California Housing Price Prediction

In [1]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from scipy.special import boxcox1p
from scipy.stats import boxcox_normmax
from scipy.stats import boxcox
import math
from scipy import stats
from scipy.stats import skew

from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
```

In [2]:

```
train=pd.read_csv('train.csv')
test=pd.read_csv('test.csv')
```

In [3]:

```
train.shape
```

Out[3]:

```
(1460, 81)
```

In [4]:

```
test.shape
```

Out[4]:

```
(1459, 80)
```

In [5]:

```
#train.options.display.max_columns = None
train.head()
```

Out[5]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Ut
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 rows × 81 columns

In [6]:

```
test.head()
```

Out[6]:

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

5 rows × 80 columns

In [7]:

```
test_id = test['Id']
```

Skewness of target Feature

In [8]:

```
#Checking Skewness

fig=plt.figure(figsize=(12,16))
fig.tight_layout()
ax1 = fig.add_subplot(3, 1, 1)
sns.set_style("darkgrid")
sns.histplot(train.loc[:, 'SalePrice'],kde=True,ax=ax1)

ax2 = fig.add_subplot(3, 1, 2)
stats.probplot(train.loc[:,'SalePrice'],plot=ax2)

ax3 = fig.add_subplot(3, 1, 3)
sns.boxplot(x=train.loc[:, 'SalePrice'],ax=ax3)
```

Out[8]:

```
<AxesSubplot:xlabel='SalePrice'>
```

In [9]:

```
#checking skewness

print("Skewness of the SalesPrice is", train['SalePrice'].skew())
```

```
Skewness of the SalesPrice is 1.8828757597682129
```

In [ ]:

```

```

In [10]:

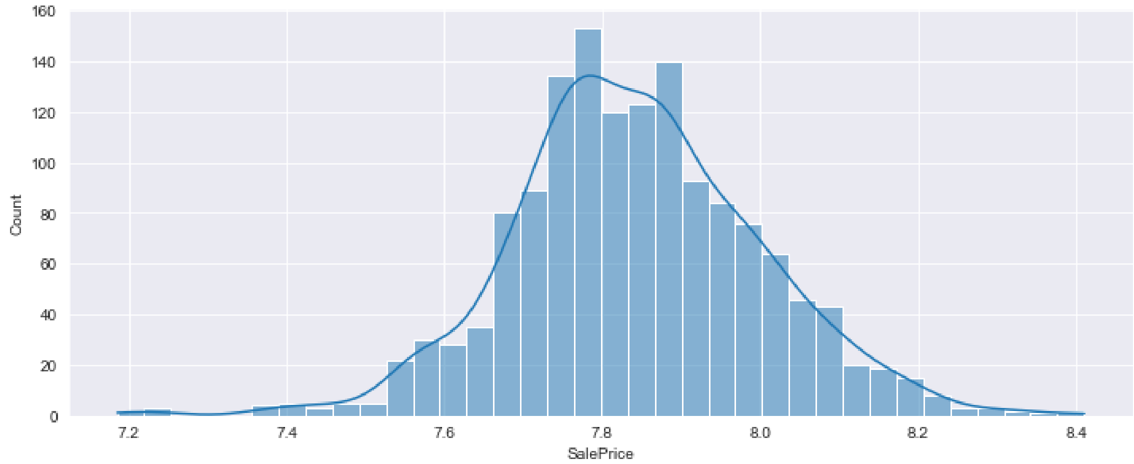
```
#train["SalePrice"] = np.log1p(train["SalePrice"])
train['SalePrice']= stats.boxcox(train['SalePrice'])[0]
```

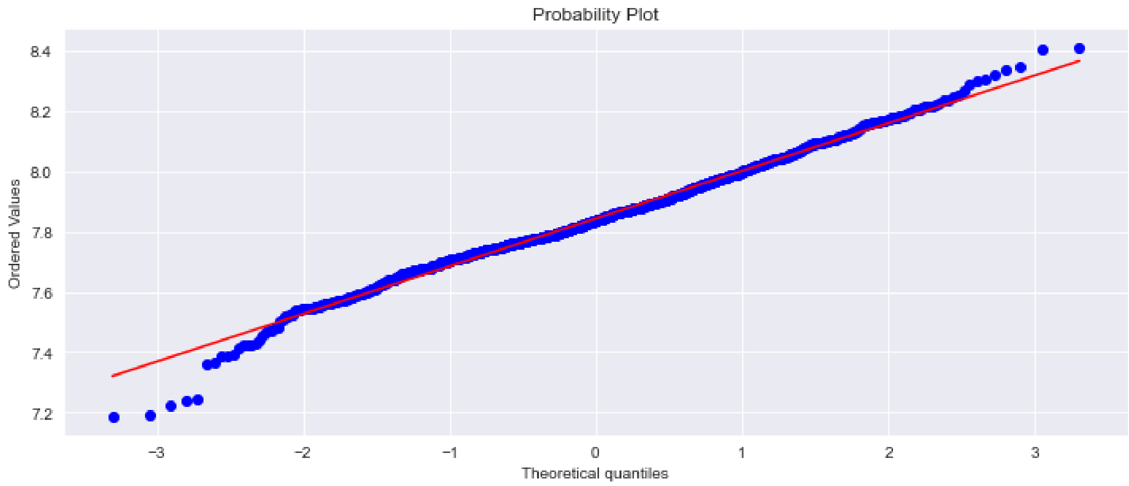
In [11]:

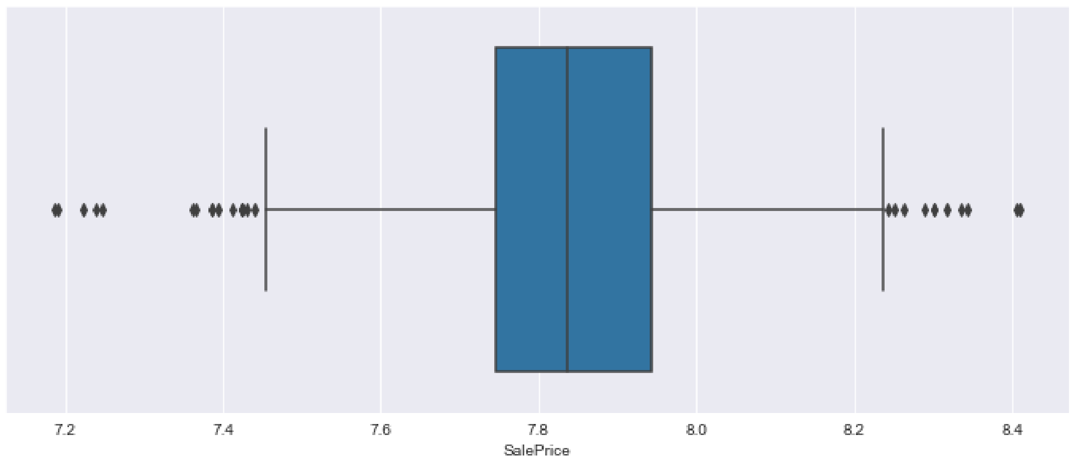
```
#checking skewness

