Predicting Houseprice:

This project was completed has part of Kaggle competition.

In this project we will dive deep into the following areas

1. Feature Engineering: a. Impute missing values b. Standardization c. Engineer meaningful features d. Dealing with categorical encoding

In [9]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from scipy.stats import skew, norm
from scipy.special import boxcox1p
from scipy.stats import boxcox_normmax
from scipy.stats.stats import pearsonr

# Models
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.linear_model import Ridge, RidgeCV
from sklearn.linear_model import ElasticNet, ElasticNetCV
from sklearn.svm import SVR
from mlxtend.regressor import StackingCVRegressor
import lightgbm as lgb
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor

# Stats
from scipy.stats import skew, norm
from scipy.special import boxcox1p
from scipy.stats import boxcox_normmax

# Misc
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import scale
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.decomposition import PCA


# Ignore useless warnings
import warnings
warnings.filterwarnings(action="ignore")
pd.options.display.max_seq_items = 8000
pd.options.display.max_rows = 8000
```

In [10]:

```
df=pd.read_csv('C:/Users/thand/Downloads/house-prices-advanced-regression-tec
df.head()
```

Out[10]:

	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 rows × 81 columns

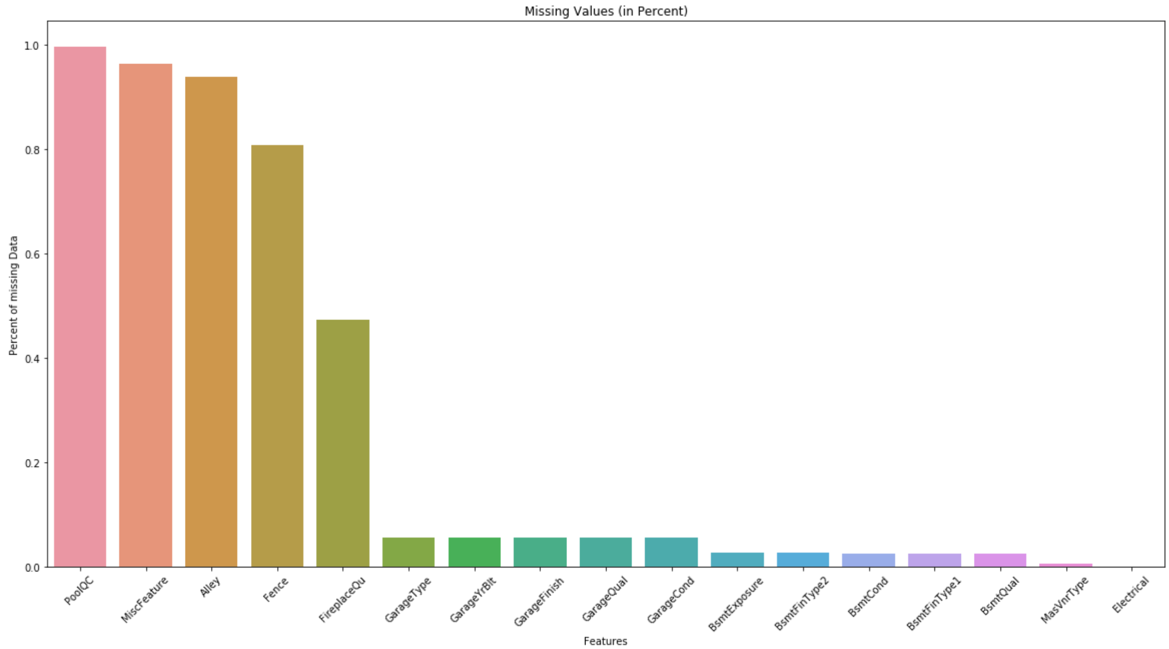
In [13]:

```
# Variables by data types

#In this section, we will make the following lists
#num_cols: list of all numerical variables
#cat_cols: list of all categorical variables
#ordinal_cols: list of all mean encoded ordinal variables


#discrete ratio/interval to categorical features to get more info on them
numericToCategory(df,['MoSold','YrSold','TotRmsAbvGrd','OverallQual','Overall
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
num_cols= df.select_dtypes(include=numerics).columns   #list of all numerical
#Nominal/ordinal
```

In [ ]:

```
# Feature Engineering:
```

In [19]:
```
#Missing Values
missing_values(df)
```



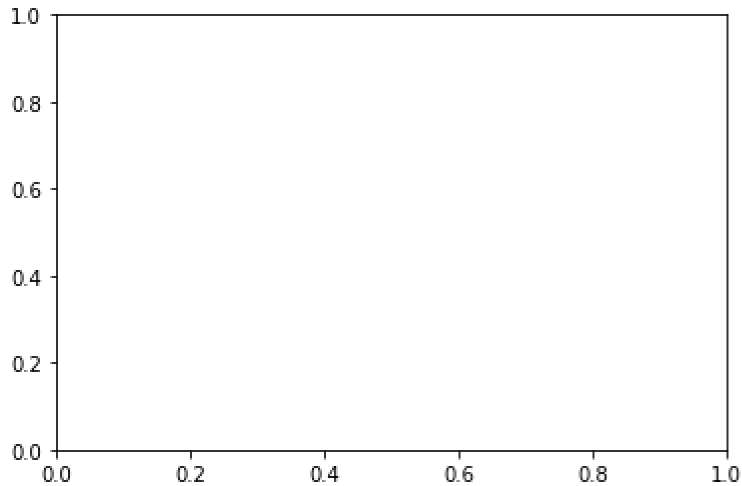
Teating Missing Values:

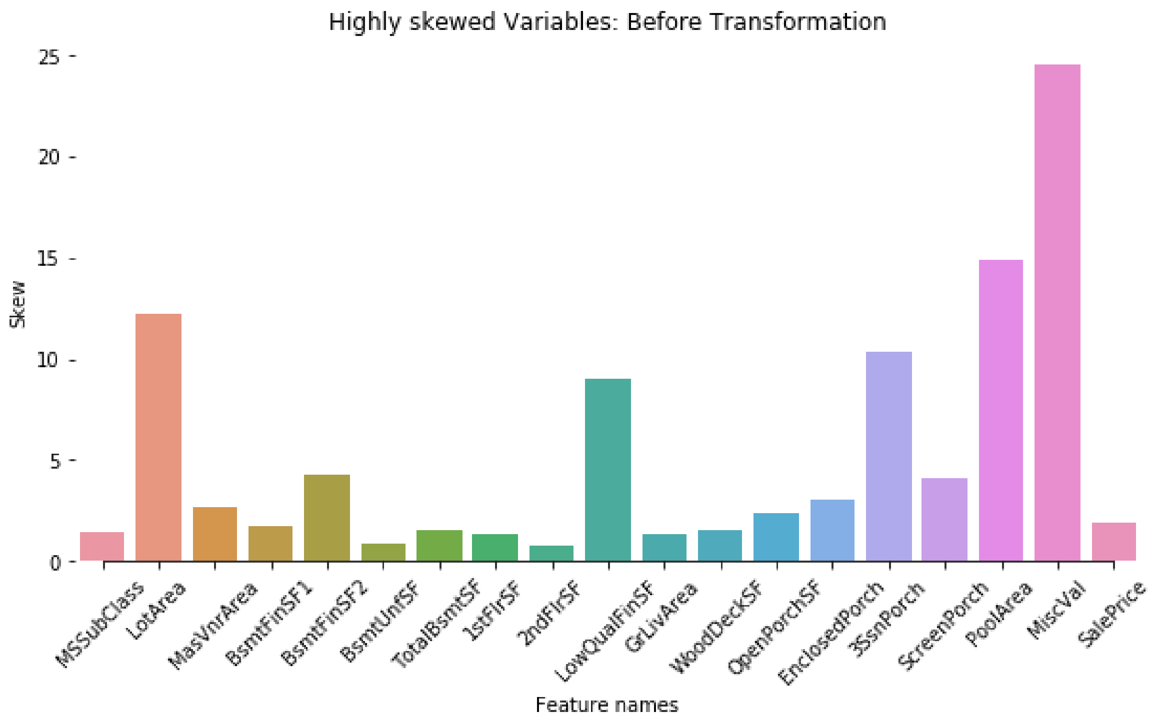
1. PoolQc: Missing PoolQc might suggest that there was no Pool in first Place. Lets fill na's with Missing
2. MiscFeature: we will drop the variable as there is>95% missing
3. Alley: 'Missing' will make more sense
4. Fence: 'Missing' will make more sense
5. FireplaceQu:-1 will make more sense
6. GarageYrBlt, lets fill with the most common Year
7. For other variables lets fill Na's with Missing

In [20]:

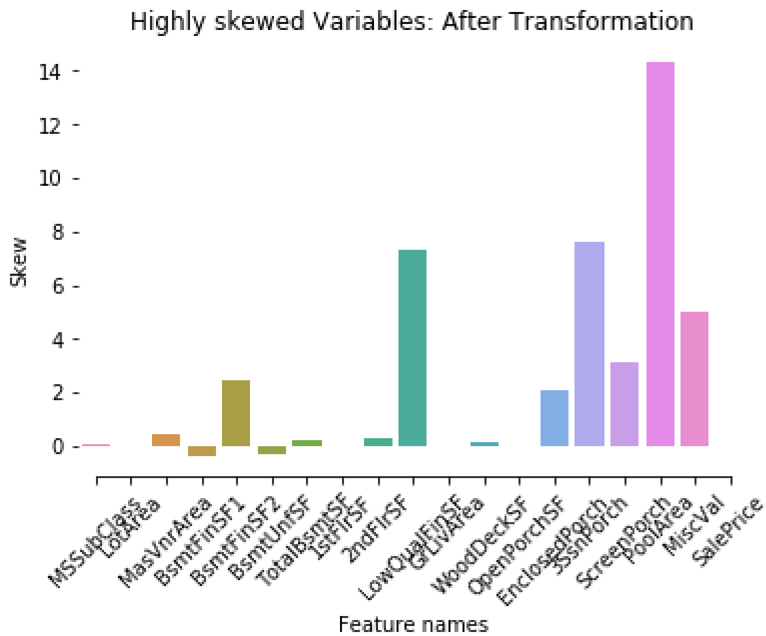
```
# Treating Missing Values
#Ordinal
missingValueImputation(data=df,categorical_cols=['PoolQC','GarageType','Alley
missingValueImputation(data=df,categorical_cols=['GarageYrBlt'],categorical_m
df.drop(columns='MiscFeature',inplace=True)
df.drop(columns='LotFrontage',inplace=True)
```

In [21]:
```
#skew Features
normalize(data=df,columns=df.select_dtypes(include=numerics).columns)
```






```
<Figure size 720x360 with 0 Axes>
```




```
<Figure size 720x360 with 0 Axes>
```

In [22]:

```
for col in ['MoSold','YrSold','TotRmsAbvGrd','OverallQual','OverallCond','Yea
    df[col]=df[col].astype(int)
```

Adding additional features for tree based algorithms

In [25]:

```
df['BsmtFinType1_Unf'] = 1*(df['BsmtFinType1'] == 'Unf')
df['HasWoodDeck'] = (df['WoodDeckSF'] == 0) * 1
df['HasOpenPorch'] = (df['OpenPorchSF'] == 0) * 1
df['HasEnclosedPorch'] = (df['EnclosedPorch'] == 0) * 1
df['Has3SsnPorch'] = (df['3SsnPorch'] == 0) * 1
df['HasScreenPorch'] = (df['ScreenPorch'] == 0) * 1
df['YearsSinceRemodel'] = df['YrSold'].astype(int) - df['YearRemodAdd'].astyp
df['Total_Home_Quality'] = df['OverallQual'].astype(int) + df['OverallCond'].
df = df.drop(['Utilities', 'Street', 'PoolQC',], axis=1)
df['TotalSF'] = df['TotalBsmtSF'] + df['1stFlrSF'] + df['2ndFlrSF']
df['YrBltAndRemod'] = df['YearBuilt'].astype(int) + df['YearRemodAdd'].astype
df['Total_sqr_footage'] = (df['BsmtFinSF1'] + df['BsmtFinSF2'] +df['1stFlrSF'
df['Total_Bathrooms'] = (df['FullBath'].astype(int) + (0.5 * df['HalfBath'])
df['Total_porch_sf'] = (df['OpenPorchSF'] + df['3SsnPorch'] +df['EnclosedPorc
df['TotalBsmtSF'] = df['TotalBsmtSF'].apply(lambda x: np.exp(6) if x <= 0.0 e
df['2ndFlrSF'] = df['2ndFlrSF'].apply(lambda x: np.exp(6.5) if x <= 0.0 else
df['GarageArea'] = df['GarageArea'].apply(lambda x: np.exp(6) if x <= 0.0 els
df['GarageCars'] = df['GarageCars'].apply(lambda x: 0 if x <= 0.0 else x)
df['MasVnrArea'] = df['MasVnrArea'].apply(lambda x: np.exp(4) if x <= 0.0 els
df['BsmtFinSF1'] = df['BsmtFinSF1'].apply(lambda x: np.exp(6.5) if x <= 0.0 e
df['haspool'] = df['PoolArea'].apply(lambda x: 1 if x > 0 else 0)
df['has2ndfloor'] = df['2ndFlrSF'].apply(lambda x: 1 if x > 0 else 0)
df['hasgarage'] = df['GarageArea'].apply(lambda x: 1 if x > 0 else 0)
df['hasbsmt'] = df['TotalBsmtSF'].apply(lambda x: 1 if x > 0 else 0)
df['hasfireplace'] = df['Fireplaces'].apply(lambda x: 1 if x > 0 else 0)
```

For simiplicity,

1. We are leaving ordinal variables as is
2. One hot encoding for other categorical variables

In [26]:

```
#get dummies
dummy = pd.get_dummies(df[df.select_dtypes(exclude=numerics).columns])
df=df.merge(dummy,left_index=True,right_index=True)
df.head()
```

Out[26]:

Oth	SaleType_WD	SaleCondition_Abnorml	SaleCondition_AdjLand	SaleCondition_Alloca	SaleCo
0	1	0	0	0	
0	1	0	0	0	
0	1	0	0	0	
0	1	1	0	0	
0	1	0	0	0	

Below are the functions used for the model

In [5]:

```
#missing value
def missing_values(data=None,influential_variable=None,label=None):
    total=data.isnull().sum()
    percent=(data.isnull().sum()/data.isnull().count()).sort_values(ascending
    missing_data=pd.concat([total,percent],axis=1,keys=['total','Percent'],so
    dat=missing_data.loc[missing_data['total']>0,:].reset_index().sort_values
    fig=plt.figure(figsize=(20,10))
    ax=sns.barplot(x='index',y='Percent',data=dat)
    ax.set(ylabel="Percent of missing Data")
    ax.set(xlabel="Features")
    ax.set(title="Missing Values (in Percent)")
    plt.xticks(rotation=45)
```

In [6]:

```
def numericToCategory(data=None,columns=None):

    for col in columns:
        data[col]=data[col].astype('category')
```

In [7]:

```
def missingValueImputation(data=None,numerical_col=None,categorical_cols=None


    number=[]
    cat=[]


    if numerical_col is not None:

        if numercial_method=='mean_imputation':
            for col in numerical_col:
                data[col]=data[col].fillna(data[col].mean())

        if numercial_method=='median_imputation':
            for col in numerical_col:
                data[col]=data[col].fillna(data[col].median())

        if numercial_method=='value_fill':
            for col in numerical_col:
                data[col]=data[col].fillna(numercial_value)

        for col in number:
            data=data.drop(col,inplace=True)


    if categorical_cols is not None:

        if categorical_method=='popular_imputation':
            for col in categorical_cols:
                data[col]=data[col].fillna(data[col].value_counts().index[0])

        if categorical_method=='value_fill':
            for col in categorical_cols:
                data[col]=data[col].fillna(categorical_value)

        for col in cat:
            data=data.drop(col,inplace=True)
```

In [8]:

```
n=[]
s=[]
def normalize(data=None, columns=None, std=True):



    if std==True:
        highly_skewed_col=data[columns].columns[df[columns].apply(lambda x: x
        f,ax=plt.subplots()
        plt.figure(figsize=(10,5))
        ax=sns.barplot(x=data[highly_skewed_col].columns,y=data[highly_skewed
        ax.xaxis.grid(False)
        ax.set(ylabel="Skew")
        ax.set(xlabel="Feature names")
        ax.set(title="Highly skewed Variables: Before Transformation")
        sns.despine(trim=True, left=True)
        plt.xticks(rotation=45)
        plt.figure(figsize=(10,5))
        plt.show()

        for cols in highly_skewed_col:
            power=boxcox_normmax(data[cols] + 1)
            n.append((cols,power))
            data[cols] = stats.boxcox(df[[cols]]+1)[0]

        scalar=StandardScaler().fit(df[columns])
        df[columns] = scalar.transform(df[columns])
        s.append(scalar)



    if std==False:

        highly_skewed_col=data[columns].columns[df[columns].apply(lambda x: x
        f,ax=plt.subplots()
        plt.figure(figsize=(10,5))
        ax=sns.barplot(x=data[highly_skewed_col].columns,y=data[highly_skewed
        ax.xaxis.grid(False)
        ax.set(ylabel="Skew")
        ax.set(xlabel="Feature names")
        ax.set(title="Highly skewed Variables: Before Transformation")
        sns.despine(trim=True, left=True)
        plt.xticks(rotation=45)
        plt.show()

        for cols in highly_skewed_col:
            power=boxcox_normmax(data[cols] + 1)
            n.append((cols,power))
            data[cols] = stats.boxcox(df[[cols]]+1)[0]

    ax=sns.barplot(x=data[highly_skewed_col].columns,y=data[highly_skewed_col
    ax.xaxis.grid(False)
    ax.set(ylabel="Skew")
    ax.set(xlabel="Feature names")
    ax.set(title="Highly skewed Variables: After Transformation")
    sns.despine(trim=True, left=True)
```

```
plt.xticks(rotation=45)
plt.figure(figsize=(10,5))
plt.show()
```