Written by Muhammed Yaseen

111701032

CSE | B.Tech | IIT-PKD

In [1]:
```
import numpy as np                          #for linear algebra
import matplotlib.pyplot as plt             #helper library for plotting
import pandas as pd                         #data processing with csv files
import seaborn as sns                       #library for statistical graphics

#add plots to the jupyter notebook
%matplotlib inline

import warnings
#from scipy import stats
from scipy.stats import norm, skew      #some math utilities
```

In [2]:
```
#Libraries for modelling
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
```

In [3]:
```
#load the train and test data provided into a pandas dataframe from the csv fil

train = pd.read_csv('houseprice_train.csv')

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

In [4]:
```
train.info()        #metadata of train dataframe
```

```
<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 
```

In [5]:
```
train.describe() #data values with some relavant statistics of the train data
```

Out[5]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	Yea
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2

8 rows × 38 columns

In [6]:
```
test.info() #metadata of test data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1459 entries, 0 to 1458
Data columns (total 80 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             1459 non-null   int64  
 1   MSSubClass     1459 non-null   int64  
 2   MSZoning       1455 non-null   object 
 3   LotFrontage    1232 non-null   float64
 4   LotArea        1459 non-null   int64  
 5   Street         1459 non-null   object 
 6   Alley          107 non-null    object 
 7   LotShape       1459 non-null   object 
 8   LandContour    1459 non-null   object 
 9   Utilities      1457 non-null   object 
 10  LotConfig      1459 non-null   object 
 11  LandSlope      1459 non-null   object 
 12  Neighborhood   1459 non-null   object 
 13  Condition1     1459 non-null   object 
 14  Condition2     1459 non-null   object 
 15  BldgType       1459 non-null   object 
 16  HouseStyle     1459 non-null   object 
 17  OverallQual    1459 non-null   int64  
 18  OverallCond    1459 non-null   int64  
 19  YearBuilt      1459 non-null   int64  
 20  YearRemodAdd   1459 non-null   int64  
 21  RoofStyle      1459 non-null   object 
 22  RoofMatl       1459 non-null   object 
 23  Exterior1st    1458 non-null   object 
 24  Exterior2nd    1458 non-null   object 
 25  MasVnrType     1443 non-null   object 
 26  MasVnrArea     1444 non-null   float64
 27  ExterQual      1459 non-null   object 
 28  ExterCond      1459 non-null   object 
 29  Foundation     1459 non-null   object 
 30  BsmtQual       1415 non-null   object 
 31  BsmtCond       1414 non-null   object 
 32  BsmtExposure   1415 non-null   object 
 33  BsmtFinType1   1417 non-null   object 
 34  BsmtFinSF1     1458 non-null   float64
 35  BsmtFinType2   1417 non-null   object 
 36  BsmtFinSF2     1458 non-null   float64
 37  BsmtUnfSF      1458 non-null   float64
 38  TotalBsmtSF    1458 non-null   float64
 39  Heating        1459 non-null   object 
 40  HeatingQC      1459 non-null   object 
 41  CentralAir     1459 non-null   object 
 42  Electrical     1459 non-null   object 
 43  1stFlrSF       1459 non-null   int64  
 44  2ndFlrSF       1459 non-null   int64  
 45  LowQualFinSF   1459 non-null   int64  
 46  GrLivArea      1459 non-null   int64  
 47  BsmtFullBath   1457 non-null   float64
 48  BsmtHalfBath   1457 non-null   float64
 49  FullBath       1459 non-null   int64  
 50  HalfBath       1459 non-null   int64  
 51  BedroomAbvGr   1459 non-null   int64  
 52  KitchenAbvGr   1459 non-null   int64  
 53  KitchenQual    1458 non-null   object 
 54  TotRmsAbvGrd   1459 non-null   int64  
 55  Functional     1457 non-null   object 
 56  Fireplaces     1459 non-null   int64  
 57  FireplaceQu    729 non-null    object 
 58  GarageType     1383 non-null   object 
 59  GarageYrBlt    1381 non-null   float64
 60  GarageFinish   1381 non-null   object 
```

In [7]:
```
test.describe()       ##data values with some relavant statistics of the test da
```

Out[7]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearR
count	1459.000000	1459.000000	1232.000000	1459.000000	1459.000000	1459.000000	1459.000000	14
mean	2190.000000	57.378341	68.580357	9819.161069	6.078821	5.553804	1971.357779	19
std	421.321334	42.746880	22.376841	4955.517327	1.436812	1.113740	30.390071	
min	1461.000000	20.000000	21.000000	1470.000000	1.000000	1.000000	1879.000000	19
25%	1825.500000	20.000000	58.000000	7391.000000	5.000000	5.000000	1953.000000	19
50%	2190.000000	50.000000	67.000000	9399.000000	6.000000	5.000000	1973.000000	19
75%	2554.500000	70.000000	80.000000	11517.500000	7.000000	6.000000	2001.000000	20
max	2919.000000	190.000000	200.000000	56600.000000	10.000000	9.000000	2010.000000	20

8 rows × 37 columns

In [8]:
```
print ("Size of train data : {}" .format(train.shape))

print ("Size of test data : {}" .format(test.shape))

