Ref.

[https://www.kaggle.com/pmarcelino/comprehensiv data-exploration-with-python (https://www.kaggle.com/pmarcelino/comprehensi data-exploration-with-python)](https://www.kaggle.com/pmarcelino/comprehensive-data-exploration-with-python)

In [77]:
```
#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
```

In [78]:
```
#bring in the six packs
df_train = pd.read_csv("/home/hduser/jupyter/Comprehensive_data_exploration_with_
```

In [79]:
```
#check the decoration
df_train.columns
```

Out[79]:
```
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')
```

In [80]:
```
df_train.shape
```

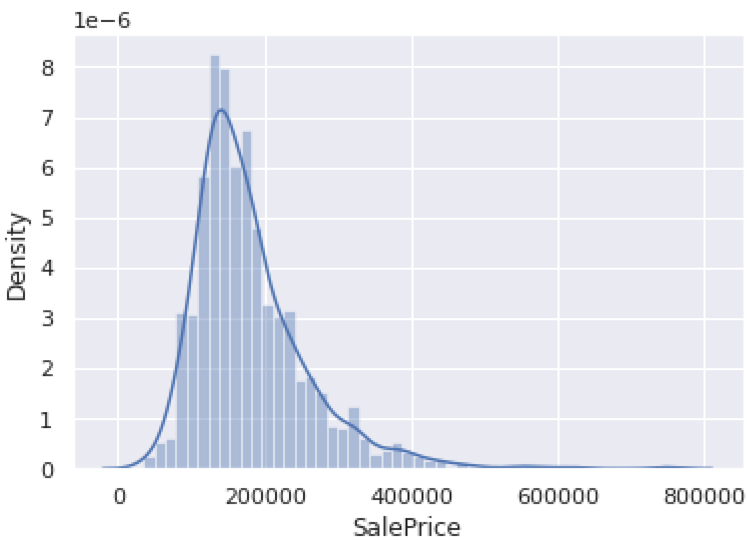
Out[80]:
```
(1460, 81)
```

In [81]:
```
#descriptive statistics summary
df_train['SalePrice'].describe()
```

Out[81]:
```
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
```

In [82]:
```
#histogram
sns.distplot(df_train['SalePrice'])
```

Out[82]:
```
<AxesSubplot:xlabel='SalePrice', ylabel='Density'>
```



In [83]:
```
#skewness and kurtosis
print("Skewness: %f" % df_train['SalePrice'].skew())
print("Kurtosis: %f" % df_train['SalePrice'].kurt())
```

```
Skewness: 1.882876
Kurtosis: 6.536282
```

In [84]:
```
#scatter plot grlivarea/saleprice
var = 'GrLivArea'
data = pd.concat([df_train['SalePrice'], df_train[var]], axis=1)
data
```

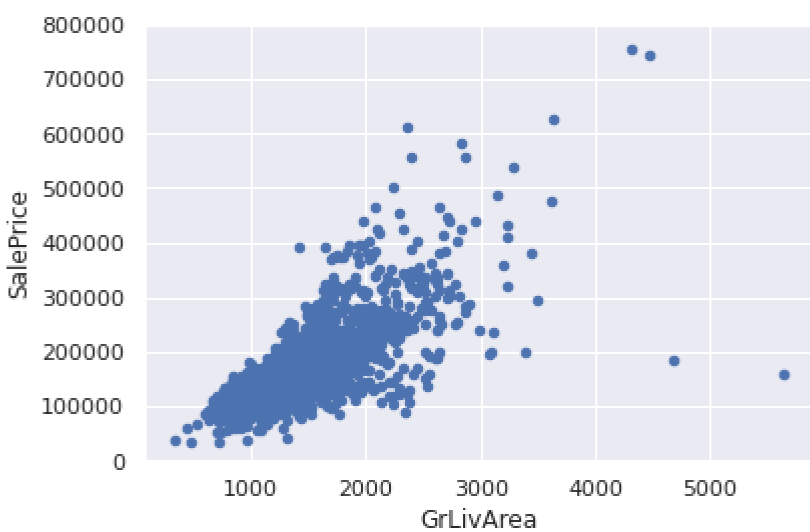
Out[84]:

	SalePrice	GrLivArea
0	208500	1710
1	181500	1262
2	223500	1786
3	140000	1717
4	250000	2198
...	...	...
1455	175000	1647
1456	210000	2073
1457	266500	2340
1458	142125	1078
1459	147500	1256

1460 rows × 2 columns

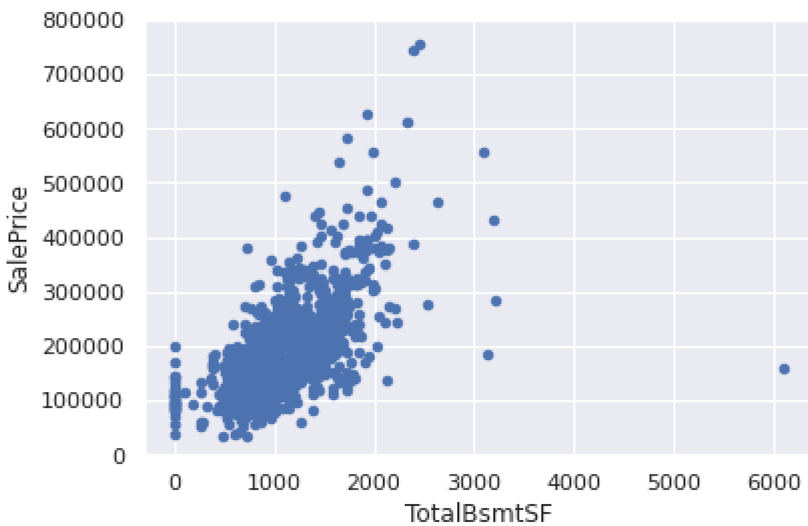
In [85]:
```
data.plot.scatter(x=var, y='SalePrice', ylim=(0,800000));
```

```
*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 the *color* keyword-argument or provide a 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all point
s.
```

