage	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	BsmtFullBath	BsmtHalfBath	GarageYrBlt	GarageCars	GarageArea	SalePrice
0	65.0	196.0	706.0	0.0	150.0	856.0	1.0	0.0	2003.0	2.0	548.0	208500.0
1	80.0	0.0	978.0	0.0	284.0	1262.0	0.0	1.0	1976.0	2.0	460.0	181500.0
2	68.0	162.0	486.0	0.0	434.0	920.0	1.0	0.0	2001.0	2.0	608.0	223500.0
3	60.0	0.0	216.0	0.0	540.0	756.0	1.0	0.0	1998.0	3.0	642.0	140000.0
4	84.0	350.0	655.0	0.0	490.0	1145.0	1.0	0.0	2000.0	3.0	836.0	250000.0

In [391]:
```
# Next I will check for null values in categorical columns
train_test_combi.select_dtypes(include='object').isnull().sum()[train_test_combi.select_dtypes(include='object').isnull().sum()>0]
```

Out[391]:
```
MSZoning           4
Alley           2721
Utilities          2
Exterior1st        1
Exterior2nd        1
MasVnrType        24
BsmtQual          81
BsmtCond          82
BsmtExposure      82
BsmtFinType1      79
BsmtFinType2      80
Electrical         1
KitchenQual        1
Functional         2
FireplaceQu     1420
GarageType       157
GarageFinish     159
GarageQual       159
GarageCond       159
PoolQC          2909
Fence           2348
MiscFeature     2814
SaleType           1
dtype: int64
```

In [392]:
```
# Next I will fill such Null values with the string 'None', for both train and test datasets
for columns in ('Alley','Utilities','MasVnrType','BsmtQual','BsmtCond','BsmtExposure','BsmtFinType1',
            'BsmtFinType2','Electrical','FireplaceQu','GarageType','GarageFinish','GarageQual','GarageCond',
           'PoolQC','Fence','MiscFeature'):
    data_train[columns]=data_train[columns].fillna('None')
    data_test[columns]=data_test[columns].fillna('None')
```

In [393]:
```
# Further, for some columns which are not available I will fill them with the 'Mode' Value
for columns in ('MSZoning','Exterior1st','Exterior2nd','KitchenQual','SaleType','Functional'):
    data_train[columns]=data_train[columns].fillna(data_train[columns].mode()[0])
    data_test[columns]=data_test[columns].fillna(data_train[columns].mode()[0])
```

In [394]:
```
# Next I will check for null values in numerical columns
train_test_combi.select_dtypes(include=['int','float']).isnull().sum()[train_test_combi.select_dtypes(include=['int','float']).isnull().sum()>0]
```

Out[394]:
```
LotFrontage      486
MasVnrArea        23
BsmtFinSF1         1
BsmtFinSF2         1
BsmtUnfSF          1
TotalBsmtSF        1
BsmtFullBath       2
BsmtHalfBath       2
GarageYrBlt      159
GarageCars         1
GarageArea         1
SalePrice       1459
dtype: int64
```

In [395]:
```
# For columns with 'None', I will fill them with 0
for columns in ('MasVnrArea','BsmtFinSF1','BsmtFinSF2','BsmtUnfSF','TotalBsmtSF','BsmtFullBath','BsmtHalfBath','GarageYrBlt','GarageCars','GarageArea'):
    data_train[columns]=data_train[columns].fillna(0)
    data_test[columns]=data_test[columns].fillna(0)
```

In [396]:
```
# For the column LotFrontage, I will replace with Mean
data_train['LotFrontage']=data_train['LotFrontage'].fillna(data_train['LotFrontage'].mean())
data_test['LotFrontage']=data_test['LotFrontage'].fillna(data_train['LotFrontage'].mean())
```

In [397]:
```
# Confirmation that the Dataset is clean of null values
print(data_train.isnull().sum().sum())
print(data_test.isnull().sum().sum())
```

```
0
0
```

In [398]:
```
# Now, I will drop some features which I believe will not contribute much as it has a low correlation with Sale Price
data_train.drop(['GarageArea','1stFlrSF','TotRmsAbvGrd','2ndFlrSF'], axis=1, inplace=True)
data_test.drop(['GarageArea','1stFlrSF','TotRmsAbvGrd','2ndFlrSF'], axis=1, inplace=True)
```

In [399]:
```
len_train=data_train.shape[0]
print(data_train.shape)
```

```
(1460, 77)
```

In [400]:
```
# Combining both again after initial cleaning
train_test_combi=pd.concat([data_train,data_test], sort=False)
```

In [401]:
```
# Now I will perform some transformations on the data
train_test_combi['MSSubClass']=train_test_combi['MSSubClass'].astype(str)
```

In [402]:
```
# Performing Skewing
skew=train_test_combi.select_dtypes(include=['int','float']).apply(lambda x: skew(x.dropna())).sort_values(ascending=False)
skew_df=pd.DataFrame({'Skew':skew})
skewed_df=skew_df[(skew_df['Skew']>0.5)|(skew_df['Skew']<-0.5)]
```

In [403]:
```
data_train=train_test_combi[:len_train]
data_test=train_test_combi[len_train:]
```

In [404]:
```
# Perform Box Cox Transformation
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.special.boxcox1p.html
lambd=0.1
for columns in ('MiscVal', 'PoolArea', 'LotArea', 'LowQualFinSF', '3SsnPorch',
        'KitchenAbvGr', 'BsmtFinSF2', 'EnclosedPorch', 'ScreenPorch',
        'BsmtHalfBath', 'MasVnrArea', 'OpenPorchSF', 'WoodDeckSF',
        'LotFrontage', 'GrLivArea', 'BsmtFinSF1', 'BsmtUnfSF', 'Fireplaces',
        'HalfBath', 'TotalBsmtSF', 'BsmtFullBath', 'OverallCond', 'YearBuilt',
        'GarageYrBlt'):
    data_train[columns]=boxcox1p(data_train[columns],lambd)
    data_test[columns]=boxcox1p(data_test[columns],lambd)
```

In [405]:
```
data_train['SalePrice']=np.log(data_train['SalePrice'])
```

In [406]:
```
# Now, I am converting the categorical variables to dummy values. For this, I chose Pandas' get_dummies function
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html
train_test_combi=pd.concat([data_train,data_test], sort=False)
train_test_combi=pd.get_dummies(train_test_combi)
```

In [407]:
```
data_train=train_test_combi[:len_train]
data_test=train_test_combi[len_train:]
```

In [408]:
```
# Final Steps like deleting Id and SalePrice
data_train.drop('Id', axis=1, inplace=True)
data_test.drop('Id', axis=1, inplace=True)
```

In [409]:
```
x_train=data_train.drop('SalePrice', axis=1)
y_train=data_train['SalePrice']
data_test=data_test.drop('SalePrice', axis=1)
```

In [410]:
```
# Finally, before I pass on my dataset for modeling, I will perform fit/scaling with RobustScaler
# https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html
scale=RobustScaler()
x_train=scale.fit_transform(x_train)
data_test=scale.transform(data_test)
```

Part 6 - Linear Regression

In [328]:
```
# Task-6
# Linear Regression
Linear_Regression_Model = LinearRegression()
Linear_Regression_Model.fit(x_train, y_train)
```

Out[328]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [329]:
```
# Running the Linear Regression Model on the Test Data
Linear_Regression_Submission = pd.read_csv(r'I:\Data Science Fundamentals\house-prices-advanced-regression-techniques\sample_submission.csv' ,index_col=
'Id')
Linear_Regression_Submission = Linear_Regression_Model.predict(data_test)
Linear_Regression_Submissions=np.exp(Linear_Regression_Submission)
Linear_Regression_Output=pd.DataFrame({'Id':data_test_copy.Id, 'SalePrice':Linear_Regression_Submissions})
Linear_Regression_Output.to_csv('Linear_Regression.csv', index=False)
```

