House Price Prediction

In [2]:

```
# importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
%matplotlib inline
```

In [3]:

```
#reading the data using pandas.pd.read_csv()method creates a DataFrame from a csv file
#load dataset
train = pd.read_csv('./train.csv')
```

In [4]:

```
#showing the first five rows of the dataset
train.head()
```

Out[4]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	M
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	

5 rows × 81 columns

In [5]:

```
#showing the last five rows of the dataset
train.tail()
```

Out[5]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fe
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	N
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	Mn
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	Gd
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	N
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	N

5 rows × 81 columns

In [6]:

```
#shape of train data
#Showing the number of rows and columns in the dataset
train.shape
```

Out[6]:

```
(1460, 81)
```

In [7]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
Id               1460 non-null int64
MSSubClass       1460 non-null int64
MSZoning         1460 non-null object
LotFrontage      1201 non-null float64
LotArea          1460 non-null int64
Street           1460 non-null object
Alley            91 non-null object
LotShape         1460 non-null object
LandContour      1460 non-null object
Utilities        1460 non-null object
LotConfig        1460 non-null object
LandSlope        1460 non-null object
Neighborhood     1460 non-null object
Condition1       1460 non-null object
Condition2       1460 non-null object
BldgType         1460 non-null object
HouseStyle       1460 non-null object
OverallQual      1460 non-null int64
OverallCond      1460 non-null int64
YearBuilt        1460 non-null int64
YearRemodAdd     1460 non-null int64
RoofStyle        1460 non-null object
RoofMatl         1460 non-null object
Exterior1st      1460 non-null object
Exterior2nd      1460 non-null object
MasVnrType       1452 non-null object
MasVnrArea       1452 non-null float64
ExterQual        1460 non-null object
ExterCond        1460 non-null object
Foundation       1460 non-null object
BsmtQual         1423 non-null object
BsmtCond         1423 non-null object
BsmtExposure     1422 non-null object
BsmtFinType1     1423 non-null object
BsmtFinSF1       1460 non-null int64
BsmtFinType2     1422 non-null object
BsmtFinSF2       1460 non-null int64
BsmtUnfSF        1460 non-null int64
TotalBsmtSF      1460 non-null int64
Heating          1460 non-null object
HeatingQC        1460 non-null object
CentralAir       1460 non-null object
Electrical       1459 non-null object
1stFlrSF         1460 non-null int64
2ndFlrSF         1460 non-null int64
LowQualFinSF     1460 non-null int64
GrLivArea        1460 non-null int64
BsmtFullBath     1460 non-null int64
BsmtHalfBath     1460 non-null int64
FullBath         1460 non-null int64
HalfBath         1460 non-null int64
BedroomAbvGr     1460 non-null int64
KitchenAbvGr     1460 non-null int64
KitchenQual      1460 non-null object
TotRmsAbvGrd     1460 non-null int64
Functional       1460 non-null object
Fireplaces       1460 non-null int64
FireplaceQu      770 non-null object
GarageType       1379 non-null object
GarageYrBlt      1379 non-null float64
GarageFinish     1379 non-null object
GarageCars       1460 non-null int64
GarageArea       1460 non-null int64
GarageQual       1379 non-null object
GarageCond       1379 non-null object
PavedDrive       1460 non-null object
WoodDeckSF       1460 non-null int64
OpenPorchSF      1460 non-null int64
```

```
EnclosedPorch    1460 non-null int64
3SsnPorch        1460 non-null int64
ScreenPorch      1460 non-null int64
PoolArea         1460 non-null int64
PoolQC           7 non-null object
Fence            281 non-null object
MiscFeature      54 non-null object
MiscVal          1460 non-null int64
MoSold           1460 non-null int64
YrSold           1460 non-null int64
SaleType         1460 non-null object
SaleCondition    1460 non-null object
SalePrice        1460 non-null int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

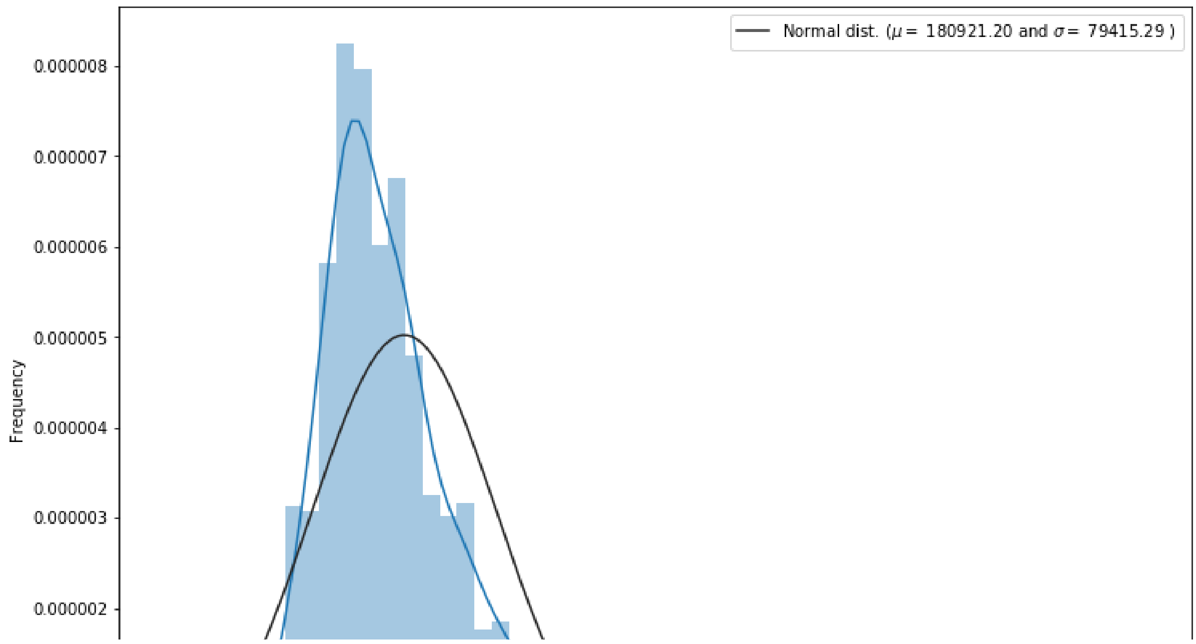
In [9]:

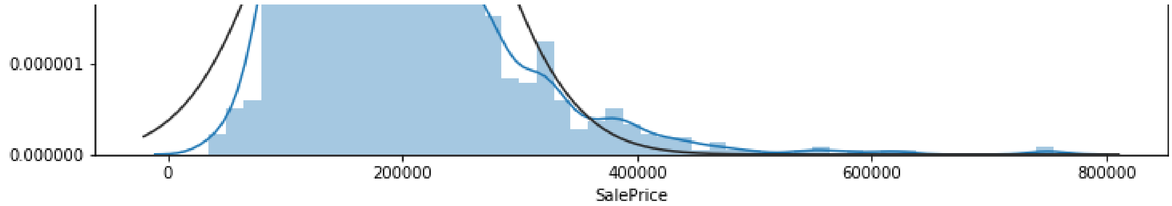
```
#now so to get the target variable we need the testing data set
#reading the test dataset using the pandas
test = pd.read_csv('./test.csv')
```

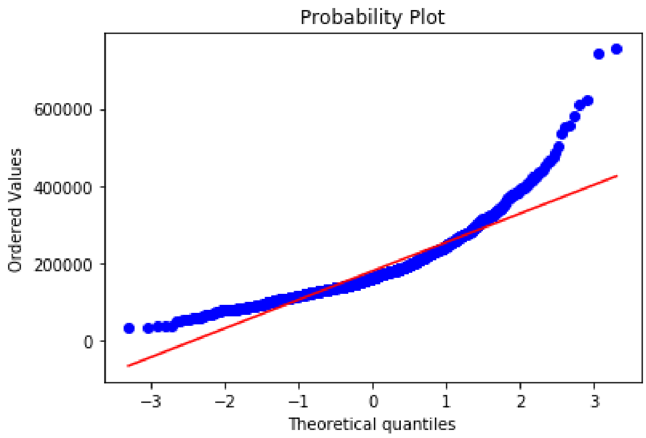
In [10]:

```
#To visualize the train data and view its distribution using graph
#some analysis on target variable
plt.subplots(figsize=(12,9))
sns.distplot(train['SalePrice'], fit=stats.norm)

# Get the fitted parameters used by the function

(mu, sigma) = stats.norm.fit(train['SalePrice'])

# plot with the distribution

plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)], loc='best')
plt.ylabel('Frequency')

#Probablity plot

fig = plt.figure()
stats.probplot(train['SalePrice'], plot=plt)
plt.show()
```

```
C:\Users\sudo\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tu
ple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[se
q]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will re
sult either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```





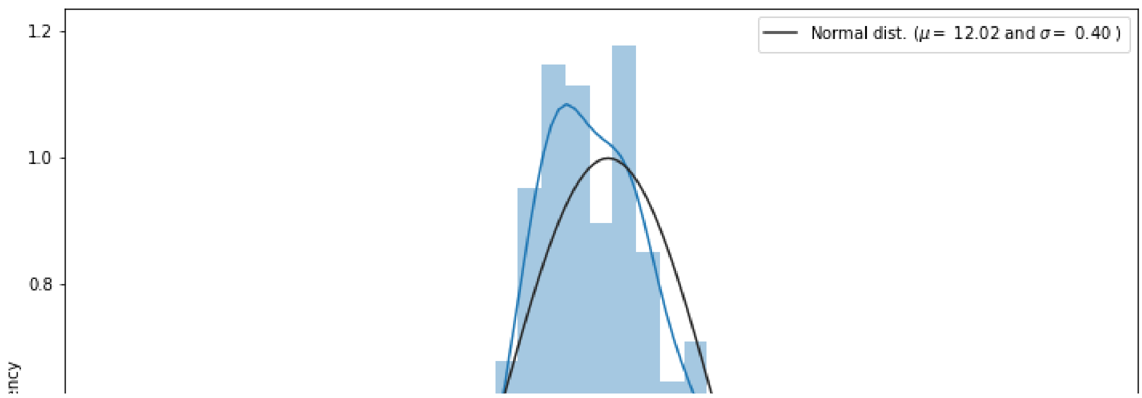


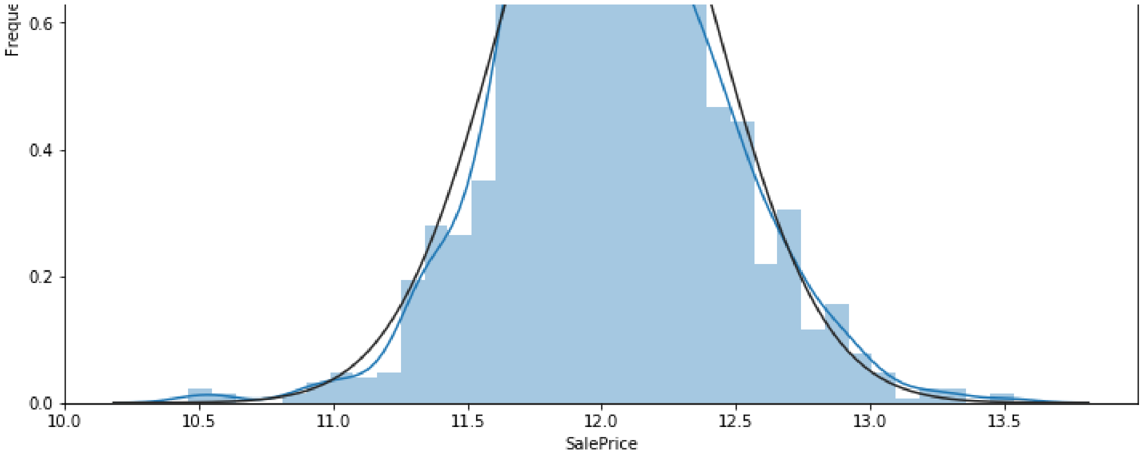
In [ ]:

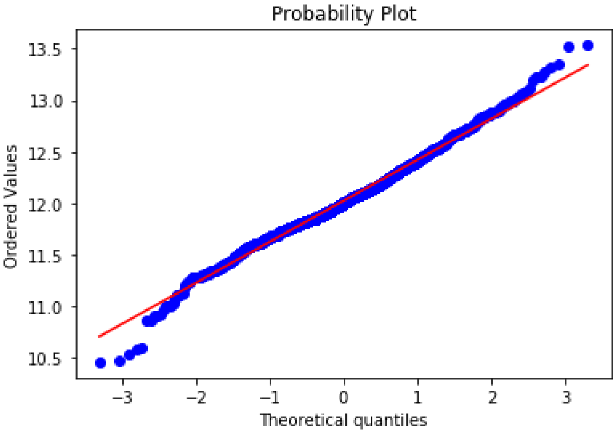
```
#As we can see from the above graph data is not normalized properly
#so we need to do normalize for better data distribution
#this target varibale is right skewed. now, we need to tranform this variable and make it normal d
istribution.
```

In [11]:

```
#Here we use log for target variable to make more normal distribution
#we use log function which is in numpy
train['SalePrice'] = np.log1p(train['SalePrice'])

#Check again for more normal distribution

plt.subplots(figsize=(12,9))
sns.distplot(train['SalePrice'], fit=stats.norm)

# Get the fitted parameters used by the function

(mu, sigma) = stats.norm.fit(train['SalePrice'])

# plot with the distribution

plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)], loc='best')
plt.ylabel('Frequency')

#Probablity plot

fig = plt.figure()
stats.probplot(train['SalePrice'], plot=plt)
plt.show()
```







In [ ]:

```
#Now we can see data is distributed in much better way
#By using the log function we were able to normalize the data
```

In [12]:

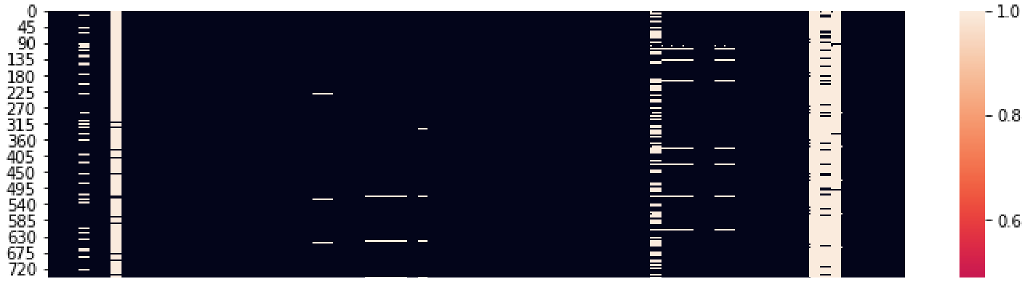
```
#Check Missing values
#Let's check if the data set has any missing values.
train.columns[train.isnull().any()]
```

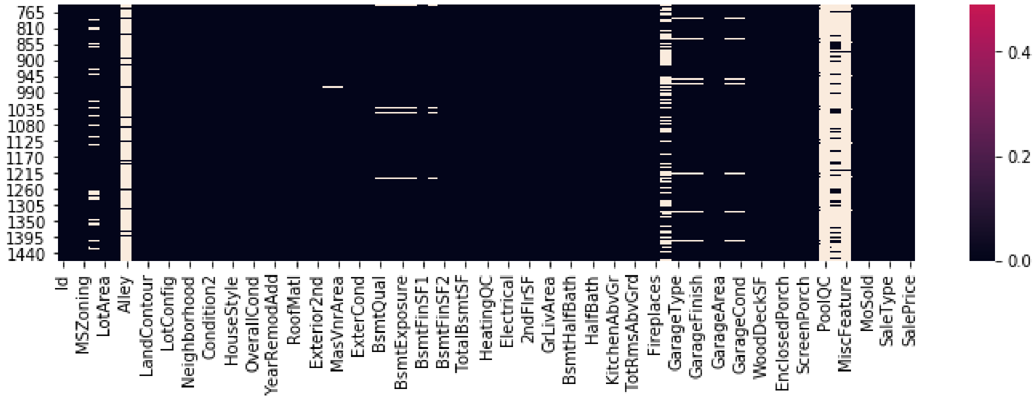
Out[12]:

```
Index(['LotFrontage', 'Alley', 'MasVnrType', 'MasVnrArea', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2',
       'Electrical', 'FireplaceQu', 'GarageType', 'GarageYrBlt',
       'GarageFinish', 'GarageQual', 'GarageCond', 'PoolQC', 'Fence',
       'MiscFeature'],
      dtype='object')
```

In [13]:

```
#plot of missing value attributes
plt.figure(figsize=(12, 6))
sns.heatmap(train.isnull())
plt.show()
```





In [ ]:

```
#As the above heatmap shows the missing values
#the white spaces on the heatmap are the missing values
#we need to fill those missing values to get the accurate result
```

In [14]:

```
#to get rid of this missing values we need to count them
#missing value counts in each of these columns
#the below code will give the detail info about number of missing value of particular columns
Isnull = train.isnull().sum()/len(train)*100
Isnull = Isnull[Isnull>0]
Isnull.sort_values(inplace=True, ascending=False)
Isnull
```

Out[14]:

```
PoolQC          99.520548
MiscFeature     96.301370
Alley           93.767123
Fence           80.753425
FireplaceQu     47.260274
LotFrontage     17.739726
GarageYrBlt      5.547945
GarageType       5.547945
GarageFinish     5.547945
GarageQual       5.547945
GarageCond       5.547945
BsmtFinType2     2.602740
BsmtExposure     2.602740
BsmtFinType1     2.534247
BsmtCond         2.534247
BsmtQual         2.534247
MasVnrArea       0.547945
MasVnrType       0.547945
Electrical       0.068493
dtype: float64
```

In [15]:

```
#to visualize the missing values
#we need to convert missing values into dataframe
Isnull = Isnull.to_frame()
```

In [16]:

```
Isnull.columns = ['count']
```

In [17]:

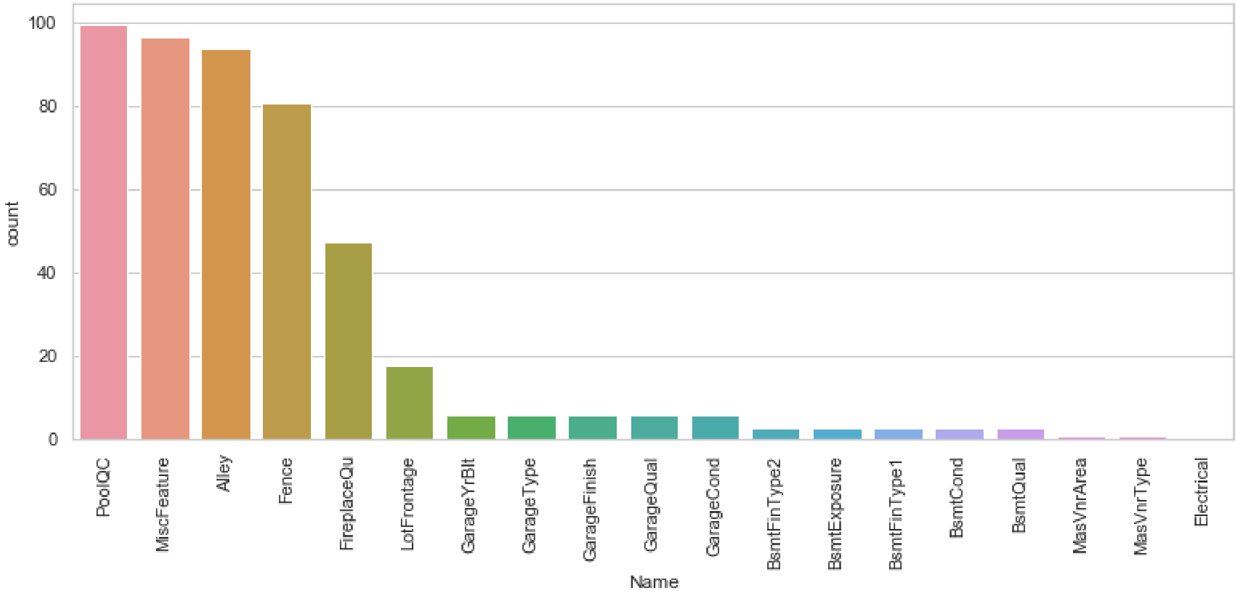
```
Isnull.index.names = ['Name']
```

In [18]:

```
Isnull['Name'] = Isnull.index
```

In [19]:

```
#plot Missing values
#ploting this missing values will give better look at the missing data
plt.figure(figsize=(13, 5))
sns.set(style='whitegrid')
sns.barplot(x='Name', y='count', data=Isnull)
plt.xticks(rotation = 90)
plt.show()
```



In [ ]:

```
#here as we can see PoolQC and MiscFeature have the highest missing values
#where as MasVnrType and Electrical have the least missing values
```

In [23]:

```
#Now we are going to show correlation between train attributes
#Separate variable into new dataframe from original dataframe which has only numerical values
#there is 38 numerical attribute from 81 attributes
train_corr = train.select_dtypes(include=[np.number])
```

In [24]:

```
#for showing the correlated dataset's rows and columns
train_corr.shape
```

Out[24]:

```
(1460, 38)
```

In [25]:

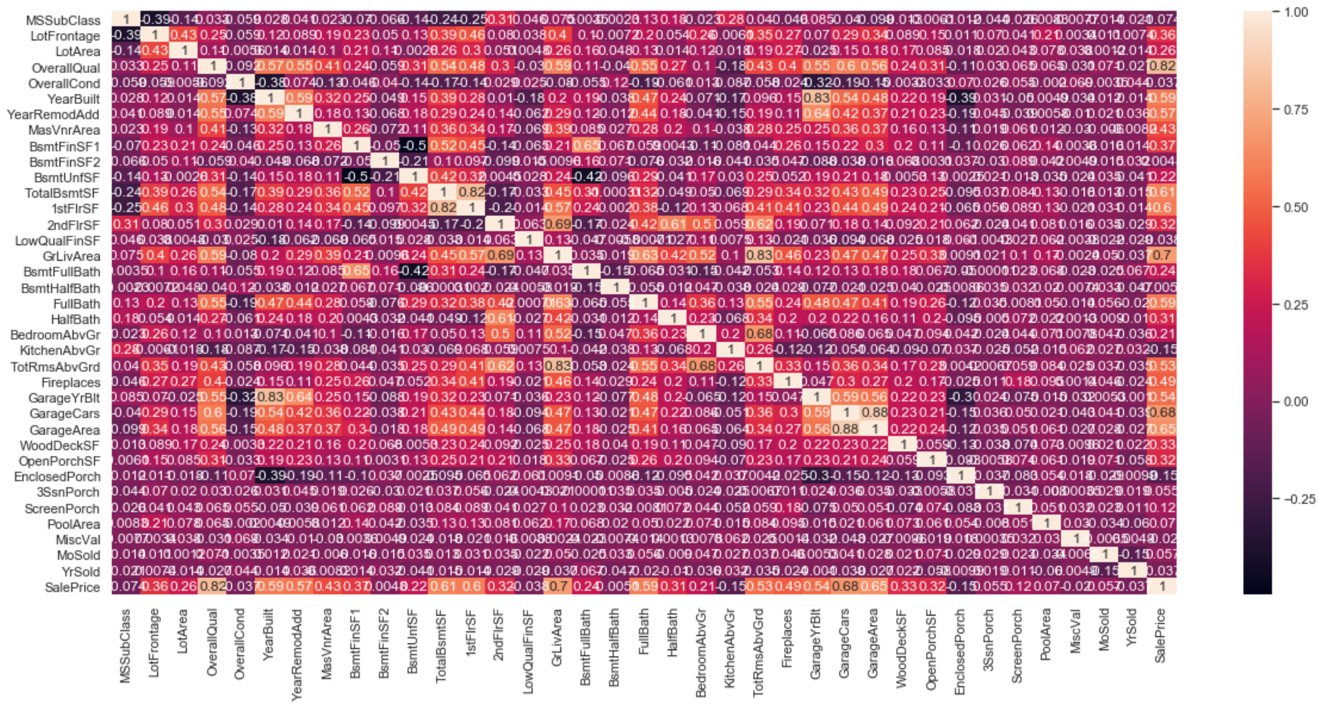
```
#Delete Id because that is not need for corralation plot
del train_corr['Id']
```

In [26]:

```
#Correlation plot
#this will give detail visual look about how the columns are correlated with each other
corr = train_corr.corr()
plt.subplots(figsize=(20,9))
sns.heatmap(corr, annot=True)
```

Out[26]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xb8bd9a8978>
```

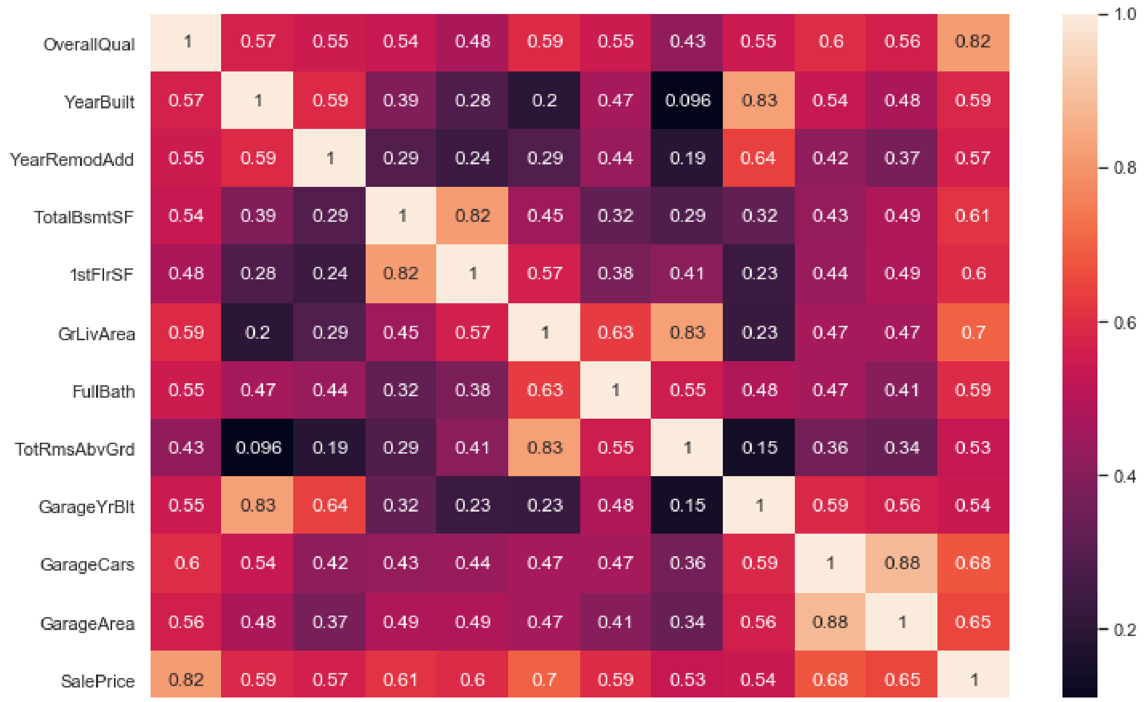


In [ ]:

```
#From the above correlation plot we can see that fields with lighter color or
# fields with higher values(from 0 to 1) are highly correalated and
# the ones with darker field or lesser values are lesser correlated
```

In [27]:

```
#Top 50% Correlation train attributes with sale-price
#this will give the columns which are correlated with respect to the Sale price i.e. more than 0.5
or 50 %
top_feature = corr.index[abs(corr['SalePrice']>0.5)]
plt.subplots(figsize=(12, 8))
top_corr = train[top_feature].corr()
sns.heatmap(top_corr, annot=True)
plt.show()
```



In [ ]:

```
#As from the above correlation plot we can see that
#Here TotRmsAbvgrd is least correlated with target feature of saleprice by 53%
#Here OverallQual is highly correlated with target feature of saleprice by 82%
```

In [31]:

```
#unique value of OverallQual
train.OverallQual.unique()
```

Out[31]:

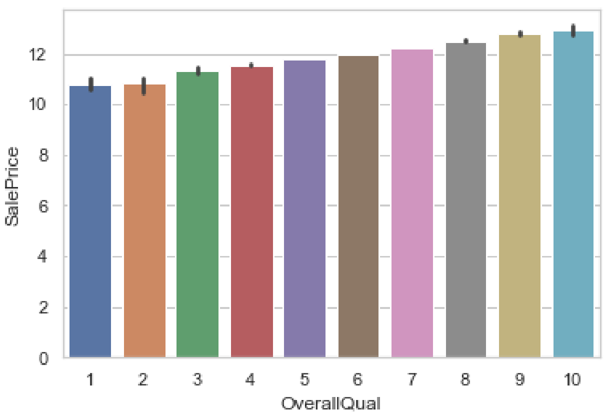
```
array([ 7,  6,  8,  5,  9,  4, 10,  3,  1,  2], dtype=int64)
```

In [32]:

```
sns.barplot(train.OverallQual, train.SalePrice)
```

Out[32]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xb8bed75d30>
```



In [ ]:

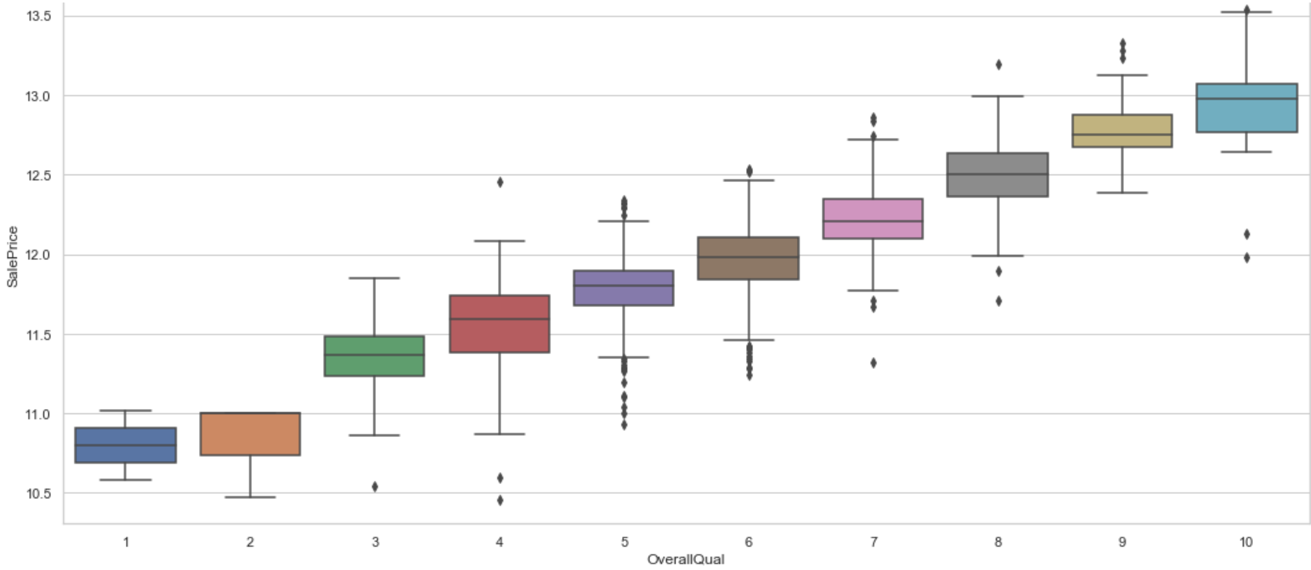
```
#the above graph shows the effects or relation of unique values of OverallQual on the saleprice
#we can see that higher the unique value of OverallQual higher is the saleprice for that unique sa
leprice
```

In [33]:

```
#Plotting a boxplot to show relation between OverallQual and Saleprice
#A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that
#facilitates comparisons between variables or across levels of a categorical variable.
#The box shows the quartiles of the dataset while the whiskers extend to show the rest of the dist
ribution,
#except for points that are determined to be “outliers” using a method that is a function of the i
nter-quartile range.
plt.figure(figsize=(18, 8))
sns.boxplot(x=train.OverallQual, y=train.SalePrice)
```

Out[33]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xb8bfb9e898>
```



In [ ]:

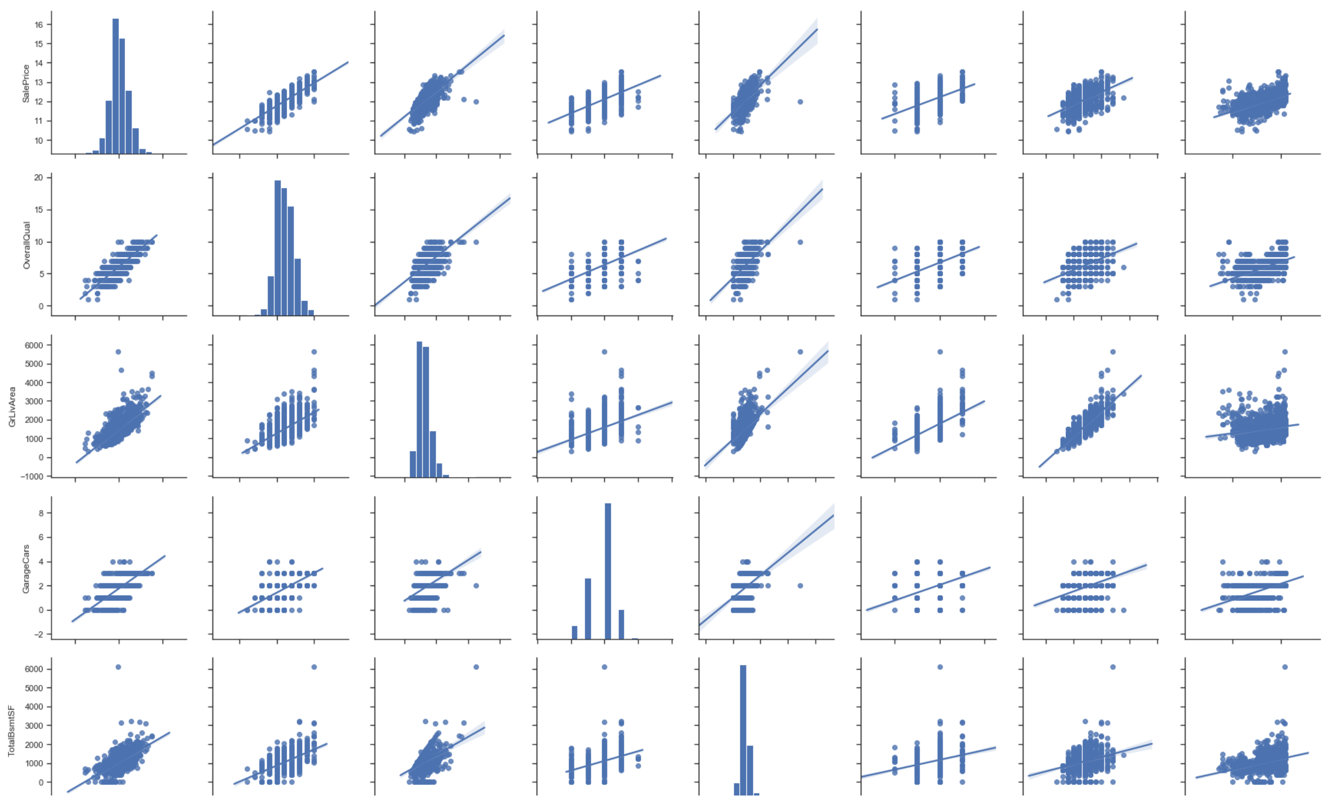
```
#the above boxplot shows the top and bottom quartile and using the box
#the outliers are shown using dotted line and
# whiskers using the lines
```

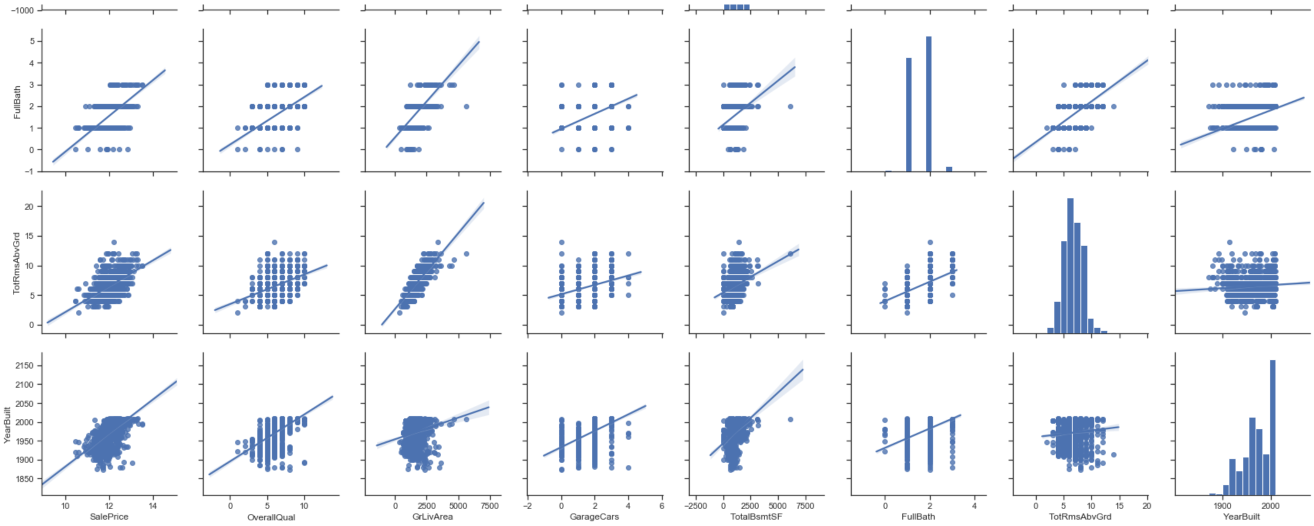
In [34]:

```
#Ploting pairplot to show relation between diffrent columns
#Plot pairwise relationships in a dataset.
#By default, this function will create a grid of Axes such that
#each variable in data will by shared in the y-axis across a single row and
#in the x-axis across a single column. The diagonal Axes are treated differently,
#drawing a plot to show the univariate distribution of the data for the variable in that column.
col = ['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'TotalBsmtSF', 'FullBath', 'TotRmsAbv
Grd', 'YearBuilt']
sns.set(style='ticks')
sns.pairplot(train[col], height=3, kind='reg')
```

Out[34]:

```
<seaborn.axisgrid.PairGrid at 0xb8bfcc8278>
```





In [ ]:

```
#The pairs plot builds on two basic figures, the histogram and the scatter plot.
#The histogram on the diagonal allows us to see the distribution of a single variable
#while the scatter plots on the upper and lower triangles show the relationship between two variab
les.
```

In [35]:

```
#this will give features that are related to target i.e. saleprice in descending order
print("Find most important features relative to target")
corr = train.corr()
corr.sort_values(['SalePrice'], ascending=False, inplace=True)
corr.SalePrice
```

```
Find most important features relative to target
```

Out[35]:

```
SalePrice        1.000000
OverallQual      0.817185
GrLivArea        0.700927
GarageCars       0.680625
GarageArea       0.650888
TotalBsmtSF      0.612134
1stFlrSF         0.596981
FullBath         0.594771
YearBuilt        0.586570
YearRemodAdd     0.565608
GarageYrBlt      0.541073
TotRmsAbvGrd     0.534422
Fireplaces       0.489450
MasVnrArea       0.430809
BsmtFinSF1       0.372023
LotFrontage      0.355879
WoodDeckSF       0.334135
OpenPorchSF      0.321053
2ndFlrSF         0.319300
HalfBath         0.313982
LotArea          0.257320
BsmtFullBath     0.236224
BsmtUnfSF        0.221985
BedroomAbvGr     0.209043
ScreenPorch      0.121208
PoolArea         0.069798
MoSold           0.057330
3SsnPorch        0.054900
BsmtFinSF2       0.004832
BsmtHalfBath    -0.005149
Id              -0.017942
MiscVal         -0.020021
OverallCond     -0.036868
YrSold          -0.037263
LowQualFinSF    -0.037963
```

```
MSSubClass      -0.073959
KitchenAbvGr    -0.147548
EnclosedPorch   -0.149050
Name: SalePrice, dtype: float64
```

In [36]:

```
#Now we are going to fill the missing value from each columns with missing values
# PoolQC has missing value ratio is 99%+. So, there is fill by None
train['PoolQC'] = train['PoolQC'].fillna('None')
```

In [37]:

```
#Around 50% missing values attributes have been fill by None
train['MiscFeature'] = train['MiscFeature'].fillna('None')
train['Alley'] = train['Alley'].fillna('None')
train['Fence'] = train['Fence'].fillna('None')
train['FireplaceQu'] = train['FireplaceQu'].fillna('None')
```

In [38]:

```
#Group by neighborhood and fill in missing value by the median LotFrontage of all the neighborhood
train['LotFrontage'] = train.groupby("Neighborhood")["LotFrontage"].transform(
    lambda x: x.fillna(x.median()))
```

In [39]:

```
#GarageType, GarageFinish, GarageQual and GarageCond these are replacing with None
for col in ['GarageType', 'GarageFinish', 'GarageQual', 'GarageCond']:
    train[col] = train[col].fillna('None')
```

In [40]:

```
#GarageYrBlt, GarageArea and GarageCars these are replacing with zero
for col in ['GarageYrBlt', 'GarageArea', 'GarageCars']:
    train[col] = train[col].fillna(int(0))
```

In [41]:

```
#BsmtFinType2, BsmtExposure, BsmtFinType1, BsmtCond, BsmtQual these are replacing with None
for col in ('BsmtFinType2', 'BsmtExposure', 'BsmtFinType1', 'BsmtCond', 'BsmtQual'):
    train[col] = train[col].fillna('None')
```

In [42]:

```
#MasVnrArea : replace with zero
train['MasVnrArea'] = train['MasVnrArea'].fillna(int(0))
```

In [43]:

```
#MasVnrType : replace with None
train['MasVnrType'] = train['MasVnrType'].fillna('None')
```

In [44]:

```
#There is put mode value
train['Electrical'] = train['Electrical'].fillna(train['Electrical']).mode()[0]
```

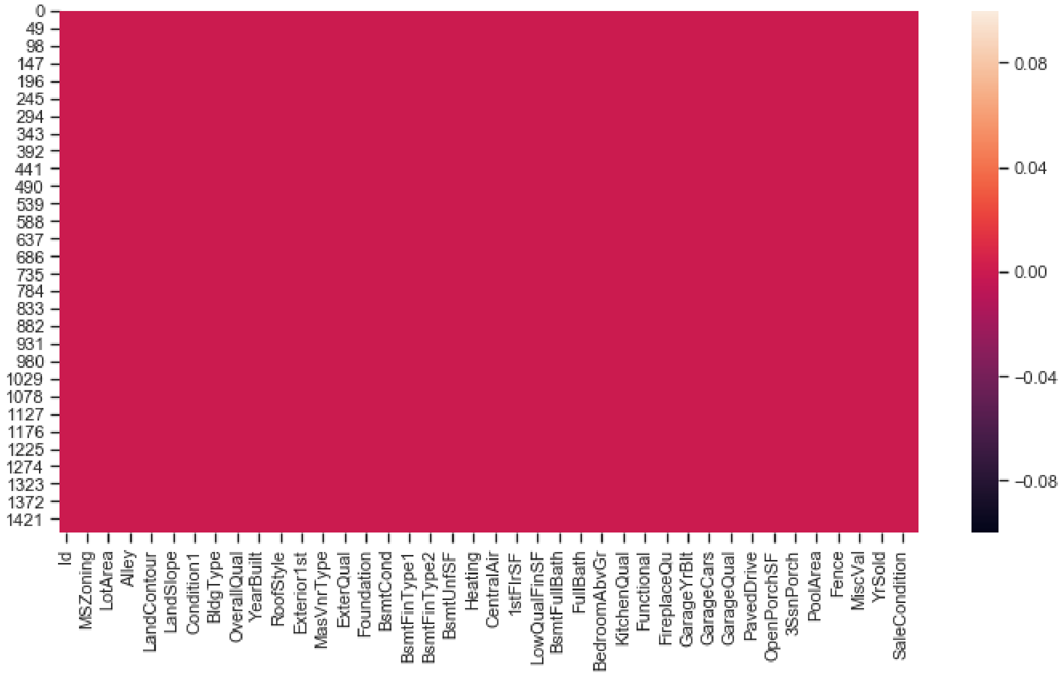
In [45]:

```
#There is no need of Utilities
train = train.drop(['Utilities'], axis=1)
```

In [47]:

```
#Checking there is any null value or not
```

```
plt.figure(figsize=(12, 6))
sns.heatmap(train.isnull())
plt.show()
```



In [ ]:

```
#from the above graph we can see that there are no missing values
#all the missing values are filed by None or zero
```

In [48]:

```
#Encoding str to int
cols = ('FireplaceQu', 'BsmtQual', 'BsmtCond', 'GarageQual', 'GarageCond',
        'ExterQual', 'ExterCond','HeatingQC', 'PoolQC', 'KitchenQual', 'BsmtFinType1',
        'BsmtFinType2', 'Functional', 'Fence', 'BsmtExposure', 'GarageFinish', 'LandSlope',
        'LotShape', 'PavedDrive', 'Street', 'Alley', 'CentralAir', 'MSSubClass', 'OverallCond',
        'YrSold', 'MoSold', 'MSZoning', 'LandContour', 'LotConfig', 'Neighborhood',
        'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st
',
        'Exterior2nd', 'MasVnrType', 'MasVnrArea', 'Foundation', 'GarageType', 'MiscFeature',
        'SaleType', 'SaleCondition', 'Electrical', 'Heating')
```

In [49]:

```
from sklearn.preprocessing import LabelEncoder
for c in cols:
    lbl = LabelEncoder()
    lbl.fit(list(train[c].values))
    train[c] = lbl.transform(list(train[c].values))
```

In [50]:

```
#for preparing the data for prediction
#Take target variable into y
#storing the saleprice into a variable y
y = train['SalePrice']
```

In [51]:

```
#As the saleprice price is stored in a variable y
#Delete the saleprice
del train['SalePrice']
```

In [52]:

In [52]:

```
#Take their values in X and y
#store the values of train and y(saleprice) in variables x and y respectively
X = train.values
y = y.values
```

In [53]:

```
# Split data into train and test format
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=7)
```

In [54]:

```
#Linear Regression
#The objective of a linear regression model is to find a relationship between one or more features
(independent variables)
#and a continuous target variable(dependent variable).
#Train the model
from sklearn import linear_model
model = linear_model.LinearRegression()
```

In [55]:

```
#Fitting the model
model.fit(X_train, y_train)
```

Out[55]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [56]:

```
#Prediction
print("Predict value " + str(model.predict([X_test[142]])))
print("Real value " + str(y_test[142]))
```

```
Predict value [11.62221633]
Real value 11.767187766223199
```

In [57]:

```
#Checking the Score/Accuracy
print("Accuracy --> ", model.score(X_test, y_test)*100)
```

```
Accuracy -->  89.26708677161409
```

In [58]:

```
#Random Forest Regression
#A Random Forest is an ensemble technique capable of performing both regression and classification
tasks
#with the use of multiple decision trees and a technique called Bootstrap Aggregation, commonly kn
own as bagging.
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators=1000)
```

In [59]:

```
#Fitting the model
model.fit(X_train, y_train)
```

Out[59]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
```

```
             min_samples_leaf=1, min_samples_split=2,
             min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=None,
             oob_score=False, random_state=None, verbose=0, warm_start=False)
```

In [60]:

```
#Prediction
print("Predict value " + str(model.predict([X_test[142]])))
print("Real value " + str(y_test[142]))
```

```
Predict value [11.70141357]
Real value 11.767187766223199
```

In [61]:

```
#Checking the Score/Accuracy
print("Accuracy --> ", model.score(X_test, y_test)*100)
```

```
Accuracy -->  89.43512782614363
```

In [68]:

```
#Gradient Boosting Regression
#Boosting is a sequential technique which works on the principle of ensemble.
#It combines a set of weak learners and delivers improved prediction accuracy.
#At any instant t, the model outcomes are weighed based on the outcomes of previous instant t-1.
#The outcomes predicted correctly are given a lower weight and the ones miss-classified are weight
ed higher.
#Train the model
from sklearn.ensemble import GradientBoostingRegressor
model = GradientBoostingRegressor(n_estimators=100, max_depth=4)
```

In [69]:

```
#Fitting the model
model.fit(X_train, y_train)
```

Out[69]:

```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
             learning_rate=0.1, loss='ls', max_depth=4, max_features=None,
             max_leaf_nodes=None, min_impurity_decrease=0.0,
             min_impurity_split=None, min_samples_leaf=1,
             min_samples_split=2, min_weight_fraction_leaf=0.0,
             n_estimators=100, n_iter_no_change=None, presort='auto',
             random_state=None, subsample=1.0, tol=0.0001,
             validation_fraction=0.1, verbose=0, warm_start=False)
```

In [70]:

```
#Prediction
print("Predict value " + str(model.predict([X_test[142]])))
print("Real value " + str(y_test[142]))
```

```
Predict value [11.65783563]
Real value 11.767187766223199
```

In [72]:

```
#Score/Accuracy
print("Accuracy --> ", model.score(X_test, y_test)*100)
```

```
Accuracy -->  91.83783816608693
```