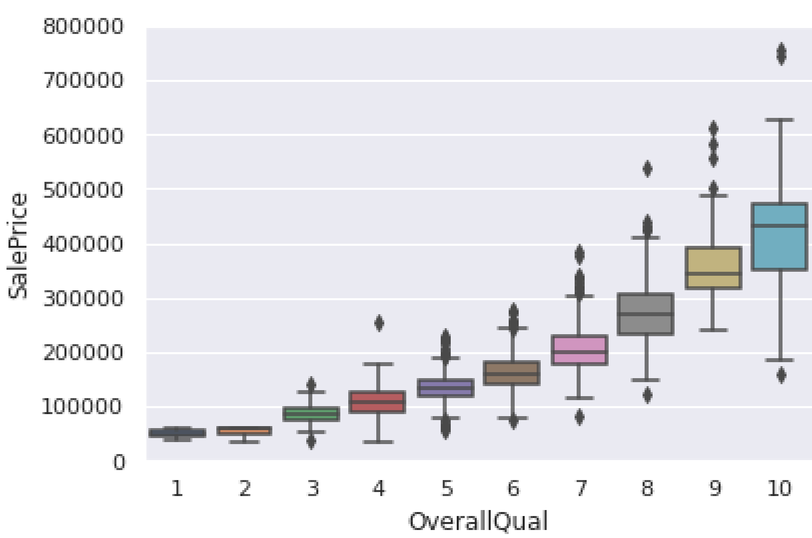


In [86]:
```
#scatter plot totalbsmtsf/saleprice
var = 'TotalBsmtSF'
data = pd.concat([df_train['SalePrice'], df_train[var]], axis=1)
data.plot.scatter(x=var, y='SalePrice', ylim=(0,800000));
```

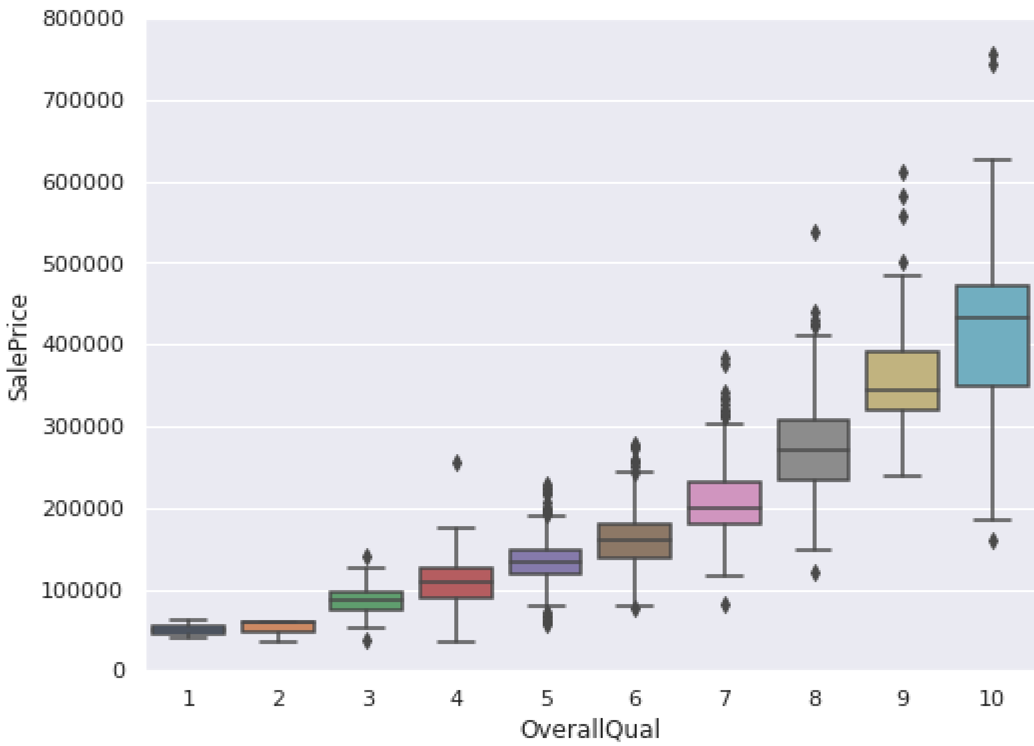
```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with *
x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with
a single row if you intend to specify the same RGB or RGBA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



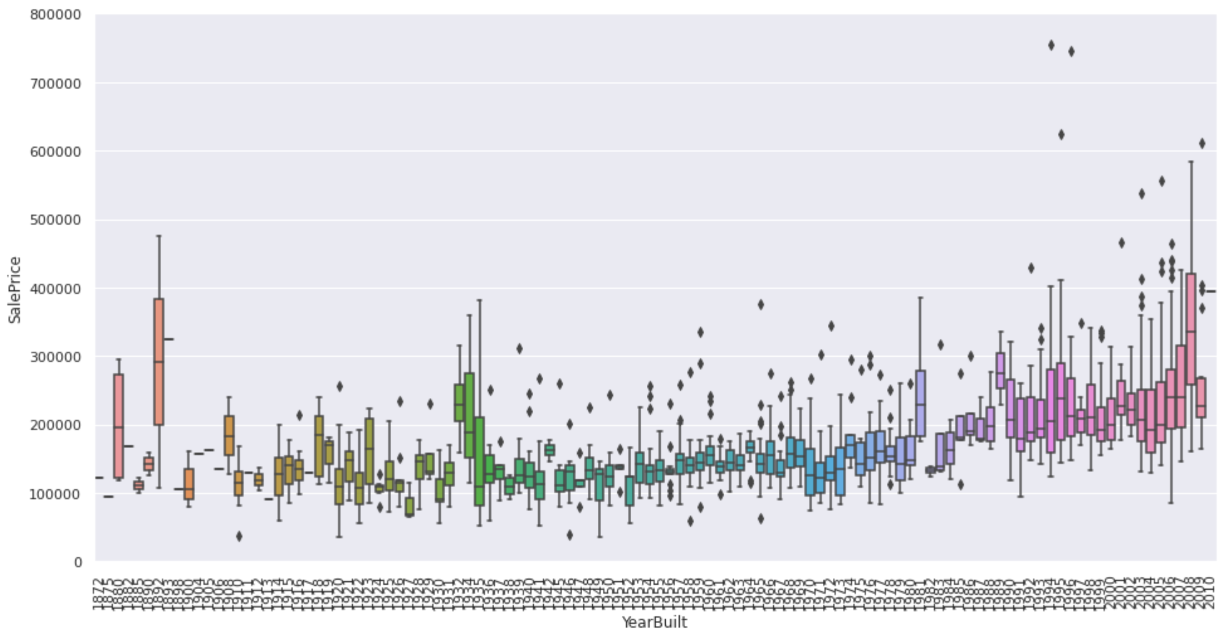
In [87]:
```
#box plot overallqual/saleprice
var = 'OverallQual'
data = pd.concat([df_train['SalePrice'], df_train[var]], axis=1)
# f, ax = plt.subplots(figsize=(8, 6))
fig = sns.boxplot(x=var, y="SalePrice", data=data)
fig.axis(ymin=0, ymax=800000);
```



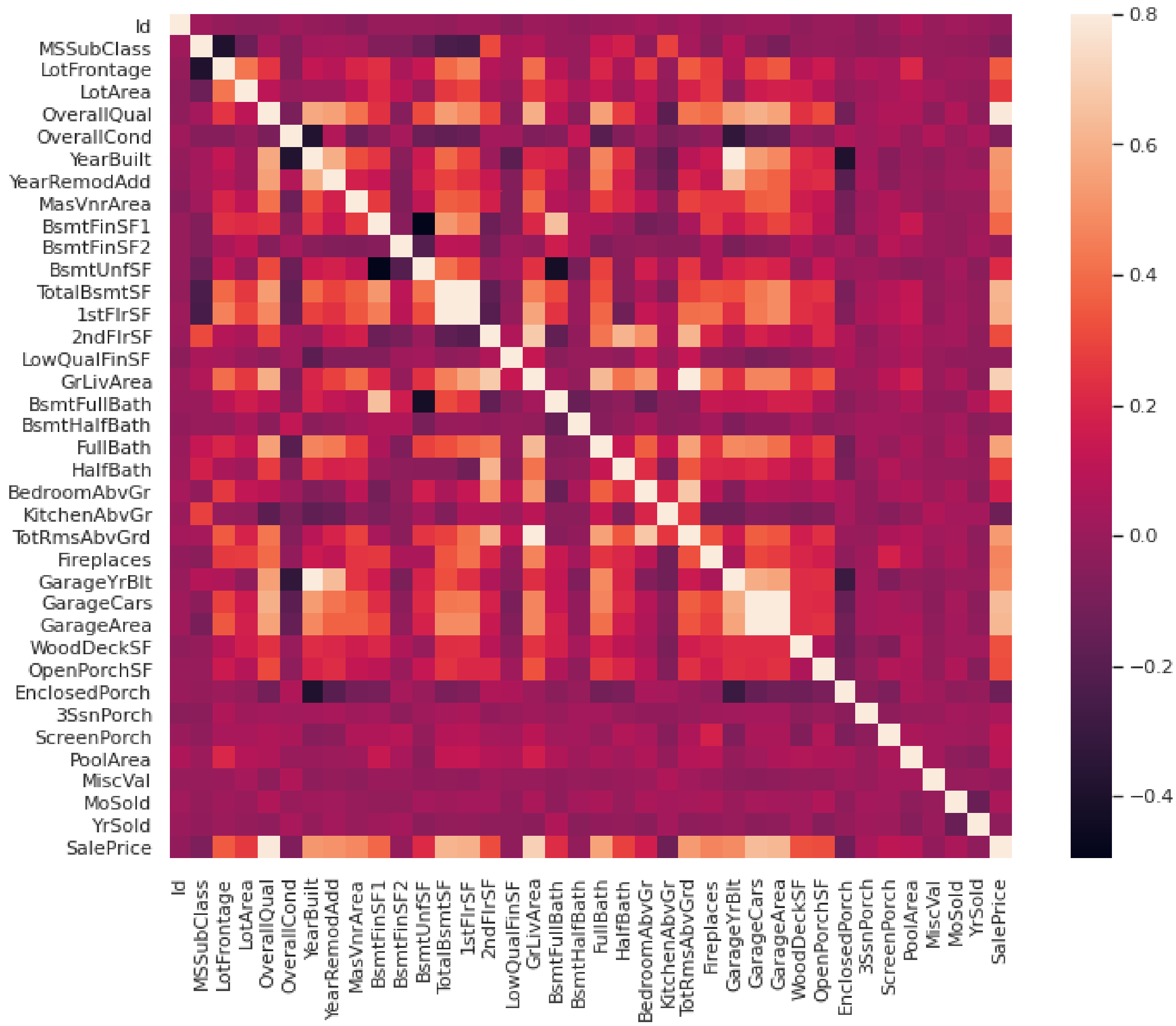
In [88]:
```
#box plot overallqual/saleprice
var = 'OverallQual'
data = pd.concat([df_train['SalePrice'], df_train[var]], axis=1)
f, ax = plt.subplots(figsize=(8, 6))
fig = sns.boxplot(x=var, y="SalePrice", data=data)
fig.axis(ymin=0, ymax=800000);
```



In [89]:
```
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);
```



In [90]:
```
#correlation matrix
corrmat = df_train.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True);
```



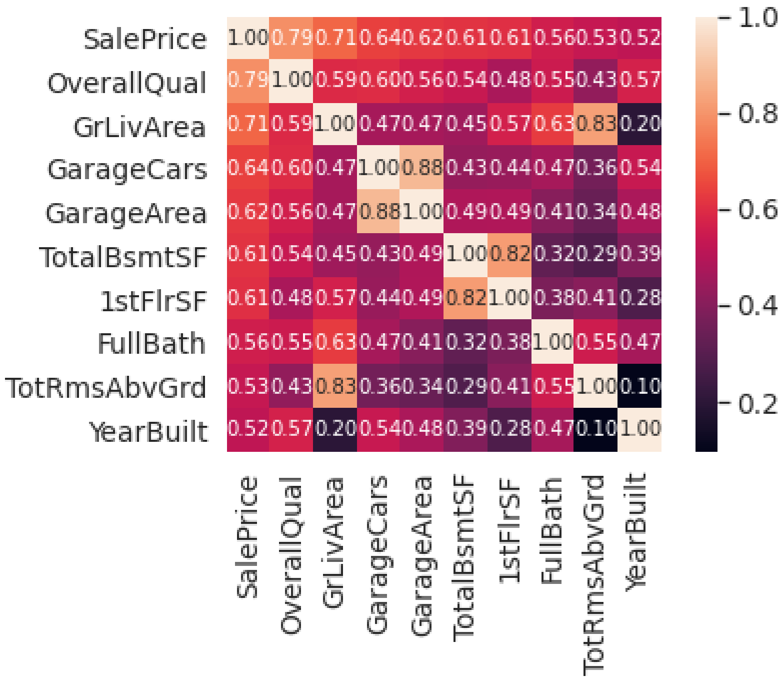
At first sight, there are **two red colored squares that get my attention**. The first one refers to the **'TotalBsmtSF' and '1stFlrSF'** variables, and the second one refers to the **'GarageX'** variables. **Both cases show how significant the correlation is between these variables.** Actually, this correlation is so strong that it can indicate a situation of multicollinearity. If we think about these variables, we can conclude that they give almost the same information so multicollinearity really occurs. Heatmaps are great to detect this kind of situations and in problems dominated by feature selection, like ours, they are an essential tool.

Another thing that got my attention was the 'SalePrice' correlations. We can see our well-known 'GrLivArea', 'TotalBsmtSF', and 'OverallQual' saying a big 'Hi!', but we can also see many other variables that should be taken into account. That's what we will do next.

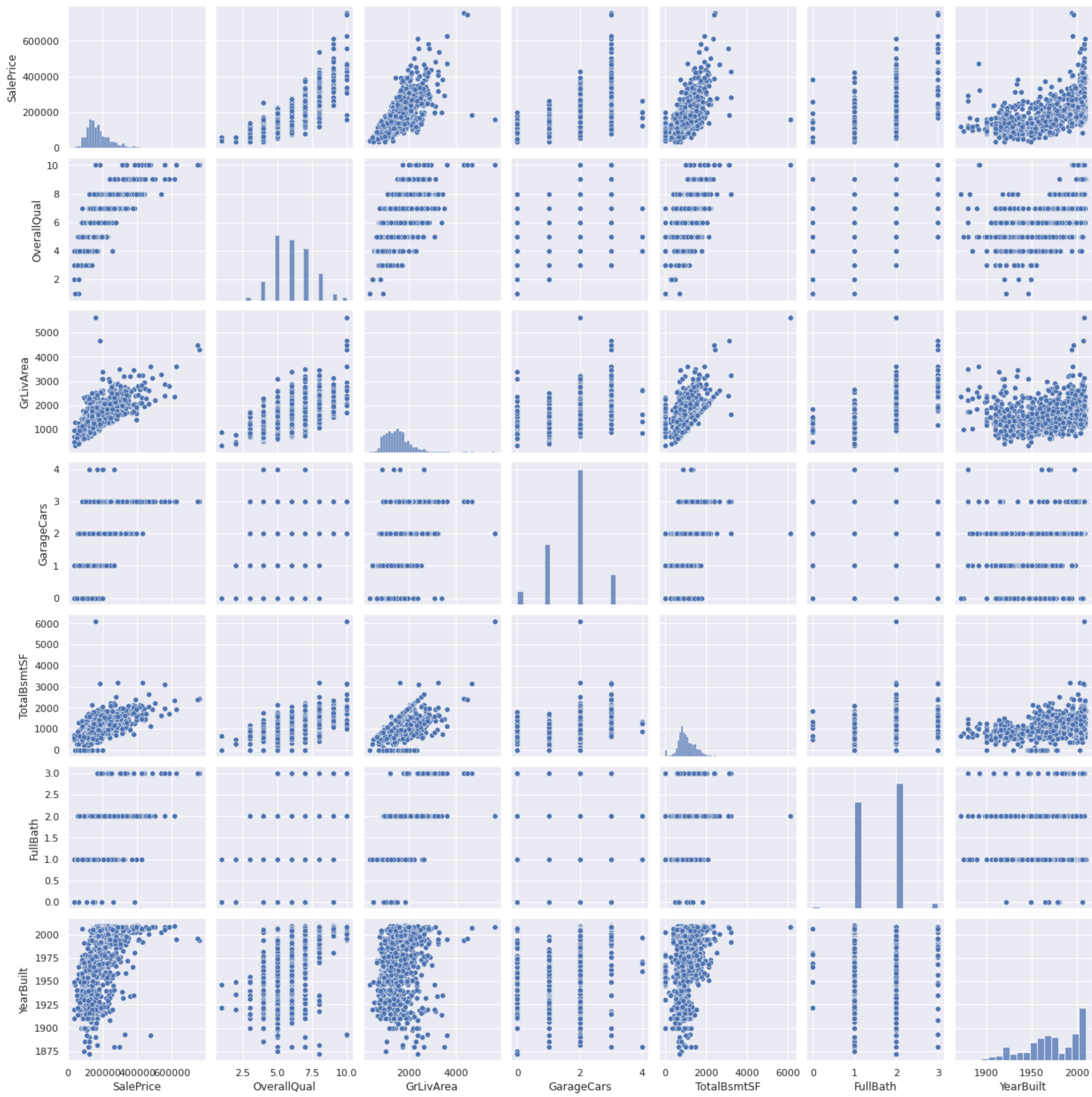
In [91]:
```
#saleprice correlation matrix
k = 10 #number of variables for heatmap
cols = corrmat.nlargest(k, 'SalePrice')['SalePrice'].index
cols
```

Out[91]:
```
Index(['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'GarageArea',
       'TotalBsmtSF', '1stFlrSF', 'FullBath', 'TotRmsAbvGrd', 'YearBuilt'],
      dtype='object')
```

In [92]:
```
cm = np.corrcoef(df_train[cols].values.T)
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'s
                 yticklabels=cols.values, xticklabels=cols.values)
plt.show()
```



In [93]:
```
#scatterplot
sns.set()
cols = ['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'TotalBsmtSF', 'Fu
        'YearBuilt']
sns.pairplot(df_train[cols], size = 2.5)
plt.show()
```



In [94]:
```
#missing data
total = df_train.isnull().sum().sort_values(ascending=False)
percent = (df_train.isnull().sum()/df_train.isnull().count().sort_values(ascendir
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(20)
```

Out[94]:

	Total	Percent
1stFlrSF	0	0.000000
2ndFlrSF	0	0.000000
3SsnPorch	0	0.000000
Alley	1369	0.937671
BedroomAbvGr	0	0.000000
BldgType	0	0.000000
BsmtCond	37	0.025342
BsmtExposure	38	0.026027
BsmtFinSF1	0	0.000000
BsmtFinSF2	0	0.000000
BsmtFinType1	37	0.025342
BsmtFinType2	38	0.026027
BsmtFullBath	0	0.000000
BsmtHalfBath	0	0.000000
BsmtQual	37	0.025342
BsmtUnfSF	0	0.000000
CentralAir	0	0.000000
Condition1	0	0.000000
Condition2	0	0.000000
Electrical	1	0.000685

In [95]:
```
df_train.isnull().sum().max() #just checking that there's any missing data missir
```

Out[95]:
```
1453
```

In [96]:
```
#dealing with missing data
df_train = df_train.drop((missing_data[missing_data['Total'] > 1]).index,1)
df_train = df_train.drop(df_train.loc[df_train['Electrical'].isnull()].index)
df_train.isnull().sum().max() #just checking that there's no missing data missing
```

Out[96]:
```
0
```

outliers can markedly affect our models and can be a valuable source of information

Outliers is a complex subject and it deserves more attention. Here, we'll just do a quick analysis through the standard deviation of 'SalePrice' and a set of scatter plots.

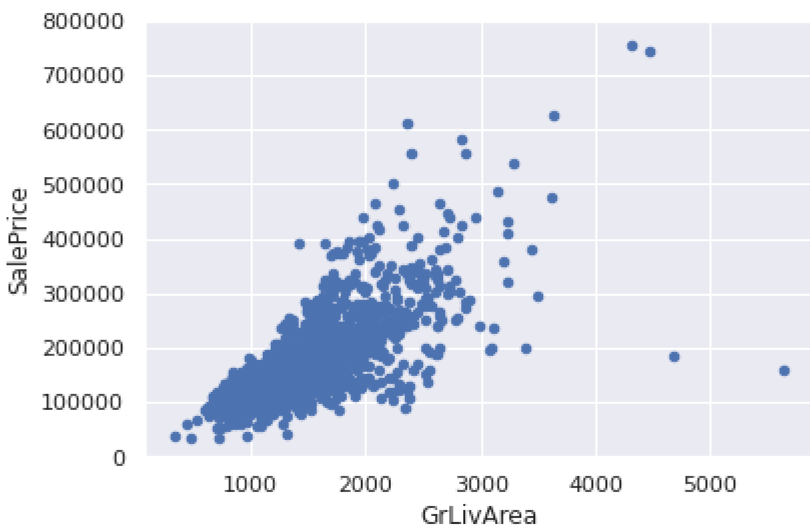
In [97]:
```
#standardizing data
saleprice_scaled = StandardScaler().fit_transform(df_train['SalePrice'][:,np.newa
low_range = saleprice_scaled[saleprice_scaled[:,0].argsort()][:10]
high_range= saleprice_scaled[saleprice_scaled[:,0].argsort()][-10:]
print('outer range (low) of the distribution:')
print(low_range)
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]]
```

In [100]:
```
#bivariate analysis saleprice/grlivarea
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 the *color* keyword-argument or provide a 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all point
s.
```



In [101]:
```
#deleting points
df_train.sort_values(by = 'GrLivArea', ascending = False)[:2]
```

Out[101]:

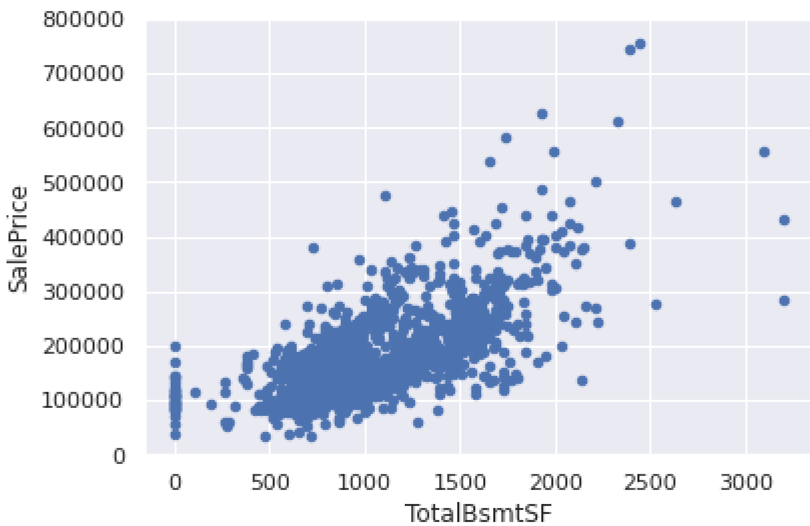
	Id	MSSubClass	MSZoning	LotArea	Street	LotShape	LandContour	Utilities	LotConfig
1298	1299	60	RL	63887	Pave	IR3	Bnk	AllPub	Corner
523	524	60	RL	40094	Pave	IR1	Bnk	AllPub	Inside

2 rows × 63 columns

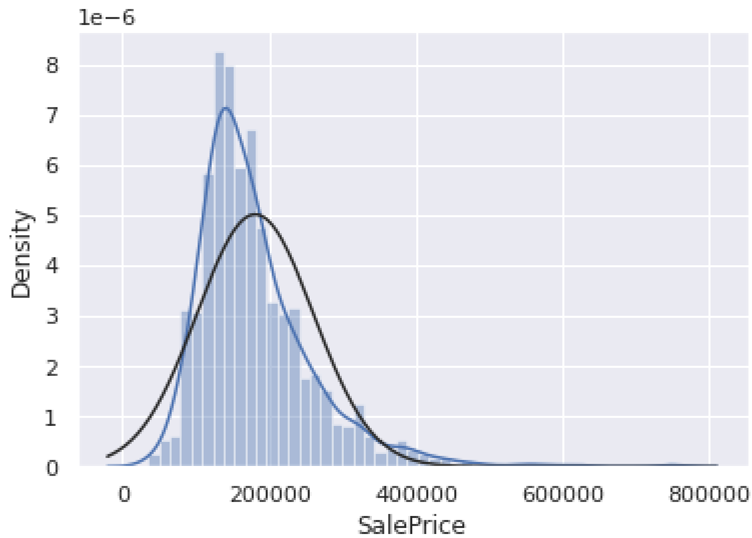
In [102]:
```
df_train = df_train.drop(df_train[df_train['Id'] == 1299].index)
df_train = df_train.drop(df_train[df_train['Id'] == 524].index)
```

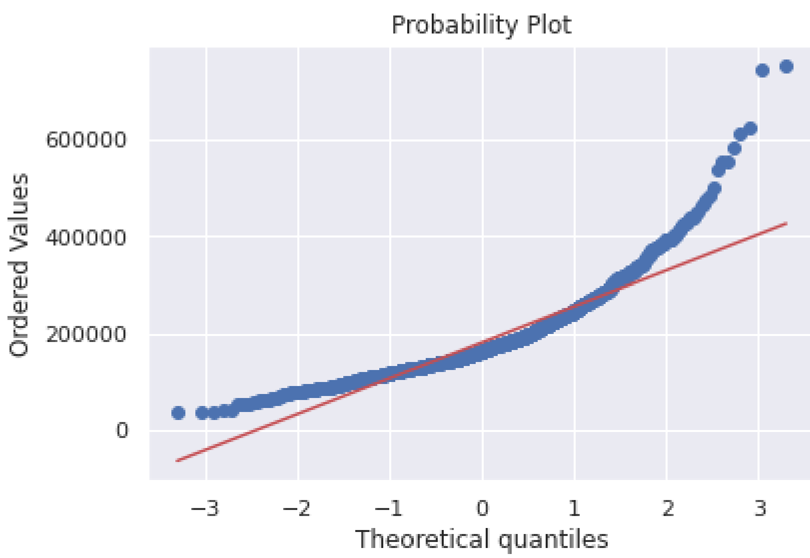
In [103]:
```
#bivariate analysis saleprice/grlivarea
var = 'TotalBsmtSF'
data = pd.concat([df_train['SalePrice'], df_train[var]], axis=1)
data.plot.scatter(x=var, y='SalePrice', ylim=(0,800000));
```

```
*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 the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



In [105]:
```
#histogram and normal probability plot
sns.distplot(df_train['SalePrice'], fit=norm)
fig = plt.figure()
res = stats.probplot(df_train['SalePrice'], plot=plt)
```





In [106]:
```
#applying log transformation
df_train['SalePrice'] = np.log(df_train['SalePrice'])
```

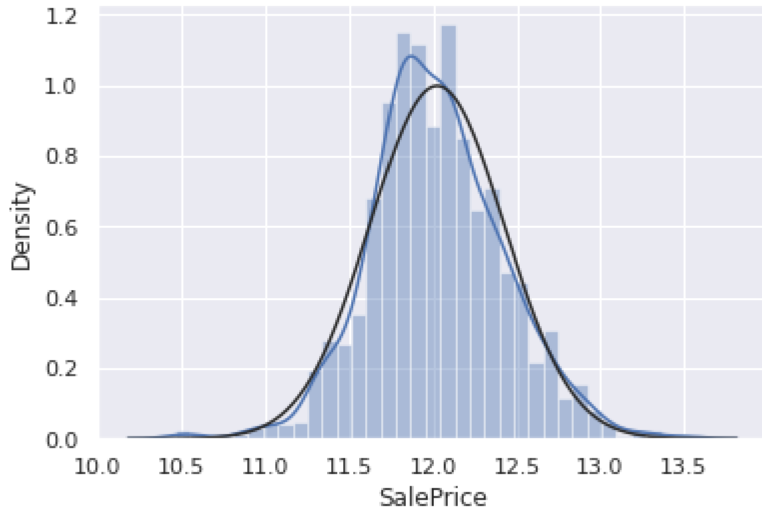
In [107]:
```
df_train
```

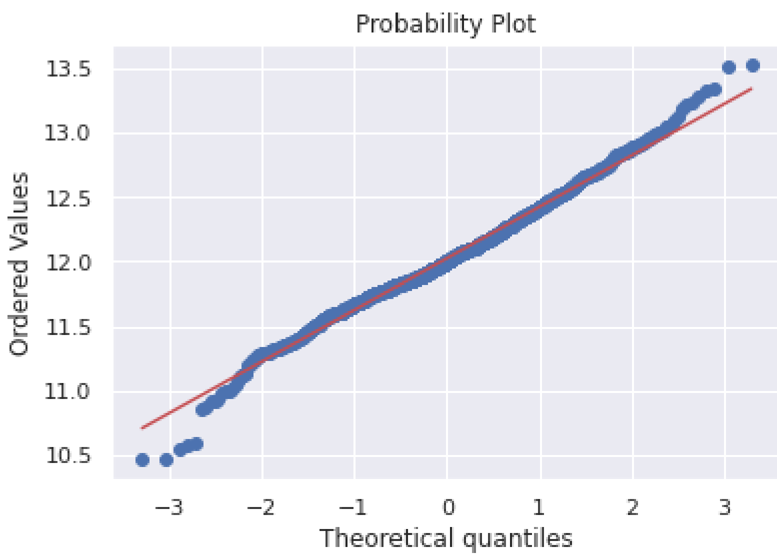
Out[107]:

	Id	MSSubClass	MSZoning	LotArea	Street	LotShape	LandContour	Utilities	LotConfig
0	1	60	RL	8450	Pave	Reg	Lvl	AllPub	Inside
1	2	20	RL	9600	Pave	Reg	Lvl	AllPub	FR2
2	3	60	RL	11250	Pave	IR1	Lvl	AllPub	Inside
3	4	70	RL	9550	Pave	IR1	Lvl	AllPub	Corner
4	5	60	RL	14260	Pave	IR1	Lvl	AllPub	FR2
...	...	...	...	...	...	...	...	...	...
1455	1456	60	RL	7917	Pave	Reg	Lvl	AllPub	Inside
1456	1457	20	RL	13175	Pave	Reg	Lvl	AllPub	Inside
1457	1458	70	RL	9042	Pave	Reg	Lvl	AllPub	Inside
1458	1459	20	RL	9717	Pave	Reg	Lvl	AllPub	Inside
1459	1460	20	RL	9937	Pave	Reg	Lvl	AllPub	Inside

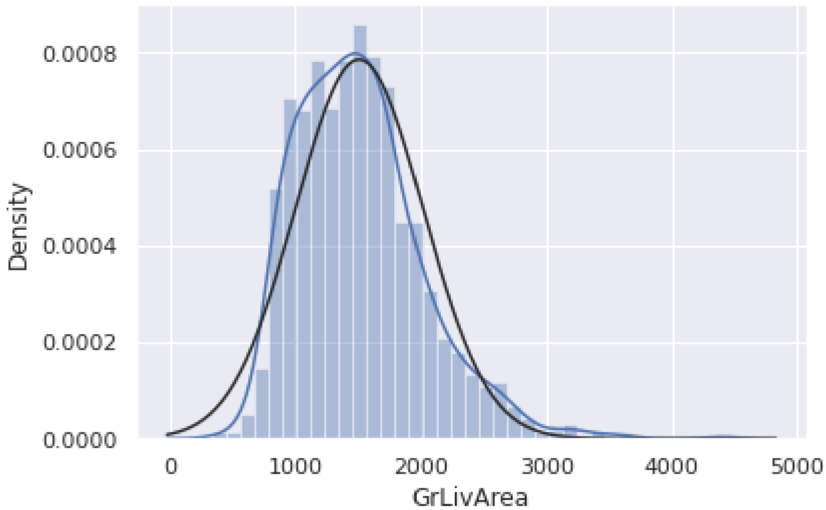
1457 rows × 63 columns

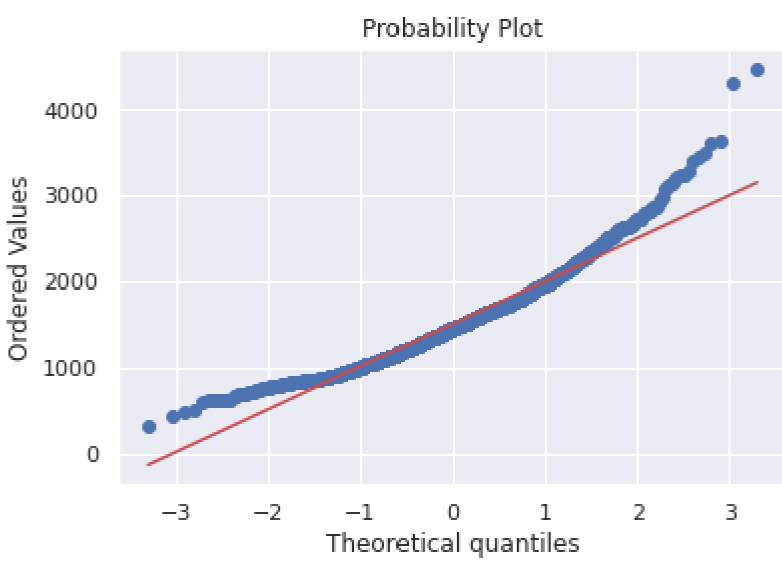
In [108]:
```
#transformed histogram and normal probability plot
sns.distplot(df_train['SalePrice'], fit=norm);
fig = plt.figure()
res = stats.probplot(df_train['SalePrice'], plot=plt)
```





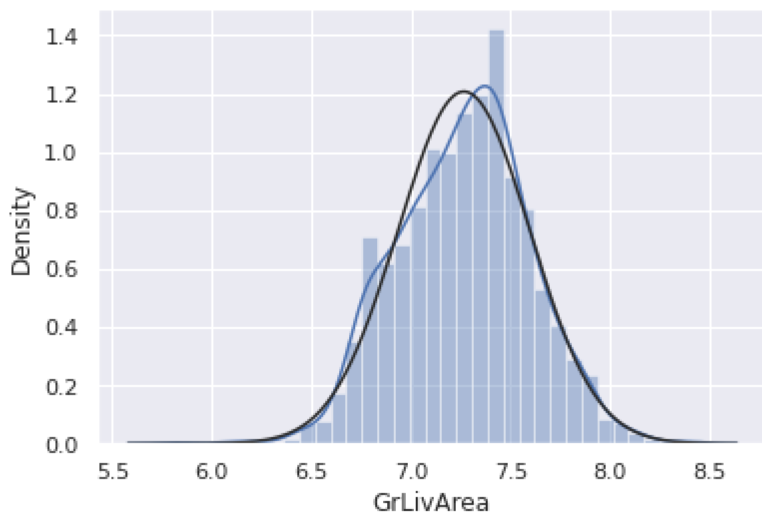
In [109]:
```
#histogram and normal probability plot
sns.distplot(df_train['GrLivArea'], fit=norm);
fig = plt.figure()
res = stats.probplot(df_train['GrLivArea'], plot=plt)
```

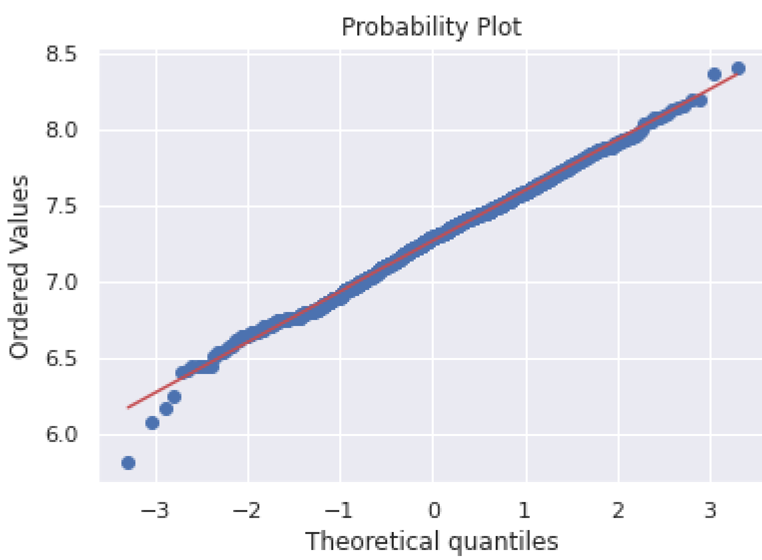




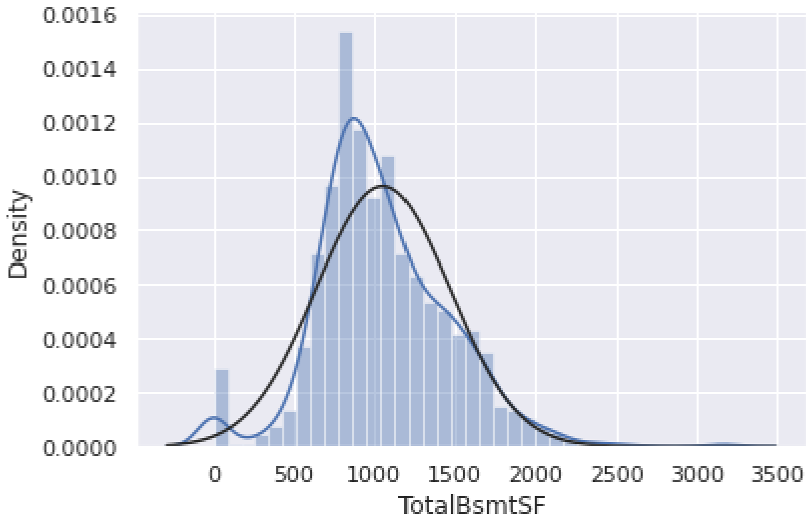
In [110]:
```
#data transformation
df_train['GrLivArea'] = np.log(df_train['GrLivArea'])
```

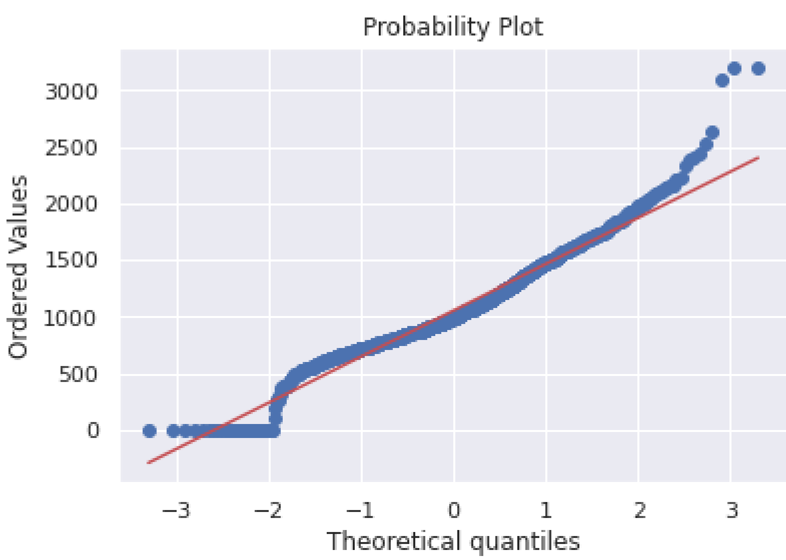
In [111]:
```
#transformed histogram and normal probability plot
sns.distplot(df_train['GrLivArea'], fit=norm);
fig = plt.figure()
res = stats.probplot(df_train['GrLivArea'], plot=plt)
```





In [112]:
```
#histogram and normal probability plot
sns.distplot(df_train['TotalBsmtSF'], fit=norm)
fig = plt.figure()
res = stats.probplot(df_train['TotalBsmtSF'], plot=plt)
```

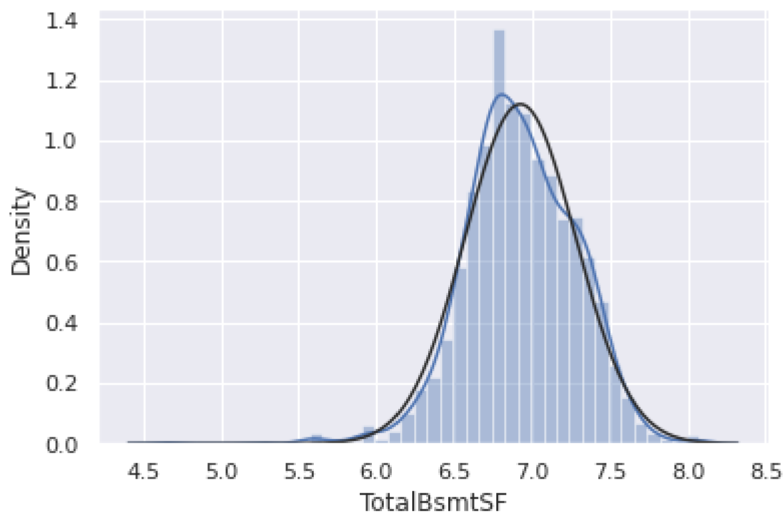


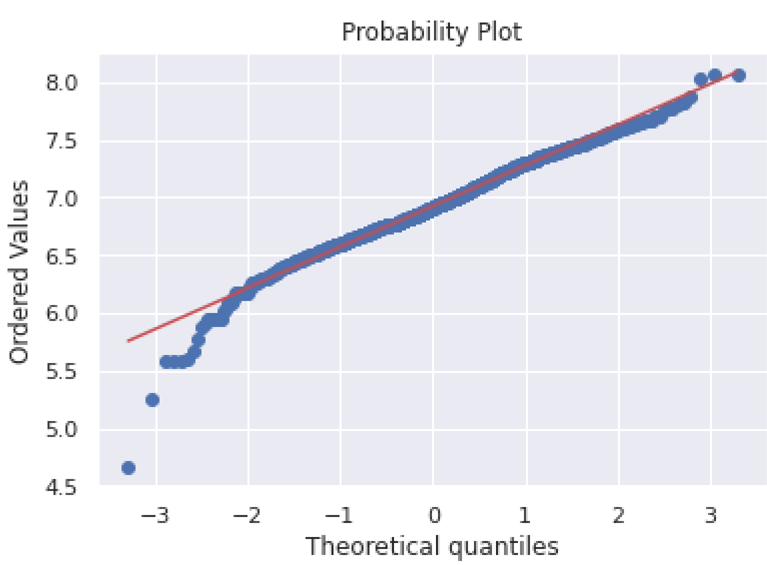


In [113]:
```
#create column for new variable (one is enough because it's a binary categorical
#if area>0 it gets 1, for area==0 it gets 0
df_train['HasBsmt'] = pd.Series(len(df_train['TotalBsmtSF']), index = df_train.in
df_train['HasBsmt'] = 0
df_train.loc[df_train['TotalBsmtSF']>0,'HasBsmt'] = 1
```

In [114]:
```
#transform data
df_train.loc[df_train['HasBsmt']==1,'TotalBsmtSF'] = np.log(df_train['TotalBsmtSF
```

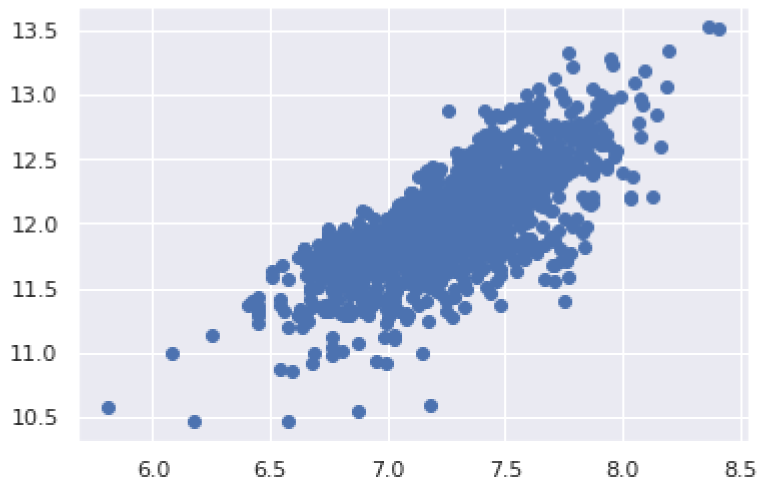
In [115]:
```
#histogram and normal probability plot
sns.distplot(df_train[df_train['TotalBsmtSF']>0]['TotalBsmtSF'], fit=norm);
fig = plt.figure()
res = stats.probplot(df_train[df_train['TotalBsmtSF']>0]['TotalBsmtSF'], plot=plt
```



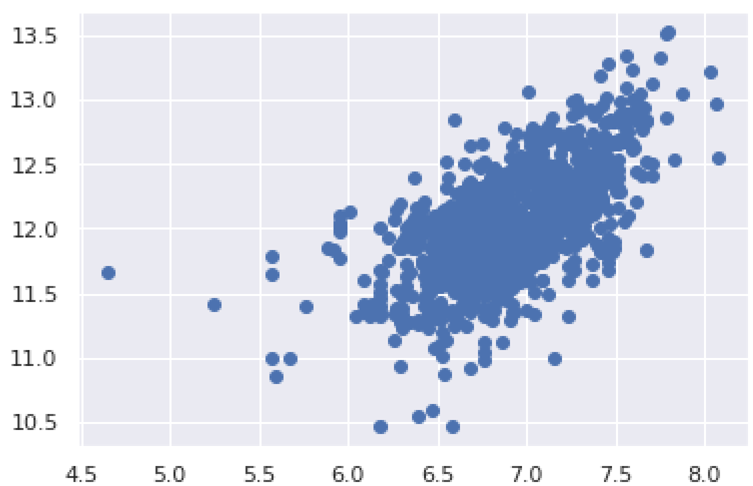


In [116]:
```
#scatter plot
plt.scatter(df_train['GrLivArea'], df_train['SalePrice'])
```

Out[116]: <matplotlib.collections.PathCollection at 0x7f0693b49be0>



In [117]:
```
#scatter plot
plt.scatter(df_train[df_train['TotalBsmtSF']>0]['TotalBsmtSF'], df_train[df_trair
```



In [118]:
```
#convert categorical variable into dummy
df_train = pd.get_dummies(df_train)
```

In [119]:
```
df_train
```

Out[119]:

	Id	MSSubClass	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	BsmtFinSF
0	1	60	8450	7	5	2003	2003	70
1	2	20	9600	6	8	1976	1976	97
2	3	60	11250	7	5	2001	2002	48
3	4	70	9550	7	5	1915	1970	21
4	5	60	14260	8	5	2000	2000	65
...	...	...	...	...	...	...	...	
1455	1456	60	7917	6	5	1999	2000	
1456	1457	20	13175	6	6	1978	1988	79
1457	1458	70	9042	7	9	1941	2006	27
1458	1459	20	9717	5	6	1950	1996	4
1459	1460	20	9937	5	6	1965	1965	83

1457 rows × 222 columns

In [ ]: