Second Project

Ask a home buyer to describe their dream house, and they probably won't begin with the height of the basement ceiling or the proximity to an east-west railroad. But this dataset proves that much more influences price negotiations than the number of bedrooms or a white-picket fence.

With 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa, this data set is a challenge to predict the final price of each home.

In the Data set we have 79 independent variables or explanatory variables we will utilize to estimate the price of the houses.

This approach is achieved by the following steps:

1. Importing the Data
2. Wrangling the data
3. Performing EDA on the data set
4. Predicting the house price by applying XGBoost machine learning algorithm.

1. Importing the Data

Importing the needed Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import scipy.stats as st
import xgboost as xgb
from sklearn.feature_extraction import DictVectorizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from xgboost import XGBRegressor
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
seed=42
```

The Data has two sets the training set and a description data set

In [2]:
```
df=pd.read_csv(r'C:\Users\issam\OneDrive\Desktop\House Price\train.csv')
with open (r'C:\Users\issam\OneDrive\Desktop\House Price\data_description.txt')as f:
    description = f.read()
```

After importing the data set we will perform some initial inspection

In [3]:
```
df.head()
```

Out[3]:

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

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
Id               1460 non-null int64
MSSubClass       1460 non-null int64
MSZoning         1460 non-null object
LotFrontage      1201 non-null float64
LotArea          1460 non-null int64
Street           1460 non-null object
Alley            91 non-null object
```

Eyeballing the information shows that the training data set has a lot of missing values.The Data has 79 features and these features are a mix of numeric and Categorical features so we will dive to get to know what is categorical and numeric for further analysis.

In [8]:
```
df.drop(['SalePrice'],axis=1).describe()
```

Out[8]:

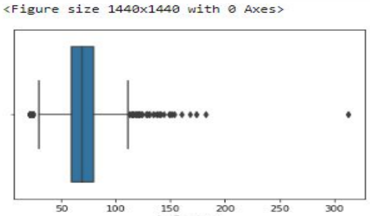
	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	M
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	14
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	16

8 rows × 37 columns

As the above chart shows we have some features with anomalies (Outliers) like MSSubClass and lotFrontage and this what the following boxplots confirms

In [10]:
```
sns.boxplot('LotFrontage',data=df)
plt.figure(figsize=(20, 20))
```

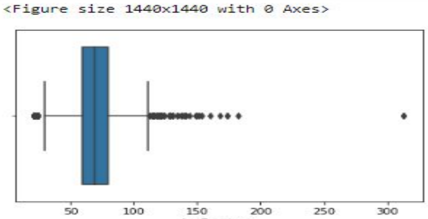
Out[10]: <Figure size 1440x1440 with 0 Axes>