#train data has 81 features
#test data has 80 features, excluding SalePrice
#train data has 1460 data samples
#test data has 1459 samples
```

```
Size of train data : (1460, 81)
Size of test data : (1459, 80)
```

In [9]:
```
# a utility function to check the skewness of a feature
def check_skewness(col):
    sns.distplot(train[col] , fit=norm);   #A distplot plots a univariate distr
    fig = plt.figure()
    (mu, sigma) = norm.fit(train[col])     #calculate mean and std.dev of the p
    print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))
```

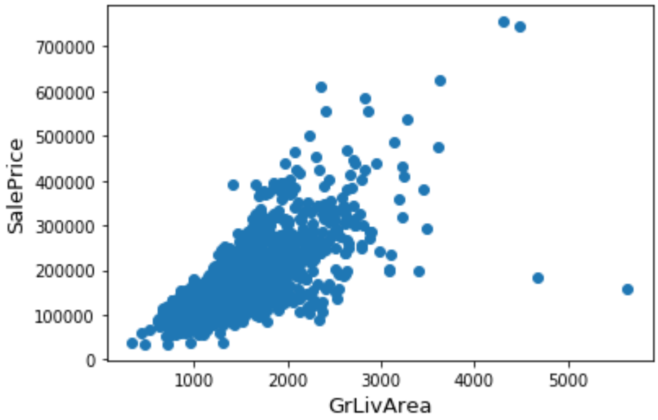
Cleaning the data

In [10]:
```
#Save the 'Id' column for future use
train_ID = train['Id']
test_ID = test['Id']

#'Id' is dropped as it is irrelevant in the prediction process
train.drop("Id", axis = 1, inplace = True)
test.drop("Id", axis = 1, inplace = True)
```

In [11]:
```
print ("Size of train data after dropping Id: {}" .format(train.shape))
print ("Size of test data after dropping Id: {}" .format(test.shape))

#train has 80 features, test has 79 features excluding SalePrice
```

```
Size of train data after dropping Id: (1460, 80)
Size of test data after dropping Id: (1459, 79)
```

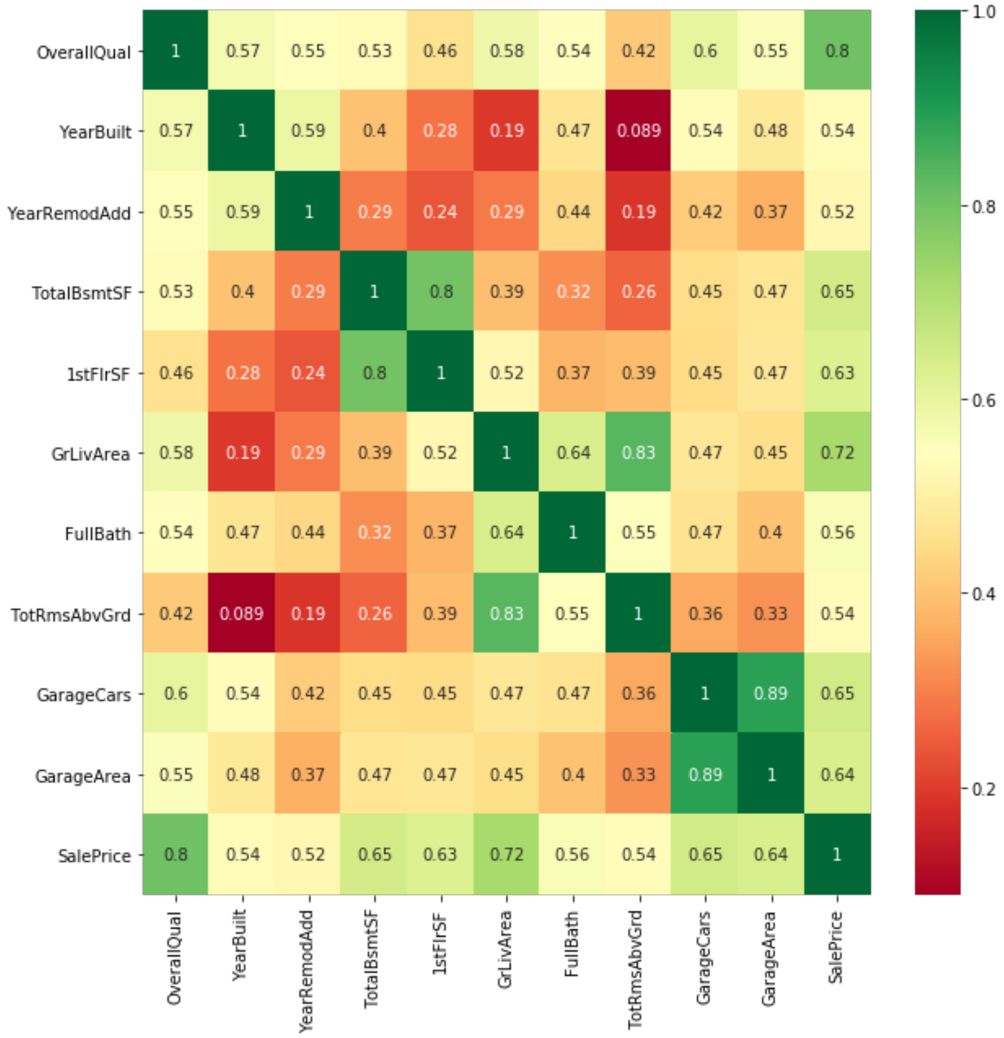
In [12]:
```
'''