In [330]:
```
Linear_Regression_Output.head()
```

Out[330]:

	Id	SalePrice
0	1461	116620.986581
1	1462	166223.569469
2	1463	189342.040355
3	1464	201605.945125
4	1465	200755.693885

Analysis: I tested two versions of my Linear Regression Model. First, I chose only the numerical values in the dataset. When I submitted this on Kaggle, I got a score of 0.24091 and had a rank 2430. Then, I tested on the whole dataset (that is, with all columns), after applying preprocessing techniques. However this brought about a worse score of 0.37545. This has led me to believe that a simple model like Linear Regression, works well, when we take into account less variables (preferably, numeric). According to my observation, OverallQual and GrLivArea are the two attributes that play a major role in a better prediction probability.

Part 7 - External Dataset

In [71]:
```python
# Task-7
# External Dataset - Housing Price Index
# https://www.fhfa.gov/DataTools/Downloads/Pages/House-Price-Index-Datasets.aspx#mpo
hpi_data=pd.read_csv(r'I:\Data Science Fundamentals\house-prices-advanced-regression-techniques\Ames_House_Price_Index_Master_Data.csv')
data_train_hpi = pd.merge(data_train, hpi_data, on='YrSold', how='left')
data_train_hpi.head(10)
```

Out[71]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	S
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	NaN	NaN	NaN	0	2	2008	WD	Normal	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	NaN	NaN	NaN	0	5	2007	WD	Normal	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	NaN	NaN	NaN	0	9	2008	WD	Normal	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	NaN	NaN	NaN	0	2	2006	WD	Abnorml	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	NaN	NaN	NaN	0	12	2008	WD	Normal	
5	6	50	RL	85.0	14115	Pave	NaN	IR1	Lvl	AllPub	...	NaN	MnPrv	Shed	700	10	2009	WD	Normal	
6	7	20	RL	75.0	10084	Pave	NaN	Reg	Lvl	AllPub	...	NaN	NaN	NaN	0	8	2007	WD	Normal	
7	8	60	RL	NaN	10382	Pave	NaN	IR1	Lvl	AllPub	...	NaN	NaN	Shed	350	11	2009	WD	Normal	
8	9	50	RM	51.0	6120	Pave	NaN	Reg	Lvl	AllPub	...	NaN	NaN	NaN	0	4	2008	WD	Abnorml	
9	10	190	RL	50.0	7420	Pave	NaN	Reg	Lvl	AllPub	...	NaN	NaN	NaN	0	1	2008	WD	Normal	

10 rows × 82 columns

In [79]:
```python
linear_reg_columns_hpi = ['Id','OverallQual', 'YearBuilt', 'BedroomAbvGr', 'FullBath', 'GarageArea', 'GrLivArea', 'Fireplaces', 'TotRmsAbvGrd', 'TotalBsm
tSF', '1stFlrSF', 'LotArea','HPI']
linear_reg_train_hpi= data_train_hpi[linear_reg_columns_hpi]
data_test =  pd.read_csv(r'I:\Data Science Fundamentals\house-prices-advanced-regression-techniques\test.csv')
data_test_hpi = pd.merge(data_test, hpi_data, on='YrSold', how='left')
data_test_hpi = data_test_hpi[linear_reg_columns_hpi]
linear_reg_train_hpi.head(10)
```

Out[79]:

	Id	OverallQual	YearBuilt	BedroomAbvGr	FullBath	GarageArea	GrLivArea	Fireplaces	TotRmsAbvGrd	TotalBsmtSF	1stFlrSF	LotArea	HPI
0	1	7	2003	3	2	548	1710	0	8	856	856	8450	167.2900
1	2	6	1976	3	2	460	1262	1	6	1262	1262	9600	167.0875
2	3	7	2001	3	2	608	1786	1	6	920	920	11250	167.2900
3	4	7	1915	3	1	642	1717	1	7	756	961	9550	163.1200
4	5	8	2000	4	2	836	2198	1	9	1145	1145	14260	167.2900
5	6	5	1993	1	1	480	1362	0	5	796	796	14115	167.5950
6	7	8	2004	3	2	636	1694	1	7	1686	1694	10084	167.0875
7	8	7	1973	3	2	484	2090	2	7	1107	1107	10382	167.5950
8	9	7	1931	2	2	468	1774	2	8	952	1022	6120	167.2900
9	10	5	1939	2	1	205	1077	2	5	991	1077	7420	167.2900

In [80]:
```
linear_reg_train_hpi = linear_reg_train_hpi.fillna(0)
data_test_hpi_without_na = data_test_hpi.fillna(0)
print(linear_reg_train_hpi.head(10))
print(data_test_hpi_without_na.head(10))
```

```
   Id  OverallQual  YearBuilt  BedroomAbvGr  FullBath  GarageArea  GrLivArea  \
0   1            7       2003             3         2         548       1710
1   2            6       1976             3         2         460       1262
2   3            7       2001             3         2         608       1786
3   4            7       1915             3         1         642       1717
4   5            8       2000             4         2         836       2198
5   6            5       1993             1         1         480       1362
6   7            8       2004             3         2         636       1694
7   8            7       1973             3         2         484       2090
8   9            7       1931             2         2         468       1774
9  10            5       1939             2         1         205       1077

   Fireplaces  TotRmsAbvGrd  TotalBsmtSF  1stFlrSF  LotArea       HPI
0           0             8          856       856     8450  167.2900
1           1             6         1262      1262     9600  167.0875
2           1             6          920       920    11250  167.2900
3           1             7          756       961     9550  163.1200
4           1             9         1145      1145    14260  167.2900
5           0             5          796       796    14115  167.5950
6           1             7         1686      1694    10084  167.0875
7           2             7         1107      1107    10382  167.5950
8           2             8          952      1022     6120  167.2900
9           2             5          991      1077     7420  167.2900
     Id  OverallQual  YearBuilt  BedroomAbvGr  FullBath  GarageArea  \
0  1461            5       1961             2         1       730.0
1  1462            6       1958             3         1       312.0
2  1463            5       1997             3         2       482.0
3  1464            6       1998             3         2       470.0
4  1465            8       1992             2         2       506.0
5  1466            6       1993             3         2       440.0
6  1467            6       1992             3         2       420.0
7  1468            6       1998             3         2       393.0
8  1469            7       1990             2         1       506.0
9  1470            4       1970             2         1       525.0

   GrLivArea  Fireplaces  TotRmsAbvGrd  TotalBsmtSF  1stFlrSF  LotArea  \
0        896           0             5        882.0       896    11622
1       1329           0             6       1329.0      1329    14267
2       1629           1             6        928.0       928    13830
3       1604           1             7        926.0       926     9978
4       1280           0             5       1280.0      1280     5005
5       1655           1             7        763.0       763    10000
6       1187           0             6       1168.0      1187     7980
7       1465           1             7        789.0       789     8402
8       1341           1             5       1300.0      1341    10176
9        882           0             4        882.0       882     8400

        HPI
0  165.7375
1  165.7375
2  165.7375
3  165.7375
4  165.7375
5  165.7375
6  165.7375
7  165.7375
8  165.7375
9  165.7375
```

In [81]:
```
train_y = data_train_hpi['SalePrice']
train_x = linear_reg_train_hpi
print(train_y.shape, train_x.shape)
```

```
(1460,) (1460, 13)
```

In [82]:
```
Linear_Regression_Model = LinearRegression()
Linear_Regression_Model.fit(train_x, train_y)
```

Out[82]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [84]:
```
# Predict probability of the test dataframe and add result to the submission csv
Linear_Regression_Submission_HPI = pd.read_csv(r'I:\Data Science Fundamentals\house-prices-advanced-regression-techniques\sample_submission.csv' ,index_c
ol='Id')
Linear_Regression_Submission_HPI['SalePrice'] = Linear_Regression_Model.predict(data_test_hpi_without_na)
Linear_Regression_Submission_HPI.to_csv('linear_regression_submission_hpi.csv')
Linear_Regression_Submission_HPI.head()
```

Out[84]:

	SalePrice
Id	
1461	129964.198313
1462	159231.649633
1463	169889.227841
1464	190521.797650
1465	215017.745902

Analysis: I chose Housing Price Index (HPI) of Ames,Iowa as my external dataset. The FHFA House Price Index (HPI) is a broad measure of the movement of single-family house prices. The HPI is a weighted, repeat-sales index, meaning that it measures average price changes in repeat sales or refinancings on the same properties. In simpler terms, a house price index (HPI) measures the price changes of residential housing as a percentage change from some specific start date (which has HPI of 100). I first preprocessed the data to include the HPI for only the years 2006-2010, as our dataset is also from the same year span. Then I merged this dataset on the column YrSold (or Year in which the house was sold) in the dataset. Finally, I ran a Linear Regression Model on this new dataset, and my assumption was proved to be true, as the score improved from 0.24091 to 0.23963.
Source: [https://www.fhfa.gov/DataTools/Downloads/Pages/House-Price-Index-Datasets.aspx#mpo (https://www.fhfa.gov/DataTools/Downloads/Pages/House-Price-Index-Datasets.aspx#mpo)](https://www.fhfa.gov/DataTools/Downloads/Pages/House-Price-Index-Datasets.aspx#mpo)

Part 8 - Permutation Test

In [347]:
```
def rmsle(y, y0):
    score = np.sqrt(np.mean(np.square(np.log1p(y) - np.log1p(y0))))
    return score

cols = ['YearBuilt', 'OverallQual', 'MiscVal', 'TotRmsAbvGrd', 'LowQualFinSF', 'GrLivArea', 'Fireplaces', 'GarageArea', '3SsnPorch', 'PoolArea']
y_train = data_train['SalePrice']
Linear_Regression_Model = LinearRegression()
for colm in cols:
    col = [colm]
    linear_model = data_train_copies[col]
    linear_model = linear_model.fillna(0)
    data_test_perm = pd.read_csv(r'I:\Data Science Fundamentals\house-prices-advanced-regression-techniques\test.csv')
    data_test_perm = data_test_perm[col]
    data_test_perm = data_test_perm.fillna(0)
    x_train = linear_model
    Linear_Regression_Model.fit(x_train, y_train)
    sale_price = Linear_Regression_Model.predict(data_test_perm)
    rmsle_score =make_scorer(rmsle,greater_is_better=False)
    score, permutation_scores, pvalue = permutation_test_score(Linear_Regression_Model, x_train, y_train,
                                                       cv=5,n_permutations=100,scoring=rmsle_score)
    print(colm, pvalue)
```

```
YearBuilt 0.009900990099009901
OverallQual 0.009900990099009901
MiscVal 0.7821782178217822
TotRmsAbvGrd 0.009900990099009901
LowQualFinSF 0.32673267326732675
GrLivArea 0.009900990099009901
Fireplaces 0.009900990099009901
GarageArea 0.009900990099009901
3SsnPorch 0.7623762376237624
PoolArea 0.039603960396039604
```

Analysis: I ran single-variable regression model on the following variables: 'YearBuilt', 'OverallQual', 'MiscVal', 'TotRmsAbvGrd', 'LowQualFinSF', 'GrLivArea', 'Fireplaces', 'GarageArea', '3SsnPorch' and 'PoolArea'. Then I performed permutation test to determine the p-value of my predictions. For this, I used the Root-Mean-Squared Error (RMSE) of the log(price) to score the models. Its general consensus that p<0.05 does a good job of predicting. As the above results show, 7 out of 10 variables pass this criteria, in my case.

Part 9 - Models

In [93]:
```
# Task-9
# Model 1, which is Linear Regression, is given in Task-6
# Model 2: Light GBM Regressor
LGB_Model = LGBMRegressor(n_estimators = 400,
max_bin = 8,
learning_rate = 0.037,
num_leaves = 20,
min_data = 10,
min_hessian = 0.05,
verbose = 0,
feature_fraction_seed = 2,
bagging_seed = 3)

LGB_Model.fit(train_x, train_y)
LGB_Submission = pd.read_csv(r'I:\Data Science Fundamentals\house-prices-advanced-regression-techniques\sample_submission.csv' ,index_col='Id')
LGB_Submission['SalePrice'] = LGB_Model.predict(data_test_hpi_without_na)
LGB_Submission.to_csv('lgb_submission.csv')
LGB_Submission.head()
```

Out[93]:

Id	SalePrice
1461	145515.717499
1462	163489.297214
1463	174702.452364
1464	177063.543348
1465	194671.346504

In [94]:
```
# Model 3: Gradient Boosting Regressor Model
GBR_Model = GradientBoostingRegressor(n_estimators=100,
     learning_rate=0.1,
     max_depth=2,
     loss='ls',
     criterion = 'mse')
GBR_Model.fit(train_x, train_y)
GBR_Submission = pd.read_csv(r'I:\Data Science Fundamentals\house-prices-advanced-regression-techniques\sample_submission.csv' ,index_col='Id')
GBR_Submission['SalePrice'] = LGB_Model.predict(data_test_hpi_without_na)
GBR_Submission.to_csv('gbr_submission.csv')
GBR_Submission.head()
```

Out[94]:

Id	SalePrice
1461	145515.717499
1462	163489.297214
1463	174702.452364
1464	177063.543348
1465	194671.346504

In [96]:
```
# Model 4: XGB Regressor
train_X = data_train.loc[:,:'SaleCondition'] #'SaleCondition' is the second last column before 'SalePrice'
train_y = data_train.loc[:,'SalePrice']
data_test = pd.read_csv(r'I:\Data Science Fundamentals\house-prices-advanced-regression-techniques\test.csv')
test_X = data_test.copy()
numeric_cols = train_X.dtypes[train_X.dtypes != 'object'].index
train_X = train_X[numeric_cols]

test_X = test_X[numeric_cols]

train_X = train_X.fillna(train_X.mean())

test_X = test_X.fillna(test_X.mean())

from xgboost import XGBRegressor

XGBmodel = XGBRegressor()

XGBmodel.fit(train_X, train_y, verbose=False)
predictions = XGBmodel.predict(test_X)
XGB_Submission = pd.read_csv(r'I:\Data Science Fundamentals\house-prices-advanced-regression-techniques\sample_submission.csv' ,index_col='Id')
XGB_Submission['SalePrice'] = XGBmodel.predict(test_X)
XGB_Submission.to_csv('XGB_Submission.csv')
XGB_Submission.head()
```

```
[01:39:13] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

Out[96]:

	SalePrice
Id	
1461	125763.687500
1462	160744.171875
1463	177163.156250
1464	183517.875000
1465	187548.187500

In [97]:
```
# Model 5: XGB Classifier
train_X = data_train.loc[:,:'SaleCondition'] #'SaleCondition' is the second last column before 'SalePrice'
train_y = data_train.loc[:,'SalePrice']

#This DataFrame will be used for the predictions
#we will submit.
data_test = pd.read_csv(r'I:\Data Science Fundamentals\house-prices-advanced-regression-techniques\test.csv')
test_X = data_test.copy()
numeric_cols = train_X.dtypes[train_X.dtypes != 'object'].index
train_X = train_X[numeric_cols]

test_X = test_X[numeric_cols]

train_X = train_X.fillna(train_X.mean())

test_X = test_X.fillna(test_X.mean())

XGBClassifiermodel = XGBClassifier()

XGBClassifiermodel.fit(train_X, train_y, verbose=False)
predictions = XGBClassifiermodel.predict(test_X)
XGB_ClassifierSubmission = pd.read_csv(r'I:\Data Science Fundamentals\house-prices-advanced-regression-techniques\sample_submission.csv' ,index_col='Id')
XGB_ClassifierSubmission['SalePrice'] = XGBClassifiermodel.predict(test_X)
XGB_ClassifierSubmission.to_csv('XGB_Classifier_Submission.csv')
XGB_ClassifierSubmission.head()
```

Out[97]:

	SalePrice
Id	
1461	144000
1462	145000
1463	181000
1464	181000
1465	179200

In [335]:
```
# Model 6: Lasso Model
Lasso_Model = Lasso(alpha =0.001, random_state=1)
Lasso_Model.fit(x_train, y_train)
# Predict probability of the test dataframe and add result to the submission csv
Lasso_Model_Submission = pd.read_csv(r'I:\Data Science Fundamentals\house-prices-advanced-regression-techniques\sample_submission.csv' ,index_col='Id')
Lasso_Model_Submission = Lasso_Model.predict(data_test)
Lasso_Model_Submissions=np.exp(Lasso_Model_Submission)
#Linear_Regression_Submission.to_csv('Linear_Regression.csv')
#Linear_Regression_Submission.head()
Lasso_Model_Output=pd.DataFrame({'Id':data_test_copy.Id, 'SalePrice':Lasso_Model_Submissions})
Lasso_Model_Output.to_csv('Lasso.csv', index=False)
```

In [350]:
```
Lasso_Model_Output.head()
```

Out[350]:

	Id	SalePrice
0	1461	117049.317523
1	1462	149353.883545
2	1463	179438.082301
3	1464	197525.928291
4	1465	197874.393480

Final Analysis of Models: I have a made of total of 6 models for this project. The following are the Kaggles scores I received for each of them:

1. **Linear Regression: 0.24091 (without HPI dataset), 0.23963 (with HPI dataset)**
2. **Light GBM: 0.16225**
3. **Gradient Boosting Regressor: 0.16225**
4. **XGB Regressor: 0.14251**
5. **XGB Classifier: 0.24067**
6. **Lasso: 0.12011**

As we can see, the Lasso Model performs exceptionally well and the best amongst all the models, and I got a Kaggle rank of 970 for my efforts.

Part 10 - Final Result

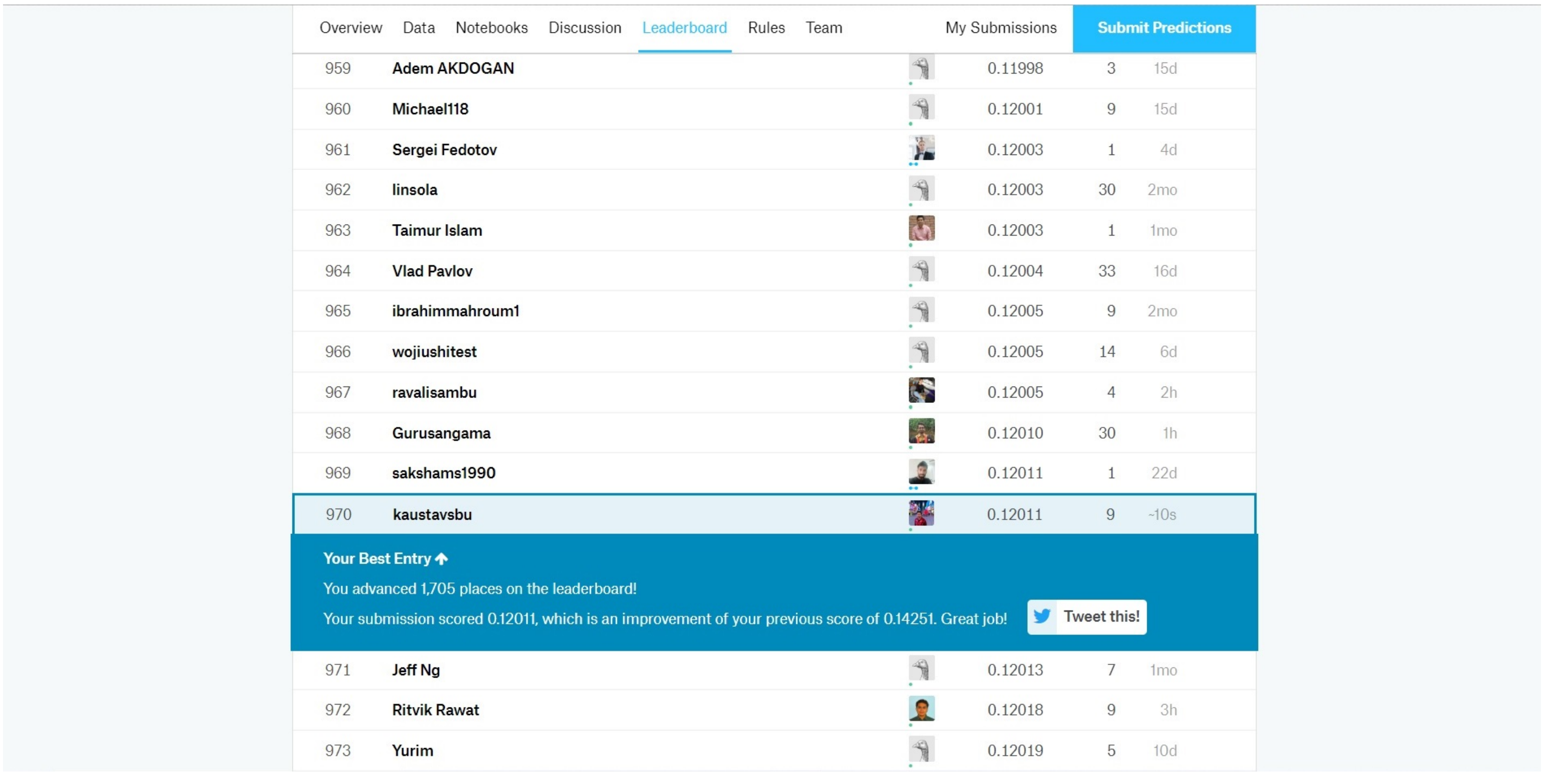
Report the rank, score, number of entries, for your highest rank. Include a snapshot of your best score on the leaderboard as confirmation. Be sure to provide a link to your Kaggle profile. Make sure to include a screenshot of your ranking. Make sure your profile includes your face and affiliation with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

Kaggle Link: [https://www.kaggle.com/kaustavsbu (https://www.kaggle.com/kaustavsbu)](https://www.kaggle.com/kaustavsbu)

Highest Rank: 970

Score: 0.12011

Number of entries: 9



Overview Data Notebooks Discussion Leaderboard Rules Team My Submissions Submit Predictions

959	Adem AKDOGAN	0.11998	3	15d
960	Michael118	0.12001	9	15d
961	Sergei Fedotov	0.12003	1	4d
962	linsola	0.12003	30	2mo
963	Taimur Islam	0.12003	1	1mo
964	Vlad Pavlov	0.12004	33	16d
965	ibrahimmahroum1	0.12005	9	2mo
966	wojiushitest	0.12005	14	6d
967	ravalisambu	0.12005	4	2h
968	Gurusangama	0.12010	30	1h
969	sakshams1990	0.12011	1	22d
970	kaustavsbu	0.12011	9	~10s

Your Best Entry

You advanced 1,705 places on the leaderboard!

Your submission scored 0.12011, which is an improvement of your previous score of 0.14251. Great job! Tweet this!

971	Jeff Ng	0.12013	7	1mo
972	Ritvik Rawat	0.12018	9	3h
973	Yurim	0.12019	5	10d