<Figure size 1440x1440 with 0 Axes>

In [10]:
```
sns.boxplot('LotFrontage',data=df)
plt.figure(figsize=(20, 20))
```

Out[10]: <Figure size 1440x1440 with 0 Axes>

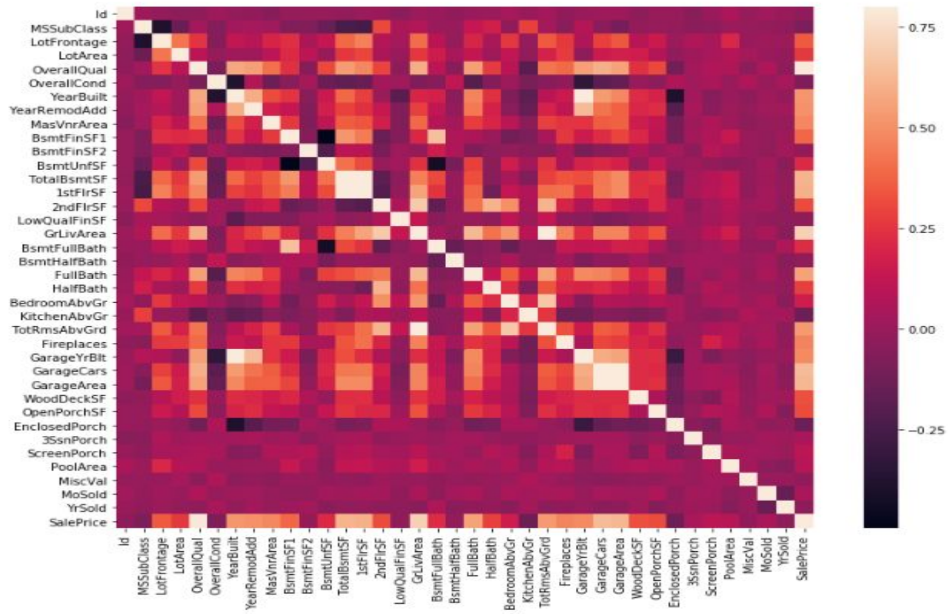


<Figure size 1440x1440 with 0 Axes>

Now part of EDA is to detect the correlation between the dependent and the independent variables. This is done by calling or plotting a heatmap

Ploting a Heatmap to vesualize the correlation between the features and the SalePrice

In [15]:
```
corr=df.corr()
ax1,ax2=plt.subplots(figsize=(14,10))
sns.heatmap(corr,vmax=0.8,square=True);
```



The next step is visualising the correlation figures or rates by a correlation matrix

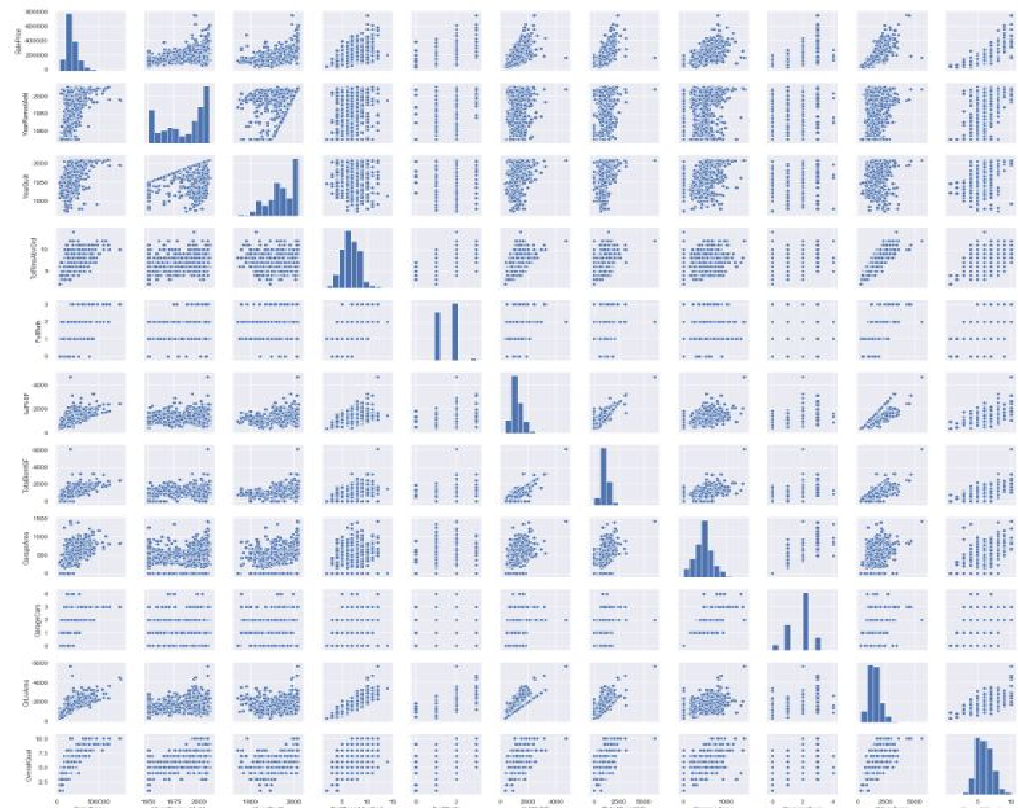
In [16]:
```
Matrix_corr=df[['YearRemodAdd','YearBuilt','TotRmsAbvGrd','FullBath','1stFlrSF','TotalBsmtSF','GarageArea',
               'GarageCars','GrLivArea','OverallQual','SalePrice']].corr()

sns.set(font_scale=1.10)
plt.figure(figsize=(12, 12))
sns.heatmap(Matrix_corr, xticklabels=Matrix_corr.columns, yticklabels=Matrix_corr.columns,vmax=.8, linewidths=0.01,
            square=True,cmap='viridis',linecolor="white",annot=True)
plt.title('Correlation between features');
```

As the correlation matrix shows the OverallQual is highly correlated to the SalePrice and so GrLivArea which explains the magnitude of effect both have on SalePrice also we have some of the Features are highly correlated to each other which suggests Collinearity which would lead to misleading Model like (GrLivArea ,TotRmsAbGrd) , (TotalBsmtsf,1stFliSF) and (GarageArea , GarageCars). This Multicollinearity could be solved by Features Engineering.

Also one important measure to track is the distribution of the important predictors. And the relation among them

In [32]:

```
sns.set()
columns=['SalePrice','YearRemodAdd','YearBuilt','TotRmsAbvGrd','FullBath'
,'1stFlrSF','TotalBsmtSF','GarageArea',
          'GarageCars','GrLivArea','OverallQual']
sns.pairplot(df[columns],height=2)
plt.show()
```



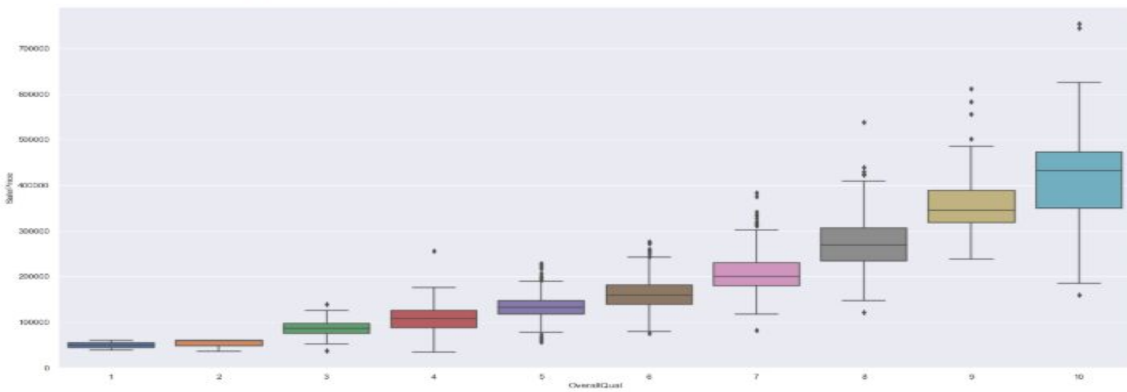
Looking at the first Row of the matrix we can see how these Predictors are correlated to the SalePrice and how the Data is distributed among the Price range. Also we can recognize the collinearity of some of the features like the relation between 1stFirSF and GrLivArea which we need to work on these features in the features engineering section later. The pairplot is an effective method to show the correlation between the Predictors and the Predicted Variables. next I will plot the relation between each of the above Predictors and the Predicted Variable (SalePrice).

Now let us do some Visual EDA for some of the features

```
def Relation_Plot(x,y):
    style.use('fivethirtyeight')
    plt.sublpots(figsize=(20,10))
    return sns.boxplot(y=y,x=x);
```

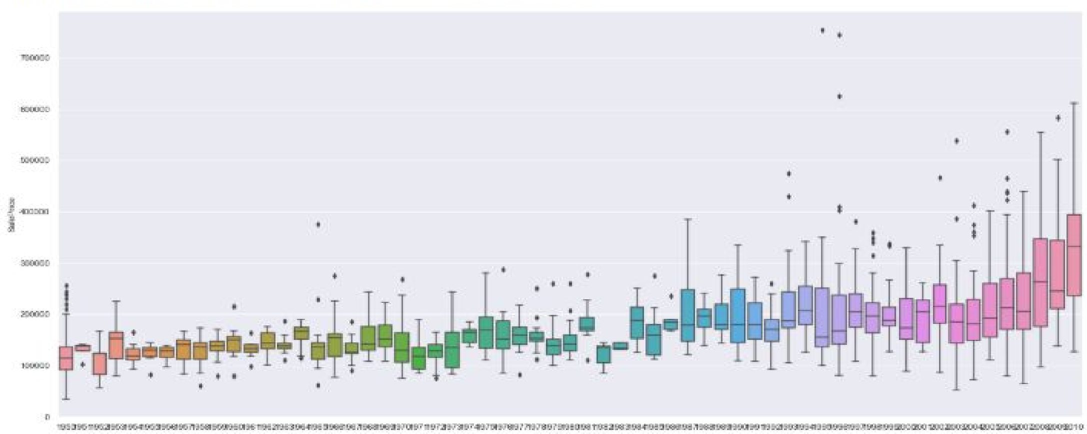
```
sns.boxplot(df.OverallQual,df.SalePrice)
```

<matplotlib.axes._subplots.AxesSubplot at 0x1f8b953a7f0>



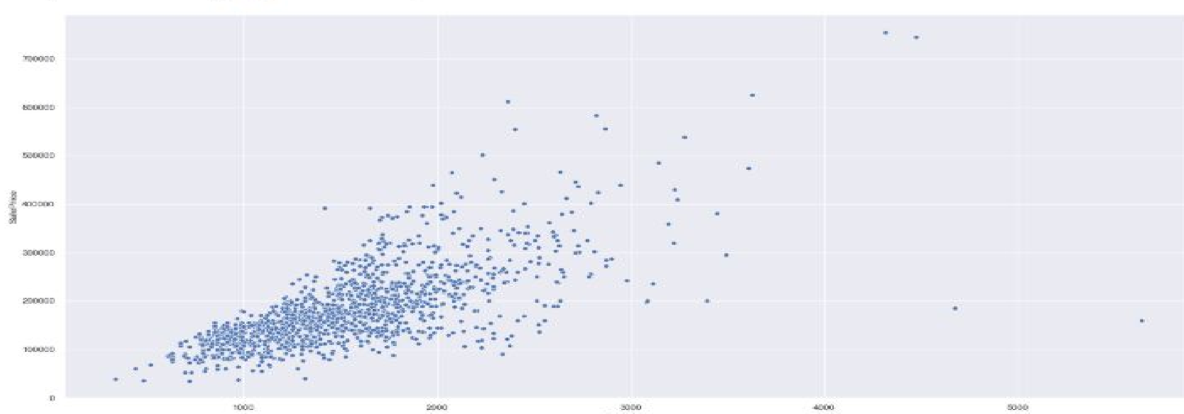

```
sns.boxplot(df.YearRemodAdd,df.SalePrice)
```

<matplotlib.axes._subplots.AxesSubplot at 0x1f8b9f5e198>



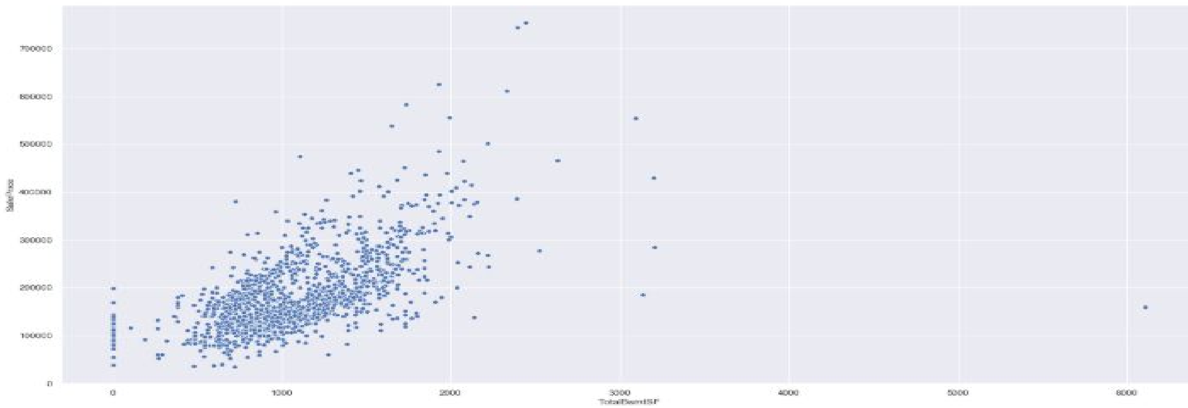

```
sns.scatterplot(df.GrLivArea,df.SalePrice)
```

<matplotlib.axes._subplots.AxesSubplot at 0x1f8bac376d8>



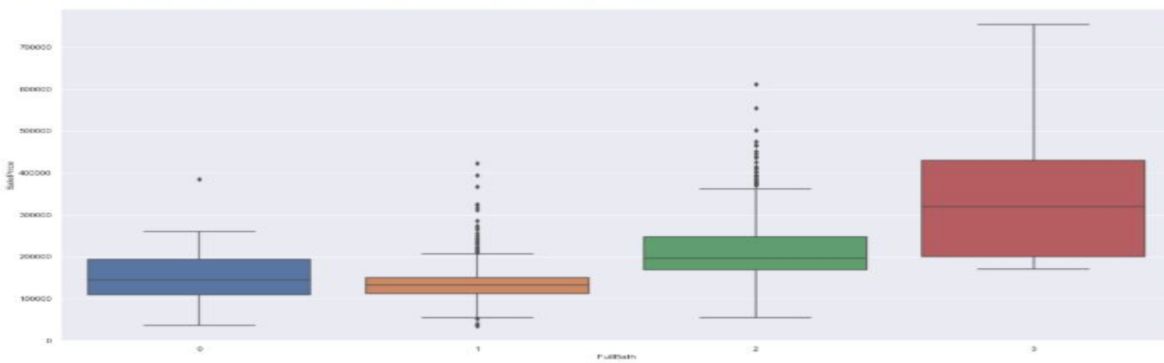
```
sns.scatterplot(df.TotalBsmtSF,df.SalePrice)
```

<matplotlib.axes._subplots.AxesSubplot at 0x1f8ba7c4a20>



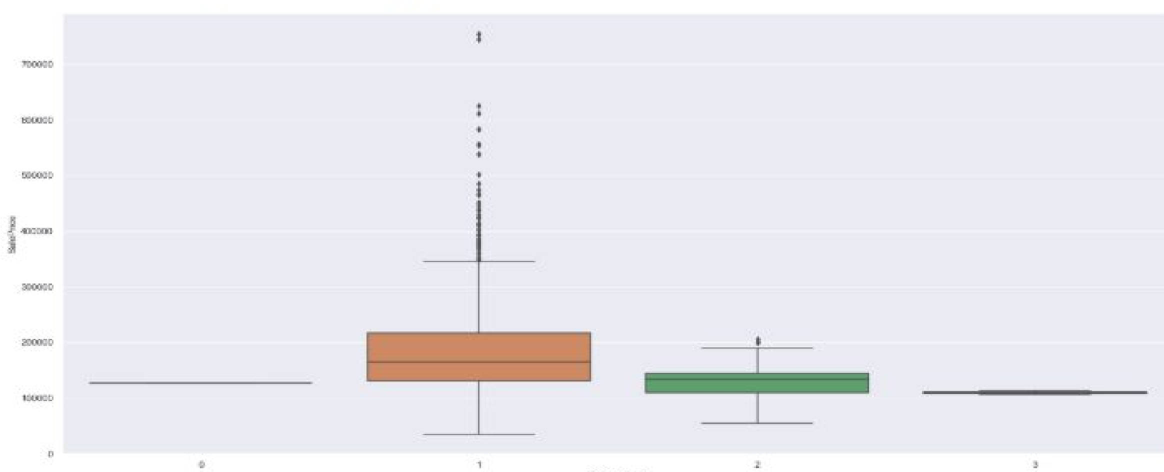

```
sns.boxplot(df.FullBath,df.SalePrice)
```

<matplotlib.axes._subplots.AxesSubplot at 0x1f8ba7dd748>




```
sns.boxplot(df.KitchenAbvGr,df.SalePrice)
```

<matplotlib.axes._subplots.AxesSubplot at 0x1f8ba7c8470>



Next part is feature engineering

```python
# Creating a DataFrame of the missing values totals and percentages
def missing_percentage(dfd):
    total=dfd.isnull().sum().sort_values(ascending=False)
    [dfd.isnull().sum().sort_values(ascending=False)!=0]
    percent=round(dfd.isnull().sum().sort_values(ascending=False)/len(dfd)*100,2)
    [round(df.isnull().sum().sort_values(ascending=False)/len(df)*100,2)!=0]
    return pd.concat([total,percent],axis=1,keys=['Total','Percent'])

missing_percentage(df)
```

]:

	Total	Percent
PoolQC	1453	99.52
MiscFeature	1406	96.30
Alley	1369	93.77
Fence	1179	80.75
FireplaceQu	690	47.26
LotFrontage	259	17.74
GarageCond	81	5.55
GarageType	81	5.55
GarageYrBlt	81	5.55
GarageFinish	81	5.55
GarageQual	81	5.55
BsmtExposure	38	2.60
BsmtFinType2	38	2.60
BsmtFinType1	37	2.53
BsmtCond	37	2.53
BsmtQual	37	2.53
MasVnrArea	8	0.55
MasVnrType	8	0.55
Electrical	1	0.07

Data Preprocessing and Feature Engineering

1.Filling missing NAN
2.separate numerical ,categorical features with dtypes
3.use one hot encoding for categorical
4.impute NaN data with their means
5.log1p transform highly skewed features (threshold =.75) and the target y 6.normalize the features to N(0,1)

Now I have to fill in the missing values of the data I have so I will print the data description to get to know the variable of each of the features so when to fill the missing values I fill with the appropriate value

```
print(description)
```

```
MSSubClass: 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

MSZoning: Identifies the general zoning classification of the sale.
```

Some of the features left with missing Data has the same purpose or the same meaning like (GarageCond,GarageFinish, GarageQual and GarageYrBlt)so I keep GarageType and will fill the missing values with NA (not available as the description of the data depicts the. YrBlt I will drop the feature it has no significant effect on making a decision of Buying a house or House price. also I will drop the BsmtFinType1 ,BsmtFinType2,BsmtExposure,BsmtCond all of them refer to the same meaning and I will keep the BsmtQual and fill the nulls with NA no basement. as for MasVnrArea I will drop and keep the MasVnrType since incase of Masonry exist they mean the same. as for the Electrical Null I will drop the observation it self (the row) dropping an observation of 1460 will not affect the Data

```
df_droped.isnull().sum().sort_values(ascending=False).head(20)
```

44]:
```
LotFrontage     259
GarageFinish     81
GarageType       81
GarageCond       81
GarageQual       81
GarageYrBlt      81
BsmtExposure     38
BsmtFinType2     38
BsmtFinType1     37
BsmtCond         37
BsmtQual         37
MasVnrType        8
MasVnrArea        8
Electrical        1
```

```python
df_droped['LotFrontage']=df_droped['LotFrontage'].fillna(np.mean(df['LotFrontage']))
df_droped['GarageType']= df_droped['GarageType'].fillna('NA')
df_droped['GarageFinish']= df_droped['GarageFinish'].fillna('NA')
df_droped['GarageQual']= df_droped['GarageQual'].fillna('NA')
df_droped['GarageCond']= df_droped['GarageCond'].fillna('NA')
df_droped['GarageYrBlt']= df_droped['GarageYrBlt'].fillna('NA')
df_droped['GarageQual']= df_droped['GarageQual'].fillna('NA')
df_droped['BsmtQual']=df_droped['BsmtQual'].fillna('NA')
df_droped['BsmtExposure']=df_droped['BsmtExposure'].fillna('NA')
df_droped['BsmtFinType2']=df_droped['BsmtFinType2'].fillna('NA')
df_droped['BsmtFinType1']=df_droped['BsmtFinType1'].fillna('NA')
df_droped['BsmtCond']=df_droped['BsmtCond'].fillna('NA')
df_droped['MasVnrType']=df_droped['MasVnrType'].fillna('None')
df_droped['MasVnrArea']=df_droped['MasVnrArea'].fillna('None')
df_droped.isnull().sum().sort_values(ascending=False).head()
```

```
]: Electrical    1
   SalePrice     0
   ExterCond     0
   RoofStyle     0
   RoofMatl      0
   dtype: int64
```

Now I am left with one missing value in my data set without affecting the count of the observations or affecting the data. so now I will drop the missing value by dropping one observation

```python
df_droped=df_droped.dropna()
df_droped.isnull().sum().sort_values(ascending=False).head()
```

```
]: SalePrice      0
   ExterCond      0
   RoofStyle      0
   RoofMatl       0
   Exterior1st    0
   dtype: int64
```

```python
df_droped.shape
```

```
8]: (1459, 76)
```

Now the data is ready for Feature engineering and this Feature engineering involves creating new features and transforming or deleting existing ones

```python
df_pre=df_droped
```

```python
df_droped.select_dtypes(include=['int64','float64']).columns
```

```
0]: Index(['Id', 'MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual',
           'OverallCond', 'YearBuilt', 'YearRemodAdd', 'BsmtFinSF1', 'BsmtFinSF2',
           'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
           'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
           'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces',
           'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
           'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',
           'MoSold', 'YrSold', 'SalePrice'],
          dtype='object')
```

```
# Features to be transformed for simplicity as described by the data description
df_droped['OverallQual']=df_droped.OverallQual.replace({1:1,2:1,3:1, # as bad Quality
                                                        4:2,5:2,6:2, # as average
                                                        7:3,8:3,9:3,10:3 # as good
                                                       })
df_droped['OverallCond']=df_droped.OverallCond.replace({1:1,2:1,3:1, # as bad Quality
                                                        4:2,5:2,6:2, # as average
                                                        7:3,8:3,9:3,10:3 # as good
                                                       })
df_droped['BedroomAbvGr']= df_droped.BedroomAbvGr.replace({0:1,1:1, # average
                                                          2:2,3:2, # good
                                                          4:3,5:3,6:3 # verygood
                                                          })
df_droped['TotRmsAbvGrd']=df_droped.replace({3:1,4:1,5:1,# as bad Quality
                                                        6:2, 7:2,8:2,9:2, # as average
                                                        10:3,11:3,12:3 # as good
                                            })
# we will start with Fetures alike or that mean have the same data description
# The Total basement square feet is just the addition of ['BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF']
#'TotalBsmtSF' and add the finished Basement to 'Totalfinished '
df_droped['Totalfinished']=df_droped['BsmtFinSF1']+df_droped['BsmtFinSF2']
# Adding the Bathrooms to one variable
df_droped['Bathrooms']=df_droped['BsmtFullBath']+df_droped['BsmtHalfBath']*0.5 +df_droped['FullBath
#Adding the Porch to one variable
df_droped['Porsch']= df_droped['OpenPorchSF']+df_droped['EnclosedPorch']+df_droped['3SsnPorch']+df_
#Now the Total area square feet of 1st and 2nd floor
df_droped['TotalArea']=df_droped['1stFlrSF']+df_droped['2ndFlrSF']
```

```
df_droped=df_droped.drop(['BsmtFinSF1','BsmtFinSF2','TotalBsmtSF','BsmtFullBath',
'BsmtHalfBath', 'FullBath', 'HalfBath','OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
 'ScreenPorch','1stFlrSF', '2ndFlrSF'],axis=1)
```

Now the data is ready for applying a machine learning algorithm

Now we have to treat with the categorical variables in the dataset by converting them to dummy variables using the DectVectorizer is an efficient way to do it and thus in one line of coding but first we have to transform the data to a dictionary and apply the DectVectorizer and then transfer it back to dataframe

```
y=df_droped['SalePrice']
X=df_droped.drop(['SalePrice'],axis=1)
```

```
type(y)
```

```
']: pandas.core.series.Series
```

The second step is to transform the X features to dictionary and then apply the DictVectorizer on the independent features X to transform the features into numpy array

```
df_dict=X.to_dict("records")
dv= DictVectorizer(sparse=False)
X_en = dv.fit_transform(df_dict)
print(X_en[:5,:])
```

```
[[3.500e+00 2.000e+00 1.000e+00 ... 2.003e+03 2.003e+03 2.008e+03]
 [2.500e+00 2.000e+00 1.000e+00 ... 1.976e+03 1.976e+03 2.007e+03]
 [3.500e+00 2.000e+00 1.000e+00 ... 2.001e+03 2.002e+03 2.008e+03]
 [2.000e+00 2.000e+00 1.000e+00 ... 1.915e+03 1.970e+03 2.006e+03]
 [3.500e+00 3.000e+00 1.000e+00 ... 2.000e+03 2.000e+03 2.008e+03]]
```

Splitting the Data set into training and test data set and calling the XGBoost Model on the training data with the default parameters or with no hyper parameter tuning and call the cross_validatiion scores to check the model out of the box

```
X_train,X_test,y_train,y_test= train_test_split(X_en,y,test_size=0.3,random_state=seed)
steps = [("xgb_model", xgb.XGBRegressor(max_depth=2, objective="reg:linear"))]

# Create the pipeline: xgb_pipeline
xgb_pipeline = Pipeline(steps)

# Cross-validate the model
cross_val_scores = cross_val_score(xgb_pipeline, X_train, y_train, cv=10, scoring="neg_mean_squared_error")

print("10-fold RMSE: ", np.mean(np.sqrt(np.abs(cross_val_scores))))
```

```
[17:12:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
10-fold RMSE:  34460.74597437582
```

```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
pr = xgb_pipeline.predict(X_test)
print("Mean Absolute Error : " + str(mean_absolute_error(pr, y_test)))
print ( "The R^2 score : ", round(r2_score(y_test, pr),2))
```

```
Mean Absolute Error : 17102.22922017694
The R^2 score :  0.88
```

As we can see out of the Box model has a 0.88 coefficient of determination and a $ 17102 as a Mean Absolute Error on the test data set

Now Will work on fine tuning the hyperparameter of the Model

Creating a param grig will help us in reducing the MAE

```
## Parameters for grid search

gbm_param_grid = {
     'colsample_bytree': np.linspace(0.5, 0.9, 5),
     'n_estimators':[50,300],
     'max_depth': [2,8,10, 15 ],

}
```

```
gbm = xgb.XGBRegressor()
grid_mse = GridSearchCV(estimator = gbm, param_grid = gbm_param_grid, scoring =
                        'neg_mean_squared_error', cv = 5, verbose = 1)
```

```
[18:49:31] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Best parameters found:  {'colsample_bytree': 0.6, 'max_depth': 15, 'n_estimators': 300}
Lowest RMSE found:  35647.77753212993
```

```
from sklearn.metrics import mean_absolute_error
pred = grid_mse.predict(X_test)
print("Mean Absolute Error : " + str(mean_absolute_error(pred, y_test)))
#print("Root mean square error for test dataset: {}".format(np.round(np.sqrt(mean_squared_error(y_test, pred)), 2)))
```

```
Mean Absolute Error : 16152.983465325342
```

```
from sklearn.metrics import r2_score
print ( "The R^2 score : ", round(r2_score(y_test, pred),2))
```

```
The R^2 score :  0.9
```

The above coefficient of determination is 0.9 which is high enough and I would be satisfied with this Model and the Mean Absolute Error is 16,152.92.00 Where the average price of the house in the data set is 180,921.19.00