HousingRegression

December 28, 2020

[4]:
```
#invite people for the Kaggle party
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm
from sklearn.preprocessing import StandardScaler
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

[38]:
```
df_train = pd.read_csv('/Users/abhishekvijay/Documents/kaggle/housing/train.
 ↪csv')
#imports the file path
```

[6]:
```
df_train.columns
#displays the columns in the data
```

[6]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

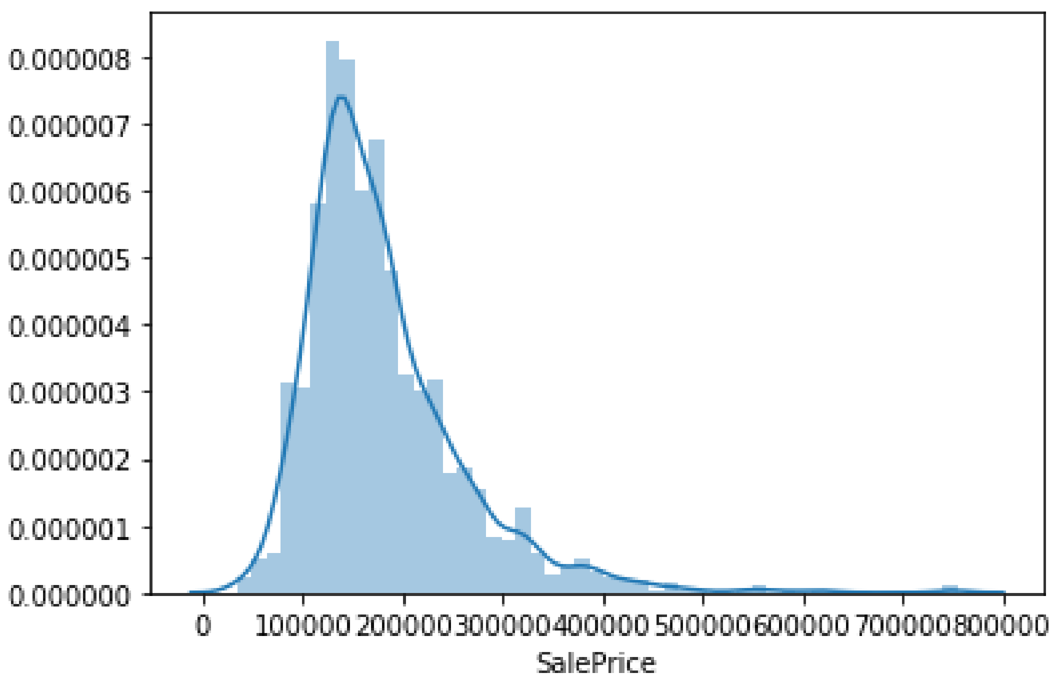
[7]:
```
df_train['SalePrice'].describe()
#describes the data with a series of output that will important to understand␣
↪the data
```

[7]:
```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

[92]:
```
#Theres no 0 values that would potentially ruin the model
```

[8]:
```
sns.distplot(df_train['SalePrice'])
#using seaborn to display a distrobution plot
```

[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f999bde39d0>
```



[38]:
```
#deviates from the normal distribution
#positive skewness: tail end on the right side is longer
#has a peak
```

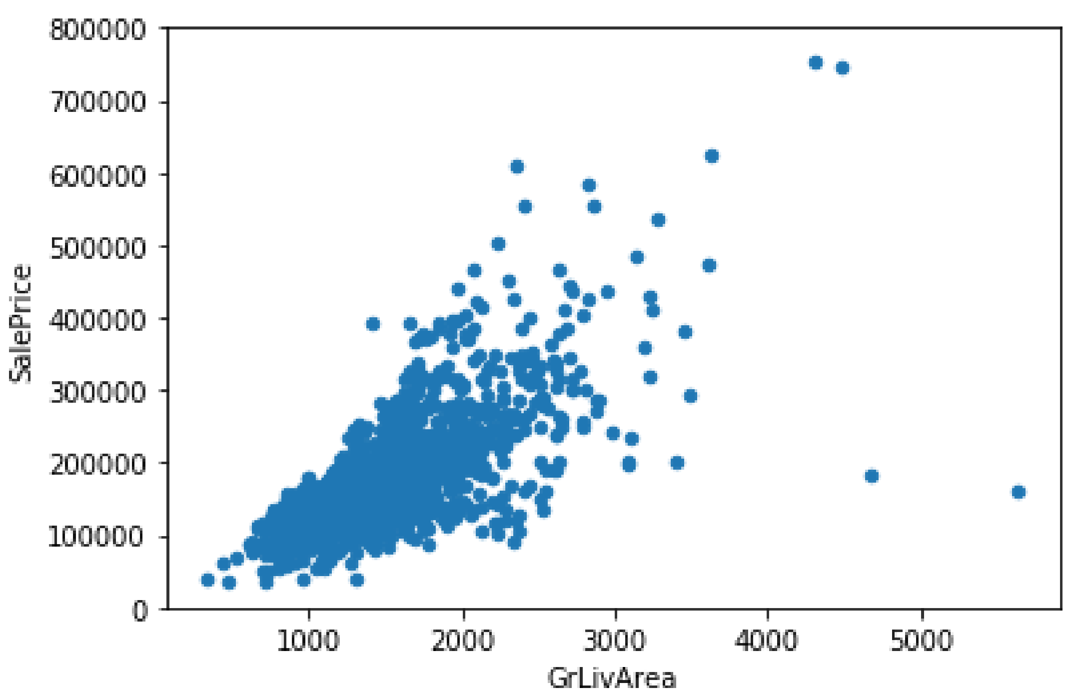
[9]:
```
print("skewness: %f" % df_train['SalePrice'].skew())
print("kurtosis: %f" % df_train['SalePrice'].kurt())
#display skewness, still need to figure out %f meaning fully
```

```
skewness: 1.882876
kurtosis: 6.536282
```

[40]:
```
#skewness is between >1 which means the data is highly skewed

#kurtosis is >3 AKA leptokurtic which means the data is heavy tailed with␣
↪outliers shown by
#the narrowness of the right tail end
```

[10]:
```
#lets see the relationship between saleprice and ground living room area

var = 'GrLivArea'
data = pd.concat([df_train['SalePrice'],df_train[var]], axis = 1)
#concat used to merge data from saleprice and grlivearea
data.plot.scatter(x=var,y='SalePrice', ylim=(0,800000))
#taking the merged data aka var and displaying it as scatter plot
#by setting the x and y axis variables and setting a limit on the y axis
```

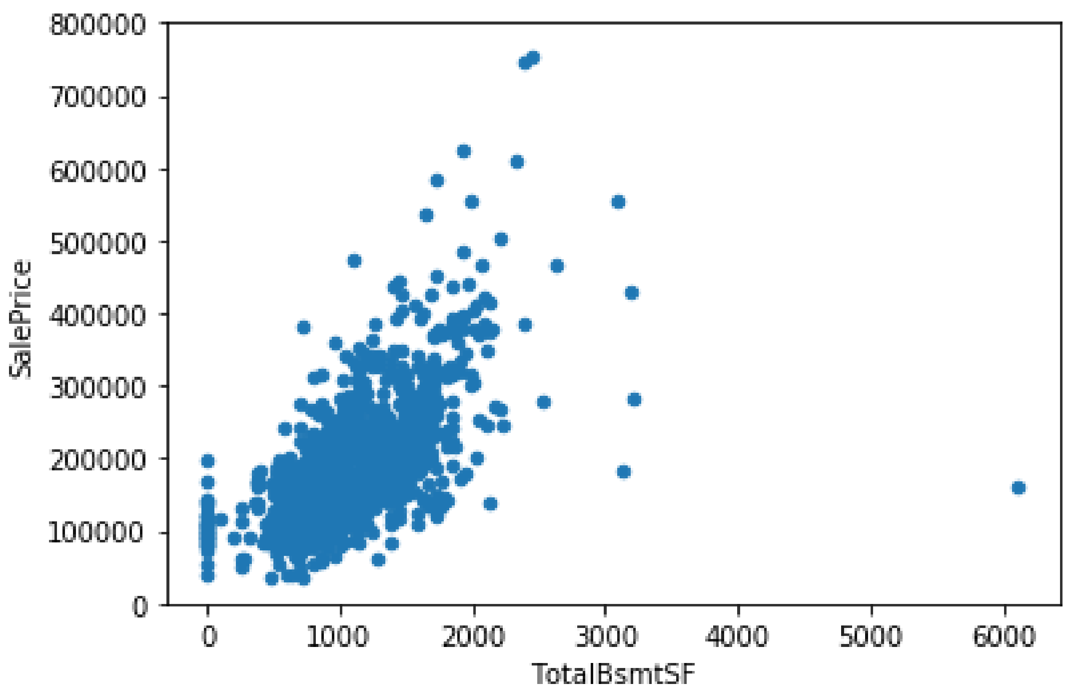
[10]: <matplotlib.axes._subplots.AxesSubplot at 0x7f999bfb1d50>



[42]:
```
#there is a linear relationship between saleprice and GrLivArea

#lets see the relationship between saleprice and total basement square foot␣
↪TotalBsmtSF
```

[11]:
```
var = 'TotalBsmtSF'
data = pd.concat([df_train['SalePrice'],df_train[var]], axis=1)
data.plot.scatter(x=var,y='SalePrice',ylim=(0,800000))
```

[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7f999c955790>

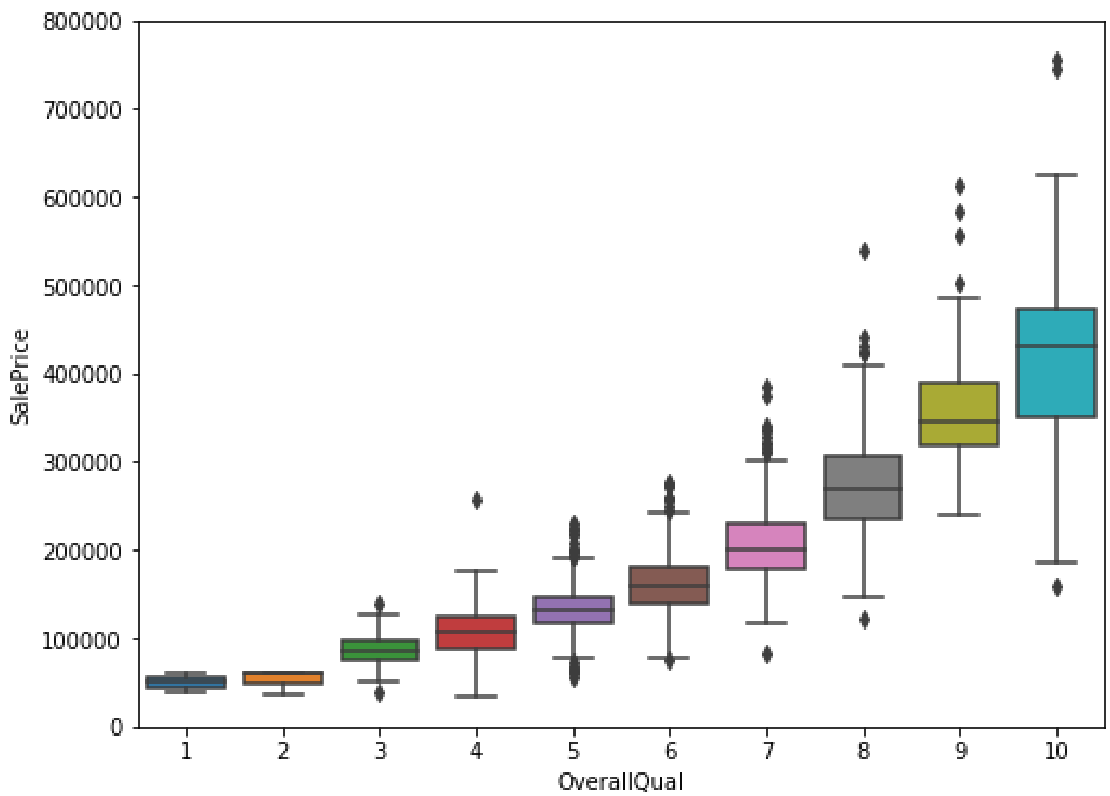


[44]:
```
#strong linear relationship based on the sudden exponential increase
#looks like having ~ 500-1500 sq doesnt impact salesprice

#lets conduct analysis on the categorical features (not numerical descriptive␣
↪traits)
```

[12]:
```
var = 'OverallQual'
#define OverallQual as the variable being used
data = pd.concat([df_train['SalePrice'],df_train[var]],axis = 1)
#concat the data of salesprice and overall quality using pandas
f, ax = plt.subplots(figsize=(8,6))
#plt.sublot is a function that returns a tuple containing a figure and axes␣
↪object(s)
```

```
#tuple is unpacked into the f and ax variables
fig = sns.boxplot(x=var,y='SalePrice', data=data)
#created the boxplot
fig.axis(ymin=0,ymax=800000)
#setting the axis size
```

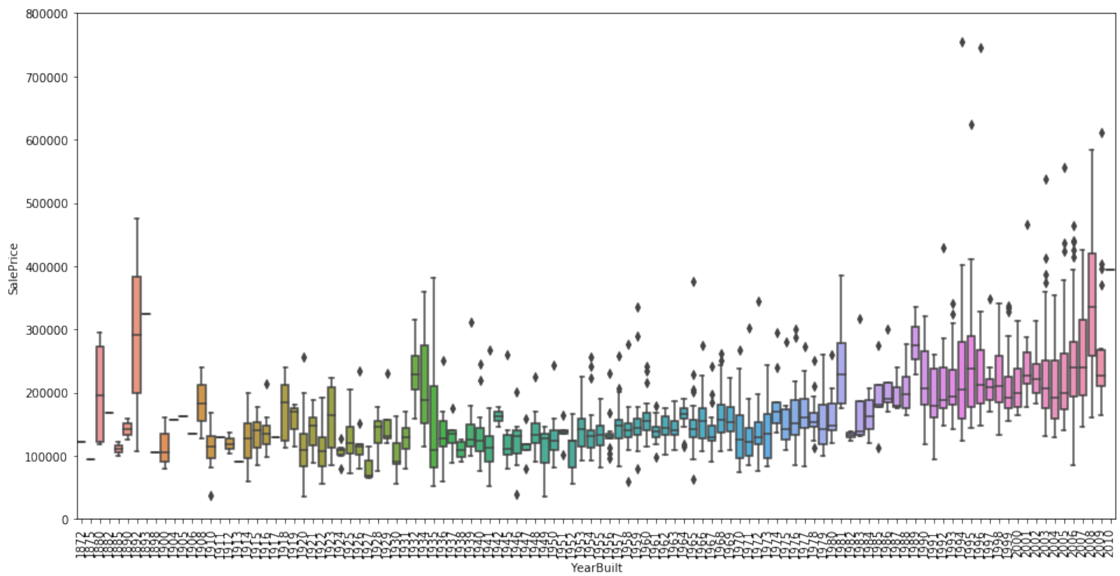
[12]: (-0.5, 9.5, 0, 800000)



[46]:
```
#looks like salesprice is driven up by the overall quality

#lets see how year built looks
```

[13]:
```
var = 'YearBuilt'
data = pd.concat([df_train['SalePrice'],df_train[var]], axis = 1)
f, ax = plt.subplots(figsize = (16,8))
fig = sns.boxplot (x=var,y='SalePrice',data=data)
fig.axis(ymin=0,ymax=800000)
plt.xticks(rotation=90)
#use this function to rotate the tick labels 90 degrees to see labels clearly
```

[13]:
```
(array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
         13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
         26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
         39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
         52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,
         65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,
         78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,
         91,  92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103,
        104, 105, 106, 107, 108, 109, 110, 111]),
 <a list of 112 Text xticklabel objects>)
```

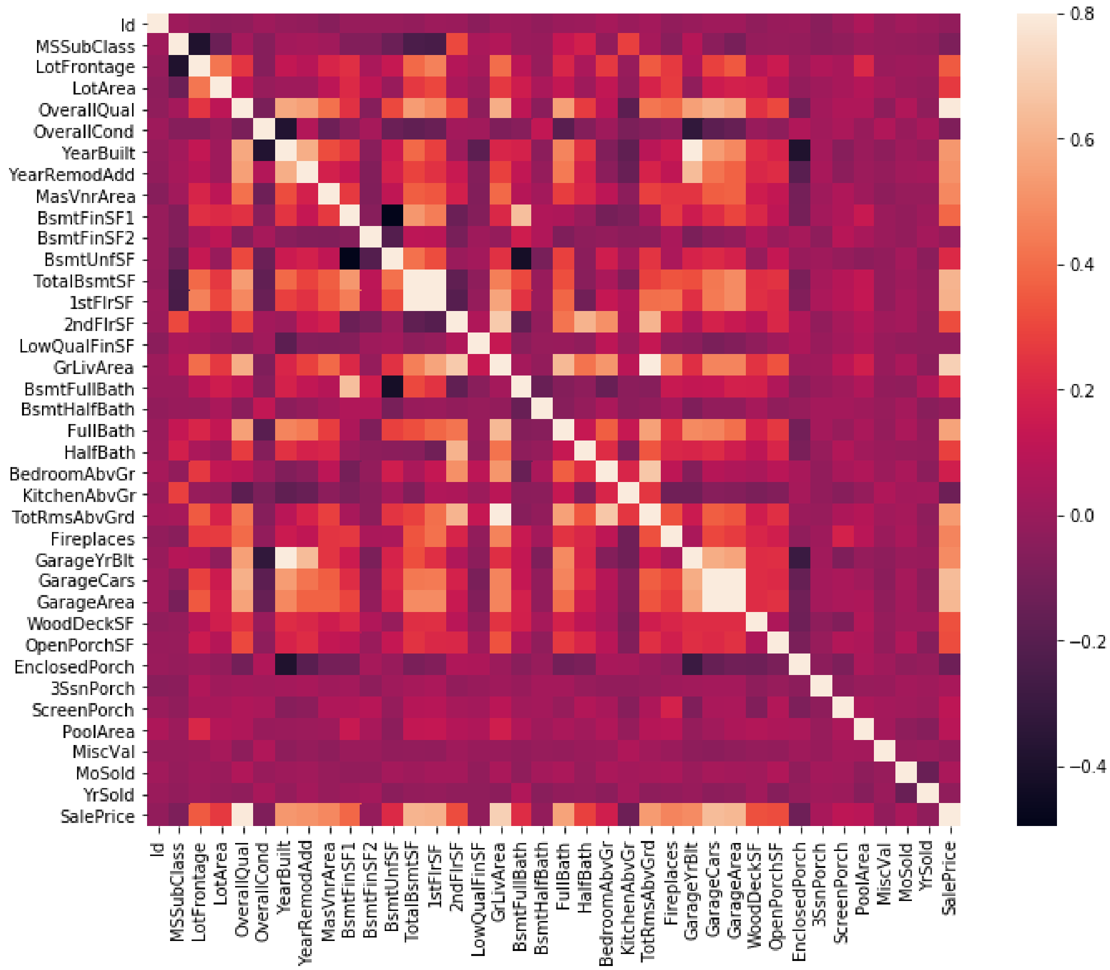


[48]:
```
#Sales price is driven by the year built although its not a strong relationship
```

[49]:
```
#Summary

