HOUSE PRICE PREDICTION SYSTEM USING MACHINE LEARNING

📝 Project Objective:

1. The aim of this project is to train a Machine Learning Model which can predict the House Sale Price using various relevant features.
2. This project is completely based for House Prices - Advanced Regression Techniques Kaggle Competition.
3. With 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa, this competition challenges us to predict the final price of each home.
4. Dataset Link:- to get the Dataset

🌟 Business Understanding:

1. In the dynamic real estate market, the significance of accurate house price predictions is increasing significantly, as they have the potential to empower homeowners, buyers, and real estate professionals by providing valuable insights into property values and facilitating informed decision-making.
2. As a result, the real estate industry faces the crucial task of determining the appropriate pricing for houses before listing them on the market. This is achieved through a comprehensive analysis of various property attributes such as location, size, amenities, condition, market trends, and more.
3. Analyzing house attributes to determine pricing helps the real estate industry strike a balance between fair market value for sellers and affordability for buyers. It ensures that house prices align with their unique characteristics, desirability, and overall value proposition. This approach also fosters transparency and facilitates fair competition among properties, allowing buyers to make well-informed

decisions based on their specific needs, preferences, and budget constraints. Moreover, accurate house price predictions enable homeowners to assess their property's worth and make informed choices regarding selling, refinancing, or investment opportunities.

🌟 ABOUT THE DATASET:

```
Here's a brief version of what you'll find in the data description file.
SalePrice - the property's sale price in dollars. This is the target variable t
hat you're trying to predict.
MSSubClass: The building class
MSZoning: The general zoning classification
LotFrontage: Linear feet of street connected to property -
LotArea: Lot size in square feet
Street: Type of road access
Alley: Type of alley access
LotShape: General shape of property
LandContour: Flatness of the property -
Utilities: Type of utilities available
LotConfig: Lot configuration
LandSlope: Slope of property
Neighborhood: Physical locations within Ames city limits
Condition1: Proximity to main road or railroad
Condition2: Proximity to main road or railroad (if a second is present)
BldgType: Type of dwelling
HouseStyle: Style of dwelling
OverallQual: Overall material and finish quality
OverallCond: Overall condition rating
YearBuilt: Original construction date
YearRemodAdd: Remodel date
RoofStyle: Type of roof
RoofMatl: Roof material
Exterior1st: Exterior covering on house
Exterior2nd: Exterior covering on house (if more than one material)
MasVnrType: Masonry veneer type
MasVnrArea: Masonry veneer area in square feet
ExterQual: Exterior material quality
ExterCond: Present condition of the material on the exterior
Foundation: Type of foundation
BsmtQual: Height of the basement
BsmtCond: General condition of the basement
BsmtExposure: Walkout or garden level basement walls
BsmtFinType1: Quality of basement finished area
BsmtFinSF1: Type 1 finished square feet
```

IMPORTING LIBRARIES

In [296]:

```
# data analysis and wrangling
import numpy as np
import pandas as pd
import random as rnd

# visualization
import matplotlib.pyplot as plt
import seaborn as sns

#Stastics
from scipy import stats as st
import pylab

# Warnings
import warnings
warnings.filterwarnings("ignore")

# Machine Learning
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, GradientBoostingR
from catboost import CatBoostRegressor
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler, MinMaxSca
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, con
from sklearn.metrics import mean_squared_error, mean_absolute_error, mean_absolute_perce
```

In [297]:

```
train=pd.read_csv("train.csv")
```

In [298]:

```
test=pd.read_csv("test.csv")
```

In [299]:

```
pd.set_option('display.max_rows',None)
pd.set_option('display.max_columns',None)
```

In [300]:

```
train.head()
```

Out[300]:

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

In [301]:

```
test.head()
```

Out[301]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContou
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	L
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	L
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	L
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	L
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	HL

In [303]:

```
train.tail()
```

Out[303]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandCo
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	

In [304]:

```
train.isnull().sum()
```

```
GarageCars          0
GarageArea          0
GarageQual         81
GarageCond         81
PavedDrive          0
WoodDeckSF          0
OpenPorchSF         0
EnclosedPorch       0
3SsnPorch           0
ScreenPorch         0
PoolArea            0
PoolQC           1453
Fence            1179
MiscFeature      1406
MiscVal             0
MoSold              0
YrSold              0
SaleType            0
SaleCondition       0
SalePrice           0
```

Combining Both Train And Test Columns For Further Processing

Basic Understanding Of Data

In [305]:

```
df=train.append(test)
```

In [306]:

```
df.head(10)
```

Out[306]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl
5	6	50	RL	85.0	14115	Pave	NaN	IR1	Lvl
6	7	20	RL	75.0	10084	Pave	NaN	Reg	Lvl
7	8	60	RL	NaN	10382	Pave	NaN	IR1	Lvl
8	9	50	RM	51.0	6120	Pave	NaN	Reg	Lvl
9	10	190	RL	50.0	7420	Pave	NaN	Reg	Lvl

In [307]:

```
df.tail(10)
```

Out[307]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandCo
1449	2910	180	RM	21.0	1470	Pave	NaN	Reg	
1450	2911	160	RM	21.0	1484	Pave	NaN	Reg	
1451	2912	20	RL	80.0	13384	Pave	NaN	Reg	
1452	2913	160	RM	21.0	1533	Pave	NaN	Reg	
1453	2914	160	RM	21.0	1526	Pave	NaN	Reg	
1454	2915	160	RM	21.0	1936	Pave	NaN	Reg	
1455	2916	160	RM	21.0	1894	Pave	NaN	Reg	
1456	2917	20	RL	160.0	20000	Pave	NaN	Reg	
1457	2918	85	RL	62.0	10441	Pave	NaN	Reg	
1458	2919	60	RL	74.0	9627	Pave	NaN	Reg	

In [308]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2919 entries, 0 to 1458
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             2919 non-null   int64  
 1   MSSubClass     2919 non-null   int64  
 2   MSZoning       2915 non-null   object 
 3   LotFrontage    2433 non-null   float64
 4   LotArea        2919 non-null   int64  
 5   Street         2919 non-null   object 
 6   Alley          198 non-null    object 
 7   LotShape       2919 non-null   object 
 8   LandContour    2919 non-null   object 
 9   Utilities      2917 non-null   object 
 10  LotConfig      2919 non-null   object 
 11  LandSlope      2919 non-null   object 
 12  Neighborhood   2919 non-null   object 
 13  Condition1     2919 non-null   object 
```

In [309]:

```
df.describe()
```

Out[309]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	
count	2919.000000	2919.000000	2433.000000	2919.000000	2919.000000	2919.000000	291
mean	1460.000000	57.137718	69.305795	10168.114080	6.089072	5.564577	197
std	842.787043	42.517628	23.344905	7886.996359	1.409947	1.113131	3
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	187
25%	730.500000	20.000000	59.000000	7478.000000	5.000000	5.000000	195
50%	1460.000000	50.000000	68.000000	9453.000000	6.000000	5.000000	197
75%	2189.500000	70.000000	80.000000	11570.000000	7.000000	6.000000	200
max	2919.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	201

In [310]:

```
df.describe(include=object)
```

Out[310]:

	MSZoning	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Ne
count	2915	2919	198	2919	2919	2917	2919	2919	
unique	5	2	2	4	4	2	5	3	
top	RL	Pave	Grvl	Reg	Lvl	AllPub	Inside	Gtl	
freq	2265	2907	120	1859	2622	2916	2133	2778	

In [311]:

```
df.shape
```

Out[311]:

```
(2919, 81)
```

In [312]:

```
df.isnull().sum()/len(df)*100
```

Out[312]:

```
Id               0.000000
MSSubClass       0.000000
MSZoning         0.137033
LotFrontage     16.649538
LotArea          0.000000
Street           0.000000
Alley           93.216855
LotShape         0.000000
LandContour      0.000000
Utilities        0.068517
LotConfig        0.000000
LandSlope        0.000000
Neighborhood     0.000000
Condition1       0.000000
Condition2       0.000000
BldgType         0.000000
HouseStyle       0.000000
OverallQual      0.000000
```

In [313]:

```
df.duplicated().sum()/len(df)*100
```

Out[313]:

```
0.0
```

In [314]:

```
df.head()
```

Out[314]:

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

Droping of ireelevant features

In [315]:

```
df.drop(columns=['Id','Alley','FireplaceQu','PoolQC','Fence','MiscFeature'],inplace=True
```

In [316]:

```
df.head()
```

Out[316]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	I
0	60	RL	65.0	8450	Pave	Reg	Lvl	AllPub	
1	20	RL	80.0	9600	Pave	Reg	Lvl	AllPub	
2	60	RL	68.0	11250	Pave	IR1	Lvl	AllPub	
3	70	RL	60.0	9550	Pave	IR1	Lvl	AllPub	
4	60	RL	84.0	14260	Pave	IR1	Lvl	AllPub	

In [317]:

```
df.shape
```

Out[317]:

```
(2919, 75)
```

In [318]:

```
for i in df:
    print(i,"---",df[i].dtype,"==",df[i].nunique())
```

```
MSSubClass --- int64 == 16
MSZoning --- object == 5
LotFrontage --- float64 == 128
LotArea --- int64 == 1951
Street --- object == 2
LotShape --- object == 4
LandContour --- object == 4
Utilities --- object == 2
LotConfig --- object == 5
LandSlope --- object == 3
Neighborhood --- object == 25
Condition1 --- object == 9
Condition2 --- object == 8
BldgType --- object == 5
HouseStyle --- object == 8
OverallQual --- int64 == 10
OverallCond --- int64 == 9
YearBuilt --- int64 == 118
YearRemodAdd --- int64 == 61
```

In [319]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2919 entries, 0 to 1458
Data columns (total 75 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   MSSubClass     2919 non-null   int64
 1   MSZoning       2915 non-null   object
 2   LotFrontage    2433 non-null   float64
 3   LotArea        2919 non-null   int64
 4   Street         2919 non-null   object
 5   LotShape       2919 non-null   object
 6   LandContour    2919 non-null   object
 7   Utilities      2917 non-null   object
 8   LotConfig      2919 non-null   object
 9   LandSlope      2919 non-null   object
 10  Neighborhood   2919 non-null   object
 11  Condition1     2919 non-null   object
 12  Condition2     2919 non-null   object
 13  BldgType       2919 non-null   object
```

SEPRATING NUMERIC AND CATEGORICAL COLUMNS

In [320]:

```
num=["int64","float64"]
num=df.select_dtypes(include=num)
num.head()
```

Out[320]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd
0	60	65.0	8450	7	5	2003	2003
1	20	80.0	9600	6	8	1976	1976
2	60	68.0	11250	7	5	2001	2002
3	70	60.0	9550	7	5	1915	1970
4	60	84.0	14260	8	5	2000	2000

In [321]:

```
cat=['object']
cat=df.select_dtypes(include=object)
cat.head()
```

Out[321]:

	MSZoning	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood
0	RL	Pave	Reg	Lvl	AllPub	Inside	Gtl	CollgCr
1	RL	Pave	Reg	Lvl	AllPub	FR2	Gtl	Veenker
2	RL	Pave	IR1	Lvl	AllPub	Inside	Gtl	CollgCr
3	RL	Pave	IR1	Lvl	AllPub	Corner	Gtl	Crawfor
4	RL	Pave	IR1	Lvl	AllPub	FR2	Gtl	NoRidge

In [322]:

```
num.shape
```

Out[322]:

```
(2919, 37)
```

In [323]:

```
cat.shape
```

Out[323]:

```
(2919, 38)
```

Handling numerical data

In [324]:

```
num.isnull().sum()/len(df)*100
```

Out[324]:

```
MSSubClass      0.000000
LotFrontage    16.649538
LotArea         0.000000
OverallQual     0.000000
OverallCond     0.000000
YearBuilt       0.000000
YearRemodAdd    0.000000
MasVnrArea      0.787941
BsmtFinSF1      0.034258
BsmtFinSF2      0.034258
BsmtUnfSF       0.034258
TotalBsmtSF     0.034258
1stFlrSF        0.000000
2ndFlrSF        0.000000
LowQualFinSF    0.000000
GrLivArea       0.000000
BsmtFullBath    0.068517
BsmtHalfBath    0.068517
```

Filling Null values in num data

In [325]:

```
num['LotFrontage']=num['LotFrontage'].fillna(num['LotFrontage'].median())
```

In [326]:

```
num['MasVnrArea']=num['MasVnrArea'].fillna(num['MasVnrArea'].median())
```

In [327]:

```
num['BsmtFinSF1']=num['BsmtFinSF1'].fillna(num['BsmtFinSF1'].median())
```

In [328]:

```
num['BsmtFinSF2']=num['BsmtFinSF2'].fillna(num['BsmtFinSF2'].median())
```

In [329]:

```
num['BsmtUnfSF']=num['BsmtUnfSF'].fillna(num['BsmtUnfSF'].median())
```

In [330]:

```
num['TotalBsmtSF']=num['TotalBsmtSF'].fillna(num['TotalBsmtSF'].median())
```

In [331]:

```
num['BsmtFullBath']=num['BsmtFullBath'].fillna(num['BsmtFullBath'].median())
```

In [332]:

```
num['BsmtHalfBath']=num['BsmtHalfBath'].fillna(num['BsmtHalfBath'].median())
```

In [333]:

```
num['GarageYrBlt']=num['GarageYrBlt'].fillna(num['GarageYrBlt'].median())
```

In [334]:

```
num['GarageCars']=num['GarageCars'].fillna(num['GarageCars'].median())
```

In [335]:

```
num['GarageArea']=num['GarageArea'].fillna(num['GarageArea'].median())
```

In [337]:

```
num.isnull().sum()
```

Out[337]:

```
MSSubClass          0
LotFrontage         0
LotArea             0
OverallQual         0
OverallCond         0
YearBuilt           0
YearRemodAdd        0
MasVnrArea          0
BsmtFinSF1          0
BsmtFinSF2          0
BsmtUnfSF           0
TotalBsmtSF         0
1stFlrSF            0
2ndFlrSF            0
LowQualFinSF        0
GrLivArea           0
BsmtFullBath        0
BsmtHalfBath        0
```

Dropping irrelevent features from num data

In [338]:

```
num.head()
```

Out[338]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd
0	60	65.0	8450	7	5	2003	2003
1	20	80.0	9600	6	8	1976	1976
2	60	68.0	11250	7	5	2001	2002
3	70	60.0	9550	7	5	1915	1970
4	60	84.0	14260	8	5	2000	2000

In [339]:

```
num.drop(columns=['BsmtFinSF2','LowQualFinSF','BsmtFullBath','GarageYrBlt','BsmtHalfBath
```

In [340]:

```
num.isnull().sum()
```

Out[340]:

```
MSSubClass      0
LotFrontage     0
LotArea         0
OverallQual     0
OverallCond     0
YearBuilt       0
YearRemodAdd    0
MasVnrArea      0
TotalBsmtSF     0
1stFlrSF        0
2ndFlrSF        0
GrLivArea       0
FullBath        0
HalfBath        0
BedroomAbvGr    0
KitchenAbvGr    0
TotRmsAbvGrd    0
Fireplaces      0
```

In [341]:

```
num.shape
```

Out[341]:

```
(2919, 23)
```

Handling Categorical data

In [342]:

```
cat.head()
```

Out[342]:

	MSZoning	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood
0	RL	Pave	Reg	Lvl	AllPub	Inside	Gtl	CollgCr
1	RL	Pave	Reg	Lvl	AllPub	FR2	Gtl	Veenker
2	RL	Pave	IR1	Lvl	AllPub	Inside	Gtl	CollgCr
3	RL	Pave	IR1	Lvl	AllPub	Corner	Gtl	Crawfor
4	RL	Pave	IR1	Lvl	AllPub	FR2	Gtl	NoRidge

Dropping irrelevent data from cat

In [343]:

```
cat.drop(columns=['GarageQual','LotConfig',"GarageFinish","MasVnrType","Exterior1st","Bs
```

In [344]:

```
cat.head()
```

Out[344]:

	MSZoning	Street	LotShape	LandContour	Utilities	Neighborhood	Condition1	HouseStyl
0	RL	Pave	Reg	Lvl	AllPub	CollgCr	Norm	2Stor
1	RL	Pave	Reg	Lvl	AllPub	Veenker	Feedr	1Stor
2	RL	Pave	IR1	Lvl	AllPub	CollgCr	Norm	2Stor
3	RL	Pave	IR1	Lvl	AllPub	Crawfor	Norm	2Stor
4	RL	Pave	IR1	Lvl	AllPub	NoRidge	Norm	2Stor

In [345]:

```
cat.shape
```

Out[345]:

```
(2919, 25)
```

In [346]:

```
cat.isnull().sum()
```

Out[346]:

```
MSZoning           4
Street             0
LotShape           0
LandContour        0
Utilities          2
Neighborhood       0
Condition1         0
HouseStyle         0
RoofStyle          0
RoofMatl           0
Exterior2nd        1
ExterQual          0
ExterCond          0
Foundation         0
BsmtQual          81
HeatingQC          0
CentralAir         0
Electrical         1
KitchenQual        1
Functional         2
GarageType       157
GarageCond       159
PavedDrive         0
SaleType           1
SaleCondition      0
dtype: int64
```

Filling null Values in categorical data data

In [347]:

```
for column in cat.columns:
    if cat[column].dtype=='object':
        cat[column]=cat[column].fillna(cat[column].mode()[0])
```

In [348]:

```
print('Missing values in Cat data:', cat.isnull().sum().sum())
```

```
Missing values in Cat data: 0
```

In [349]:

```
cat.isnull().sum()
```

Out[349]:

```
MSZoning         0
Street           0
LotShape         0
LandContour      0
Utilities        0
Neighborhood     0
Condition1       0
HouseStyle       0
RoofStyle        0
RoofMatl         0
Exterior2nd      0
ExterQual        0
ExterCond        0
Foundation       0
BsmtQual         0
HeatingQC        0
CentralAir       0
Electrical       0
KitchenQual      0
Functional       0
GarageType       0
GarageCond       0
PavedDrive       0
SaleType         0
SaleCondition    0
dtype: int64
```

concatination of both num and cat data

In [350]:

```
df=pd.concat([num,cat],axis=1)
```

In [351]:

```
df.head()
```

Out[351]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd
0	60	65.0	8450	7	5	2003	2003
1	20	80.0	9600	6	8	1976	1976
2	60	68.0	11250	7	5	2001	2002
3	70	60.0	9550	7	5	1915	1970
4	60	84.0	14260	8	5	2000	2000

In [352]:

```
df.shape
```

Out[352]:

```
(2919, 48)
```

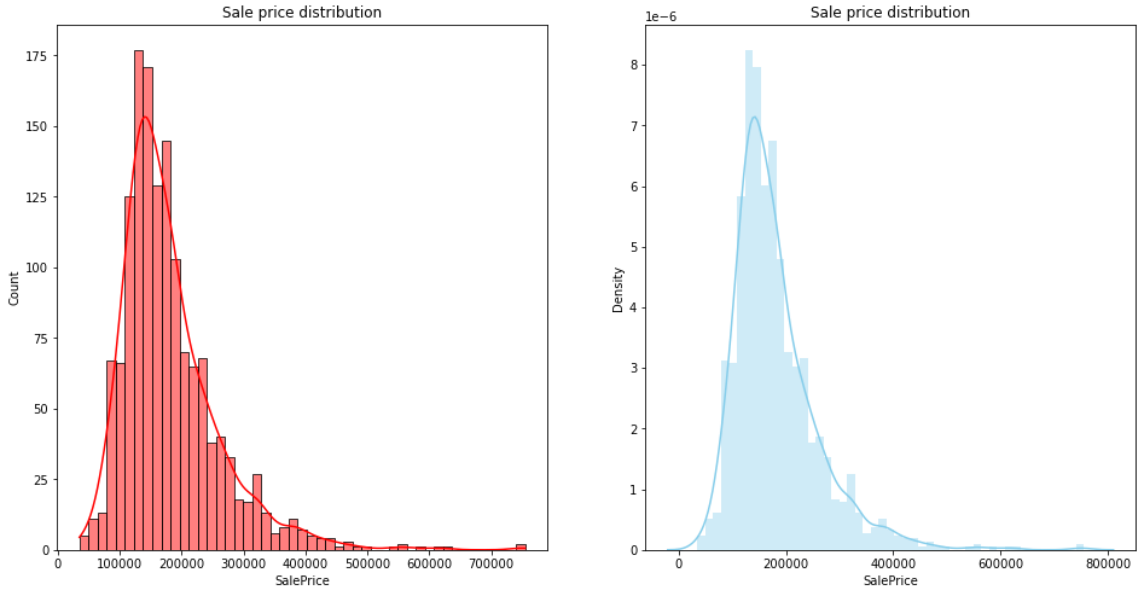
EXPLORATORY DATA ANALYSIS

VISULALIZING Numerical Features

VISULALIZING TARGET VARIABLE (SALEPRICE):

In [353]:

```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.title("Sale price distribution")
sns.histplot(df['SalePrice'],color='red',kde=True)

plt.subplot(1,2,2)
plt.title("Sale price distribution")
sns.distplot(df['SalePrice'],color='skyblue');
```



observation:

most of the sale price of houses lied between 100000 to 250000
Highest price of the house is 755000.0

In [354]:

```
df[df['SalePrice']>600000]
```

Out[354]:

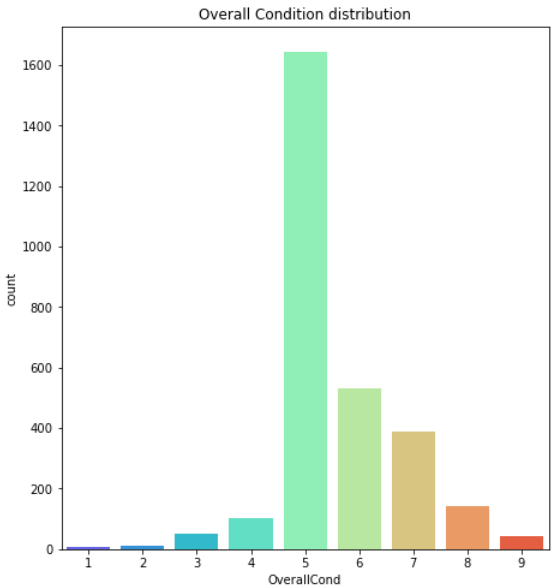
	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAd
691	60	104.0	21535	10	6	1994	199
898	20	100.0	12919	9	5	2009	201
1169	60	118.0	35760	10	5	1995	199
1182	60	160.0	15623	10	5	1996	199

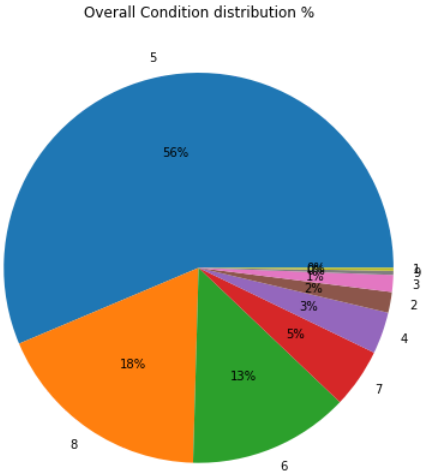
Viualizing OverallCond: Rates the overall condition of the house

```
10    Very Excellent
9     Excellent
8     Very Good
7     Good
6     Above Average
5     Average
4     Below Average
3     Fair
2     Poor
1     Very Poor
```

In [355]:

```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.title("Overall Condition distribution")
sns.countplot(df['OverallCond'],palette='rainbow',data=df)

plt.subplot(1,2,2)
plt.title("Overall Condition distribution %")
plt.pie(df['OverallCond'].value_counts(),labels=df['OverallCond'].unique(),autopct='%0.0
```





observation:

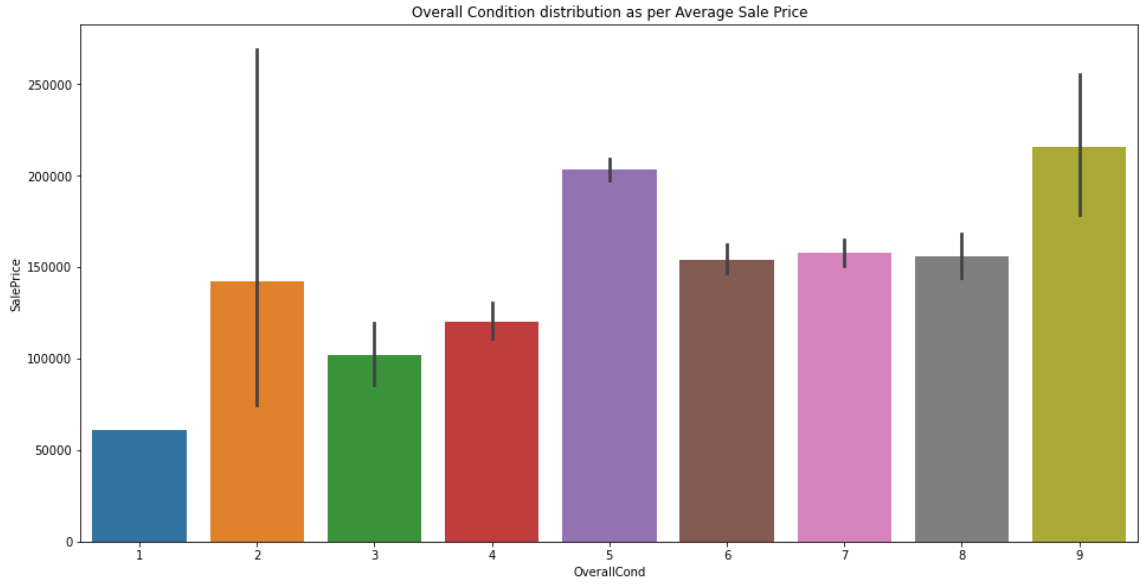
most demandable houses conditions is 5,6,7
Around 0.5% of the houses have Very Excellent overall condition
Most of the houses Around 56% of the houses have Average overall condition

In [356]:

```
plt.figure(figsize=(16,8))
plt.title("Overall Condition distribution as per Average Sale Price")
sns.barplot(df['OverallCond'],df['SalePrice'])
```

Out[356]:

```
<AxesSubplot:title={'center':'Overall Condition distribution as per Averag
e Sale Price'}, xlabel='OverallCond', ylabel='SalePrice'>
```



Observation:

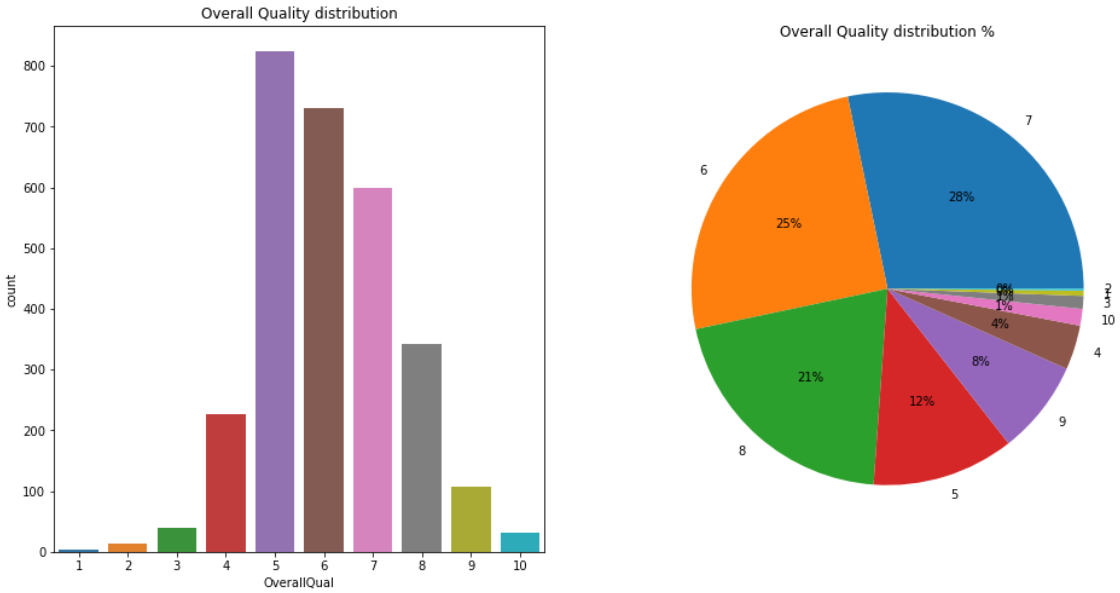
Average price for Excellent overall condition house is High

Viualizing OverallQual: Rates the overall material and finish of the house

10	Very Excellent
9	Excellent
8	Very Good
7	Good
6	Above Average
5	Average
4	Below Average
3	Fair
2	Poor
1	Very Poor

In [357]:

```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.title("Overall Quality distribution")
sns.countplot(df['OverallQual'],data=df)

plt.subplot(1,2,2)
plt.title("Overall Quality distribution %")
plt.pie(df['OverallQual'].value_counts(),labels=df['OverallQual'].unique(),autopct='%0.0
```



observation:

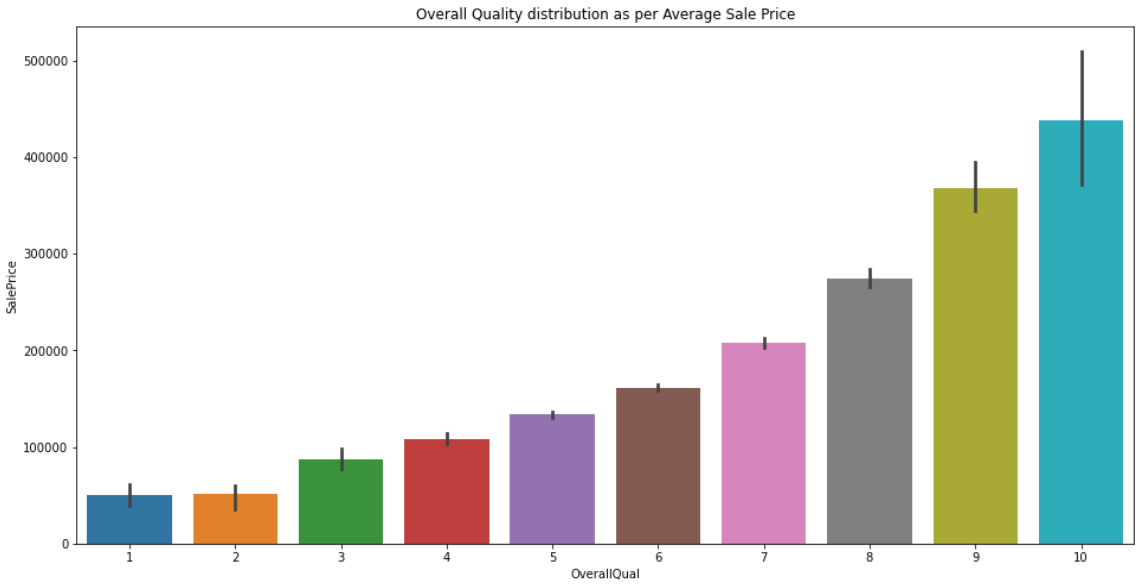
most demandable houses quality is 5,6,7
Around 1% of the houses have Very Excellent overall Quality
Most of the houses Around 28% of the houses have Average overall Quality

In [358]:

```
plt.figure(figsize=(16,8))
plt.title("Overall Quality distribution as per Average Sale Price")
sns.barplot(df['OverallQual'],df['SalePrice'])
```

Out[358]:

```
<AxesSubplot:title={'center':'Overall Quality distribution as per Average
Sale Price'}, xlabel='OverallQual', ylabel='SalePrice'>
```



Visualizing Year Built :

In [359]:

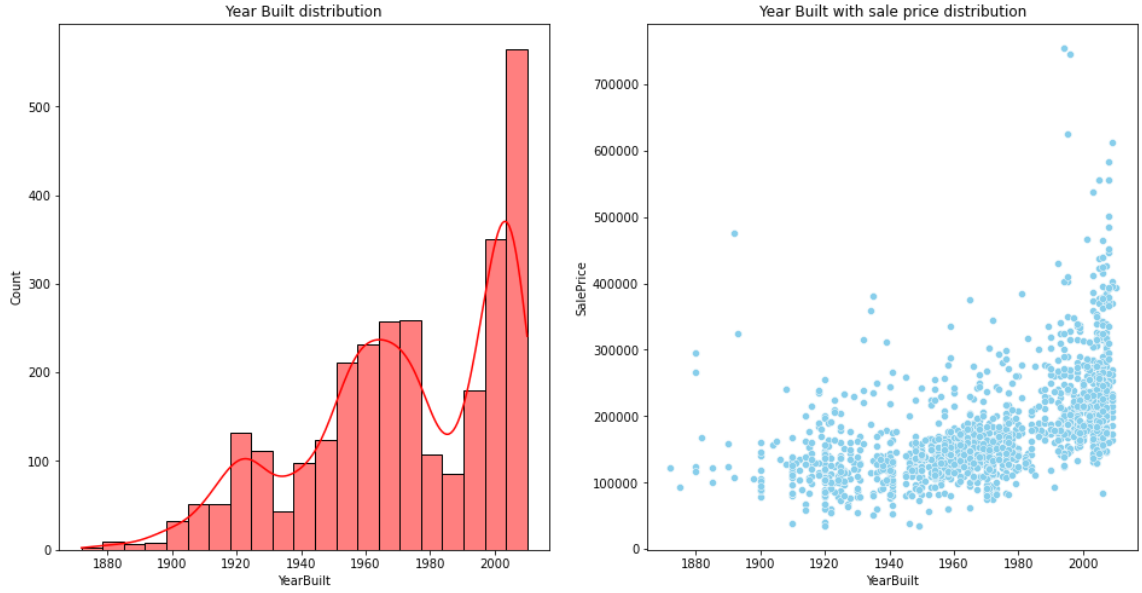
```
df['YearBuilt'].value_counts().head()
```

Out[359]:

```
2005    142
2006    138
2007    109
2004     99
2003     88
Name: YearBuilt, dtype: int64
```

In [360]:

```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.title("Year Built distribution")
sns.histplot(df['YearBuilt'],color='red',kde=True)


plt.subplot(1,2,2)
plt.title("Year Built with sale price distribution")
sns.scatterplot(df['YearBuilt'],df['SalePrice'],color='skyblue');
```



observations:

year with most of houses built is 2005 with 142 houses
most of the houses built between year (1950 to 1970) and (2003 to 2007)

visualizing YearRemodAdd : house remodeled

In [361]:

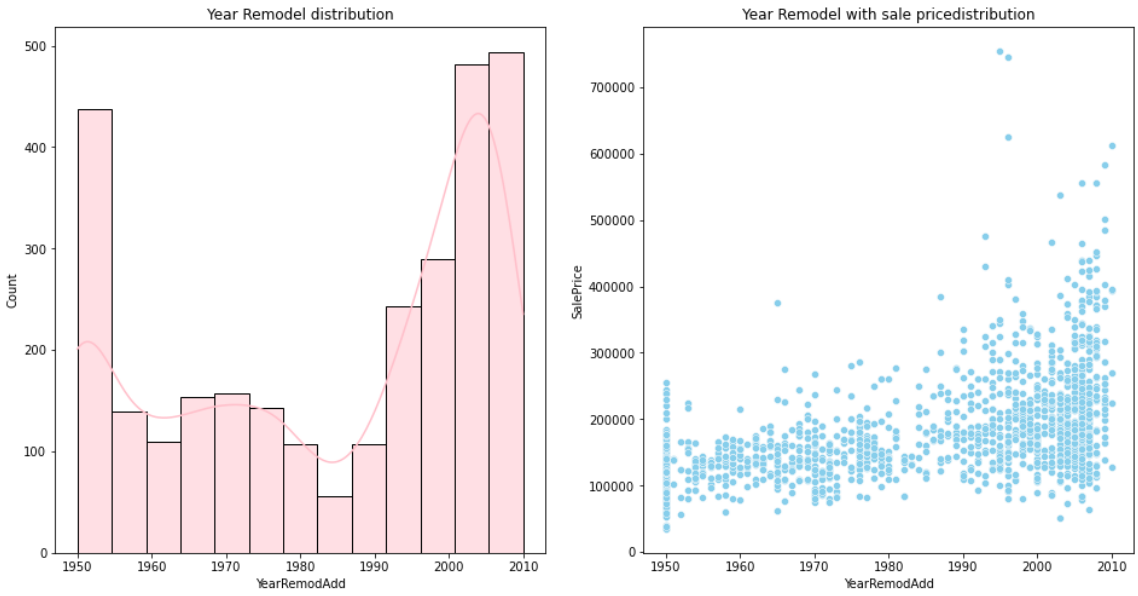
```
df['YearRemodAdd'].value_counts().head()
```

Out[361]:

```
1950    361
2006    202
2007    164
2005    141
2004    111
Name: YearRemodAdd, dtype: int64
```

In [362]:

```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.title("Year Remodel distribution")
sns.histplot(df['YearRemodAdd'],color='pink',kde=True)


plt.subplot(1,2,2)
plt.title("Year Remodel with sale pricedistribution")
sns.scatterplot(df['YearRemodAdd'],df['SalePrice'],color='skyblue');
```



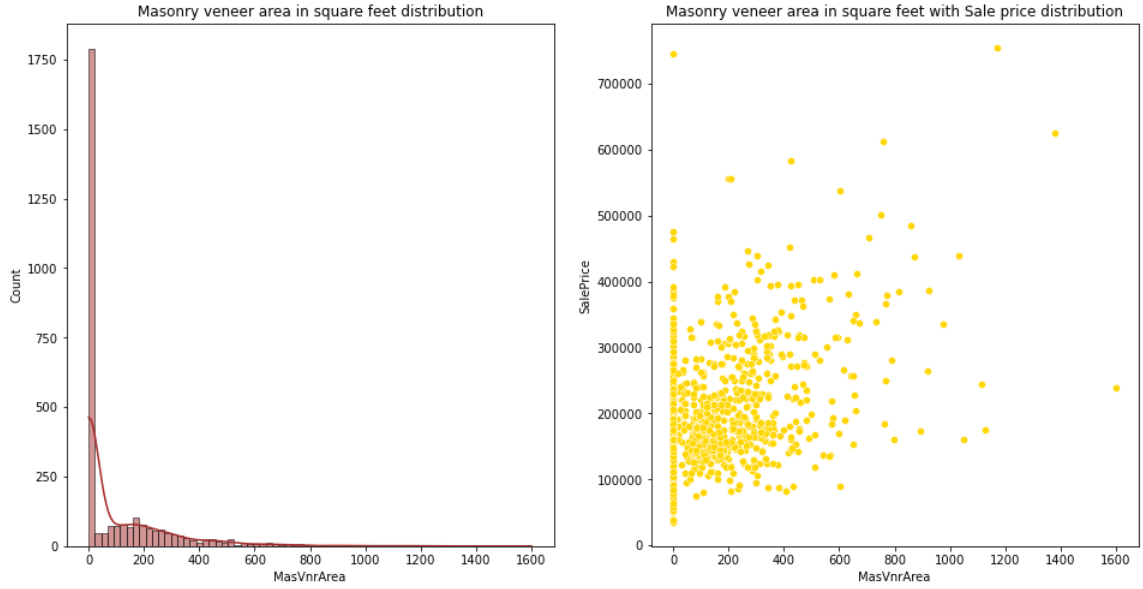
Observation:

highest house build in 1950 around 361
most of the houses were remodeled between 1990 to 2010

Visualizing MasVnrArea :Masonry veneer area in square feet

In [363]:

```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.title("Masonry veneer area in square feet distribution")
sns.histplot(df['MasVnrArea'],color='brown',kde=True)


plt.subplot(1,2,2)
plt.title("Masonry veneer area in square feet with Sale price distribution")
sns.scatterplot(df['MasVnrArea'],df['SalePrice'],color='gold');
```



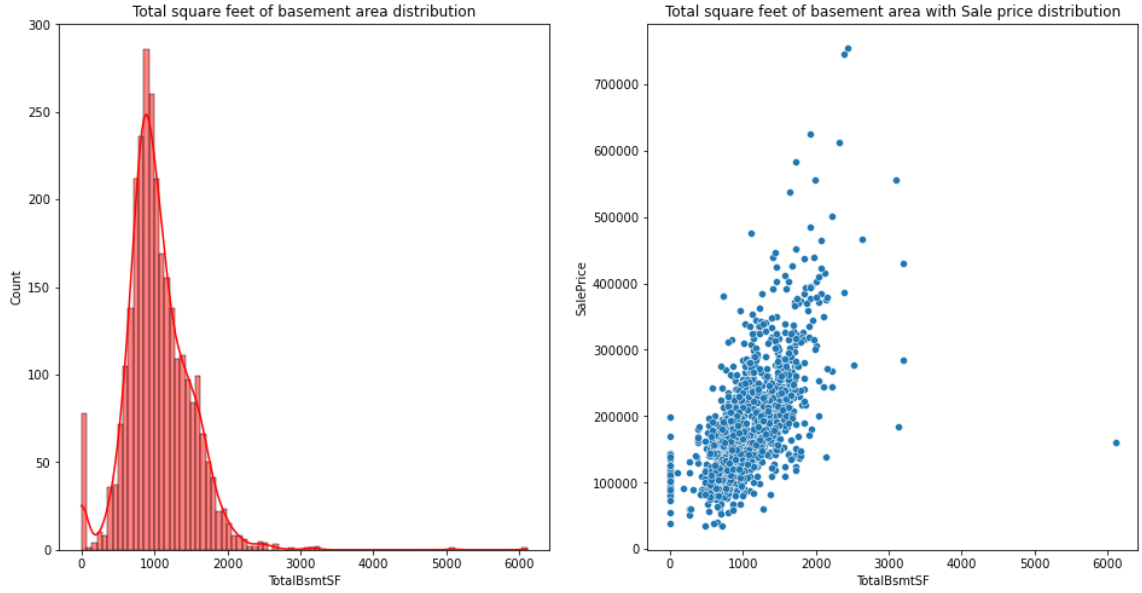
observation:

Most of the houses have 0.0 Masonry veneer area

Visualizing TotalBsmtSF: Total square feet of basement area

In [364]:

```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.title("Total square feet of basement area distribution")
sns.histplot(df['TotalBsmtSF'],color='red',kde=True)


plt.subplot(1,2,2)
plt.title("Total square feet of basement area with Sale price distribution")
sns.scatterplot(df['TotalBsmtSF'],df['SalePrice']);
```



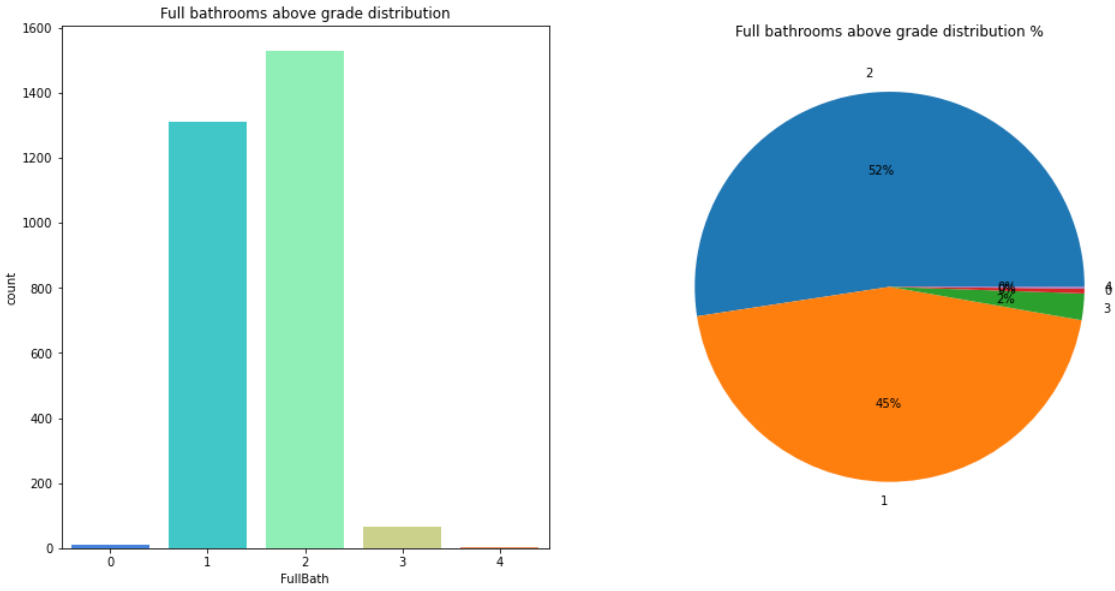
observation :

Most of the houses have 1000 sqft basement size
Highest basement sq feet of house is 6000 sqft.

Visualizing FullBath Full bathrooms above grade

In [365]:

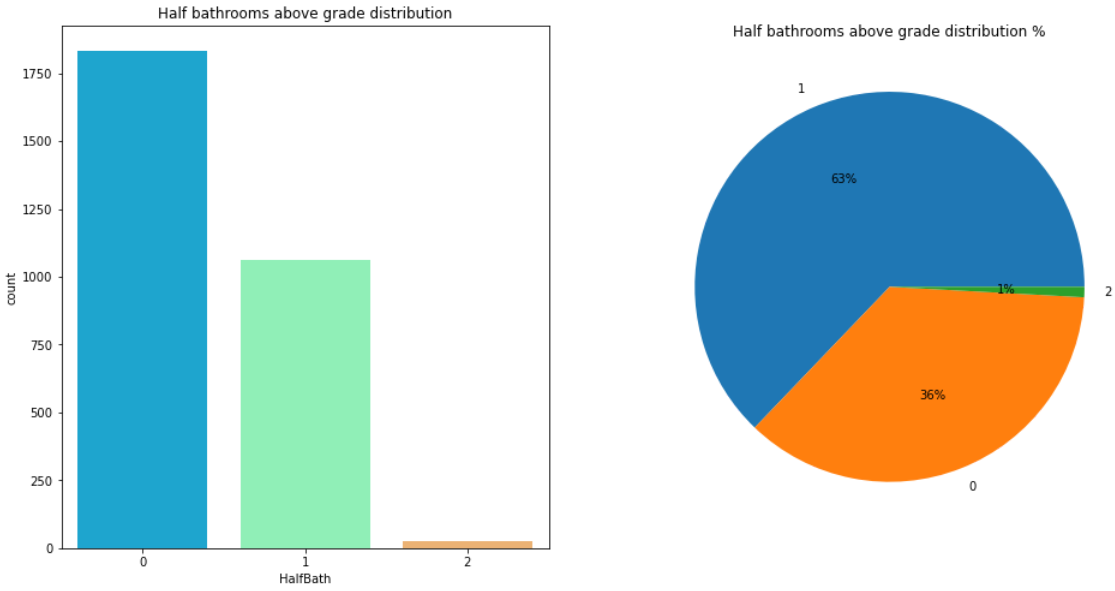
```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.title("Full bathrooms above grade distribution")
sns.countplot(df['FullBath'],palette='rainbow',data=df)

plt.subplot(1,2,2)
plt.title("Full bathrooms above grade distribution %")
plt.pie(df['FullBath'].value_counts(),labels=df['FullBath'].unique(),autopct='%0.0f%%',d
```



Visualizing HalfBath: Half baths above grade

In [366]:

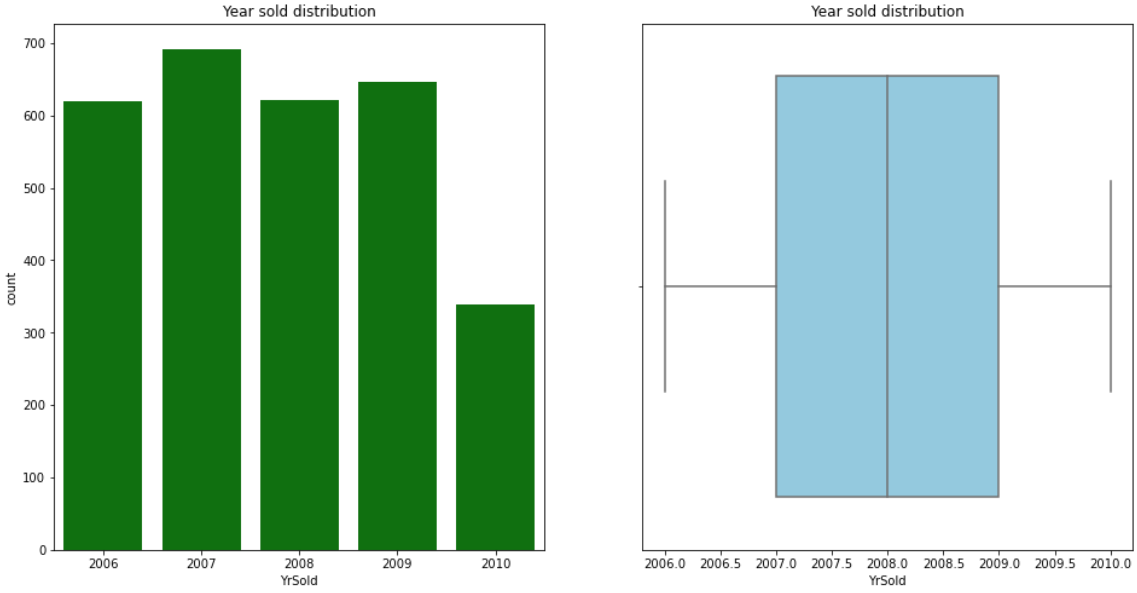
```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.title("Half bathrooms above grade distribution")
sns.countplot(df['HalfBath'],palette='rainbow',data=df)

plt.subplot(1,2,2)
plt.title("Half bathrooms above grade distribution %")
plt.pie(df['HalfBath'].value_counts(),labels=df['HalfBath'].unique(),autopct='%0.0f%%',d
```



Visualizing YrSold year sold

In [367]:

```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.title("Year sold distribution")
sns.countplot(df['YrSold'],color='green')


plt.subplot(1,2,2)
plt.title("Year sold distribution")
sns.boxplot(df['YrSold'],color='skyblue');
```



Observation:

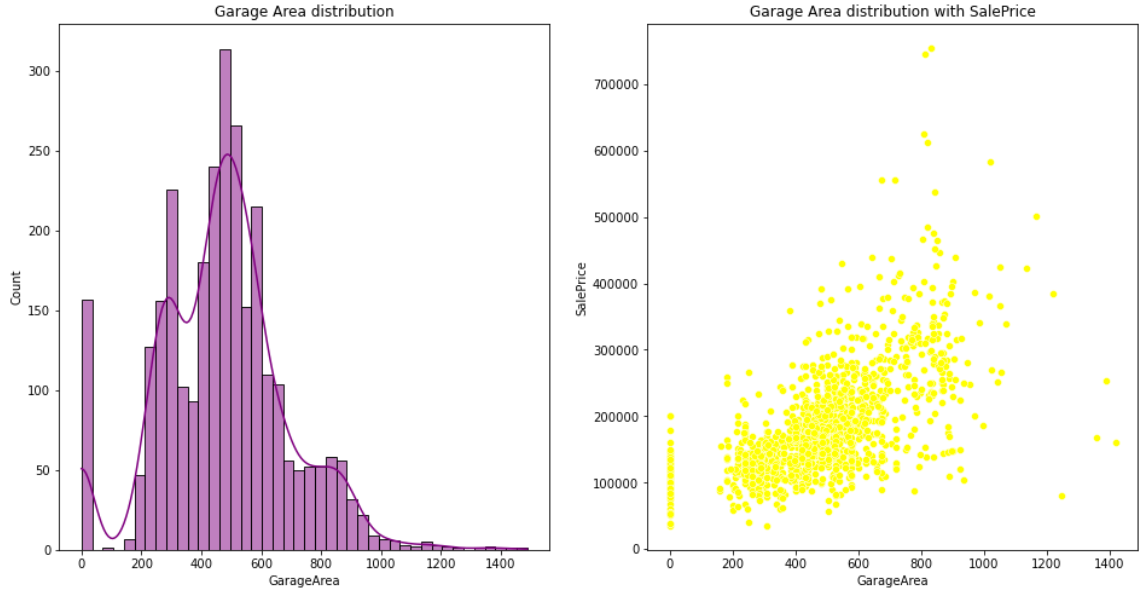
most of the house sold in 2007

least house sold in 2010

Visualizing GarageArea

In [368]:

```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.title("Garage Area distribution")
sns.histplot(df['GarageArea'],color='purple',kde=True)


plt.subplot(1,2,2)
plt.title("Garage Area distribution with SalePrice")
sns.scatterplot(df['GarageArea'],df['SalePrice'],color='yellow');
```



In [369]:

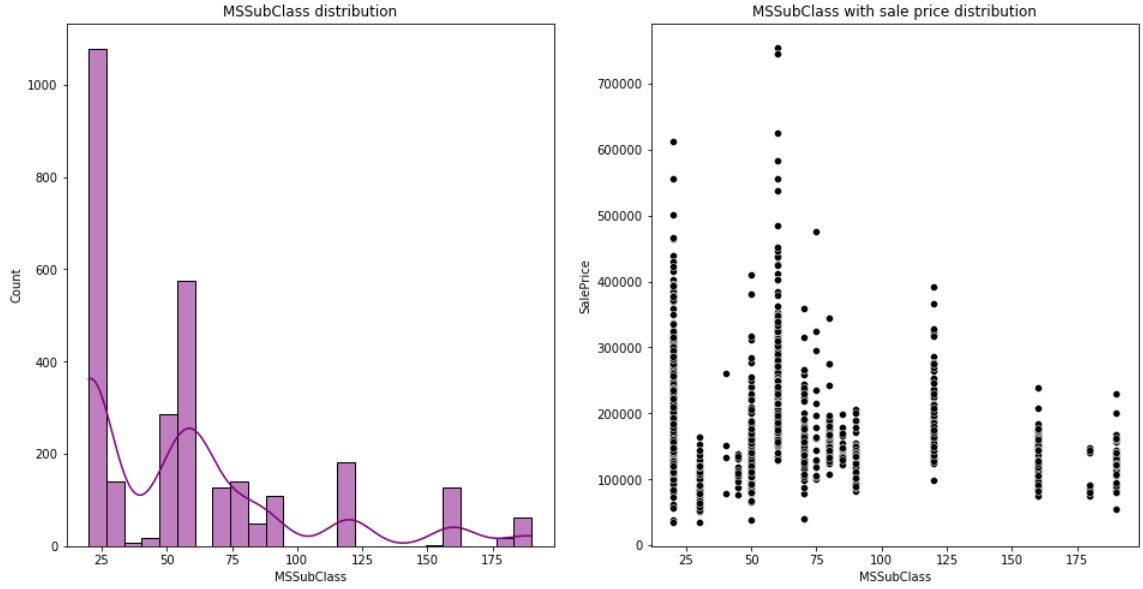
```
# MSSubClass    LotFrontage LotArea
```

Visualizing MSSubClass feature:

```
    Identifies the type of dwelling involved in the sale.

        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
```

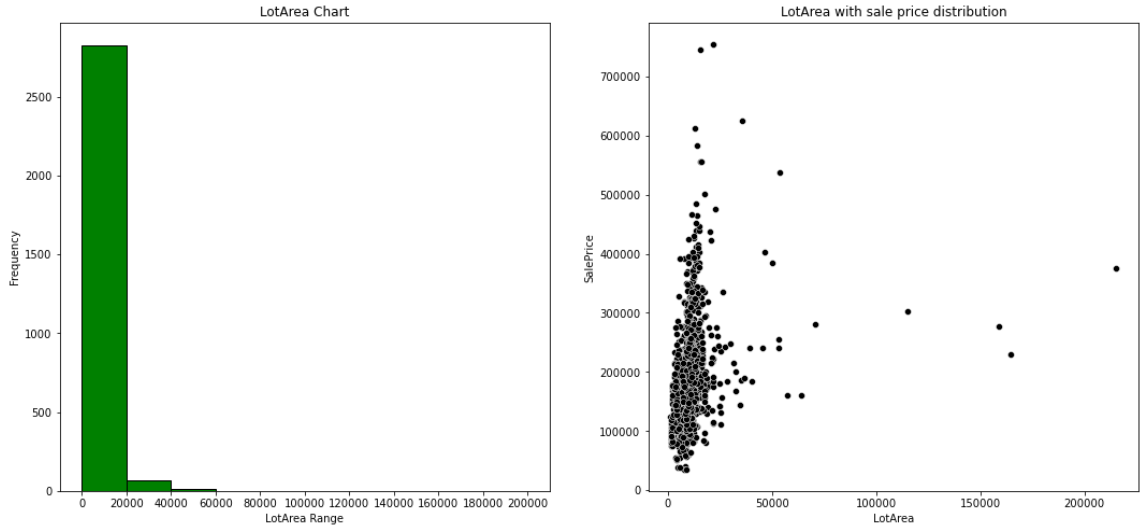
In [370]:

```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.title("MSSubClass distribution")
sns.histplot(df['MSSubClass'],color='purple',kde=True)


plt.subplot(1,2,2)
plt.title("MSSubClass with sale price distribution")
sns.scatterplot(df['MSSubClass'],df['SalePrice'],color='black');
```



In [371]:

```
plt.figure(figsize=(18,8));
plt.subplot(1,2,1)
plt.hist(df["LotArea"],edgecolor="black",color="g",bins=[0,20000,40000,60000,80000,10000
plt.xticks([0,20000,40000,60000,80000,100000,120000,140000,160000,180000,200000]);
plt.title("LotArea Chart");
plt.xlabel("LotArea Range");
plt.ylabel("Frequency");

plt.subplot(1,2,2)
plt.title("LotArea with sale price distribution")
sns.scatterplot(df['LotArea'],df['SalePrice'],color='black');
```



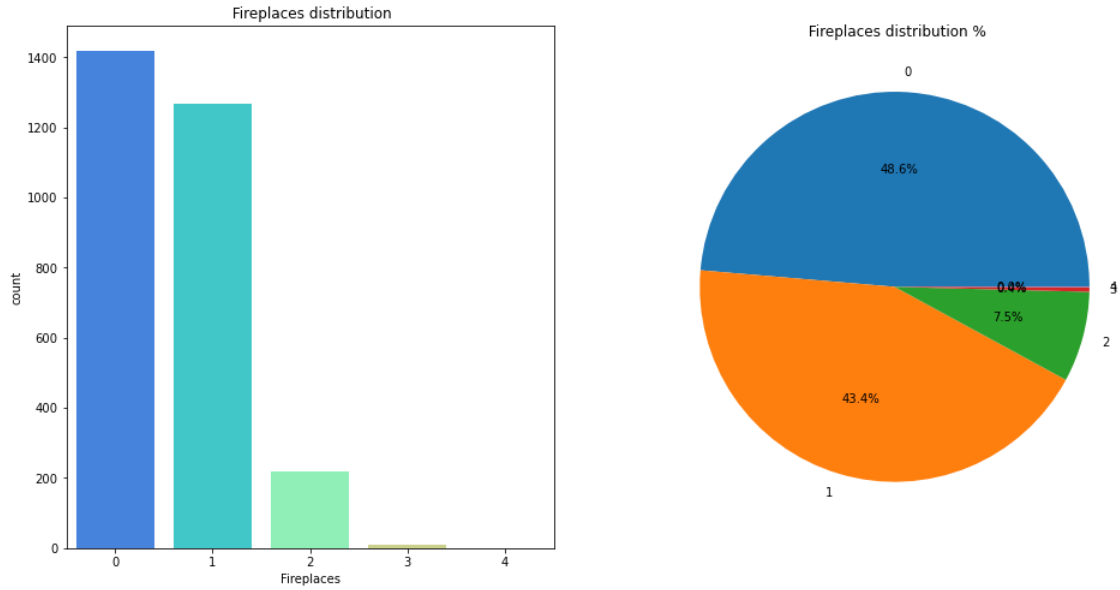
Observation:

Lot size in square feet ranges between 0 to 20000

Visualizing Fireplaces Feature

In [372]:

```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.title(" Fireplaces distribution")
sns.countplot(df['Fireplaces'],palette='rainbow',data=df)

plt.subplot(1,2,2)
plt.title(" Fireplaces distribution %")
plt.pie(df['Fireplaces'].value_counts(),labels=df['Fireplaces'].unique(),autopct='%0.1f%
```



Observation :

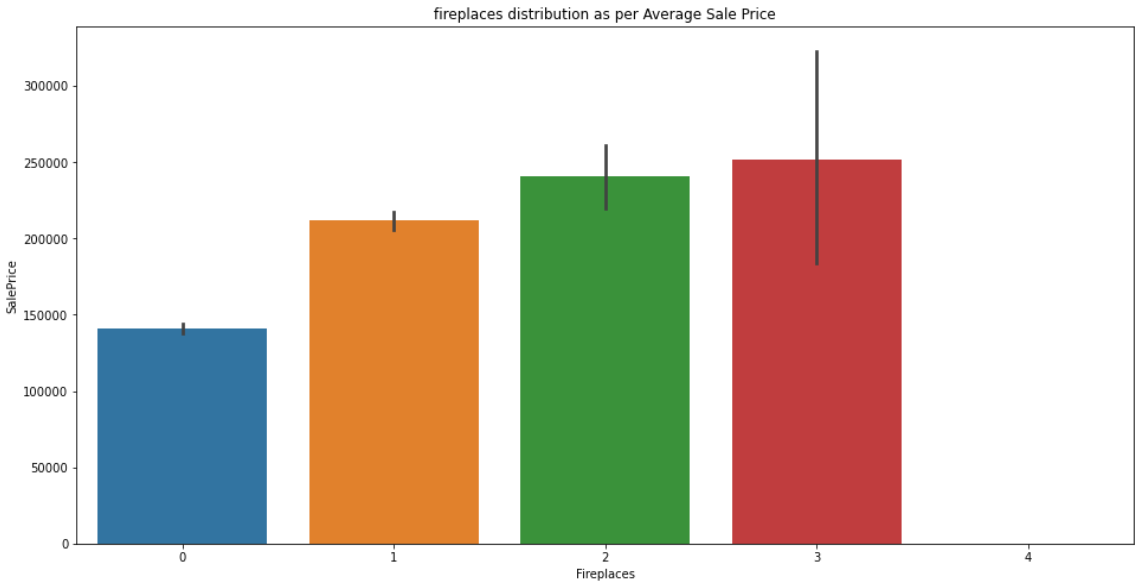
Most of the houses have zero fireplaces around 49%

In [373]:

```
plt.figure(figsize=(16,8))
plt.title("fireplaces distribution as per Average Sale Price")
sns.barplot(df['Fireplaces'],df['SalePrice'])
```

Out[373]:

```
<AxesSubplot:title={'center':'fireplaces distribution as per Average Sale
Price'}, xlabel='Fireplaces', ylabel='SalePrice'>
```



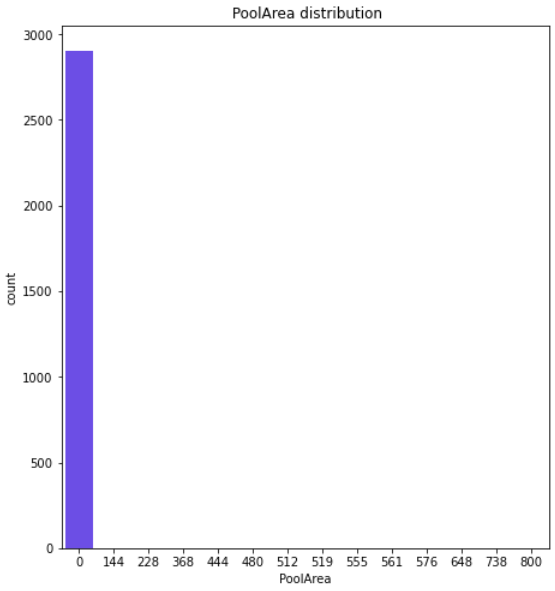
Observation :

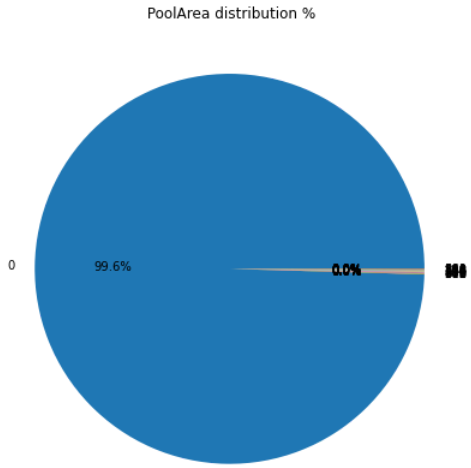
the house which have 2 firplaces have high Sale price

Visualizing Pool Area Feature

In [374]:

```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.title(" PoolArea distribution")
sns.countplot(df['PoolArea'],palette='rainbow',data=df)

plt.subplot(1,2,2)
plt.title(" PoolArea distribution %")
plt.pie(df['PoolArea'].value_counts(),labels=df['PoolArea'].unique(),autopct='%0.1f%%',d
```





Observation:

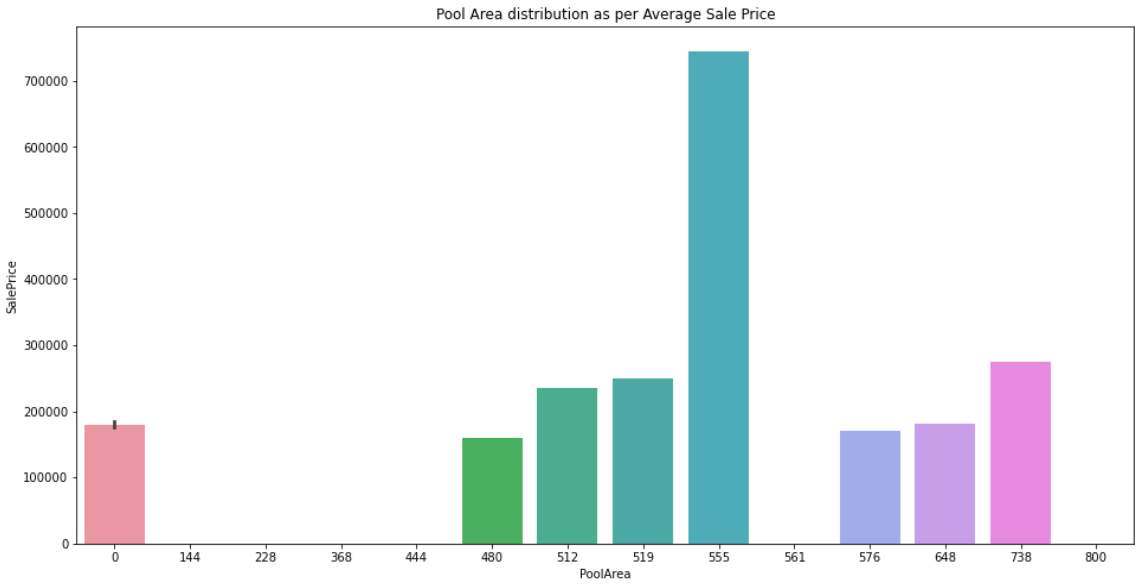
Most of the houses have no pool area

In [375]:

```
plt.figure(figsize=(16,8))
plt.title("Pool Area distribution as per Average Sale Price")
sns.barplot(df['PoolArea'],df['SalePrice'])
```

Out[375]:

```
<AxesSubplot:title={'center':'Pool Area distribution as per Average Sale P
rice'}, xlabel='PoolArea', ylabel='SalePrice'>
```



Observation:

Average price is high of pool area with 555.

Visualizing Categorical Features:

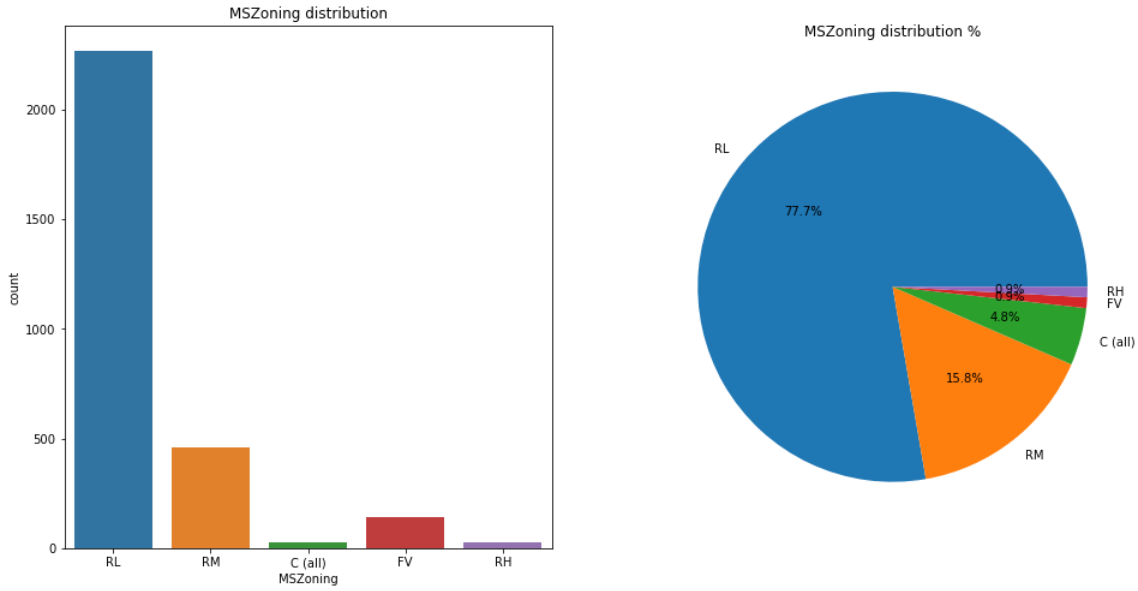
Visualizing MSZoning feature:

MSZoning: Identifies the general zoning classification of the sale.

A Agriculture
C Commercial
FV Floating Village Residential
I Industrial
RH Residential High Density
RL Residential Low Density
RP Residential Low Density Park
RM Residential Medium Density

In [376]:

```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.title("MSZoning distribution")
sns.countplot(df['MSZoning'])

plt.subplot(1,2,2)
plt.title("MSZoning distribution %")
plt.pie(df['MSZoning'].value_counts(),labels=df['MSZoning'].unique(),autopct='%0.1f%%',d
```

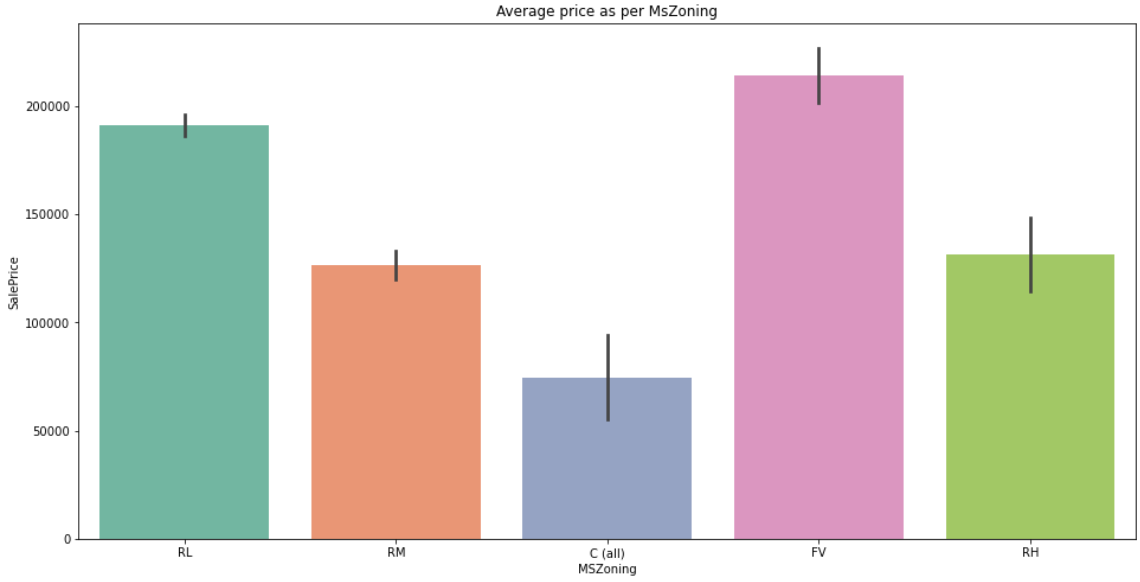


Observation :

Most of the house have RL (Residential Low Density) zoning area around 78%

In [377]:

```
plt.figure(figsize=(16,8))
plt.title("Average price as per MsZoning")
sns.barplot(df['MSZoning'],df['SalePrice'],palette='Set2',data=df);
```



observation :

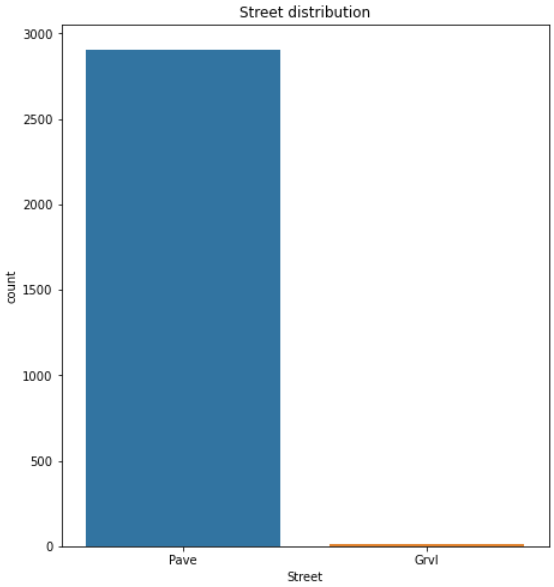
```
sale price of the FV (Floating Village Residential) zoining is high
```

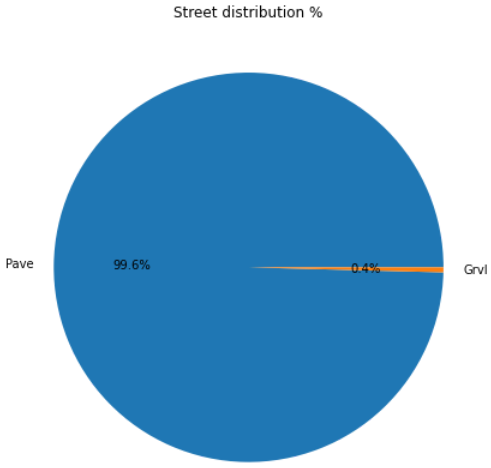
Visualizing Street feature:

```
Type of road access to property

Grvl    Gravel
Pave    Paved
```

In [378]:

```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.title("Street distribution")
sns.countplot(df['Street'])

plt.subplot(1,2,2)
plt.title("Street distribution %")
plt.pie(df['Street'].value_counts(),labels=df['Street'].unique(),autopct='%0.1f%%',data=
```





Observation:

```
most of the houses has paved streets around 99.6%
```

In [379]:

```
plt.figure(figsize=(16,8))
plt.title("Average price as per MsZoning")
sns.barplot(df['Street'],df['SalePrice'],palette=['red','green'],data=df)
```

Out[379]:

```
<AxesSubplot:title={'center':'Average price as per MsZoning'}, xlabel='Str
eet', ylabel='SalePrice'>
```

Observation:

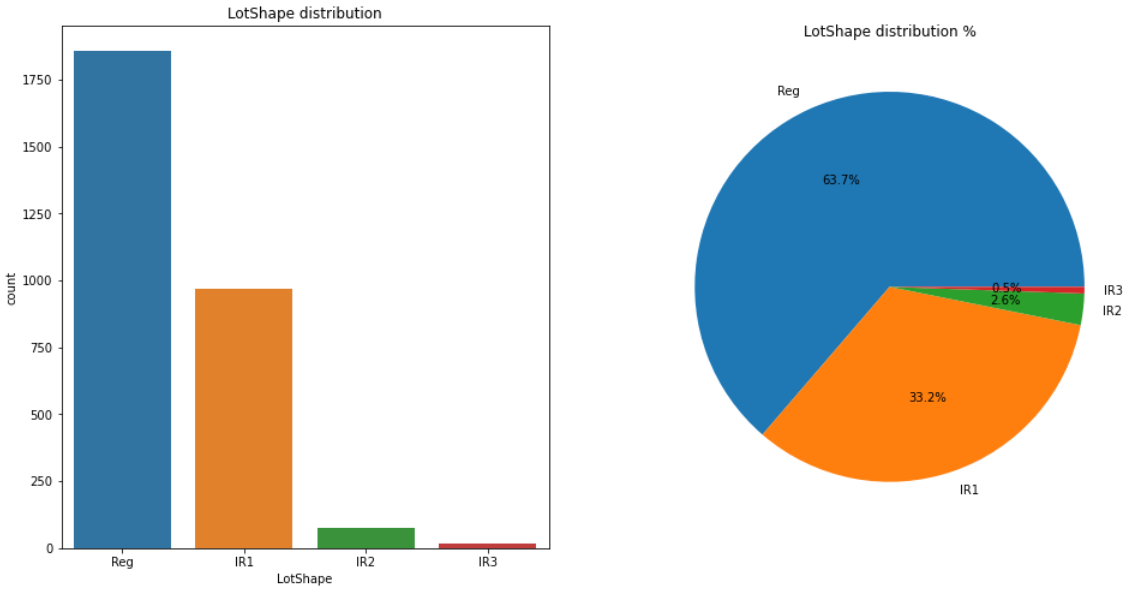
Average price for paved house is high

Visualizing LotShape: General shape of property

Reg	Regular
IR1	Slightly irregular
IR2	Moderately Irregular
IR3	Irregular

In [380]:

```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.title("LotShape distribution")
sns.countplot(df['LotShape'])

plt.subplot(1,2,2)
plt.title("LotShape distribution %")
plt.pie(df['LotShape'].value_counts(),labels=df['LotShape'].unique(),autopct='%0.1f%%',d
```



Observation :

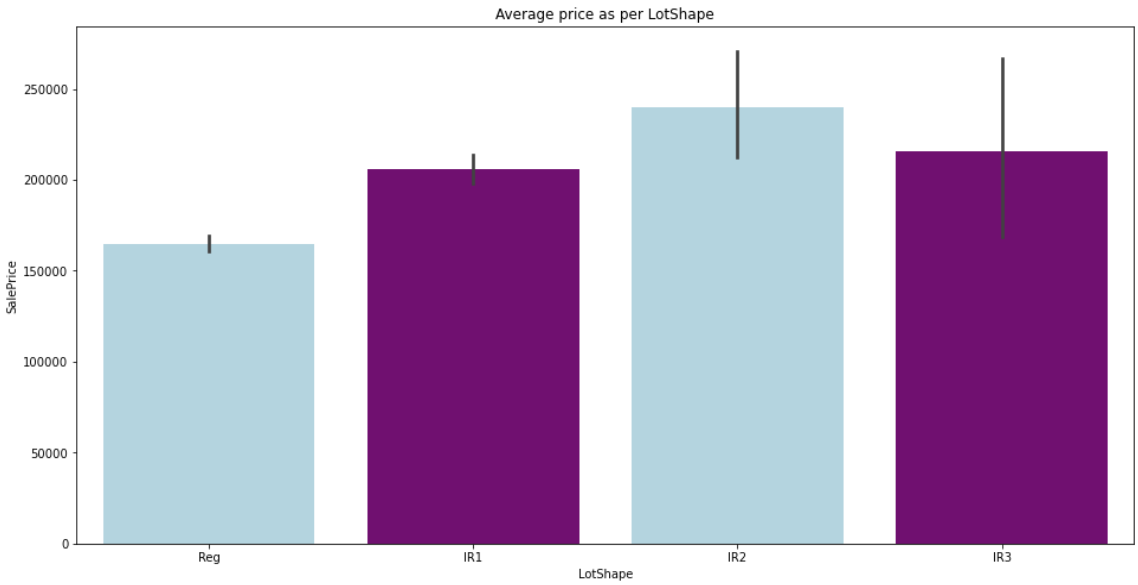
Most of the houses have regular shape of property around 63.7 %
and demand for IR1 Slightly irregular shape of property is also high around 33.
2%

In [381]:

```
plt.figure(figsize=(16,8))
plt.title("Average price as per LotShape")
sns.barplot(df['LotShape'],df['SalePrice'],palette=['lightblue','purple'],data=df)
```

Out[381]:

```
<AxesSubplot:title={'center':'Average price as per LotShape'}, xlabel='Lot
Shape', ylabel='SalePrice'>
```



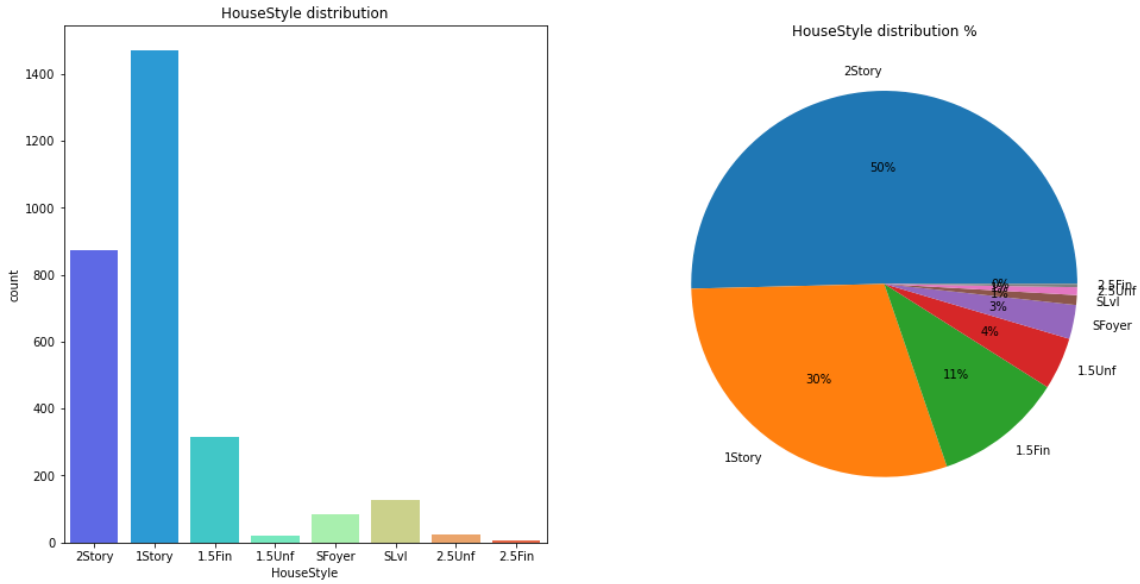
observation

The sale price of ir3 is high among all others.

visualizing HouseStyle Feature:

In [382]:

```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.title("HouseStyle distribution")
sns.countplot(df['HouseStyle'],palette='rainbow')

plt.subplot(1,2,2)
plt.title("HouseStyle distribution %")
plt.pie(df['HouseStyle'].value_counts(),labels=df['HouseStyle'].unique(),autopct='%0.0f%
```

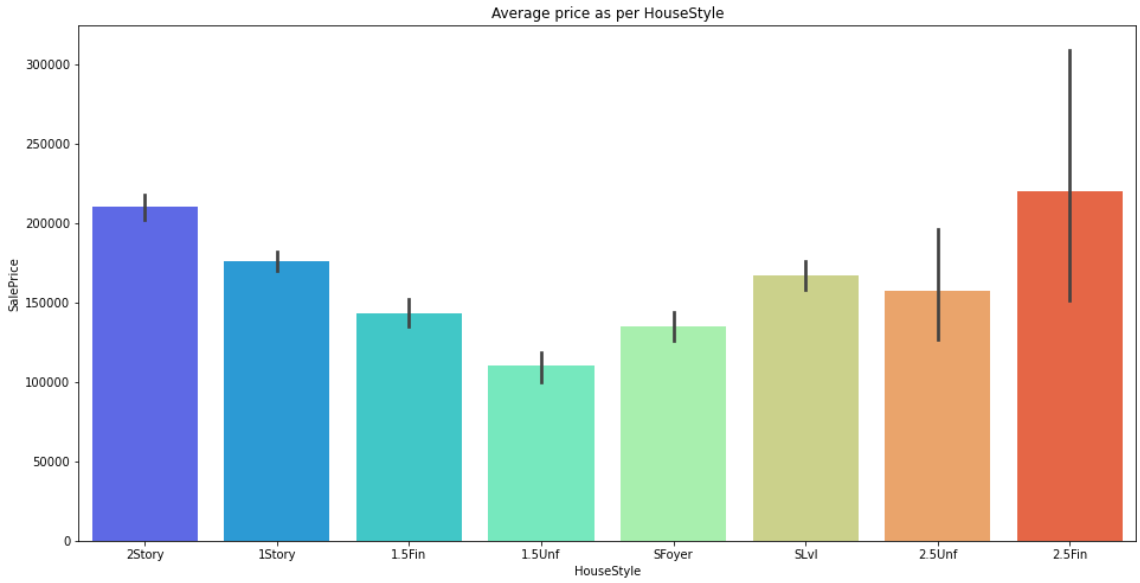


Observation :

Most of the houses are 1 story around 50%

In [383]:

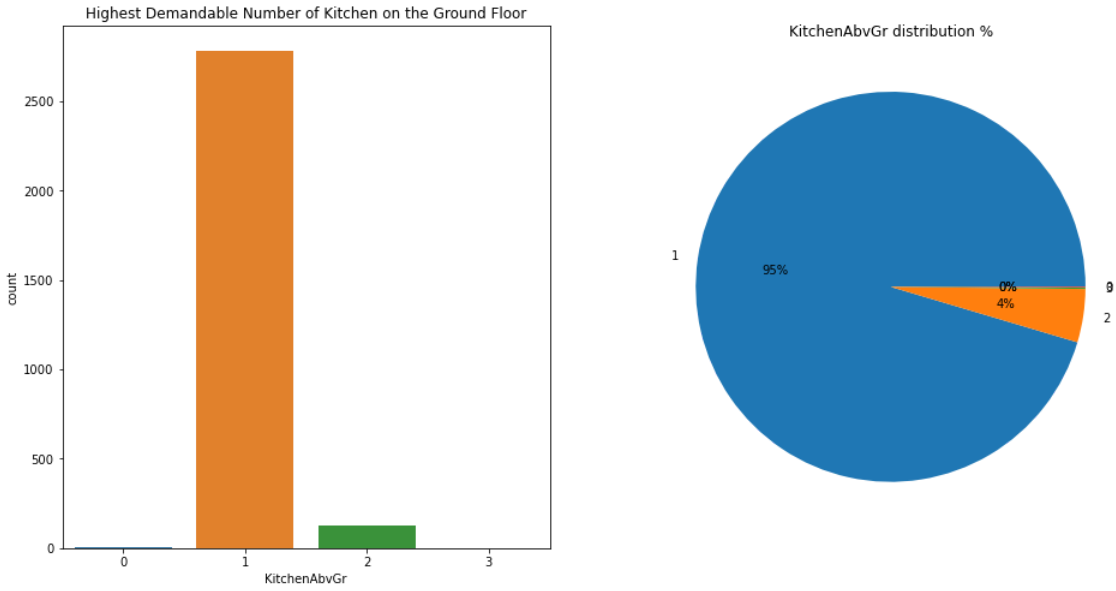
```
plt.figure(figsize=(16,8))
plt.title("Average price as per HouseStyle")
sns.barplot(df['HouseStyle'],df['SalePrice'],palette='rainbow',data=df);
```



Visualizing KitchenAbvGr Feature:

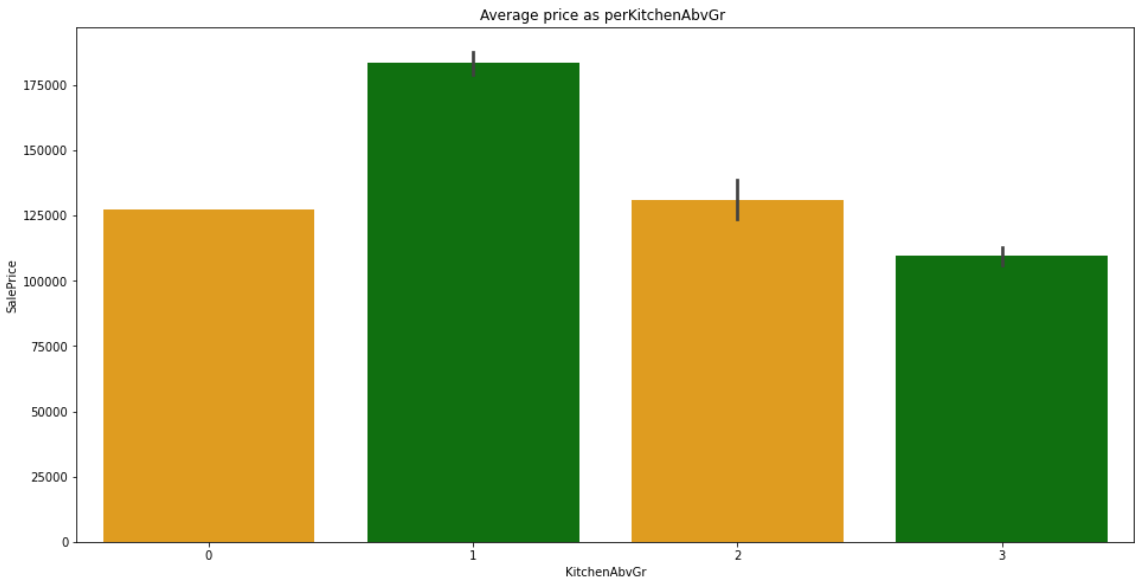
In [384]:

```
plt.figure(figsize=(16,8));
plt.subplot(1,2,1)
sns.countplot(x="KitchenAbvGr",data=df);
plt.title("Highest Demandable Number of Kitchen on the Ground Floor");

plt.subplot(1,2,2)
plt.title("KitchenAbvGr distribution %")
plt.pie(df['KitchenAbvGr'].value_counts(),labels=df['KitchenAbvGr'].unique(),autopct='%0
```



In [385]:

```
plt.figure(figsize=(16,8))
plt.title("Average price as perKitchenAbvGr ")
sns.barplot(df['KitchenAbvGr'],df['SalePrice'],palette=['orange','green'],data=df);
```



Observation: We can see the highest demandable kitchen on th ground floor is 1 and we can also see the averge price of the above chart.

In [386]:

```
df.head()
```

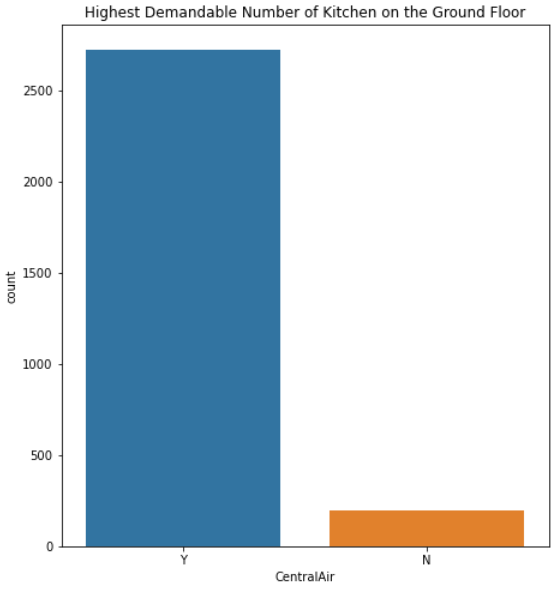
Out[386]:

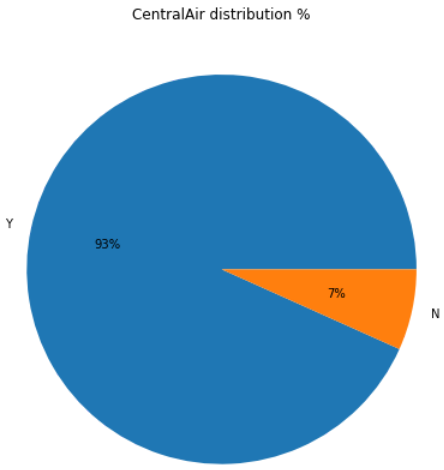
	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd
0	60	65.0	8450	7	5	2003	2003
1	20	80.0	9600	6	8	1976	1976
2	60	68.0	11250	7	5	2001	2002
3	70	60.0	9550	7	5	1915	1970
4	60	84.0	14260	8	5	2000	2000

Visualizing Central Air:

In [387]:

```
plt.figure(figsize=(16,8));
plt.subplot(1,2,1)
sns.countplot(x="CentralAir",data=df);
plt.title("Highest Demandable Number of Kitchen on the Ground Floor");

plt.subplot(1,2,2)
plt.title("CentralAir distribution %")
plt.pie(df['CentralAir'].value_counts(),labels=df['CentralAir'].unique(),autopct='%0.0f%
```

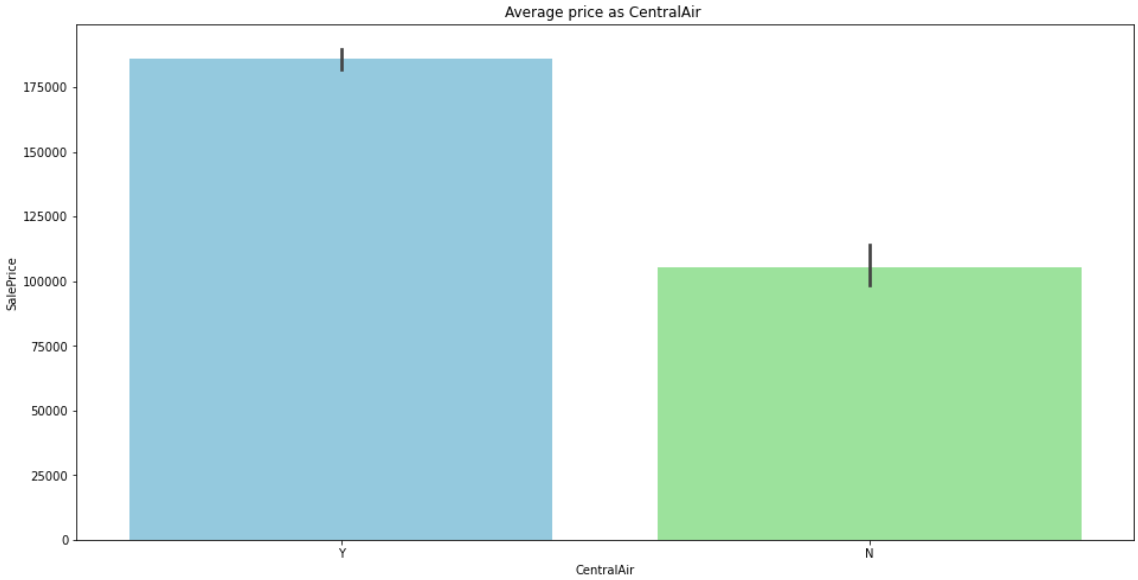




Observations : 93 % of houses have central air condition

In [388]:

```
plt.figure(figsize=(16,8))
plt.title("Average price as CentralAir ")
sns.barplot(df['CentralAir'],df['SalePrice'],palette=['skyblue','lightgreen'],data=df);
```



observation

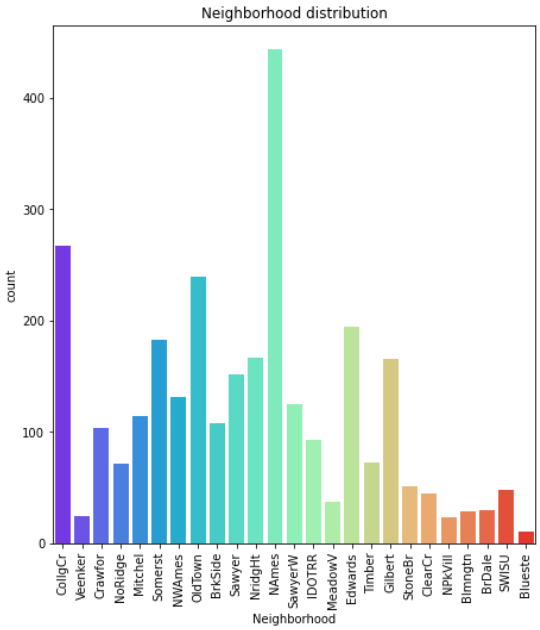
houses with central air condition having a high sales price

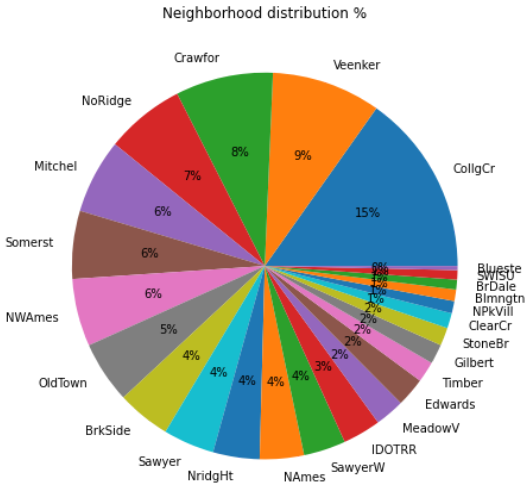
Visualizing Neighborhood feature:

```
Neighborhood: Physical locations within Ames city limits

   Blmngtn    Bloomington Heights
   Blueste    Bluestem
```

In [389]:

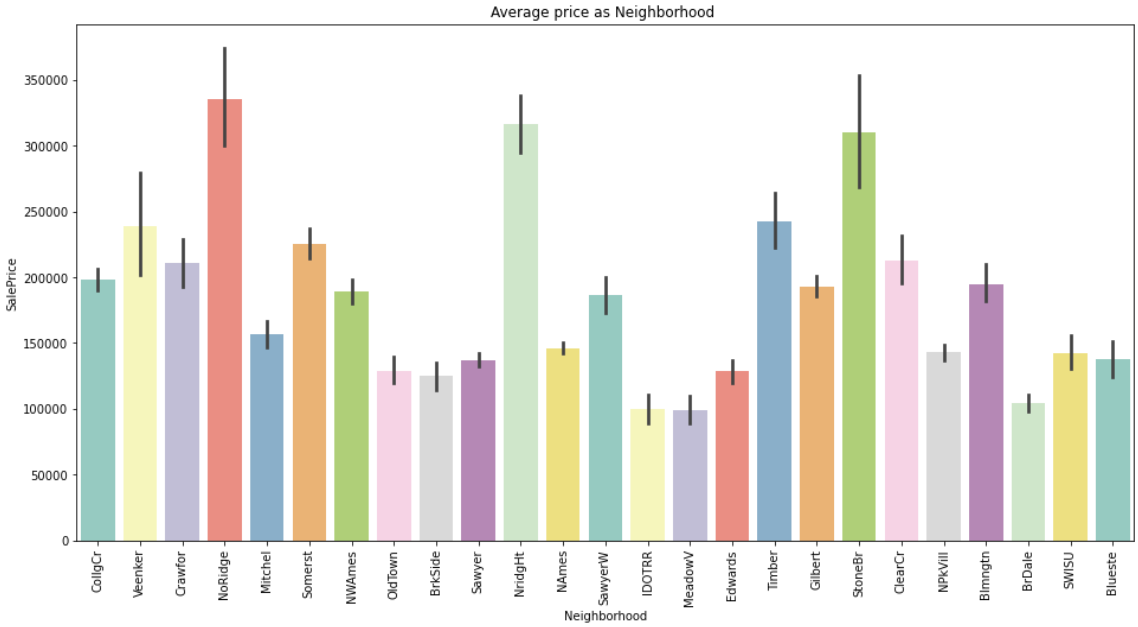
```
plt.figure(figsize=(16,8));
plt.subplot(1,2,1)
sns.countplot(x="Neighborhood",palette='rainbow',data=df);
plt.xticks(rotation=90)
plt.title("Neighborhood distribution");

plt.subplot(1,2,2)
plt.title("Neighborhood distribution %")
plt.pie(df['Neighborhood'].value_counts(),labels=df['Neighborhood'].unique(),autopct='%0
```





In [390]:

```
plt.figure(figsize=(16,8))
plt.title("Average price as Neighborhood ")
sns.barplot(df['Neighborhood'],df['SalePrice'],palette='Set3',data=df)
plt.xticks(rotation=90);
```

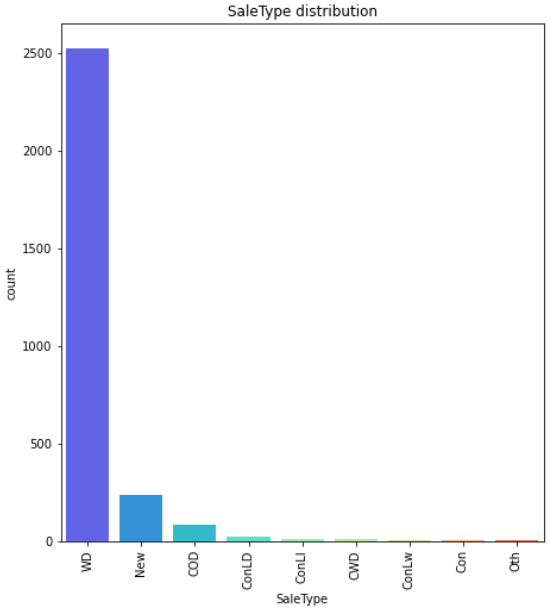


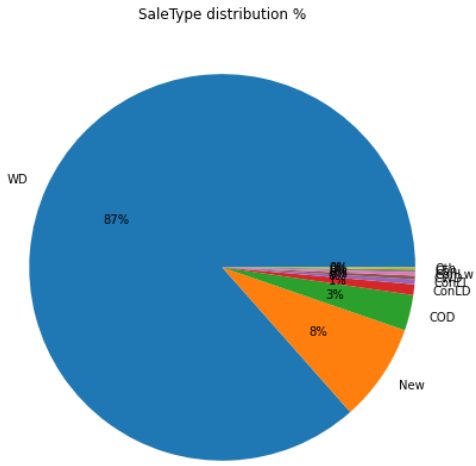
Visualizing SaleType: Type of sale

```
WD     Warranty Deed - Conventional
CWD    Warranty Deed - Cash
VWD    Warranty Deed - VA Loan
New    Home just constructed and sold
COD    Court Officer Deed/Estate
Con    Contract 15% Down payment regular terms
ConLw    Contract Low Down payment and low interest
ConLI    Contract Low Interest
ConLD    Contract Low Down
Oth    Other
```

In [391]:

```
plt.figure(figsize=(16,8));
plt.subplot(1,2,1)
sns.countplot(x="SaleType",palette='rainbow',data=df);
plt.xticks(rotation=90)
plt.title("SaleType distribution");

plt.subplot(1,2,2)
plt.title("SaleType distribution %")
plt.pie(df['SaleType'].value_counts(),labels=df['SaleType'].unique(),autopct='%0.0f%%',d
```



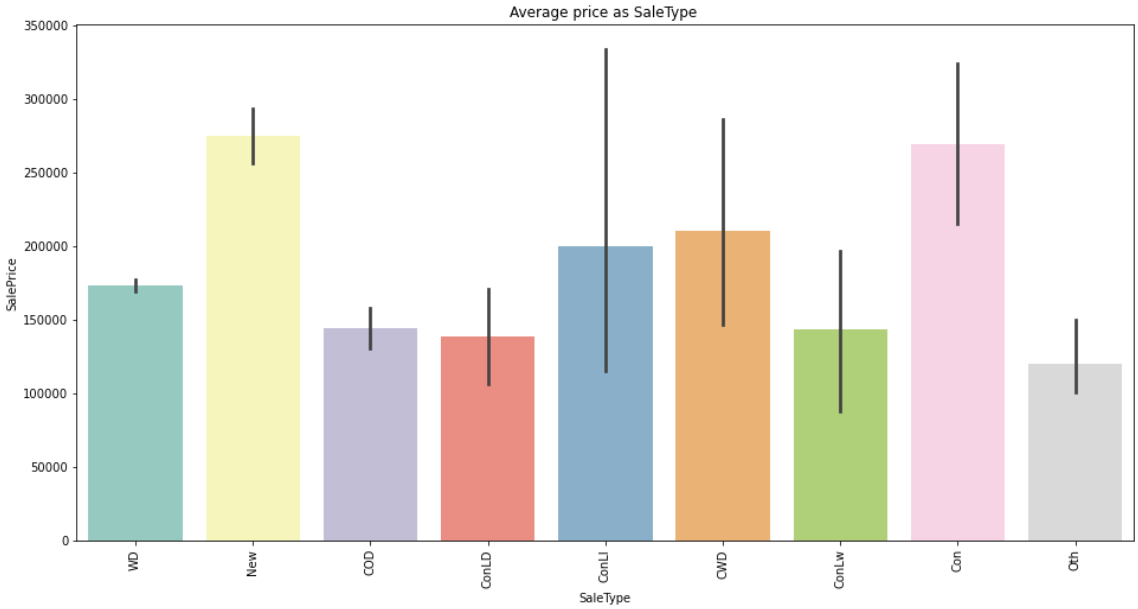


observation:

Most of the houses deals was Warranty Deed - Conventional around 87%.

In [392]:

```
plt.figure(figsize=(16,8))
plt.title("Average price as SaleType ")
sns.barplot(df['SaleType'],df['SalePrice'],palette='Set3',data=df)
plt.xticks(rotation=90);
```

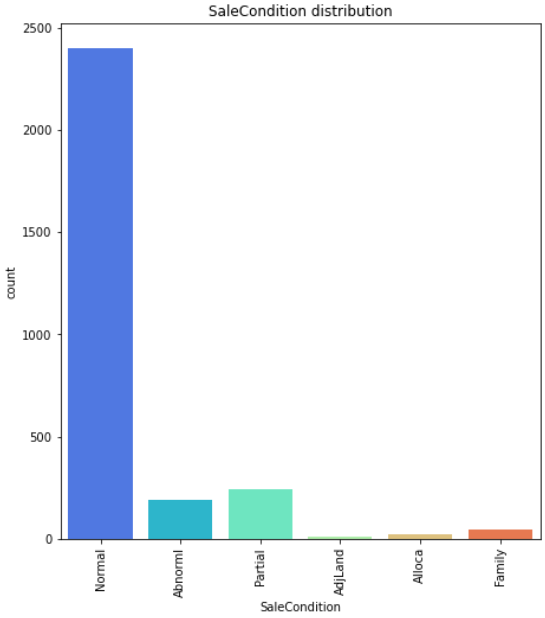


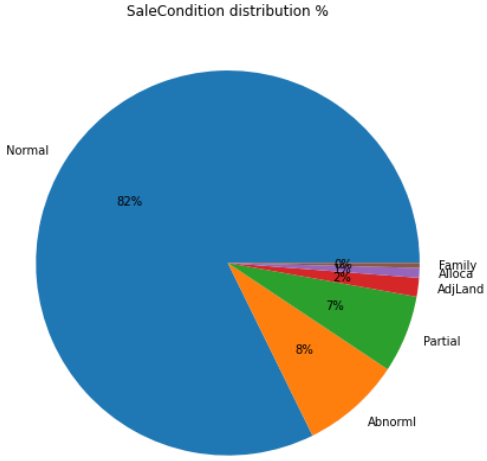
Visualizing SaleCondition Condition of sale

```
   Normal    Normal Sale
   Abnorml    Abnormal Sale -  trade, foreclosure, short sale
   AdjLand    Adjoining Land Purchase
   Alloca    Allocation - two linked properties with separate deeds, typically
condo with a garage unit
   Family    Sale between family members
   Partial    Home was not completed when last assessed (associated with New Ho
mes)
```

In [393]:

```
plt.figure(figsize=(16,8));
plt.subplot(1,2,1)
sns.countplot(x="SaleCondition",palette='rainbow',data=df);
plt.xticks(rotation=90)
plt.title("SaleCondition distribution");

plt.subplot(1,2,2)
plt.title("SaleCondition distribution %")
plt.pie(df['SaleCondition'].value_counts(),labels=df['SaleCondition'].unique(),autopct='
```



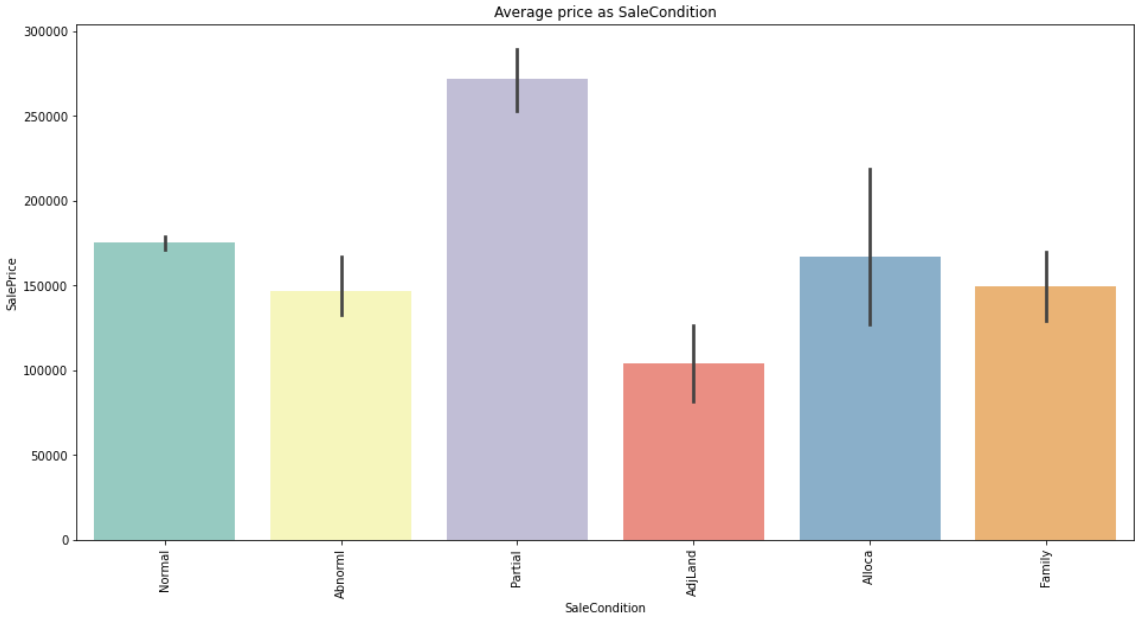


observation:

most of the houses sales condition is normal around 82%

In [394]:

```
plt.figure(figsize=(16,8))
plt.title("Average price as SaleCondition")
sns.barplot(df['SaleCondition'],df['SalePrice'],palette='Set3',data=df)
plt.xticks(rotation=90);
```



observation:

average price for partial sales condition is high

DATA PREPROCESSING

In [395]:

```
a=df.select_dtypes(include=object)
```

In [396]:

```
from sklearn import preprocessing
```

In [397]:

```
label_encoder = preprocessing.LabelEncoder()
```

In [398]:

```
df['MSZoning']= label_encoder.fit_transform(df['MSZoning'])
```

In [399]:

```
df['Street']= label_encoder.fit_transform(df['Street'])
```

In [400]:

```
df['LotShape']= label_encoder.fit_transform(df['LotShape'])
```

In [401]:

```
df['Utilities']= label_encoder.fit_transform(df['Utilities'])
```

In [402]:

```
df['Neighborhood']= label_encoder.fit_transform(df['Neighborhood'])
```

In [403]:

```
df['Condition1']= label_encoder.fit_transform(df['Condition1'])
```

In [404]:

```
df['HouseStyle']= label_encoder.fit_transform(df['HouseStyle'])
```

In [405]:

```
df['RoofStyle']= label_encoder.fit_transform(df['RoofStyle'])
```

In [406]:

```
df['RoofMatl']= label_encoder.fit_transform(df['RoofMatl'])
```

In [407]:

```
df['Exterior2nd']= label_encoder.fit_transform(df['Exterior2nd'])
```

In [408]:

```
df['ExterQual']= label_encoder.fit_transform(df['ExterQual'])
```

In [409]:

```
df['ExterCond']= label_encoder.fit_transform(df['ExterCond'])
```

In [410]:

```
df['Foundation']= label_encoder.fit_transform(df['Foundation'])
```

In [411]:

```
df['BsmtQual']= label_encoder.fit_transform(df['BsmtQual'])
```

In [412]:

```
df['HeatingQC']= label_encoder.fit_transform(df['HeatingQC'])
```

In [413]:

```
df['CentralAir']= label_encoder.fit_transform(df['CentralAir'])
```

In [414]:

```
df['Electrical']= label_encoder.fit_transform(df['Electrical'])
```

In [415]:

```
df['KitchenQual']= label_encoder.fit_transform(df['KitchenQual'])
```

In [416]:

```
df['Functional']= label_encoder.fit_transform(df['Functional'])
```

In [417]:

```
df['GarageType']= label_encoder.fit_transform(df['GarageType'])
```

In [418]:

```
df['GarageCond']= label_encoder.fit_transform(df['GarageCond'])
```

In [419]:

```
df['PavedDrive']= label_encoder.fit_transform(df['PavedDrive'])
```

In [420]:

```
df['SaleType']= label_encoder.fit_transform(df['SaleType'])
```

In [421]:

```
df['SaleCondition']= label_encoder.fit_transform(df['SaleCondition'])
```

In [422]:

```
df['LandContour']= label_encoder.fit_transform(df['LandContour'])
```

In [423]:

```
df.tail(10)
```

Out[423]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAd
1449	180	21.0	1470	4	6	1970	197
1450	160	21.0	1484	4	4	1972	197
1451	20	80.0	13384	5	5	1969	197
1452	160	21.0	1533	4	5	1970	197
1453	160	21.0	1526	4	5	1970	197
1454	160	21.0	1936	4	7	1970	197
1455	160	21.0	1894	4	5	1970	197
1456	20	160.0	20000	5	7	1960	199
1457	85	62.0	10441	5	5	1992	199
1458	60	74.0	9627	7	5	1993	199

In [424]:

```
# segregating train and test data
```

In [427]:

```
df.isnull().sum()
```

Out[427]:

```
MSSubClass          0
LotFrontage         0
LotArea             0
OverallQual         0
OverallCond         0
YearBuilt           0
YearRemodAdd        0
MasVnrArea          0
TotalBsmtSF         0
1stFlrSF            0
2ndFlrSF            0
GrLivArea           0
FullBath            0
HalfBath            0
BedroomAbvGr        0
KitchenAbvGr        0
TotRmsAbvGrd        0
Fireplaces          0
GarageCars          0
GarageArea          0
PoolArea            0
YrSold              0
SalePrice        1459
MSZoning            0
Street              0
LotShape            0
LandContour         0
Utilities           0
Neighborhood        0
Condition1          0
HouseStyle          0
RoofStyle           0
RoofMatl            0
Exterior2nd         0
ExterQual           0
ExterCond           0
Foundation          0
BsmtQual            0
HeatingQC           0
CentralAir          0
Electrical          0
KitchenQual         0
Functional          0
GarageType          0
GarageCond          0
PavedDrive          0
SaleType            0
SaleCondition       0
dtype: int64
```

In [428]:

```
train1=df[~df['SalePrice'].isnull()]
```

In [429]:

```
train1.isnull().sum()
```

Out[429]:

```
MSSubClass       0
LotFrontage      0
LotArea          0
OverallQual      0
OverallCond      0
YearBuilt        0
YearRemodAdd     0
MasVnrArea       0
TotalBsmtSF      0
1stFlrSF         0
2ndFlrSF         0
GrLivArea        0
FullBath         0
HalfBath         0
BedroomAbvGr     0
KitchenAbvGr     0
TotRmsAbvGrd     0
Fireplaces       0
```

In [430]:

```
test1=df[df['SalePrice'].isnull()]
```

In [432]:

```
test1.isnull().sum()
```

Out[432]:

```
MSSubClass          0
LotFrontage         0
LotArea             0
OverallQual         0
OverallCond         0
YearBuilt           0
YearRemodAdd        0
MasVnrArea          0
TotalBsmtSF         0
1stFlrSF            0
2ndFlrSF            0
GrLivArea           0
FullBath            0
HalfBath            0
BedroomAbvGr        0
KitchenAbvGr        0
TotRmsAbvGrd        0
Fireplaces          0
```

In [434]:

```
test1.head()
```

Out[434]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd
0	20	80.0	11622	5	6	1961	1961
1	20	81.0	14267	6	6	1958	1958
2	60	74.0	13830	5	5	1997	1998
3	60	78.0	9978	6	6	1998	1998
4	120	43.0	5005	8	5	1992	1992

In [435]:

```
train1.shape
```

Out[435]:

```
(1460, 48)
```

In [436]:

```
test1.shape
```

Out[436]:

```
(1459, 48)
```

In [437]:

```
test1.drop(columns=['SalePrice'],inplace=True)
```

In [438]:

```
train1.isnull().sum()
```

Out[438]:

```
MSSubClass      0
LotFrontage     0
LotArea         0
OverallQual     0
OverallCond     0
YearBuilt       0
YearRemodAdd    0
MasVnrArea      0
TotalBsmtSF     0
1stFlrSF        0
2ndFlrSF        0
GrLivArea       0
FullBath        0
HalfBath        0
BedroomAbvGr    0
KitchenAbvGr    0
TotRmsAbvGrd    0
Fireplaces      0
```

In [439]:

```
test1.isnull().sum()
```

Out[439]:

```
MSSubClass      0
LotFrontage     0
LotArea         0
OverallQual     0
OverallCond     0
YearBuilt       0
YearRemodAdd    0
MasVnrArea      0
TotalBsmtSF     0
1stFlrSF        0
2ndFlrSF        0
GrLivArea       0
FullBath        0
HalfBath        0
BedroomAbvGr    0
KitchenAbvGr    0
TotRmsAbvGrd    0
Fireplaces      0
```

Segregate the features to train model

In [452]:

```
x = train1.drop(columns=['SalePrice'])
y = train1['SalePrice']
```

In [453]:

```
x.isnull().sum()
```

Out[453]:

```
MSSubClass       0
LotFrontage      0
LotArea          0
OverallQual      0
OverallCond      0
YearBuilt        0
YearRemodAdd     0
MasVnrArea       0
TotalBsmtSF      0
1stFlrSF         0
2ndFlrSF         0
GrLivArea        0
FullBath         0
HalfBath         0
BedroomAbvGr     0
KitchenAbvGr     0
TotRmsAbvGrd     0
Fireplaces       0
```

In [454]:

```
y
```

Out[454]:

```
0       208500.0
1       181500.0
2       223500.0
3       140000.0
4       250000.0
5       143000.0
6       307000.0
7       200000.0
8       129900.0
9       118000.0
10      129500.0
11      345000.0
12      144000.0
13      279500.0
14      157000.0
15      132000.0
16      149000.0
17       90000.0
```

In [455]:

```
x.head()
```

Out[455]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd
0	60	65.0	8450	7	5	2003	2003
1	20	80.0	9600	6	8	1976	1976
2	60	68.0	11250	7	5	2001	2002
3	70	60.0	9550	7	5	1915	1970
4	60	84.0	14260	8	5	2000	2000

In [456]:

```
y.head()
```

Out[456]:

```
0    208500.0
1    181500.0
2    223500.0
3    140000.0
4    250000.0
Name: SalePrice, dtype: float64
```

Splitting of x and y

In [457]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2, random_state=42)
```

In [458]:

```
print("The shape of x_train :",x_train.shape)
print("The shape of x_test :",x_test.shape)
print("The shape of y_train :",y_train.shape)
print("The shape of y_test :",y_test.shape)
```

```
The shape of x_train : (1168, 47)
The shape of x_test : (292, 47)
The shape of y_train : (1168,)
The shape of y_test : (292,)
```

Model Building

In [485]:

```
Testing_Score=[]
Training_Score=[]
```

In [486]:

```
def model(model):

    model.fit(x_train,y_train)

    y_pred = model.predict(x_test)
    y_pred_train = model.predict(x_train)

    a=r2_score(y_pred,y_test)
    b=r2_score(y_pred_train,y_train)

    Testing_Score.append(a)
    Training_Score.append(b)



    print("The r2_score of test data is:",a)
    print("The r2_score of train data is:",b)
```

In [487]:

```
model(DecisionTreeRegressor())
```

```
The r2_score of test data is: 0.7556348330507344
The r2_score of train data is: 0.999995055666207
```

In [488]:

```
model(RandomForestRegressor())
```

```
The r2_score of test data is: 0.8549257936547187
The r2_score of train data is: 0.9766716823966551
```

In [489]:

```
model(AdaBoostRegressor())
```

```
The r2_score of test data is: 0.7557818085303312
The r2_score of train data is: 0.8288797421222323
```

In [490]:

```
model(GradientBoostingRegressor())
```

```
The r2_score of test data is: 0.8770927644516692
The r2_score of train data is: 0.9555139230300828
```

In [491]:

```
model(CatBoostRegressor(verbose=False))
```

```
The r2_score of test data is: 0.8771688936856052
The r2_score of train data is: 0.9928619491452245
```

In [492]:

```
model(XGBRegressor())
```

```
The r2_score of test data is: 0.8745147559850938
The r2_score of train data is: 0.999711905420369
```

In [493]:

```
model(LGBMRegressor())
```

```
The r2_score of test data is: 0.8534681403479736
The r2_score of train data is: 0.9739123359889572
```

In [495]:

```
print("Testing Score of all above algorithms:", Testing_Score)
```

```
Testing Score of all above algorithms: [0.7556348330507344, 0.854925793654
7187, 0.7557818085303312, 0.8770927644516692, 0.8771688936856052, 0.874514
7559850938, 0.8534681403479736]
```

In [496]:

```
print("Training score of all above algorithms:", Training_Score)
```

```
Training score of all above algorithms: [0.999995055666207, 0.976671682396
6551, 0.8288797421222323, 0.9555139230300828, 0.9928619491452245, 0.999711
905420369, 0.9739123359889572]
```

Creating a dataframe of the scores for better understanding

In [501]:

```
Models=['DecisionTreeRegressor','RandomForestRegressor','AdaBoostRegressor','GradientBoo
```

In [503]:

```
df= pd.DataFrame({'Algorithms':Models,"Training Score":Training_Score,"Testing Score":Te
```

In [504]:

```
df
```

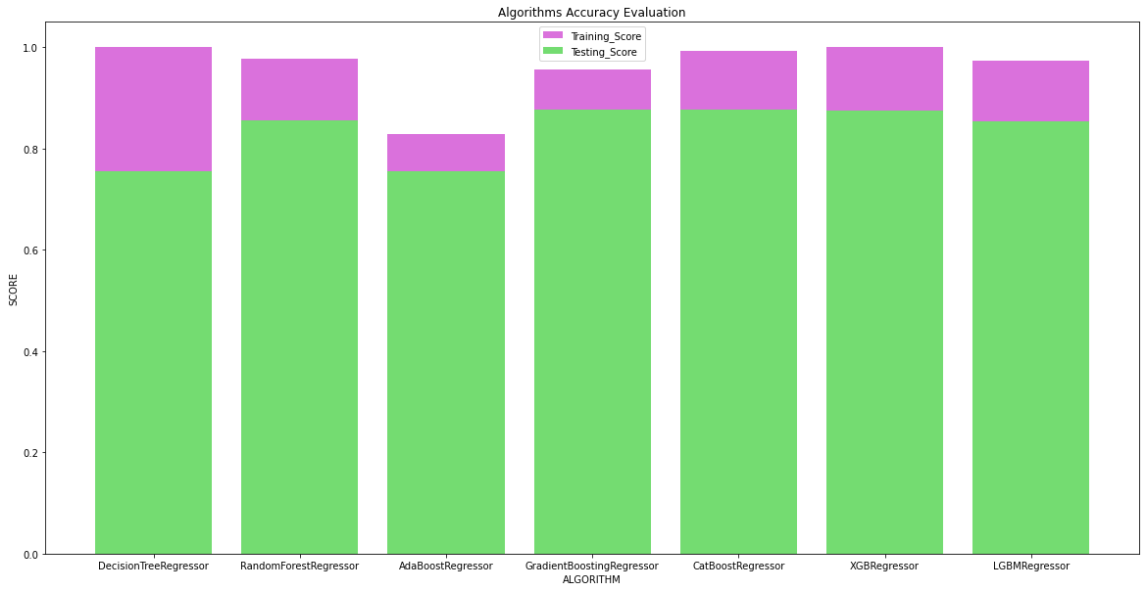
Out[504]:

	Algorithms	Training Score	Testing Score
0	DecisionTreeRegressor	0.999995	0.755635
1	RandomForestRegressor	0.976672	0.854926
2	AdaBoostRegressor	0.828880	0.755782
3	GradientBoostingRegressor	0.955514	0.877093
4	CatBoostRegressor	0.992862	0.877169
5	XGBRegressor	0.999712	0.874515
6	LGBMRegressor	0.973912	0.853468

Visualizing Scores:

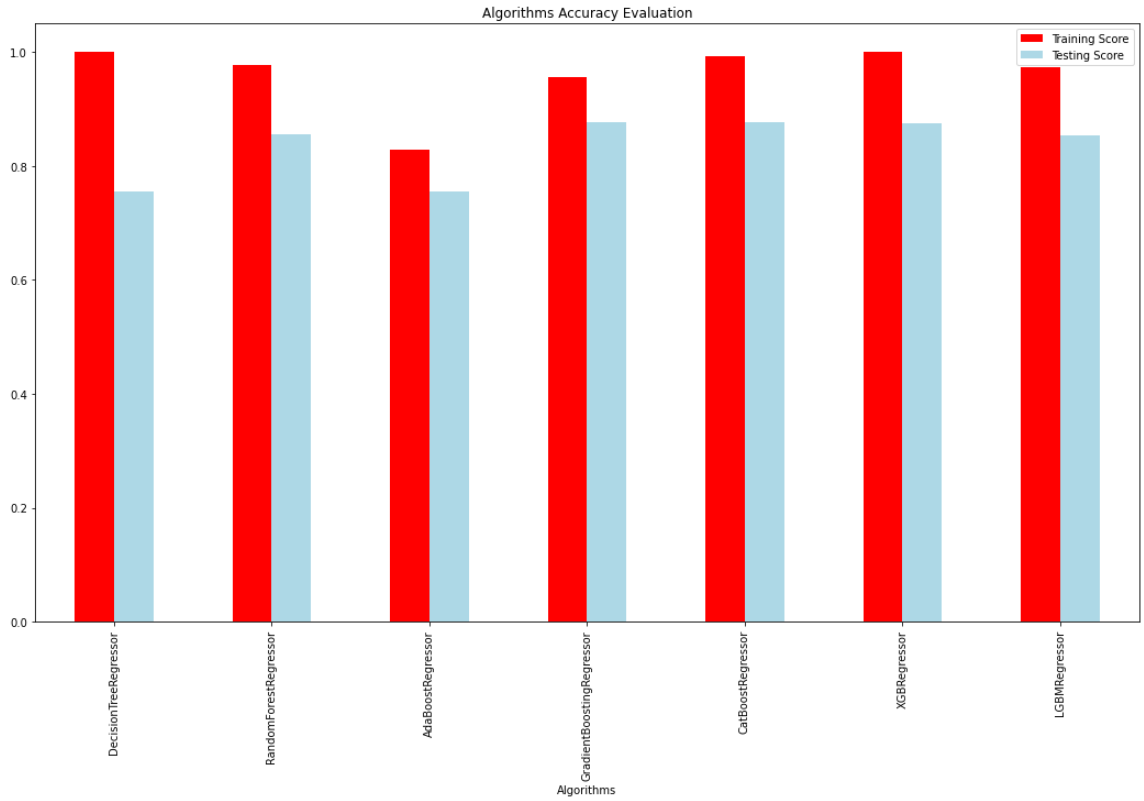
In [527]:

```
plt.figure(figsize=(20,10))
plt.title('Algorithms Accuracy Evaluation')
plt.xlabel("ALGORITHM")
plt.ylabel("SCORE")
plt.bar(df['Algorithms'],df['Training Score'],color='#DA71DC',label='Training_Score',dat
plt.bar(df['Algorithms'],df['Testing Score'],color='#74DC71',label='Testing_Score',data=
plt.legend();
```



In [528]:

```
df.plot(x='Algorithms',y=['Training Score','Testing Score'],kind="bar",color=['red','lig
```



As you can see CatBoostRegressor is having highest r2_score and having highest accuracy

Now We can perform HYPERPARAMETER TUNNING FOR GradientBoostingRegressor()

In [537]:

```
GBR=GradientBoostingRegressor()
```

In [538]:

```
grid_param = {
    'n_estimators': [10,50,100,500,200],
    'learning_rate' : [0.0001,0.01,0.001,1.0]}
```

In [539]:

```
grid_search = GridSearchCV(estimator=GBR,
                     param_grid=grid_param,
                    n_jobs =-1,cv=10,scoring="r2")
```

In [541]:

```
grid_search.fit(x_train,y_train)
```

Out[541]:

```
GridSearchCV(cv=10, estimator=GradientBoostingRegressor(), n_jobs=-1,
             param_grid={'learning_rate': [0.0001, 0.01, 0.001, 1.0],
                         'n_estimators': [10, 50, 100, 500, 200]},
             scoring='r2')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [544]:

```
print(" Results from Grid Search " )
print("\n The best estimator across ALL searched params:\n", grid_search.best_estimator_
print("\n The best score across ALL searched params:\n", grid_search.best_score_)
print("\n The best parameters across ALL searched params:\n", grid_search.best_params_)
```

```
 Results from Grid Search

 The best estimator across ALL searched params:
 GradientBoostingRegressor(learning_rate=0.01, n_estimators=500)

 The best score across ALL searched params:
 0.8368505173702785

 The best parameters across ALL searched params:
 {'learning_rate': 0.01, 'n_estimators': 500}
```

In [580]:

```
GBR= GradientBoostingRegressor(verbose=False)
```

In [581]:

```
GBR.fit(x_train,y_train)
```

Out[581]:

```
GradientBoostingRegressor(verbose=False)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [582]:

```
y_pred = GBR.predict(x_test)
y_pred_train = GBR.predict(x_train)
```

In [583]:

```
r2_score(y_pred,y_test)
```

Out[583]:

```
0.8723247055529731
```

In [584]:

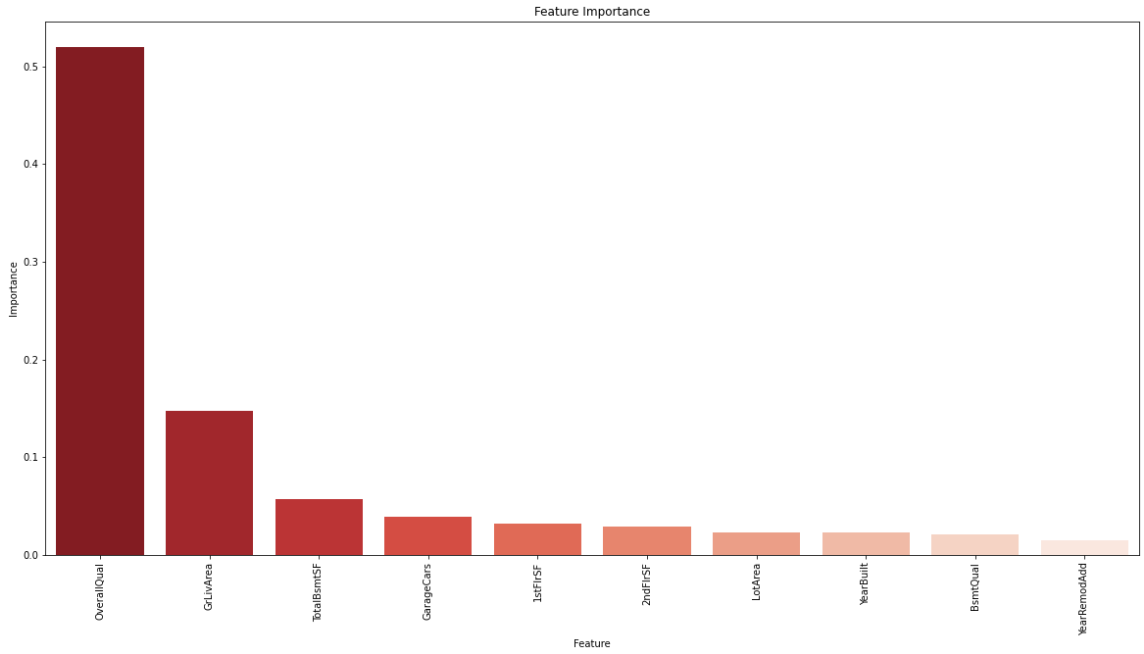
```
r2_score(y_pred_train,y_train)
```

Out[584]:

```
0.9555139230300828
```

In [587]:

```
# Calculate the feature importances
importances = GBR.feature_importances_

# Sort the feature importances in descending order and take the top 10
indices = np.argsort(importances)[::-1]
columns = x_train.columns.values[indices[:10]]
values = importances[indices][:10]

# Create a bar plot of the feature importances
plt.figure(figsize=(20, 10))
sns.barplot(x=columns, y=values, palette='Reds_r')
plt.title('Feature Importance')
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.xticks(rotation=90)
plt.show()
```



Now We can perform HYPERPARAMETER TUNNING FOR CatBoostRegressor()

In [579]:

```
CBR=CatBoostRegressor()
```

In [530]:

```
parameters = {'depth'         : [6,8,10],
                  'learning_rate' : [0.01, 0.05, 0.1],
                  'iterations'    : [30, 50, 100]
                 }
```

In [543]:

```
grid = GridSearchCV(estimator=CBR, param_grid = parameters, cv = 2, n_jobs=-1)
grid.fit(x_train, y_train)
```

In [534]:

```
print(" Results from Grid Search " )
print("\n The best estimator across ALL searched params:\n", grid.best_estimator_)
print("\n The best score across ALL searched params:\n", grid.best_score_)
print("\n The best parameters across ALL searched params:\n", grid.best_params_)
```

```
 Results from Grid Search

 The best estimator across ALL searched params:
 <catboost.core.CatBoostRegressor object at 0x0000021D0448B2B0>

 The best score across ALL searched params:
 0.8506150623211546

 The best parameters across ALL searched params:
 {'depth': 6, 'iterations': 100, 'learning_rate': 0.1}
```

Calculate the feature importances for CATBOOSTREGRESSOR() model

In [573]:

```
CBR= CatBoostRegressor(verbose=False)
```

In [574]:

```
CBR.fit(x_train,y_train)
```

Out[574]:

```
<catboost.core.CatBoostRegressor at 0x21d0a38ccd0>
```

In [575]:

```
y_pred = CBR.predict(x_test)
y_pred_train = CBR.predict(x_train)
```

In [576]:

```
r2_score(y_pred,y_test)
```

Out[576]:

```
0.8771688936856052
```

In [577]:

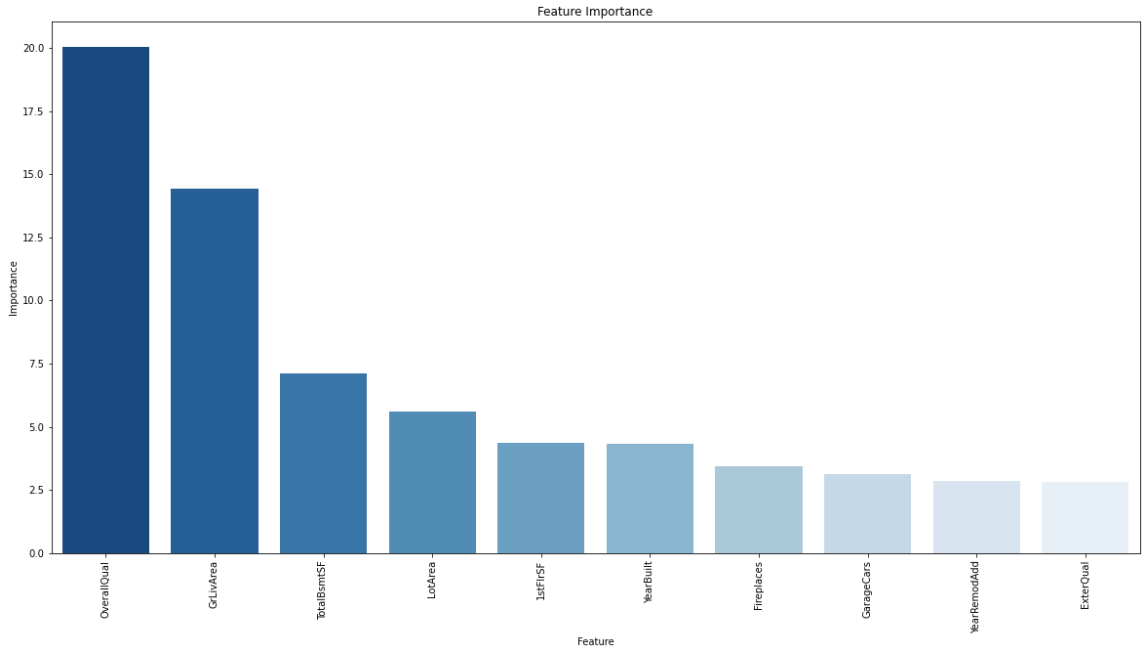
```
r2_score(y_pred_train,y_train)
```

Out[577]:

```
0.9928619491452245
```

In [578]:

```
# Calculate the feature importances
importances = CBR.feature_importances_

# Sort the feature importances in descending order and take the top 10
indices = np.argsort(importances)[::-1]
columns = x_train.columns.values[indices[:10]]
values = importances[indices][:10]

# Create a bar plot of the feature importances
plt.figure(figsize=(20, 10))
sns.barplot(x=columns, y=values, palette='Blues_r')
plt.title('Feature Importance')
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.xticks(rotation=90)
plt.show()
```



CONCLUSION

📝 Key Findings:

Features like `OverallQual` , `GrLivArea` ,'TotalBsmtSF', and `GarageArea` are having strong relation with the target variable.

The best performing model is CatBoostRegressor with highest R2 & Adjusted_R2 Scores and lowest MAE,MSE,RMSE values.

The second & third best performing model is GradientBoostingRegressor & LGBMRegressor models.

The project developed a house price prediction model with strong performance metrics.

The project effectively addresses the task of house price prediction and contributes as a valuable tool in the dynamic real estate industry.