Dealing with outliers

Outlinear in the GrLivArea is recommended by the author of the data to remove i

Ref: Ames housing dataset http://jse.amstat.org/v19n3/decock.pdf

'''

fig, ax = plt.subplots()
ax.scatter(x = train['GrLivArea'], y = train['SalePrice'])
plt.ylabel('SalePrice', fontsize=13)
plt.xlabel('GrLivArea', fontsize=13)
plt.show()
```



Please note the outliers towards the right.

In [13]:
```
train = train[train['GrLivArea'] < 4000]
#train data is rid of the samples having GrLivArea >= 4000
```

In [14]:
```
#check for correlation with the features
corrmat = train.corr()
#the features having corr > 0.5 and corr < -0.5 are considered seperately
top_corr_features = corrmat.index[abs(corrmat["SalePrice"])>0.5]
plt.figure(figsize=(10,10))
#plot the correlation map
g = sns.heatmap(train[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



Some points to note:

1. OverallQual, GrLivArea, TotalBsmtSF have maximum correlation with SalePrice

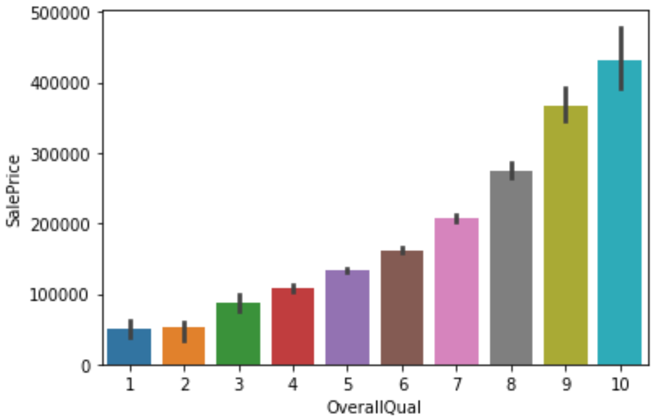
2. GarageArea and GarageCars have highest correlation, which can be understood from their nature. A similar pattern is observed across such data feature pairs.

In [15]:
```
#print the top_corr_feature types (for our knowledge)
print(train[top_corr_features].dtypes)
#we can see that all are numerical types
```

```
OverallQual     int64
YearBuilt       int64
YearRemodAdd    int64
TotalBsmtSF     int64
1stFlrSF        int64
GrLivArea       int64
FullBath        int64
TotRmsAbvGrd    int64
GarageCars      int64
GarageArea      int64
SalePrice       int64
dtype: object
```

In [16]:
```
#A barplot to see how OverallQual depends on the SalePrice
sns.barplot(train.OverallQual,train.SalePrice)
```

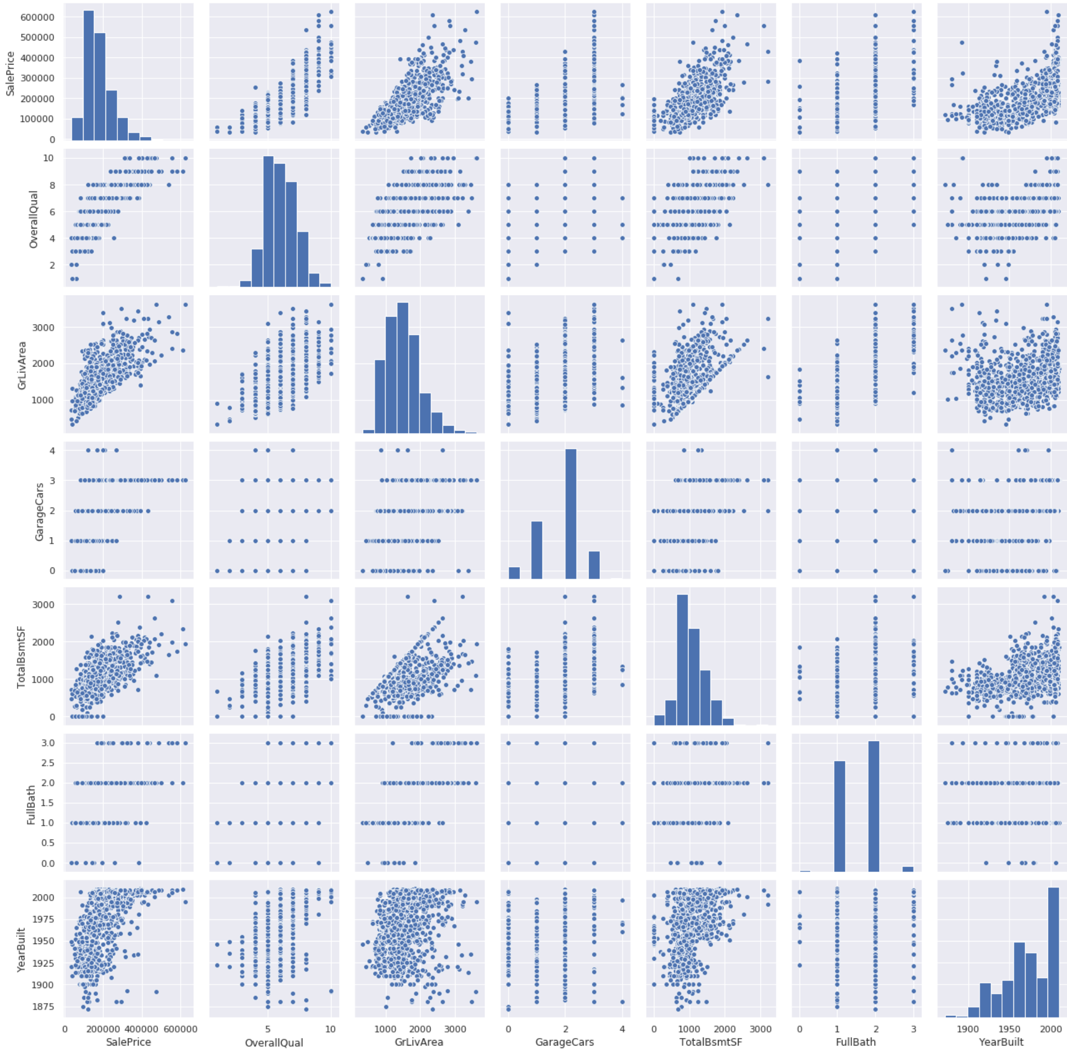
Out[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fee5c46f5d0>
```



Almost a linear variance as it should be from the correlation plot

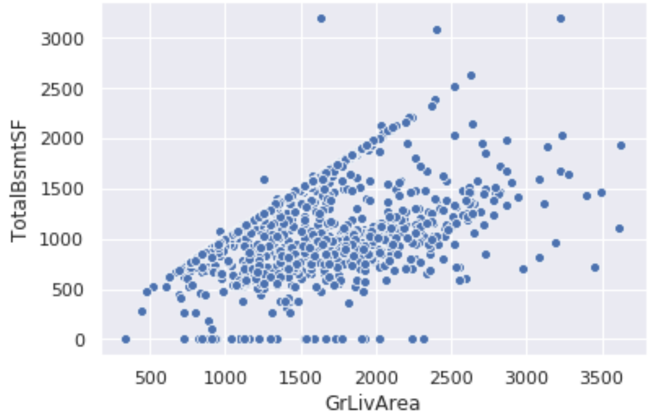
In [17]:
```
#Scatter plots between 'SalePrice' and correlated variables
print(top_corr_features)
sns.set()
cols = ['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'TotalBsmtSF', '
sns.pairplot(train[cols], height = 2.5)
plt.show();
```

```
Index(['OverallQual', 'YearBuilt', 'YearRemodAdd', 'TotalBsmtSF', '1stFlrSF',
       'GrLivArea', 'FullBath', 'TotRmsAbvGrd', 'GarageCars', 'GarageArea',
       'SalePrice'],
      dtype='object')
```



In [18]:
```
sns.scatterplot(train.GrLivArea,train.TotalBsmtSF)
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x7fee59609f90>



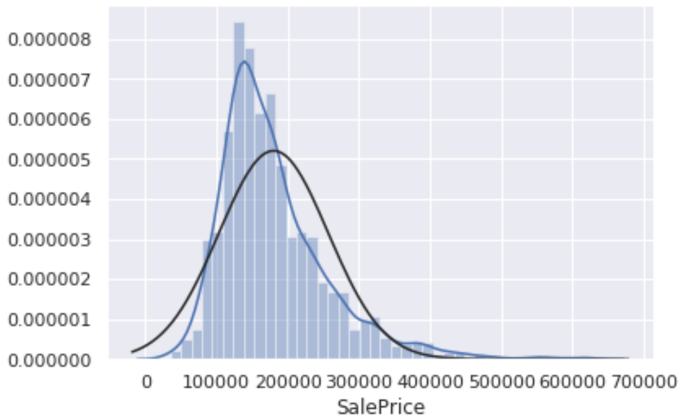
It can be observed that GrLivArea(Above grade (ground) living area square feet) is greater than

TotalBsmtSF(Total square feet of basement area)

in most of the cases

In [19]:
```
check_skewness('SalePrice')
```

```
 mu = 180151.23 and sigma = 76670.25
```



<Figure size 432x288 with 0 Axes>

SalePrice or our target variable is skewed towards the left of the median as it is evident from the above plot.

Regression model will not work if it is not a normal distribution as it can affect the parameter calculations, the above plot is skewed right so we need to normalize it. We use a log normalisation.

In [20]:
```
#We use the numpy fuction log1p which  applies log(1+x) to all elements of the
train["SalePrice"] = np.log1p(train["SalePrice"])

check_skewness('SalePrice')
```

```
 mu = 12.02 and sigma = 0.40
```


```
<Figure size 432x288 with 0 Axes>
```

We can see that the data has become fairly normal

If you log transform the response variable, it is required to also log transform feature variables that are skewed. We will do it at a later stage.

Feature Engineering

We are using some data mining techniques to make the data prediction more accurate.

We are concatinating test data and train data and are also remembering their respective sizes.

This is necessary as all the data mining techniques done to train data should be done to the test data too.

In [21]:
```
ntrain = train.shape[0]
ntest = test.shape[0]
y_train = train.SalePrice.values      # y_train is the target variable i.e. Sal
all_data = pd.concat((train, test)).reset_index(drop=True)
all_data.drop(['SalePrice'], axis=1, inplace=True)    #SalePrice from train data
print("all_data size is : {}".format(all_data.shape))
```

```
all_data size is : (2915, 79)
```

Handling missing data

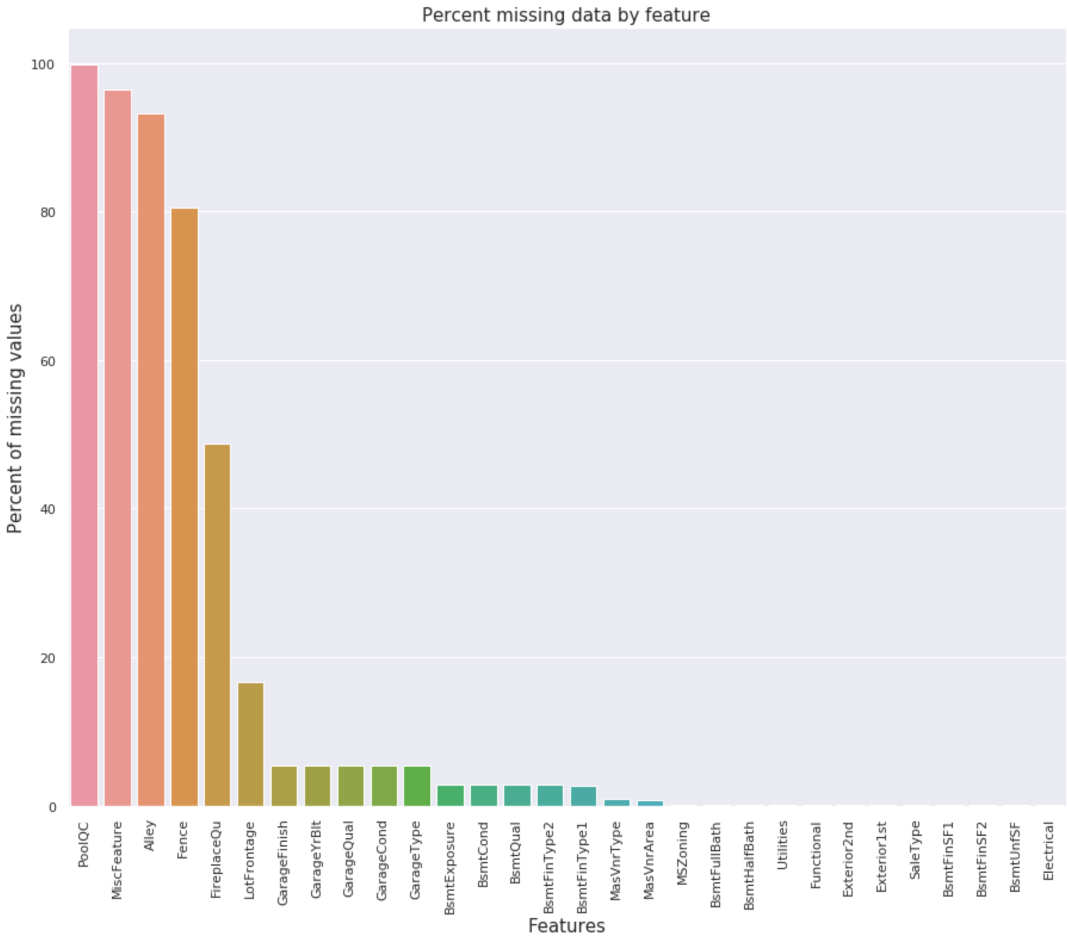
In [22]:
```
all_data_na = (all_data.isnull().sum() / len(all_data)) * 100  #percentage of m
all_data_na = all_data_na.drop(all_data_na[all_data_na == 0].index).sort_values
missing_data = pd.DataFrame({'Missing Ratio' :all_data_na}) #missing data is a
missing_data
```

Out[22]:

	Missing Ratio
PoolQC	99.725557
MiscFeature	96.397942
Alley	93.207547
Fence	80.445969
FireplaceQu	48.713551
LotFrontage	16.672384
GarageFinish	5.454545
GarageYrBlt	5.454545
GarageQual	5.454545
GarageCond	5.454545
GarageType	5.385935
BsmtExposure	2.813036
BsmtCond	2.813036
BsmtQual	2.778731
BsmtFinType2	2.744425
BsmtFinType1	2.710120
MasVnrType	0.823328
MasVnrArea	0.789022
MSZoning	0.137221
BsmtFullBath	0.068611
BsmtHalfBath	0.068611
Utilities	0.068611
Functional	0.068611
Exterior2nd	0.034305
Exterior1st	0.034305
SaleType	0.034305
BsmtFinSF1	0.034305
BsmtFinSF2	0.034305
BsmtUnfSF	0.034305
Electrical	0.034305

In [23]:
```
#plot missing data percentages (to make a relative comparison)
f, ax = plt.subplots(figsize=(15, 12))
plt.xticks(rotation='90')
sns.barplot(x=all_data_na.index, y=all_data_na)
plt.xlabel('Features', fontsize=15)
plt.ylabel('Percent of missing values', fontsize=15)
plt.title('Percent missing data by feature', fontsize=15)
```

Out[23]:
```
Text(0.5, 1.0, 'Percent missing data by feature')
```



In [24]:
```
all_data = all_data.drop(['PoolQC'], axis=1)

# As nearly 100% of PoolQC are missing, we can safely drop that feature
```

In [25]:
```
all_data["Alley"] = all_data["Alley"].fillna("None")

#From the data description, NA means No alley access
```

In [26]:
```
all_data["MiscFeature"] = all_data["MiscFeature"].fillna("None")

#NA means None from the data description
```

In [27]:
```
all_data["Fence"] = all_data["Fence"].fillna("None")

#NA means No Fence from the data description
```

In [28]:
```
all_data["FireplaceQu"] = all_data["FireplaceQu"].fillna("None")

#NA means No Fireplace from the data description
```

In [29]:
```
all_data["LotFrontage"] = all_data.groupby("Neighborhood")["LotFrontage"].trans

#LotFrontage(Linear feet of street connected to property) is assumed to be the
```

In [30]:
```
for col in ['GarageType', 'GarageFinish', 'GarageQual', 'GarageCond']:
    all_data[col] = all_data[col].fillna('None')

#NA means No Garage according to the data description
```

In [31]:
```
for col in ('GarageYrBlt', 'GarageArea', 'GarageCars'):
    all_data[col] = all_data[col].fillna(0)

#Numerical features. They might be missing because garage may not be present at
```

In [32]:
```
for col in ('BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF','TotalBsmtSF', 'BsmtFullBat
    all_data[col] = all_data[col].fillna(0)

#Similar explanation as the above one
```

In [33]:
```
for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinTyp
    all_data[col] = all_data[col].fillna('None')

#Categorical features, similar explanation as above
```

In [34]:
```
all_data["MasVnrType"] = all_data["MasVnrType"].fillna("None")
all_data["MasVnrArea"] = all_data["MasVnrArea"].fillna(0)

#MasVnrArea: Masonry veneer area in square feet
#Similar explanation
```

In [35]:
```
all_data['Functional'].value_counts()
```

Out[35]:
```
Typ     2713
Min2      70
Min1      65
Mod       35
Maj1      19
Maj2       9
Sev        2
Name: Functional, dtype: int64
```

In [36]:
```
all_data["Functional"] = all_data["Functional"].fillna("Typ")

#Home functionality: Replaced with Typ, the most occured value
```

In [37]:
```
mode_col = ['Electrical','KitchenQual', 'Exterior1st', 'Exterior2nd', 'SaleType
for col in mode_col:
    all_data[col] = all_data[col].fillna(all_data[col].mode()[0])

#Similar to functionality, these features are replaced with their mode values
```

In [38]:
```
all_data['MSSubClass'] = all_data['MSSubClass'].fillna("None")
```

In [39]:
```
all_data['MSZoning'] = all_data['MSZoning'].fillna(all_data['MSZoning'].mode()[
```

In [40]:
```
#A sanity check for missing data to confirm
all_data_na = (all_data.isnull().sum() / len(all_data)) * 100
all_data_na = all_data_na.drop(all_data_na[all_data_na == 0].index).sort_values
missing_data = pd.DataFrame({'Missing Ratio' :all_data_na})
missing_data.head()

missing_data
```

Out[40]:

	Missing Ratio
Utilities	0.068611

We have accounted for all the missing features as the above dataframe is empty.

Removing redundant data (these were identified from the skew)

In [41]:
```
all_data['Utilities'].value_counts()
```

Out[41]:
```
AllPub    2912
NoSeWa       1
Name: Utilities, dtype: int64
```

In [42]:
```
all_data['MiscVal'].value_counts()
```

Out[42]:
```
0        2812
400        18
500        13
450         9
600         8
700         7
2000        7
650         3
1200        3
1500        3
4500        2
2500        2
480         2
3000        2
12500       1
300         1
350         1
8300        1
420         1
80          1
54          1
460         1
490         1
3500        1
560         1
17000       1
15500       1
750         1
800         1
900         1
1000        1
1150        1
1300        1
1400        1
1512        1
6500        1
455         1
620         1
Name: MiscVal, dtype: int64
```

In [43]:
```
all_data = all_data.drop(['Utilities'], axis=1)
all_data = all_data.drop(['MiscVal'], axis=1)

#Except for one, all the other samples have the same value.
#Irrelevant for analysis, dropping it
```

Accounting for categorical features

The data in the following group, all have categorical variables disguised in number format.

We have to change them to the string type

In [44]:
```
#MSSubClass=The building class
all_data['MSSubClass'] = all_data['MSSubClass'].apply(str)

#Changing OverallCond into a categorical variable
all_data['OverallCond'] = all_data['OverallCond'].astype(str)

#Year and month sold are transformed into categorical features.
all_data['YrSold'] = all_data['YrSold'].astype(str)
all_data['MoSold'] = all_data['MoSold'].astype(str)
```

LabelEncoder encode labels with a value between 0 and n_classes-1 where n is the number of distinct labels.

If a label repeats it assigns the same value to as assigned earlier.

In [45]:
```
from sklearn.preprocessing import LabelEncoder
cols = ('FireplaceQu', 'BsmtQual', 'BsmtCond', 'GarageQual', 'GarageCond',
        'ExterQual', 'ExterCond','HeatingQC', 'KitchenQual', 'BsmtFinType1',
        'BsmtFinType2', 'Functional', 'Fence', 'BsmtExposure', 'GarageFinish',
        'LotShape', 'PavedDrive', 'Street', 'Alley', 'CentralAir', 'MSSubClass'
        'YrSold', 'MoSold')

# process columns, apply LabelEncoder to categorical features
for c in cols:
    lbl = LabelEncoder()
    lbl.fit(list(all_data[c].values))
    all_data[c] = lbl.transform(list(all_data[c].values))

# shape
print('Shape all_data: {}'.format(all_data.shape))
```

```
Shape all_data: (2915, 76)
```

In [46]:
```
# Adding total sqfootage feature , Usually houses are categorised by area
all_data['TotalSF'] = all_data['TotalBsmtSF'] + all_data['1stFlrSF'] + all_data
```

In [47]:
```
numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index

# Check the skew of all numerical features using skew utility
skewed_feats = all_data[numeric_feats].apply(lambda x: skew(x.dropna())).sort_v
print("\nSkew in numerical features: \n")
skewness = pd.DataFrame({'Skew' :skewed_feats})
skewness.head(15)
#top 15
```

```
Skew in numerical features:
```

Out[47]:

	Skew
PoolArea	18.701829
LotArea	13.123758
LowQualFinSF	12.080315
3SsnPorch	11.368094
LandSlope	4.971350
KitchenAbvGr	4.298845
BsmtFinSF2	4.142863
EnclosedPorch	4.000796
ScreenPorch	3.943508
BsmtHalfBath	3.942892
MasVnrArea	2.600697
OpenPorchSF	2.529245
WoodDeckSF	1.848285
1stFlrSF	1.253011
LotFrontage	1.092709

In [48]:
```
#applying log tranformation where skewness > 0.75 and skewness < -0.75
skewness = skewness[abs(skewness) > 0]
skewed_features = skewness.index
for feat in skewed_features:
    all_data[feat] = np.log1p(all_data[feat])
```

In [49]:
```
numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index

# Check the skew of all numerical features
skewed_feats = all_data[numeric_feats].apply(lambda x: skew(x.dropna())).sort_v
print("\nSkew in numerical features: \n")
skewness = pd.DataFrame({'Skew' :skewed_feats})
skewness[abs(skewness) > 0.75]
#print(skewness[skewness['']=='SalePrice'].index.values)
```

```
Skew in numerical features:
```

Out[49]:

	Skew
PoolArea	16.332187
3SsnPorch	8.818976
LowQualFinSF	8.551587
LandSlope	4.480719
BsmtHalfBath	3.785015
KitchenAbvGr	3.517415
ScreenPorch	2.943234
BsmtFinSF2	2.460035
EnclosedPorch	1.958822
HalfBath	NaN
MasVnrArea	NaN
BsmtFullBath	NaN
2ndFlrSF	NaN
HeatingQC	NaN
Fireplaces	NaN
WoodDeckSF	NaN
TotRmsAbvGrd	NaN
1stFlrSF	NaN
OpenPorchSF	NaN
GrLivArea	NaN
FullBath	NaN
TotalSF	NaN
YearRemodAdd	NaN
YrSold	NaN
LotArea	NaN
BsmtFinSF1	NaN
BsmtFinType1	NaN
YearBuilt	NaN
LotShape	NaN
OverallQual	NaN
GarageFinish	-0.894539
BedroomAbvGr	-0.982308
LotFrontage	-1.076566
FireplaceQu	-1.105723

A noticeable improvement is seen with the skewed data

In [50]:
```
all_data = pd.get_dummies(all_data)
all_data.shape
```
Out[50]:
```
(2915, 218)
```

Earlier, we had concatenated train and test data for feature engineering purposes

We have to split it back to the original form for modelling purposes

In [51]:
```
train = all_data[:ntrain]
test = all_data[ntrain:]
train.shape
```
Out[51]:
```
(1456, 218)
```

Modelling

As the problem involves predicting a variable wrt. other variables, we will use mutivariate linear regression models.

Also, to tackle the cases of overfitting, we arrive at Lasso and ridge regressions to choose from.

The methods are chosen only from ones those were covered in the class.

best_score_: Mean cross-validated score of the best_estimator. Score here means the R2 score.

In [52]:
```
linear_reg = LinearRegression(normalize = True)
parameters = [0.0001, 0.001, 0.003, 0.009, 0.01, 0.03, 0.06, 0.09, 0.1, 0.5, 1,
param_grid = {'alpha' : parameters}
linear_cv = GridSearchCV(linear_reg, param_grid = {}, cv = 3)

linear_cv.fit(train, y_train)
# Print the tuned parameters and score
print("Result of OLS Regression:\n")
#print("Tuned Logistic Regression Parameters: {}".format(linear_cv.best_params_
print("Best score is {}".format(linear_cv.best_score_))
```
```
Result of OLS Regression:

Best score is -1.502877791201857e+22
```

In [53]:
```
pd.DataFrame(linear_cv.cv_results_)
```
Out[53]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	params	split0_test_score	split1_test_scor
0	0.245425	0.282643	0.008662	0.009419	{}	-4.479660e+22	-8.851508e+1

OLS is discarded due to the negative score.

Regression is carried out with GridSearchCV utility function which performs a cross validation and selects the hyperparameter 'alpha'. Here, the cross validation is 3-fold (project guideline instructs to choose a roughly 70-30 train-test split).

Lasso Regression

In [54]:
```
lasso = Lasso()
parameters = [0.0001, 0.001, 0.003, 0.009, 0.01, 0.03, 0.06, 0.09, 0.1, 0.5, 1,
param_grid = {'alpha' : parameters}
# Instantiating the GridSearchCV object
lasso_cv = GridSearchCV(lasso, param_grid, cv = 3)

lasso_cv.fit(train, y_train)

# Print the tuned parameters and score
print("Result of Lasso Regression:\n")
print("Tuned Lasso Regression Parameters: {}".format(lasso_cv.best_params_))
print("Best score is {}".format(lasso_cv.best_score_))
```

```
Result of Lasso Regression:

Tuned Logistic Regression Parameters: {'alpha': 0.001}
Best score is 0.9112493923128753
```

Lasso feature extraction

In [55]:
```
null_coeffs = pd.Series(lasso_cv.best_estimator_.coef_, index=train.columns)
res = null_coeffs.to_list()
zero_count = 0
for val in res:
    if val == 0:
        zero_count += 1
print("Lasso eliminated ",zero_count," variables out of ",len(res)," variables"
```

```
Lasso eliminated  147  variables out of  218  variables
```

Ridge Regression

In [56]:
```
ridge = Ridge()
parameters = [0.0001, 0.001, 0.003, 0.009, 0.01, 0.03, 0.06, 0.09, 0.1, 0.5, 1,
param_grid = {'alpha' : parameters}
# Instantiating the GridSearchCV object
ridge_cv = GridSearchCV(ridge, param_grid, cv = 3)

ridge_cv.fit(train, y_train)

# Print the tuned parameters and score
print("Tuned Ridge Regression Parameters: {}".format(ridge_cv.best_params_))
print("Best score is {}".format(ridge_cv.best_score_))
```

```
Tuned Logistic Regression Parameters: {'alpha': 5}
Best score is 0.9118340189222525
```

In [57]:
```
#lasso performs better
lasso_cv.fit(train,y_train)
ridge_cv.fit(train,y_train)
```

Out[57]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=Non
e,
                             solver='auto', tol=0.001),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [0.0001, 0.001, 0.003, 0.009, 0.01, 0.03,
                                   0.06, 0.09, 0.1, 0.5, 1, 5, 10, 20, 50,
                                   100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

As LASSO and RIDGE models perform more or less similar, we shall take a weighted average of them to be the final predictor. Since, RIDGE has a marginally high score, we are giving a higher priority for RIDGE model.

Finally, SalePrice is converted back to its original form from the logarithmic transformed form

In [58]:
```
final_model = (0.45*np.expm1(lasso_cv.predict(test)) + 0.55*np.expm1(ridge_cv.p
```

In [59]:
```
sub = pd.DataFrame()
sub['Id'] = test_ID
sub['SalePrice'] = final_model
sub.to_csv('submission_project_final.csv',index=False)
```

In [60]:
```
#### The entry got top 22% in global Kaggle Leaderboard with a rank 1084 and a
#### https://kaggle.com/c/house-prices-advanced-regression-techniques
#### Improvements are possible with advanced regression techniques
```