#Ground living room area and total basement square footage seem to be linearly␣
↪related with Sale price.
#Both are positively related, as one variable increases, the other also␣
↪increases
#Overall quality and year built also show a relationship with salesprice,␣
↪although it is not as strong with year built

#lets analyze more
#correlation matrix (heatmap - seaborn)
#salesprice correlation matrix (zoomed heatmap)
#scatter plots between correlated variables
```

[14]:
```python
#correlation matrix
corrmat = df_train.corr()
f, ax = plt.subplots(figsize = (12,9))
sns.heatmap(corrmat, vmax = .8, square=True)
```

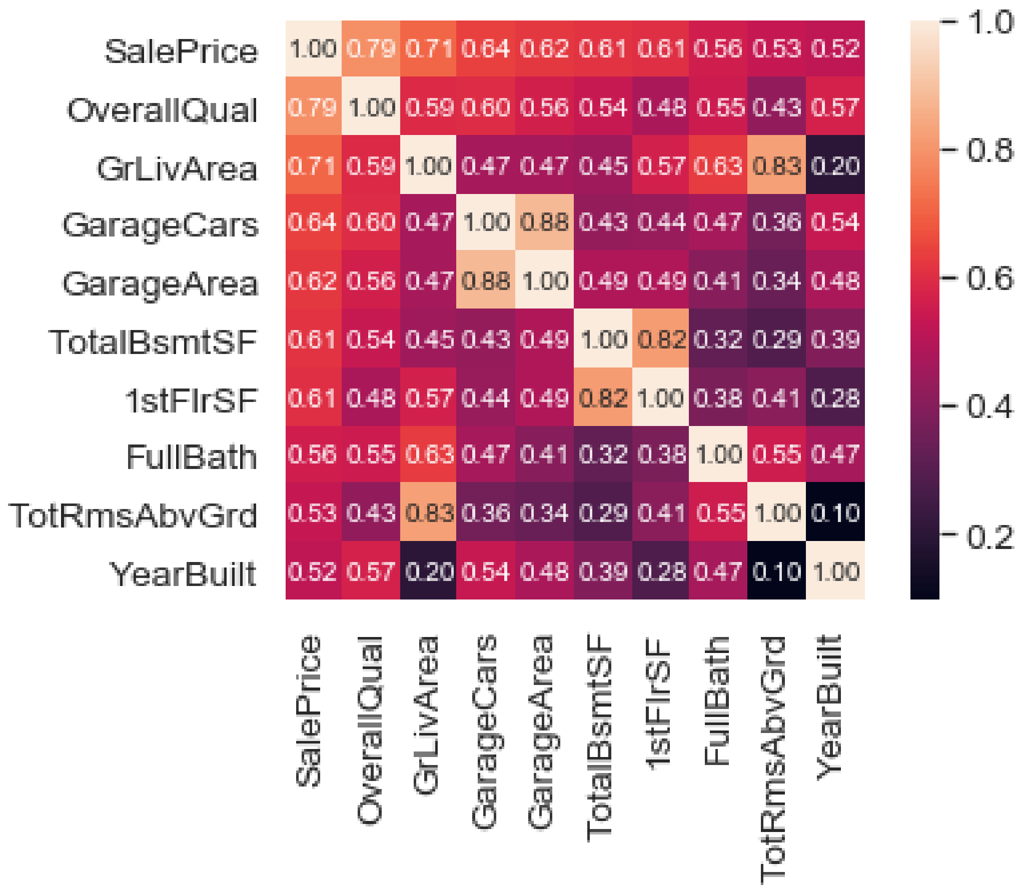
[14]: <matplotlib.axes._subplots.AxesSubplot at 0x7f999d787390>



[51]:
```python
#By first looking at this correlation matrix, 2 parts stand out:
# "total bsmt sf" and '1stFlrSF'
#the Garage x variables
#there is a high correlation between these variables which may be a case of multicollinearity
#if you think about these variables, we can see that multicollinearity is occuring
#basement size will usually be the size of the first floor and the bigger the garage size, the more cars
```

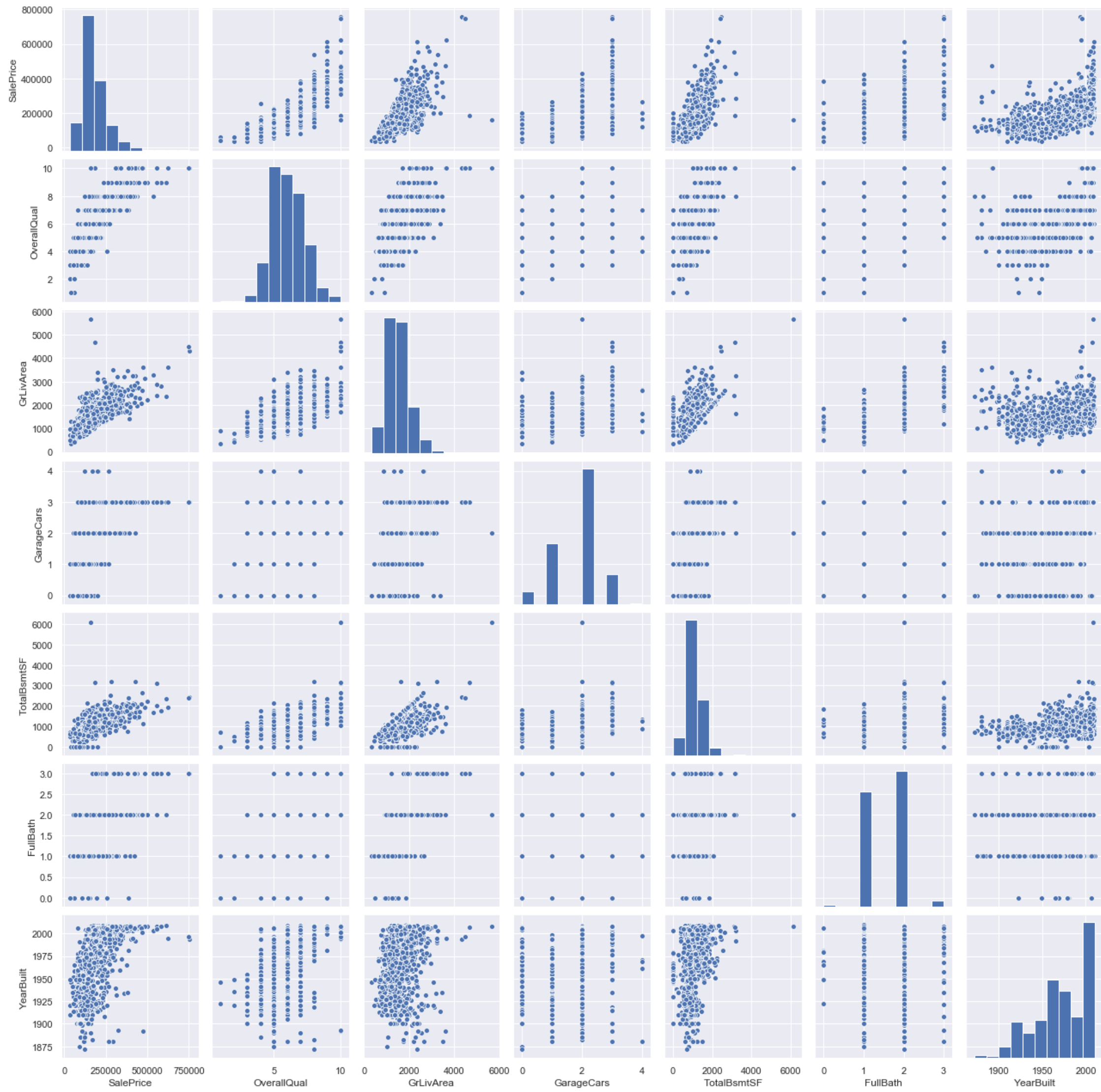
```
#salesprice has a few squares that stand out: OverallQual, GrLivArea and Total␣
↪BsmtSF

#lets take a look at the salesprice correlation matrix (zoomed heatmap style)␣
↪next
```

[15]:
```
k = 10
#number of variables for the heatmap
cols = corrmat.nlargest(k,'SalePrice')['SalePrice'].index
#getting the 10 largest correlated variables and getting their row labels
cm = np.corrcoef(df_train[cols].values.T)
#transposing values of "cols"  into correlation coefficients
sns.set(font_scale=1.25)
#set font fize
hm = sns.heatmap(cm, cbar=True,annot=True,square=True, fmt='.
↪2f',annot_kws={'size':10}, yticklabels=cols.values, xticklabels=cols.values)
#creating heatmap using seaborn
plt.show()
```



[53]:
```
#Above are the variables that are most correlated to SalePrice
#Our assumptions above that OverallQual, GrLiveArea, TotalBsmtArea are␣
↪correlated with Salesprice are true
#Garage cars and Garage area are strongly correlated but basically the same. We␣
↪will use garage cars in our analysis
#since the correlation is higher

#Same with TotalBsmtSF and 1stFloorSF. We will keep TotalBsmtSF
#Same with TotRomsAboveGrd and GrLivArea

#onto the scatter plots
```

[16]:
```
sns.set()
#apply default seaborn scheme
cols =␣
↪['SalePrice','OverallQual','GrLivArea','GarageCars','TotalBsmtSF','FullBath','YearBuilt']
#set column names into list
sns.pairplot(df_train[cols],size = 2.5)
#using seaborn to compare the variables using pairplot
#pairplot plots pairwise relationships in a dataset creating a grid of plots␣
↪with each variable in x and y axis
plt.show()
#shows the plots
```



[55]:
```
#looks like Basement size = living room area
#prices rise exponentially the newer the house
```

[56]:
```
#Important questions to ask about Missing Data
#How prevalent is the missing data?
#Is missing data random or does it have a pattern?

#missing data can imply a reduction in sample size which can prevent us from
↪proceeding with the analysis
#need to ensure the missing data process is not biased and hiding an
↪inconvenient truth
```

[35]:
```python
Total = df_train.isnull().sum().sort_values(ascending = False)
#total number of null values for variable data sorted by most missing missing
percent = (df_train.isnull().sum()/df_train.isnull().count()).
 ↪sort_values(ascending = False)
#setting a percent by dividing sum of missing values
missing_data = pd.concat([Total,percent], axis=1, keys=['Total','Percent'])
missing_data.head(20)
```

[35]:

	Total	Percent
PoolQC	1453	0.995205
MiscFeature	1406	0.963014
Alley	1369	0.937671
Fence	1179	0.807534
FireplaceQu	690	0.472603
LotFrontage	259	0.177397
GarageCond	81	0.055479
GarageType	81	0.055479
GarageYrBlt	81	0.055479
GarageFinish	81	0.055479
GarageQual	81	0.055479
BsmtExposure	38	0.026027
BsmtFinType2	38	0.026027
BsmtFinType1	37	0.025342
BsmtCond	37	0.025342
BsmtQual	37	0.025342
MasVnrArea	8	0.005479
MasVnrType	8	0.005479
Electrical	1	0.000685
Utilities	0	0.000000

[18]:
```python
#the user that is guiding this project feels that variables missing more than␣
 ↪>15% of data should be deleted
#i will do that in the next cell
#PoolQC, MiscFeature, Alley will be deleted as they are >15% and these arent␣
 ↪feature people look for when buying a home

#GarageX variables are missing the same number of variables. Garage cars will␣
 ↪represent this data
#Same logic for BasementX variables

#For electricalm
```

[40]:
```python
df_train = df_train.drop((missing_data[missing_data['Total'] > 1]).index,1)
#setting df_train now as a new data frame by dropping the variables where␣
 ↪missing values are > 1
df_train = df_train.drop(df_train.loc[df_train['Electrical'].isnull()].index)
```

```
#setting df_train now as a new data frame by dropping the null variable in the␣
↪electricity columns
```

[41]:
```
print(len(df_train.columns))
#checking to see if columns were dropped
```

63

[ ]:
```
#lets go on to do analysis on the outliers in our data
#univariate analysis

#we need to establish a threshold that defines an observation as an outlier.
#we will standardize the data - converting data values to have a mean of 0 and␣
↪standard deviation of 1
```

[43]:
```
saleprice_scaled = StandardScaler().fit_transform(df_train['SalePrice'][:,np.
↪newaxis])
#using the standardscaler for the fit_transform function to scale the training␣
↪data and the scaling parameters
#the model will learn the mean and variance of the features of the training data
#https://stackoverflow.com/questions/29241056/
↪how-does-numpy-newaxis-work-and-when-to-use-it
low_range = saleprice_scaled[saleprice_scaled[:,0].argsort()][:10]
#setting low range of distribution by taking the scaled saleprice
high_range = saleprice_scaled[saleprice_scaled[:,0].argsort()][-10:]
#setting low range of distribution by taking the scaled saleprice
print('outer range (low) of the distribution:')
print(low_range)
print('\nouter range (high) of the distribution:')
print(high_range)
```

```
outer range (low) of the distribution:
[[-1.83820775]
 [-1.83303414]
 [-1.80044422]
 [-1.78282123]
 [-1.77400974]
 [-1.62295562]
 [-1.6166617 ]
 [-1.58519209]
 [-1.58519209]
 [-1.57269236]]

outer range (high) of the distribution:
[[3.82758058]
 [4.0395221 ]
 [4.49473628]
```

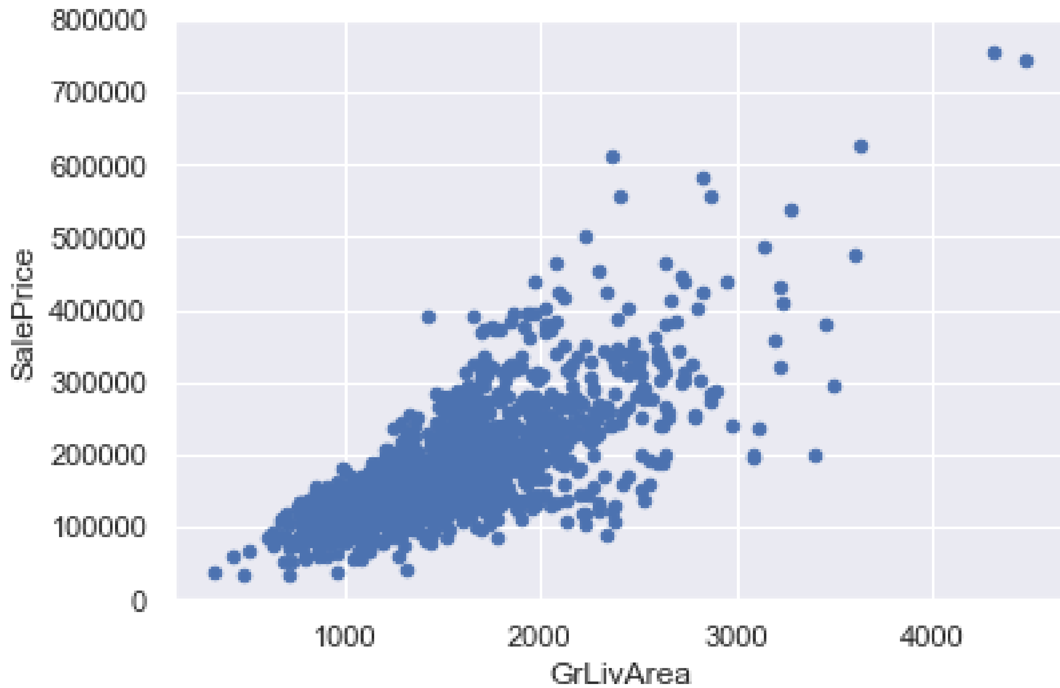
```
[4.70872962]
[4.728631  ]
[5.06034585]
[5.42191907]
[5.58987866]
[7.10041987]
[7.22629831]]
```

[ ]:
```
#Data standardization is the process of rescaling the attributes so that they␣
↪have mean as 0 and variance as 1.
#The ultimate goal to perform standardization is to bring down all the features␣
↪to a common scale
#without distorting the differences in the range of the values.
#In sklearn.preprocessing.StandardScaler(), centering and scaling happens␣
↪independently on each feature.

#observations
#low range values are similar and not too far from 0
#high range values are further from 0 (last two values are really out of range)

#lets move on to bivariate analysis of sale price vs grlivarea
```

[46]:
```
var = 'GrLivArea'
data = pd.concat([df_train['SalePrice'],df_train[var]],axis=1)
data.plot.scatter(x=var,y='SalePrice', ylim=(0,800000))
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
```

[46]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f999bdc7850>
```



[ ]:
```
#the two at the bottom right do not seem to be following the trend
#we will delete those outliers
```

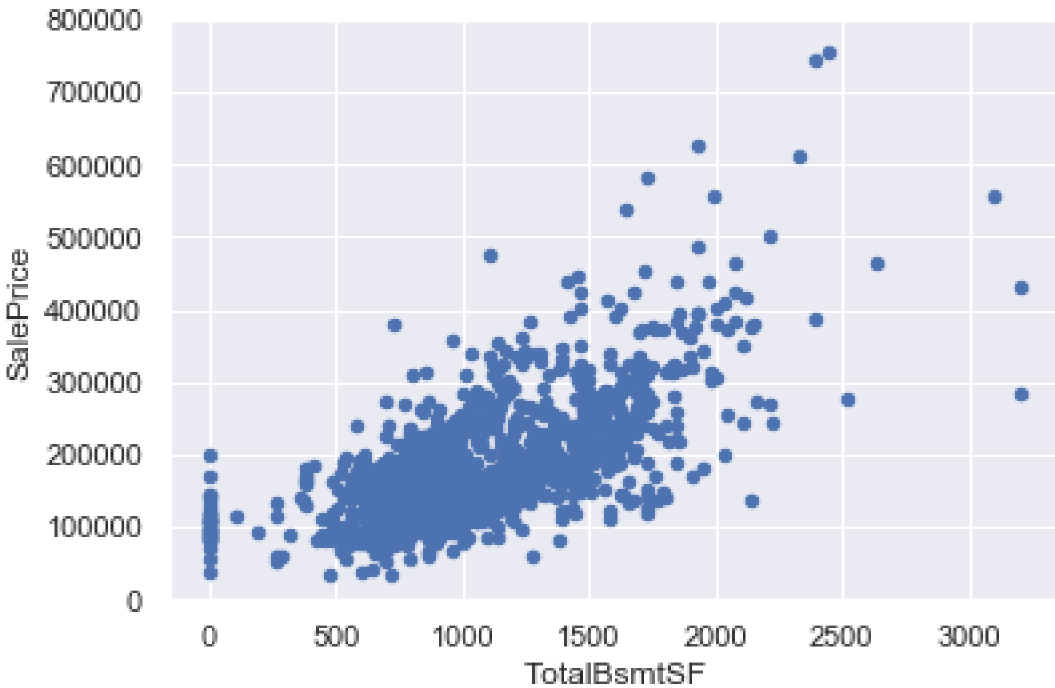
[45]:
```
df_train.sort_values(by = 'GrLivArea', ascending = False)[:2]
#sorting df_train by GrLiveArea and keeping the first 2
df_train = df_train.drop(df_train[df_train['Id'] == 1299].index)
df_train = df_train.drop(df_train[df_train['Id'] == 524].index)
#dropping the rows 1299 and 524
```

[ ]:
```
#bivariate analysis of salesprice vs totalbsmtsf
```

[47]:
```
var = 'TotalBsmtSF'
data = pd.concat([df_train['SalePrice'],df_train[var]],axis=1)
data.plot.scatter(x=var,y='SalePrice', ylim=(0,800000))
```

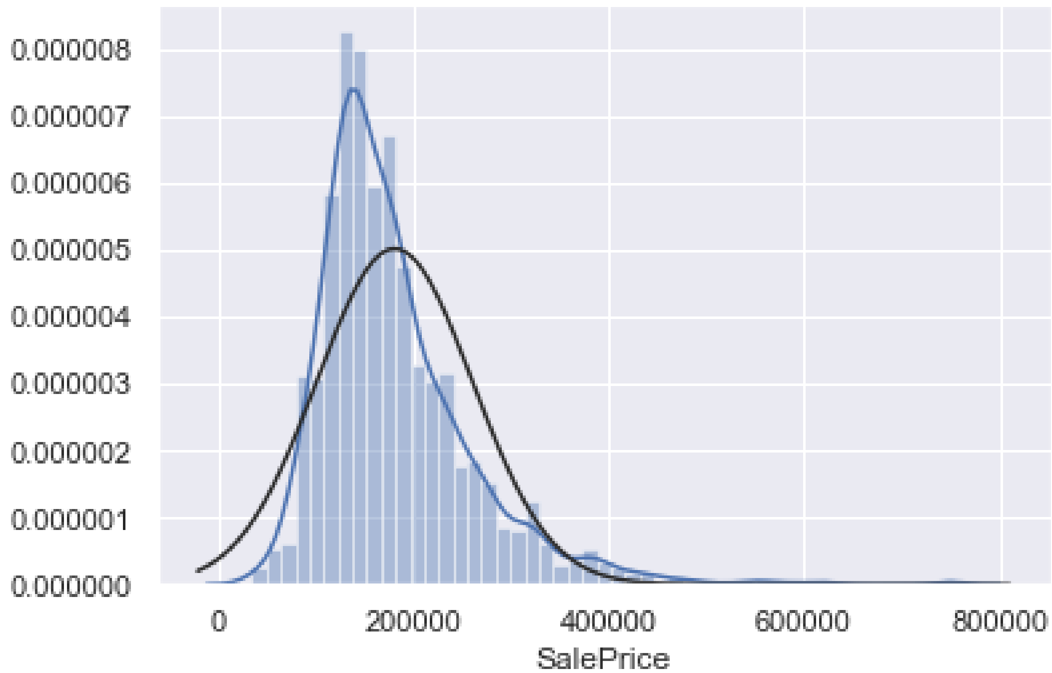
```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
```

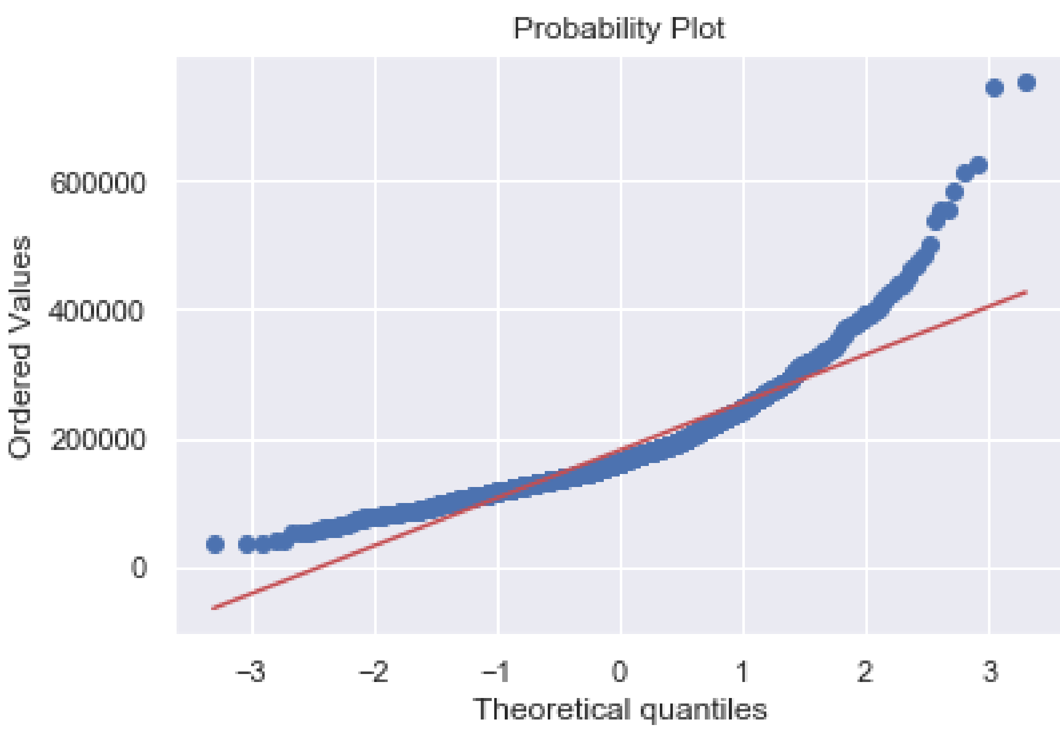
[47]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f99a0dff0d0>
```



[ ]:
```
#looks ok, no need to delete any data

#lets look for normality and homoscedascity
#histogram to see kurtosis and skewness
#normal probability plot - to see if data distribution follows the diagnal that␣
↪represents normal distribution
```

[50]:
```
sns.distplot(df_train['SalePrice'],fit=norm)
#using seaborn to plot histogram by normal distribution
fig = plt.figure()
#create figure object
res = stats.probplot(df_train['SalePrice'],plot = plt)
#use scipy to create prob plot of saleprice
```

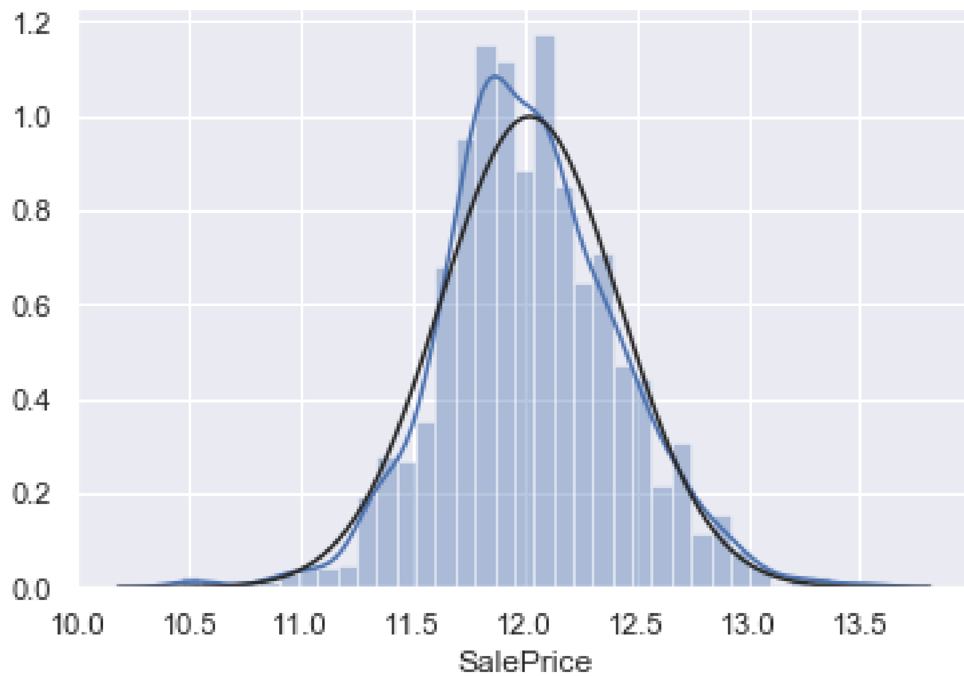

0.000008
0.000007
0.000006
0.000005
0.000004
0.000003
0.000002
0.000001
0.000000
0
200000
400000
600000
800000
SalePrice

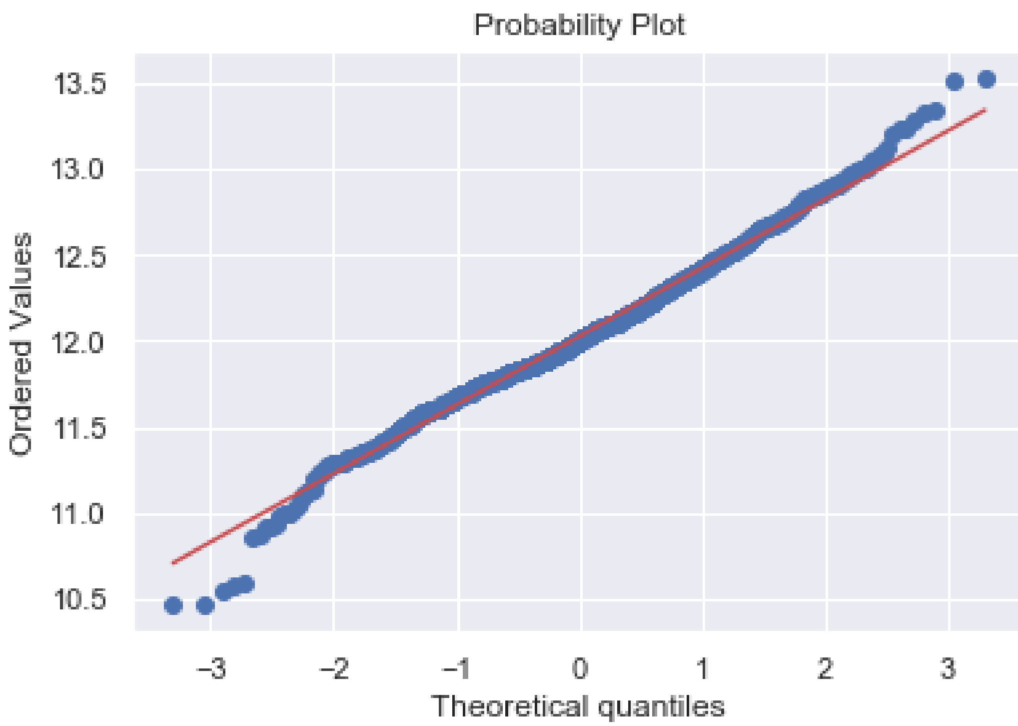

Probability Plot
600000
400000
200000
0
Ordered Values
−3
−2
−1
0
1
2
3
Theoretical quantiles

[ ]:
```
#sale price is not normal distributed
#it shows peakedness, positive skewness does not follow the diagnal line
#lets tranform this data using log transformations
```

[51]:
```
df_train['SalePrice'] = np.log(df_train['SalePrice'])
```

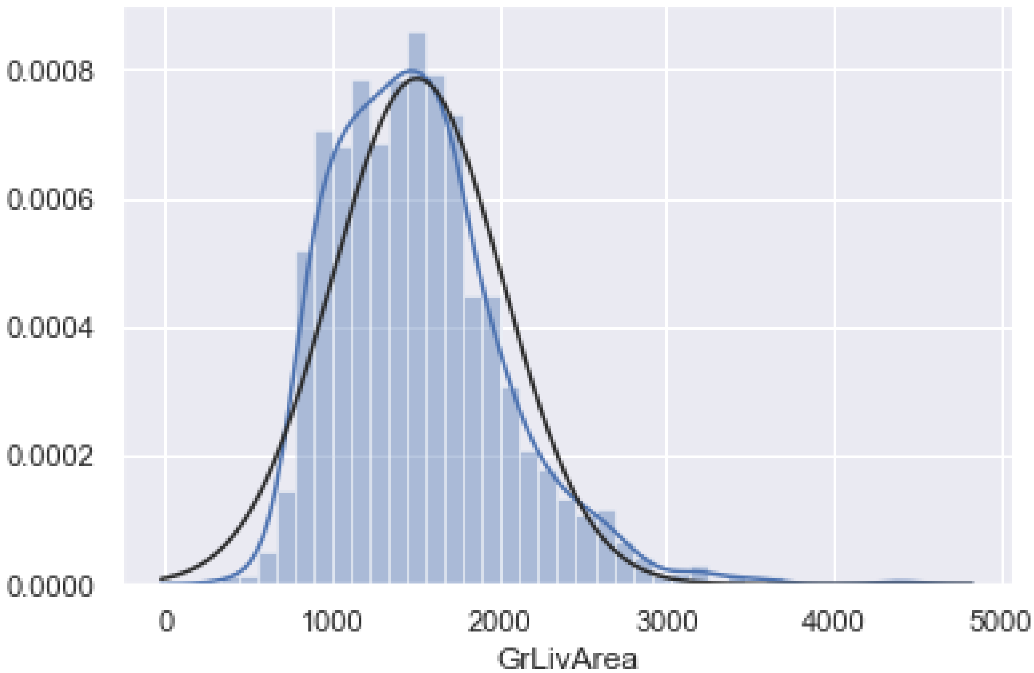
[52]:
```
#re-running histogram and probability plot

sns.distplot(df_train['SalePrice'],fit=norm)
#using seaborn to plot histogram by normal distribution
fig = plt.figure()
#create figure object
res = stats.probplot(df_train['SalePrice'],plot = plt)
#use scipy to create prob plot of saleprice
```

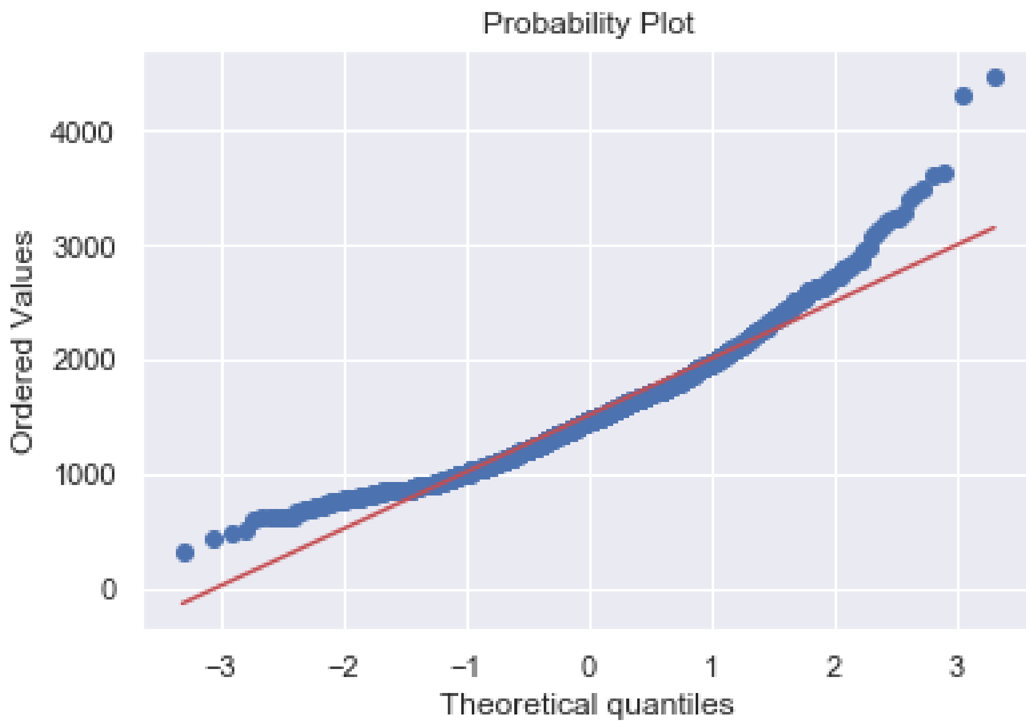




[ ]:
```
#looks good, now lets check out GrLivArea
```

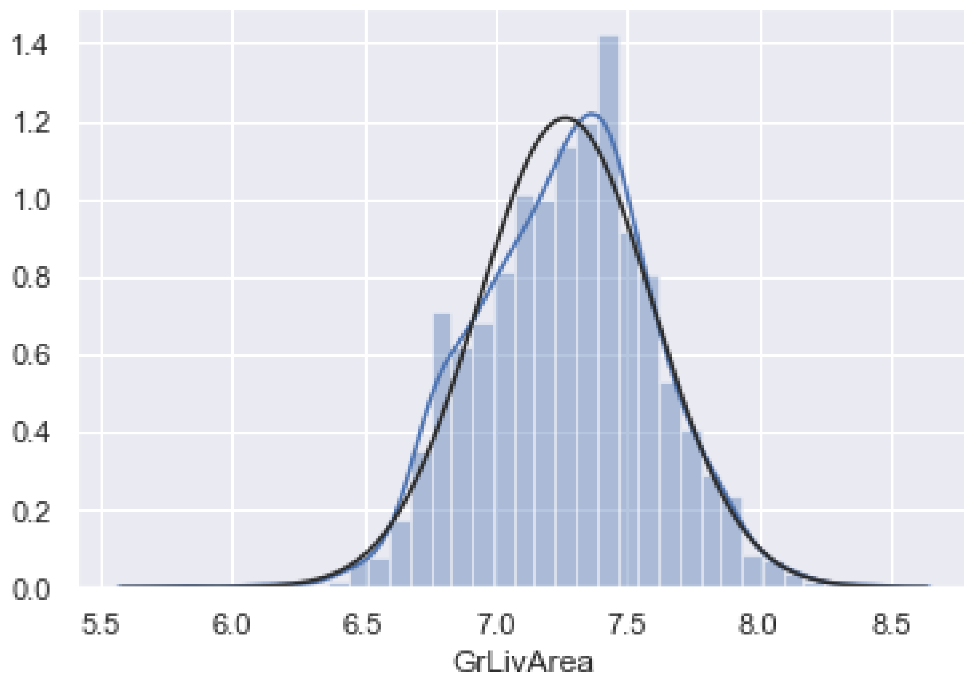
[53]:
```
sns.distplot(df_train['GrLivArea'],fit=norm)
#using seaborn to plot histogram by normal distribution
fig = plt.figure()
#create figure object
res = stats.probplot(df_train['GrLivArea'],plot = plt)
#use scipy to create prob plot of saleprice
```

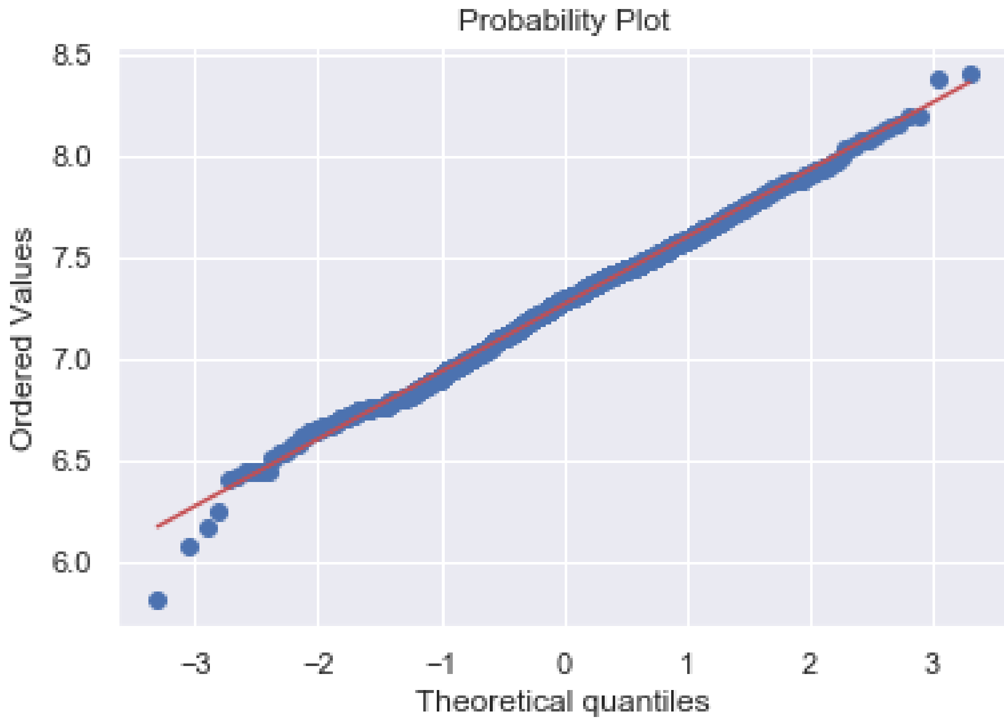

0.0008
0.0006
0.0004
0.0002
0.0000
0
1000
2000
3000
4000
5000
GrLivArea


Probability Plot
Ordered Values
4000
3000
2000
1000
0
−3
−2
−1
0
1
2
3
Theoretical quantiles

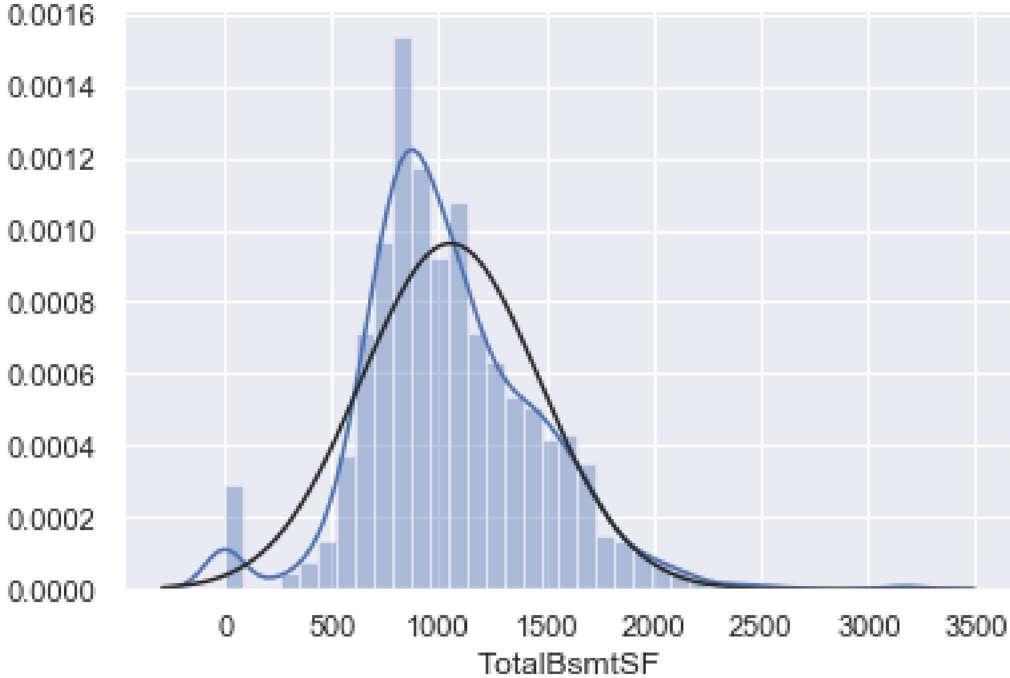
[54]:

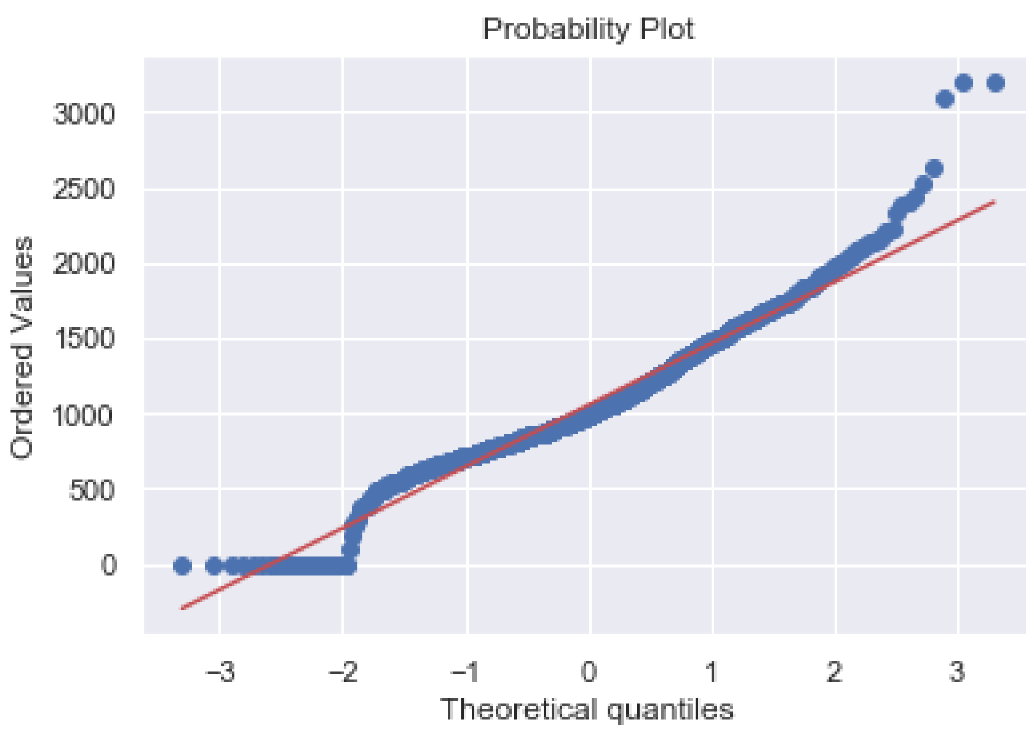
```
#there is skewness, lets transform this data
df_train['GrLivArea'] = np.log(df_train['GrLivArea'])

sns.distplot(df_train['GrLivArea'],fit=norm)
#using seaborn to plot histogram by normal distribution
fig = plt.figure()
#create figure object
res = stats.probplot(df_train['GrLivArea'],plot = plt)
#use scipy to create prob plot of saleprice
```





[55]:
```
#much better! lets try TotalBsmtSF

sns.distplot(df_train['TotalBsmtSF'],fit=norm)
#using seaborn to plot histogram by normal distribution
fig = plt.figure()
#create figure object
res = stats.probplot(df_train['TotalBsmtSF'],plot = plt)
#use scipy to create prob plot of salepric
```


0.0016
0.0014
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
0
500
1000
1500
2000
2500
3000
3500
TotalBsmtSF

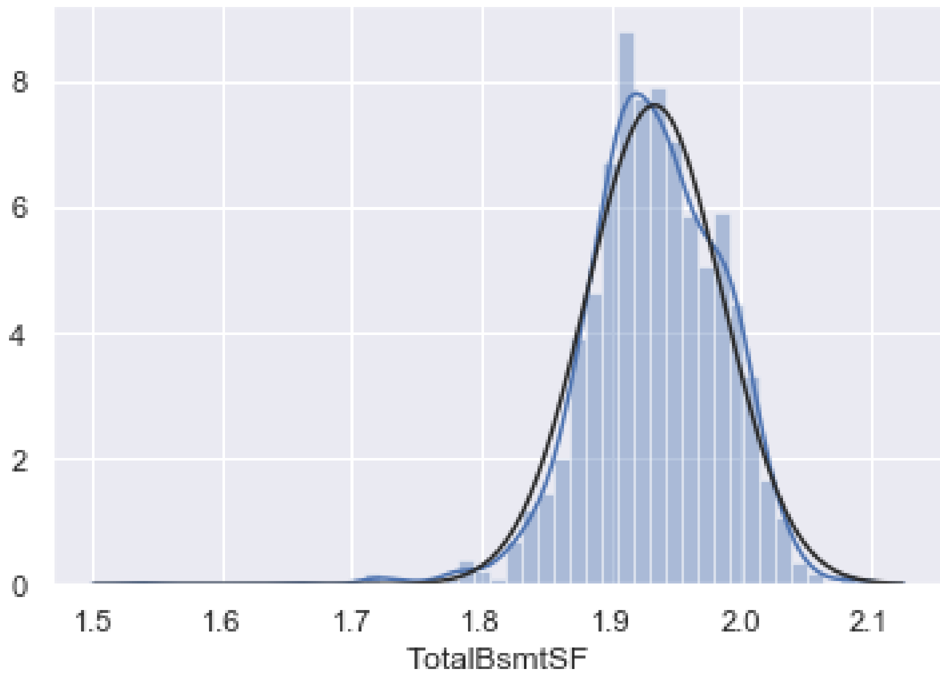

Probability Plot
Ordered Values
3000
2500
2000
1500
1000
500
0
−3
−2
−1
0
1
2
3
Theoretical quantiles

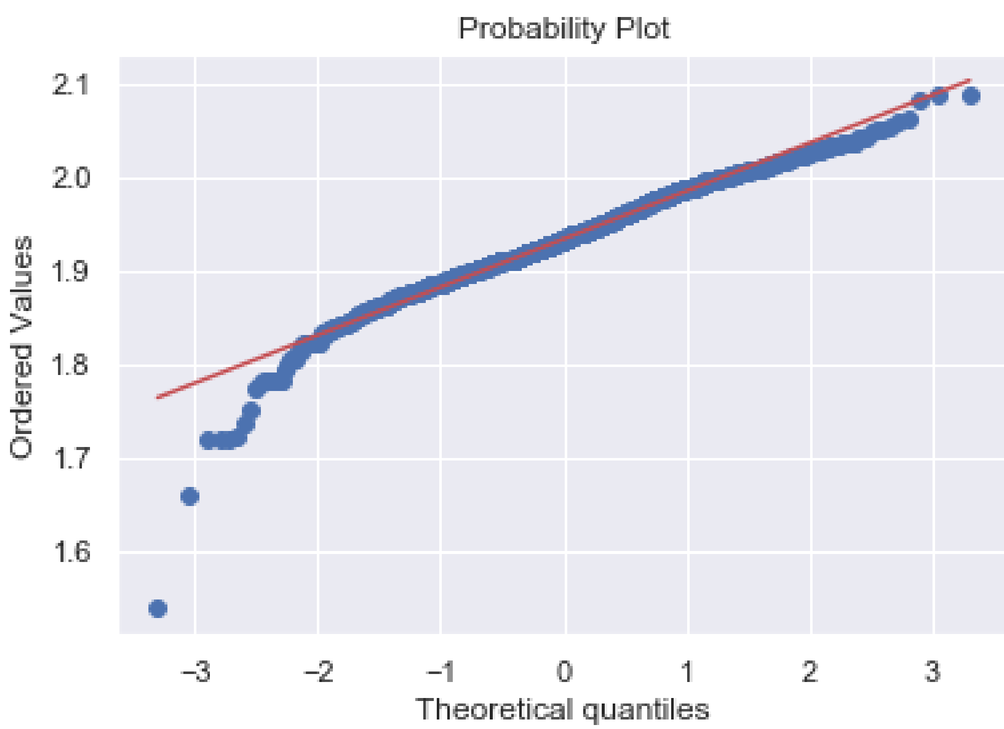
[ ]:
```
#there is skewnwess
#lots of houses with no basements (value = 0)
#we cannot do log transformation on zero values

#to conduct a log transformation, we will create a variable that can create the␣
↪effect of hacing or not having
#a basement. then we will do a log transformation to all the non-zero␣
↪observations and then ignore those with
#value 0

#a log transform of data set containing zero values can be easily handled by␣
↪numpy.log1p()
```

[57]:
```
#create column for new variable (one is enough because it's a binary␣
↪categorical feature)
#if area>0 it gets 1, for area==0 it gets 0
df_train['HasBsmt'] = pd.Series(len(df_train['TotalBsmtSF']), index=df_train.
↪index)
#create new column "HasBsmt"
df_train['HasBsmt'] = 0
#set all values to 0
df_train.loc[df_train['TotalBsmtSF']>0,'HasBsmt'] = 1
#goes thrrough each index of TotalBsmtSF and sets the value for HasBsmt to 1 if␣
↪value > 0
```

[58]:
```
#transform the data
df_train.loc[df_train['HasBsmt']==1,'TotalBsmtSF'] = np.
↪log(df_train['TotalBsmtSF'])
#find indexes where there HasBsmt = 1 and conduct a log transformation
```

[60]:
```
sns.distplot(df_train[df_train['TotalBsmtSF']>0]['TotalBsmtSF'],fit=norm)
#using seaborn to plot histogram by normal distribution where totalbsmtsf > 0
fig = plt.figure()
#create figure object
res = stats.probplot(df_train[df_train['TotalBsmtSF']>0]['TotalBsmtSF'],plot =␣
↪plt)
#use scipy to create prob plot of saleprice
```

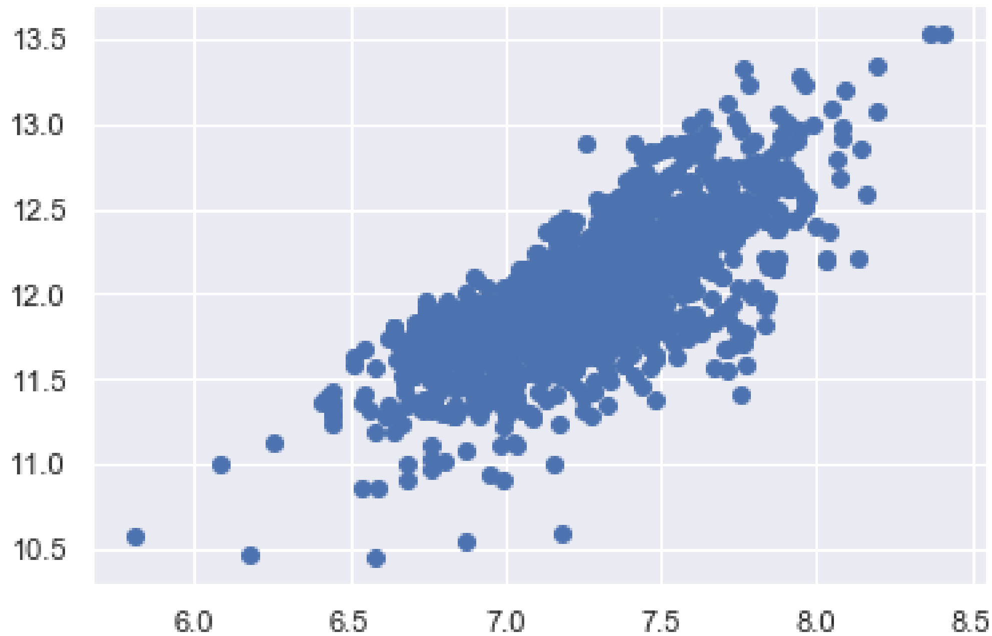




[ ]:
```
#lets search for homoscedascity
```

[62]:
```
plt.scatter(df_train['GrLivArea'],df_train['SalePrice'])
```

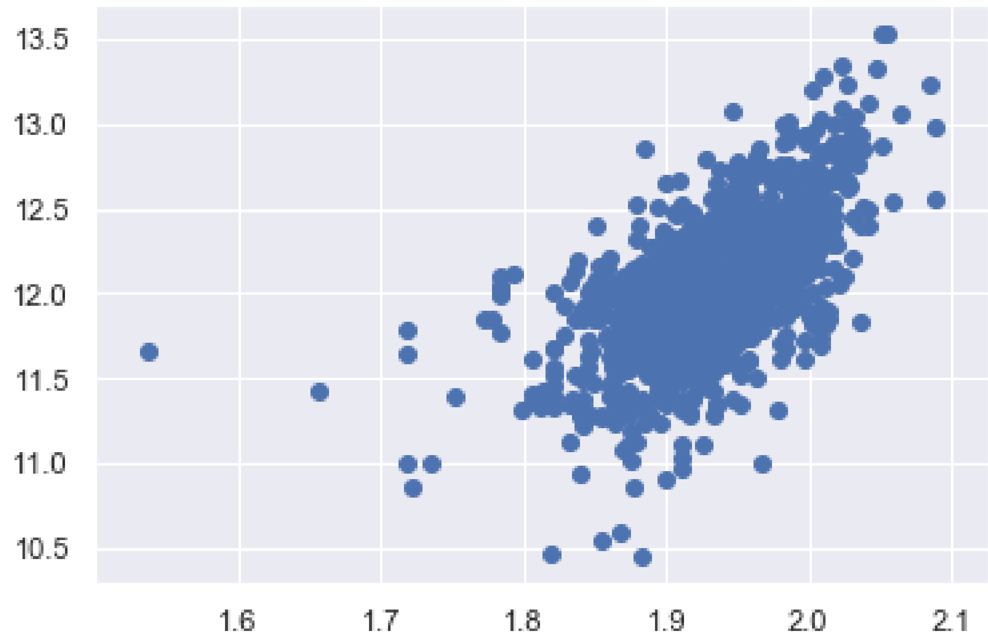
[62]: <matplotlib.collections.PathCollection at 0x7f999bb1b610>



[ ]:
```
#older versions of this plot had a conic shape - it is normalized
#now lets check saleprice with totalbsmtsf
```

[63]:
```
plt.scatter(df_train['TotalBsmtSF'],df_train['SalePrice'])
```

[63]: <matplotlib.collections.PathCollection at 0x7f99a0689710>



[ ]:
```
#equal level of variance of salesprice across range of totalbsmtsf

#now dummy variables
```

[64]:
```
df_train = pd.get_dummies(df_train)
```

[ ]:
```
#comments
```

[ ]:
```
"Understanding the problem" phase is great, "developing sixth sense" is a practice that is often disregarded.
Guessing which columns to drop, in case of correlating features ideally should be performed based on some
experiments, like their effect on the model accuracy or something of that ilk.
It's always better to retain the data than to dismiss it. The threshold of 15% for dropping appears to me
somewhat arbitrary. As a rule of thumb, statisticians dismiss some feature if it has more than 80 percent
missing data and doesn't seem to be significantly important, because at that point the bias from imputation
is very likely to be more problematic than dropping the whole feature. After that, ideally, it needs a little
bit of experimentation to decide which features to impute and which features to drop. "Best statisticians
make the fewest assumptions".
```

```
Making decisions based on univariate and scatter plots seems a little dangerous␣
↪because they can have totally
different relationships on higher dimensions.
```