fig=plt.figure(figsize=(12,16))
fig.tight_layout()

ax1 = fig.add_subplot(3, 1, 1)
sns.set_style("darkgrid")
sns.histplot(train.loc[:, 'SalePrice'],kde=True,ax=ax1)

ax2 = fig.add_subplot(3, 1, 2)
stats.probplot(train.loc[:,'SalePrice'],plot=ax2)

ax3 = fig.add_subplot(3, 1, 3)
sns.boxplot(x=train.loc[:, 'SalePrice'],ax=ax3)
```

Out[11]:

```
<AxesSubplot:xlabel='SalePrice'>
```







In [12]:

```
#checking skewness
print("Skewness of the SalesPrice is", train['SalePrice'].skew())
```

```
Skewness of the SalesPrice is -0.008652893640830044
```

In [13]:

```
train.head(100)
```

Out[13]:

ey	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSol
ıN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	:
ıN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	:
ıN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	!
ıN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	:
ıN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	1:
...	...	...	...	...	...	...	...	...	...	.
ıN	IR2	Lvl	AllPub	...	0	NaN	NaN	Shed	480	.
ıN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	!
ıN	Reg	HLS	AllPub	...	0	NaN	NaN	NaN	0	!
ıN	Reg	Lvl	AllPub	...	0	NaN	NaN	Shed	400	!
ıN	IR1	Lvl	AllPub	...	0	NaN	NaN	Shed	400	

In [14]:

```
y = train['SalePrice']
```

In [15]:

```
test.head()
```

Out[15]:

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

5 rows × 80 columns

In [16]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             1460 non-null   int64  
 1   MSSubClass     1460 non-null   int64  
 2   MSZoning       1460 non-null   object 
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64  
 5   Street         1460 non-null   object 
 6   Alley          91 non-null     object 
 7   LotShape       1460 non-null   object 
 8   LandContour    1460 non-null   object 
 9   Utilities      1460 non-null   object 
 10  LotConfig      1460 non-null   object 
 11  LandSlope      1460 non-null   object 
 12  Neighborhood   1460 non-null   object 
 13  Condition1     1460 non-null   object 
 14  Condition2     1460 non-null   object 
 15  BldgType       1460 non-null   object 
 16  HouseStyle     1460 non-null   object 
 17  OverallQual    1460 non-null   int64  
 18  OverallCond    1460 non-null   int64  
 19  YearBuilt      1460 non-null   int64  
 20  YearRemodAdd   1460 non-null   int64  
 21  RoofStyle      1460 non-null   object 
 22  RoofMatl       1460 non-null   object 
 23  Exterior1st    1460 non-null   object 
 24  Exterior2nd    1460 non-null   object 
 25  MasVnrType     1452 non-null   object 
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object 
 28  ExterCond      1460 non-null   object 
 29  Foundation     1460 non-null   object 
 30  BsmtQual       1423 non-null   object 
 31  BsmtCond       1423 non-null   object 
 32  BsmtExposure   1422 non-null   object 
 33  BsmtFinType1   1423 non-null   object 
 34  BsmtFinSF1     1460 non-null   int64  
 35  BsmtFinType2   1422 non-null   object 
 36  BsmtFinSF2     1460 non-null   int64  
 37  BsmtUnfSF      1460 non-null   int64  
 38  TotalBsmtSF    1460 non-null   int64  
 39  Heating        1460 non-null   object 
 40  HeatingQC      1460 non-null   object 
 41  CentralAir     1460 non-null   object 
 42  Electrical     1459 non-null   object 
 43  1stFlrSF       1460 non-null   int64  
 44  2ndFlrSF       1460 non-null   int64  
 45  LowQualFinSF   1460 non-null   int64  
 46  GrLivArea      1460 non-null   int64  
 47  BsmtFullBath   1460 non-null   int64  
 48  BsmtHalfBath   1460 non-null   int64  
 49  FullBath       1460 non-null   int64  
 50  HalfBath       1460 non-null   int64  
```

```
 51  BedroomAbvGr   1460 non-null   int64  
 52  KitchenAbvGr   1460 non-null   int64  
 53  KitchenQual    1460 non-null   object 
 54  TotRmsAbvGrd   1460 non-null   int64  
 55  Functional     1460 non-null   object 
 56  Fireplaces     1460 non-null   int64  
 57  FireplaceQu    770 non-null    object 
 58  GarageType     1379 non-null   object 
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object 
 61  GarageCars     1460 non-null   int64  
 62  GarageArea     1460 non-null   int64  
 63  GarageQual     1379 non-null   object 
 64  GarageCond     1379 non-null   object 
 65  PavedDrive     1460 non-null   object 
 66  WoodDeckSF     1460 non-null   int64  
 67  OpenPorchSF    1460 non-null   int64  
 68  EnclosedPorch  1460 non-null   int64  
 69  3SsnPorch      1460 non-null   int64  
 70  ScreenPorch    1460 non-null   int64  
 71  PoolArea       1460 non-null   int64  
 72  PoolQC         7 non-null      object 
 73  Fence          281 non-null    object 
 74  MiscFeature    54 non-null     object 
 75  MiscVal        1460 non-null   int64  
 76  MoSold         1460 non-null   int64  
 77  YrSold         1460 non-null   int64  
 78  SaleType       1460 non-null   object 
 79  SaleCondition  1460 non-null   object 
 80  SalePrice      1460 non-null   float64
dtypes: float64(4), int64(34), object(43)
memory usage: 924.0+ KB
```

In [ ]:

```

```

In [17]:

```
full_data=pd.concat((train,test))
full_data=full_data.drop('Id',axis="columns")

full_data
```

Out[17]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Uti
0	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	A
1	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	A
2	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	A
3	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	A
4	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	A
...	...	...	...	...	...	...	...	...	
1454	160	RM	21.0	1936	Pave	NaN	Reg	Lvl	A
1455	160	RM	21.0	1894	Pave	NaN	Reg	Lvl	A
1456	20	RL	160.0	20000	Pave	NaN	Reg	Lvl	A
1457	85	RL	62.0	10441	Pave	NaN	Reg	Lvl	A
1458	60	RL	74.0	9627	Pave	NaN	Reg	Lvl	A

2919 rows × 80 columns

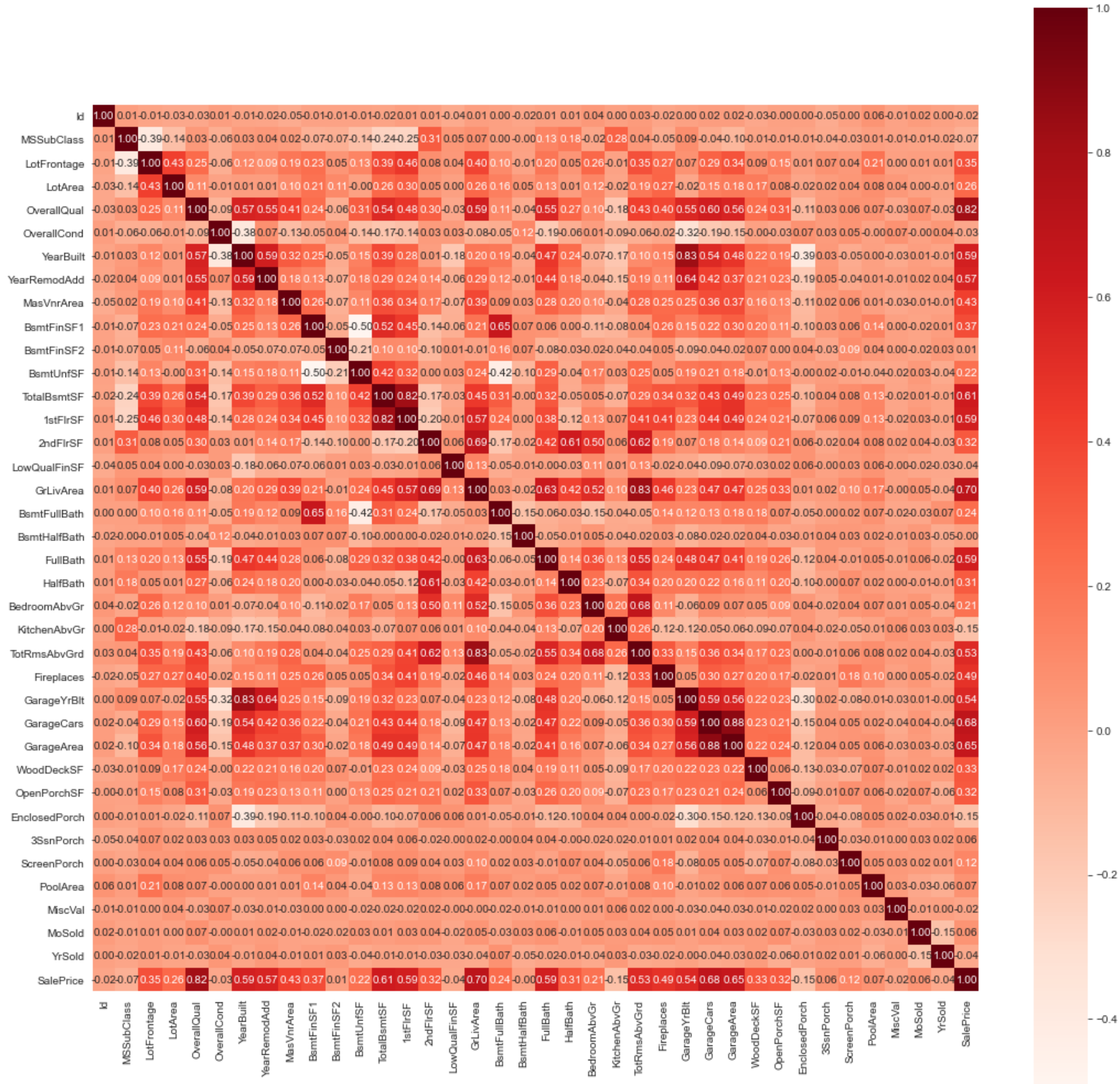
Corelationship

In [18]:

```
plt.figure(figsize=(18,18))
sns.heatmap(train.corr(),annot=True,cmap="Reds",fmt='.2f',square=True)
```

Out[18]:

<AxesSubplot:>



In [19]:

```
correlation = train.corr()['SalePrice']
correlation
```

Out[19]:

```
Id              -0.017327
MSSubClass      -0.072481
LotFrontage      0.354339
LotArea          0.255397
OverallQual      0.815044
OverallCond     -0.032282
YearBuilt        0.588037
YearRemodAdd     0.566884
MasVnrArea       0.425364
BsmtFinSF1       0.369564
BsmtFinSF2       0.006244
BsmtUnfSF        0.221045
TotalBsmtSF      0.609149
1stFlrSF         0.593533
2ndFlrSF         0.317707
LowQualFinSF    -0.038494
GrLivArea        0.697018
BsmtFullBath     0.235705
BsmtHalfBath    -0.003649
FullBath         0.594022
HalfBath         0.314675
BedroomAbvGr     0.211745
KitchenAbvGr    -0.147274
TotRmsAbvGrd     0.532020
Fireplaces       0.488518
GarageYrBlt      0.542682
GarageCars       0.680203
GarageArea       0.649638
WoodDeckSF       0.333121
OpenPorchSF      0.318872
EnclosedPorch   -0.149878
3SsnPorch        0.055339
ScreenPorch      0.121459
PoolArea         0.068403
MiscVal         -0.019930
MoSold           0.057452
YrSold          -0.037891
SalePrice        1.000000
Name: SalePrice, dtype: float64
```

Handling Missing Values

In [20]:

```
df=full_data
#df=df.drop(['SalePrice'],axis='columns')
df.isnull().sum()
```

Out[20]:

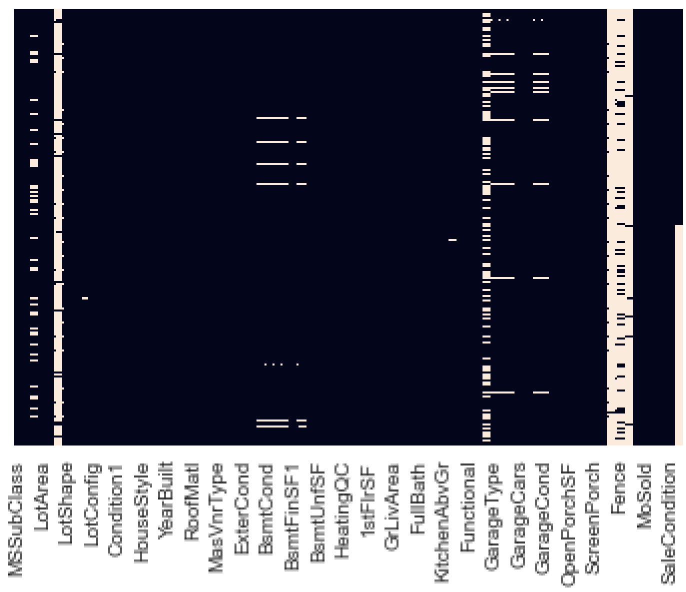
```
MSSubClass          0
MSZoning            4
LotFrontage       486
LotArea             0
Street              0
                 ...
MoSold              0
YrSold              0
SaleType            1
SaleCondition       0
SalePrice        1459
Length: 80, dtype: int64
```

In [21]:

```
#checking for missing values

sns.heatmap(df.isnull(),yticklabels=False,cbar=False)
```

Out[21]:

```
<AxesSubplot:>
```



Replacing incorrect values to nan

In [22]:

```
pd.set_option("display.max_rows", None, "display.max_columns", None)
#df
```

In [23]:

```
#The garage year built is greater is older than the built year of house, which cannot be po
df.replace(df['GarageYrBlt'].values[2592],np.NaN,inplace=True)
df.replace(df['GarageYrBlt'].values[2549],np.NaN,inplace=True)
```

In [24]:

```
#2592
#print(df['GarageYrBlt'].values[2592])
```

Checking Missing Percent

In [25]:

```
#checking missing values
#pd.set_option('display.max_rows')

null_values=df.isnull().sum().values
l=df.shape[0]
percent=[]

for i in null_values:
    percent.append((i/l)*100)

d={'Columns':df.columns,'Missing_Count':null_values, 'Missing_Percent':percent}
data=pd.DataFrame(data=d)
data=data[data.Missing_Count>0]
data=data.sort_values(by="Missing_Percent",ascending=False)
style = data.style.set_properties(**{'text-align':'left'}).set_table_styles([dict(selector=
style
```

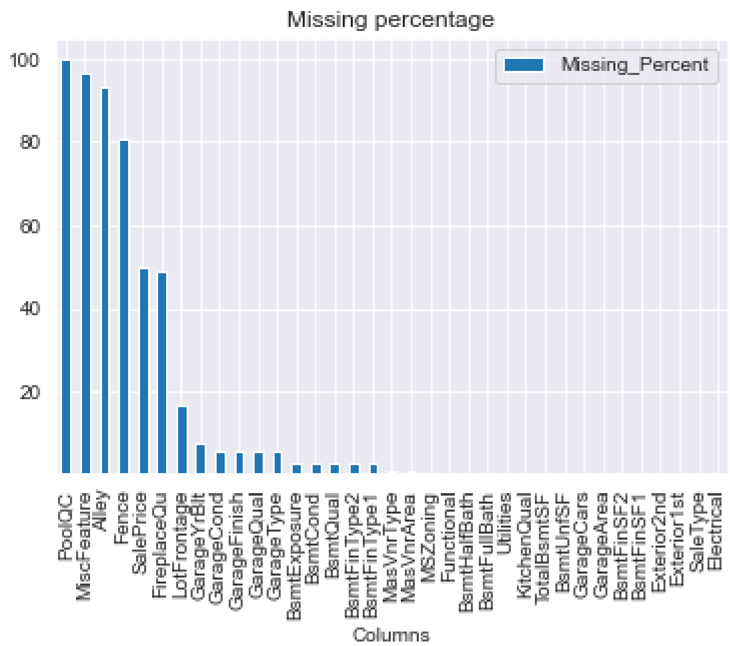
Out[25]:

	Columns	Missing_Count	Missing_Percent
71	PoolQC	2909	99.657417
73	MiscFeature	2814	96.402878
5	Alley	2721	93.216855
72	Fence	2348	80.438506
79	SalePrice	1459	49.982871
56	FireplaceQu	1420	48.646797
2	LotFrontage	486	16.649538
58	GarageYrBlt	221	7.571086
63	GarageCond	159	5.447071
59	GarageFinish	159	5.447071
62	GarageQual	159	5.447071
57	GarageType	157	5.378554
31	BsmtExposure	82	2.809181
30	BsmtCond	82	2.809181
29	BsmtQual	81	2.774923
34	BsmtFinType2	80	2.740665
32	BsmtFinType1	79	2.706406
24	MasVnrType	24	0.822199
25	MasVnrArea	23	0.787941
1	MSZoning	4	0.137033
54	Functional	2	0.068517
47	BsmtHalfBath	2	0.068517
46	BsmtFullBath	2	0.068517
8	Utilities	2	0.068517

	Columns	Missing_Count	Missing_Percent
52	KitchenQual	1	0.034258
37	TotalBsmtSF	1	0.034258
36	BsmtUnfSF	1	0.034258
60	GarageCars	1	0.034258
61	GarageArea	1	0.034258
35	BsmtFinSF2	1	0.034258
33	BsmtFinSF1	1	0.034258
23	Exterior2nd	1	0.034258
22	Exterior1st	1	0.034258
77	SaleType	1	0.034258
41	Electrical	1	0.034258

In [26]:

```
data.plot.bar(x="Columns", y="Missing_Percent", title="Missing percentage",bottom=0.1)
plt.show()
```



Dropping columns with missing values > 75%

In [27]:

```
# Dropping columns with missing_percent>75
#data=data.drop(['SalePrice'],axis=0)
while True:
    if data.iloc[0:len(data),2].values[0]>75:
        df=df.drop([data.iloc[0:len(data),0].values[0]],axis='columns')
        data.drop(index=data.index[0], axis=0, inplace=True)
    else:
        break
```

In [28]:

```
data = data.iloc[1: , :]
data
```

Out[28]:

	Columns	Missing_Count	Missing_Percent
56	FireplaceQu	1420	48.646797
2	LotFrontage	486	16.649538
58	GarageYrBlt	221	7.571086
63	GarageCond	159	5.447071
59	GarageFinish	159	5.447071
62	GarageQual	159	5.447071
57	GarageType	157	5.378554
31	BsmtExposure	82	2.809181
30	BsmtCond	82	2.809181
29	BsmtQual	81	2.774923
34	BsmtFinType2	80	2.740665
32	BsmtFinType1	79	2.706406
24	MasVnrType	24	0.822199
25	MasVnrArea	23	0.787941
1	MSZoning	4	0.137033
54	Functional	2	0.068517
47	BsmtHalfBath	2	0.068517
46	BsmtFullBath	2	0.068517
8	Utilities	2	0.068517
52	KitchenQual	1	0.034258
37	TotalBsmtSF	1	0.034258
36	BsmtUnfSF	1	0.034258
60	GarageCars	1	0.034258
61	GarageArea	1	0.034258
35	BsmtFinSF2	1	0.034258
33	BsmtFinSF1	1	0.034258
23	Exterior2nd	1	0.034258
22	Exterior1st	1	0.034258
77	SaleType	1	0.034258
41	Electrical	1	0.034258

In [ ]:

```

```

Filtering categorical and numerical values in missing values

In [29]:

```
train.head(100)
```

Out[29]:

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

In [ ]:

```

```

In [30]:

```
missing_cate=[]
missing_num=[]
```

In [31]:

```
#Appending categorical values to missing_cate and non-cate values to missing_num inorder to

i=0
data_rows=data.iloc[0:len(data),0]
while i!=len(data):
    x=data_rows.values[i]
    if df.dtypes[x] == object:
        missing_cate.append(x)
    else:
        missing_num.append(x)
    i+=1
```

In [32]:

```
print(missing_cate,"\n",missing_num)
```

```
['FireplaceQu', 'GarageCond', 'GarageFinish', 'GarageQual', 'GarageType', 'B
smtExposure', 'BsmtCond', 'BsmtQual', 'BsmtFinType2', 'BsmtFinType1', 'MasVn
rType', 'MSZoning', 'Functional', 'Utilities', 'KitchenQual', 'Exterior2nd',
'Exterior1st', 'SaleType', 'Electrical']
 ['LotFrontage', 'GarageYrBlt', 'MasVnrArea', 'BsmtHalfBath', 'BsmtFullBat
h', 'TotalBsmtSF', 'BsmtUnfSF', 'GarageCars', 'GarageArea', 'BsmtFinSF2', 'B
smtFinSF1']
```

Filling Missing Values

In [33]:

```
#filling missing categorical values with mode of the column
for i in missing_cate:
    df[i].fillna(df[i].mode()[0], inplace=True)

#filling missing non-categorical values with mean of the column
for i in missing_num:
    df[i].fillna(df[i].mean(), inplace=True)
```

In [34]:

```
#df
```

In [35]:

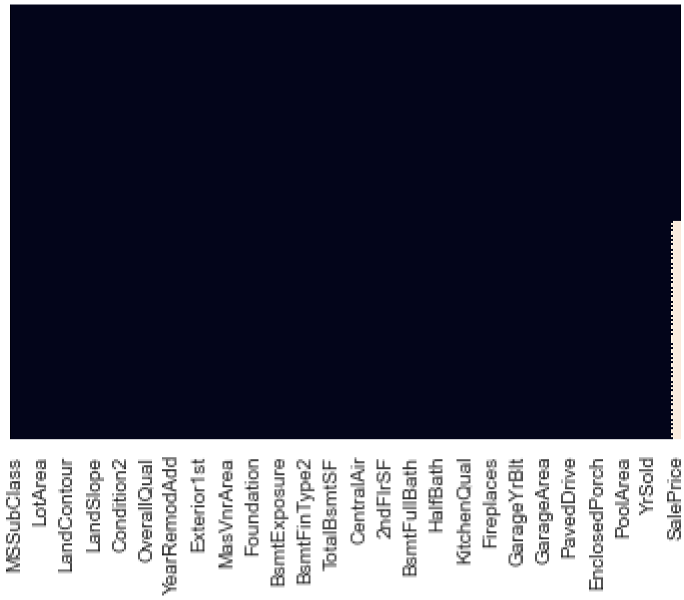
```
#df["YrSold"] = df["YrSold"].astype(str)
```

In [36]:

```
#checking for missing values

sns.heatmap(df.isnull(),yticklabels=False,cbar=False)
```

Out[36]:

```
<AxesSubplot:>
```



Feature Engineering

In [37]:

```
df.loc[df['YearRemodAdd'] ==0, 'YearRemodAdd'] = df['YearBuilt']

df['age'] = df['YrSold']-df['YearRemodAdd']

df['Garage_age'] =  df['YrSold']-df['GarageYrBlt']
```

In [ ]:

```

```

Correlationship

In [38]:

```
correlation = df.corr()['SalePrice']
correlation
```

Out[38]:

```
MSSubClass      -0.072481
LotFrontage      0.334042
LotArea          0.255397
OverallQual      0.815044
OverallCond     -0.032282
YearBuilt        0.588037
YearRemodAdd     0.566884
MasVnrArea       0.424118
BsmtFinSF1       0.369564
BsmtFinSF2       0.006244
BsmtUnfSF        0.221045
TotalBsmtSF      0.609149
1stFlrSF         0.593533
2ndFlrSF         0.317707
LowQualFinSF    -0.038494
GrLivArea        0.697018
BsmtFullBath     0.235705
BsmtHalfBath    -0.003649
FullBath         0.594022
HalfBath         0.314675
BedroomAbvGr     0.211745
KitchenAbvGr    -0.147274
TotRmsAbvGrd     0.532020
Fireplaces       0.488518
GarageYrBlt      0.475276
GarageCars       0.680203
GarageArea       0.649638
WoodDeckSF       0.333121
OpenPorchSF      0.318872
EnclosedPorch   -0.149878
3SsnPorch        0.055339
ScreenPorch      0.121459
PoolArea         0.068403
MiscVal         -0.019930
MoSold           0.057452
YrSold          -0.037891
SalePrice        1.000000
age             -0.569453
Garage_age      -0.476171
Name: SalePrice, dtype: float64
```

In [39]:

```
df = df.drop(columns = ['GarageYrBlt','YrSold','YearBuilt','SalePrice','YearRemodAdd'],axis
```

Data Transformation

In [40]:

```
#cate=['OverallQual','OverallCond','HalfBath','GarageCars','Fireplaces','TotRmsAbvGrd','Kit
#        'HalfBath','FullBath','BsmtHalfBath','BsmtFullBath']
i=0
while i<df.shape[1]:
    x=df.columns[i]
    #df[x] = scaler.fit_transform(df[[x]])
    if df[x].dtypes!=object and abs(df[x]).skew() >0.5:
        #df[x] = boxcox1p(df[x],boxcox_normmax(df[x]+1))
        df[x]=stats.boxcox(df[x]+1)[0]
        #df[x] = np.log1p(df[x])
    i+=1
```

Encoding Categorical values

In [41]:

```
#Finding ctegorical values from df
categorical=[]
for i in df:
    if df.dtypes[i] == object:

        categorical.append(i)
print(categorical)
```

```
['MSZoning', 'Street', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseS
tyle', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposur
e', 'BsmtFinType1', 'BsmtFinType2', 'Heating', 'HeatingQC', 'CentralAir', 'E
lectrical', 'KitchenQual', 'Functional', 'FireplaceQu', 'GarageType', 'Garag
eFinish', 'GarageQual', 'GarageCond', 'PavedDrive', 'SaleType', 'SaleConditi
on']
```

In [42]:

```
#convert categorical to numerical

label_encoder = preprocessing.LabelEncoder()

#for i in categorical:
df['MSZoning']= label_encoder.fit_transform(df['MSZoning'])
```

In [43]:

```
df.head()
```

Out[43]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	L
0	2.664197	3	15.967598	17.222846	Pave	Reg	Lvl	AllPub	
1	2.192747	3	18.053783	17.639401	Pave	Reg	Lvl	AllPub	
2	2.664197	3	16.400996	18.166916	Pave	IR1	Lvl	AllPub	
3	2.722440	3	15.224624	17.622218	Pave	IR1	Lvl	AllPub	
4	2.664197	3	18.579309	18.976035	Pave	IR1	Lvl	AllPub	

In [44]:

```
df.shape
```

Out[44]:

(2919, 73)

In [45]:

```
df = pd.get_dummies(df).reset_index(drop=True)
df.shape
```

Out[45]:

(2919, 269)

In [46]:

```
df.head(100)
```

Out[46]:

	MSSubClass	MSZoning	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea	BsmtFinSF1	B
0	2.664197	3	15.967598	17.222846	7	3.009505	2.961482	15.065113	
1	2.192747	3	18.053783	17.639401	6	4.193354	0.000000	16.552706	
2	2.664197	3	16.400996	18.166916	7	3.009505	2.908403	13.490633	
3	2.722440	3	15.224624	17.622218	7	3.009505	0.000000	10.499547	
4	2.664197	3	18.579309	18.976035	8	3.009505	3.108887	14.738148	
5	2.592851	3	18.708904	18.940639	5	3.009505	0.000000	15.224820	
6	2.192747	3	17.379807	17.801833	8	3.009505	2.947131	18.207643	
7	2.664197	3	16.586930	17.898500	7	3.429547	3.015319	15.947275	
8	2.592851	4	13.812805	16.199985	7	3.009505	0.000000	0.000000	

In [ ]:

```

```

Spliting data

In [47]:

```
X = df.iloc[:len(y), :]
test = df.iloc[len(y):, :]
```

In [48]:

```

```

In [49]:

```
X.shape , test.shape, y.shape
```

Out[49]:

```
((1460, 269), (1459, 269), (1460,))
```

In [50]:

```
#Split Data for Training and Testing
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error,r2_score
#from sklearn import metrics

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3,random_state=0)
```

Training

Metric Evaluation

In [51]:

```
rms=[]
mse=[]
mae=[]
r2=[]
def evaluation(model, x_test, y_test, rms , mse, mae, r2):
        pred = model.predict(x_test)

        rms.append(np.sqrt(mean_squared_error(y_test, pred)))
        mse.append(mean_squared_error(y_test, pred))
        mae.append(mean_absolute_error(y_test, pred))
        r2.append(r2_score(y_test,pred))

        print("Result\n","---------------------------------------------")
        print("\nRoot Mean Squared Error: ",np.sqrt(mean_squared_error(y_test, pred)))
        print("\nMean Squared Error: \t",mean_squared_error(y_test, pred))
        print("\nMean Absolute Error: \t",mean_absolute_error(y_test, pred))
        print("\nR2 Score: \t\t",r2_score(y_test,pred))
```

In [52]:

```
from sklearn.linear_model import Lasso
model = Lasso(alpha=1.0)
model.fit(x_train, y_train)
model.score(x_test,y_test)
```

Out[52]:

```
0.4946075988475276
```

Linear Regression

In [53]:

```
from sklearn.linear_model import LinearRegression

regressor = LinearRegression()
model=regressor.fit(x_train, y_train)

#y_pred_test_Forestreg=model.predict(x_test)
model.score(x_test,y_test)
```

Out[53]:

```
0.793913116231499З
```

In [54]:

```
evaluation(model, x_test, y_test,  rms , mse, mae, r2)
```

```
Result
 ---------------------------------------------

Root Mean Squared Error:  0.0703301703553051

Mean Squared Error:      0.004946332862206237

Mean Absolute Error:     0.03880783145285173

R2 Score:                0.7939131162314993
```

In [ ]:

```

```

KNN

In [55]:

```
from sklearn.neighbors import KNeighborsRegressor

knn=KNeighborsRegressor(n_neighbors=18)
x=knn.fit(x_train,y_train)
```

In [56]:

```
evaluation(x, x_test, y_test, rms , mse, mae, r2)
```

```
Result
 ---------------------------------------------

Root Mean Squared Error:  0.09322253607260549

Mean Squared Error:      0.00869044123180823

Mean Absolute Error:     0.06909841459225842

R2 Score:                0.6379164116266517
```

SVR

In [57]:

```
from sklearn import svm

svr = svm.SVR()
model_svr=svr.fit(x_train, y_train)
```

In [58]:

```
evaluation(model_svr, x_test, y_test,  rms , mse, mae, r2)
```

```
Result
 ---------------------------------------------

Root Mean Squared Error:  0.08154467179330871

Mean Squared Error:      0.006649533497878438

Mean Absolute Error:     0.06252627208902889

R2 Score:                0.7229499762212146
```

XGBOOST

In [59]:

```
import xgboost as xgb
xgb_regress = xgb.XGBRegressor(n_estimators = 1000, learning_rate = 0.1)
l=xgb_regress.fit(x_train, y_train)

evaluation(l, x_test, y_test,  rms , mse, mae, r2)
```

```
Result
 ---------------------------------------------

Root Mean Squared Error:  0.051257055089208546

Mean Squared Error:      0.0026272856964181597

Mean Absolute Error:     0.03230945967949397

R2 Score:                0.8905352435778314
```

ADABOOST

In [60]:

```
from sklearn import ensemble

regr = ensemble.AdaBoostRegressor(random_state=0, n_estimators=1000)
l=regr.fit(x_train, y_train)
```

In [61]:

```
evaluation(l, x_test, y_test,  rms , mse, mae, r2)
```

```
Result
 ---------------------------------------------

Root Mean Squared Error:  0.06896396227244583

Mean Squared Error:      0.004756028092315333

Mean Absolute Error:     0.0530700465556499

R2 Score:                0.8018420846381271
```

GradientBoost

In [62]:

```
params = {
    "n_estimators": 1000,
    "max_depth": 4,
    "min_samples_split": 5,
    "learning_rate": 0.1
}

reg = ensemble.GradientBoostingRegressor(**params)
l=reg.fit(x_train, y_train)
```

In [63]:

```
evaluation(l, x_test, y_test,  rms , mse, mae, r2)
```

```
Result
 ---------------------------------------------

Root Mean Squared Error:  0.04942810103350073

Mean Squared Error:      0.002443137171777956

Mean Absolute Error:     0.03398502964027502

R2 Score:                0.8982077146085697
```

RandomForest

In [64]:

```
from sklearn.ensemble import RandomForestRegressor

 # create regressor object
regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)

# fit the regressor with x and y data
r=regressor.fit(x_train, y_train)
```

In [65]:

```
evaluation(r, x_test, y_test,  rms , mse, mae, r2)
```

```
Result
 ---------------------------------------------

Root Mean Squared Error:  0.05394573137845794

Mean Squared Error:      0.0029101419339567416

Mean Absolute Error:     0.03696946084269166

R2 Score:                0.8787501570960436
```

CatBoost

In [66]:

```
import catboost as cb

#from sklearn.inspection import permutation_importance

model = cb.CatBoostRegressor(iterations=1000,
                             learning_rate=0.1,
                             depth=8)
# Fit model
cat=model.fit(x_train, y_train)
```

```
0:      learn: 0.1493480        total: 167ms    remaining: 2m 46s
1:      learn: 0.1404617        total: 187ms    remaining: 1m 33s
2:      learn: 0.1326432        total: 207ms    remaining: 1m 8s
3:      learn: 0.1249758        total: 226ms    remaining: 56.3s
4:      learn: 0.1183172        total: 245ms    remaining: 48.7s
5:      learn: 0.1125798        total: 265ms    remaining: 43.9s
6:      learn: 0.1073468        total: 286ms    remaining: 40.6s
7:      learn: 0.1018173        total: 304ms    remaining: 37.7s
8:      learn: 0.0972322        total: 319ms    remaining: 35.2s
9:      learn: 0.0929331        total: 333ms    remaining: 32.9s
10:     learn: 0.0894323        total: 344ms    remaining: 30.9s
11:     learn: 0.0855525        total: 358ms    remaining: 29.5s
12:     learn: 0.0820156        total: 375ms    remaining: 28.5s
13:     learn: 0.0788386        total: 390ms    remaining: 27.5s
14:     learn: 0.0756009        total: 403ms    remaining: 26.5s
15:     learn: 0.0730286        total: 414ms    remaining: 25.5s
16:     learn: 0.0707613        total: 427ms    remaining: 24.7s
17:     learn: 0.0685509        total: 438ms    remaining: 23.9s
18:     learn: 0.0664708        total: 450ms    remaining: 23.2s
```

In [67]:

```
evaluation(cat, x_test, y_test,  rms , mse, mae, r2)
```

```
Result
 ---------------------------------------------

Root Mean Squared Error:  0.04729808701467888

Mean Squared Error:      0.002237109035248135

Mean Absolute Error:     0.03222136676989126

R2 Score:                0.9067917904904188
```

In [ ]:

```

```

In [68]:

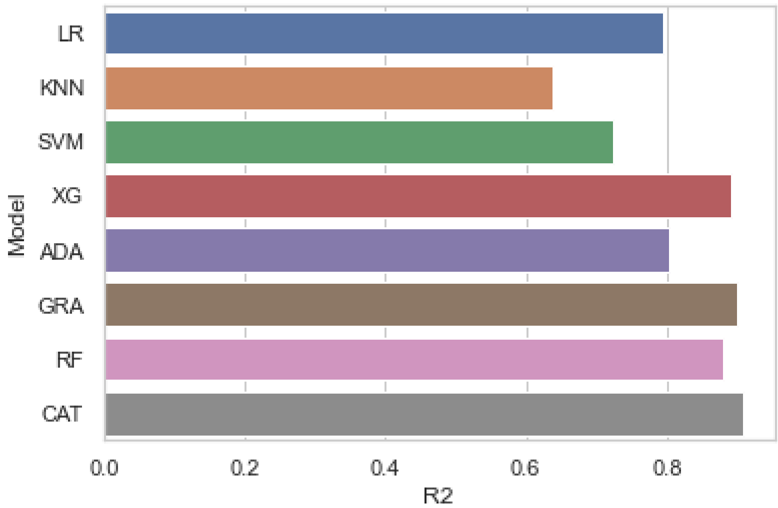
```
df=pd.DataFrame()
df['Model']=["LR","KNN","SVM","XG","ADA","GRA","RF","CAT"]
df['RMSE']=rms
df['MSE']=mse
df['MAE']=mae
df['R2']=r2

df
```

Out[68]:

	Model	RMSE	MSE	MAE	R2
0	LR	0.070330	0.004946	0.038808	0.793913
1	KNN	0.093223	0.008690	0.069098	0.637916
2	SVM	0.081545	0.006650	0.062526	0.722950
3	XG	0.051257	0.002627	0.032309	0.890535
4	ADA	0.068964	0.004756	0.053070	0.801842
5	GRA	0.049428	0.002443	0.033985	0.898208
6	RF	0.053946	0.002910	0.036969	0.878750
7	CAT	0.047298	0.002237	0.032221	0.906792

In [69]:

```
sns.set(style="whitegrid")
ax=sns.barplot(y='Model',x='R2',data=df)
```



In [ ]: