Code

In [1]:
```python
%matplotlib inline
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.feature_selection import SelectFromModel
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm,skew
from scipy import stats
```

In [2]:
```python
data = pd.read_csv("train.csv")
obj_2 = data.select_dtypes(include = ['object'])
data.head(3)
```

Out[2]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeatu
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	Na
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	Na
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	Na

3 rows × 81 columns

Number of Columns & shape of the dataset

In [3]:
```python
columns = data.columns
print(columns)
```

```
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 [4]:
```python
data.shape
```

Out[4]:
```
(1460, 81)
```

In [ ]:

In [5]:
```python
#List of columns in the dataset, and we can see that there are 80 columns in this dataset which mean that we m

categorical = ['MSSubClass','ExterQual','ExterCond']
for category in categorical:
    data[category] = data[category].astype('str')
```

In [6]:
```python
#Data Cleaning
#Check for columns with large number of missing values and drop them
null = [column for column in columns if data[column].isnull().sum() > (0.5*len(data))]
print("Columns with missing value is:\n{}".format(null))
```

```
Columns with missing value is:
['Alley', 'PoolQC', 'Fence', 'MiscFeature']
```

In [7]:
```python
for column in null:
    data.drop(column,axis = 1,inplace = True)
data.drop("Id",axis = 1, inplace = True)
columns = data.columns
```

In [8]:
```python
#Fixing null values in remaining columns
missing = [column for column in columns if data[column].isnull().sum() > 0]
missing
```

Out[8]:
```
['LotFrontage',
 'MasVnrType',
 'MasVnrArea',
 'BsmtQual',
 'BsmtCond',
 'BsmtExposure',
 'BsmtFinType1',
 'BsmtFinType2',
 'Electrical',
 'FireplaceQu',
 'GarageType',
 'GarageYrBlt',
 'GarageFinish',
 'GarageQual',
 'GarageCond']
```

In [9]:
```python
# Lot Frontage
data["LotFrontage"] = data.groupby("Neighborhood")["LotFrontage"].transform(
    lambda x: x.fillna(x.median()))
```

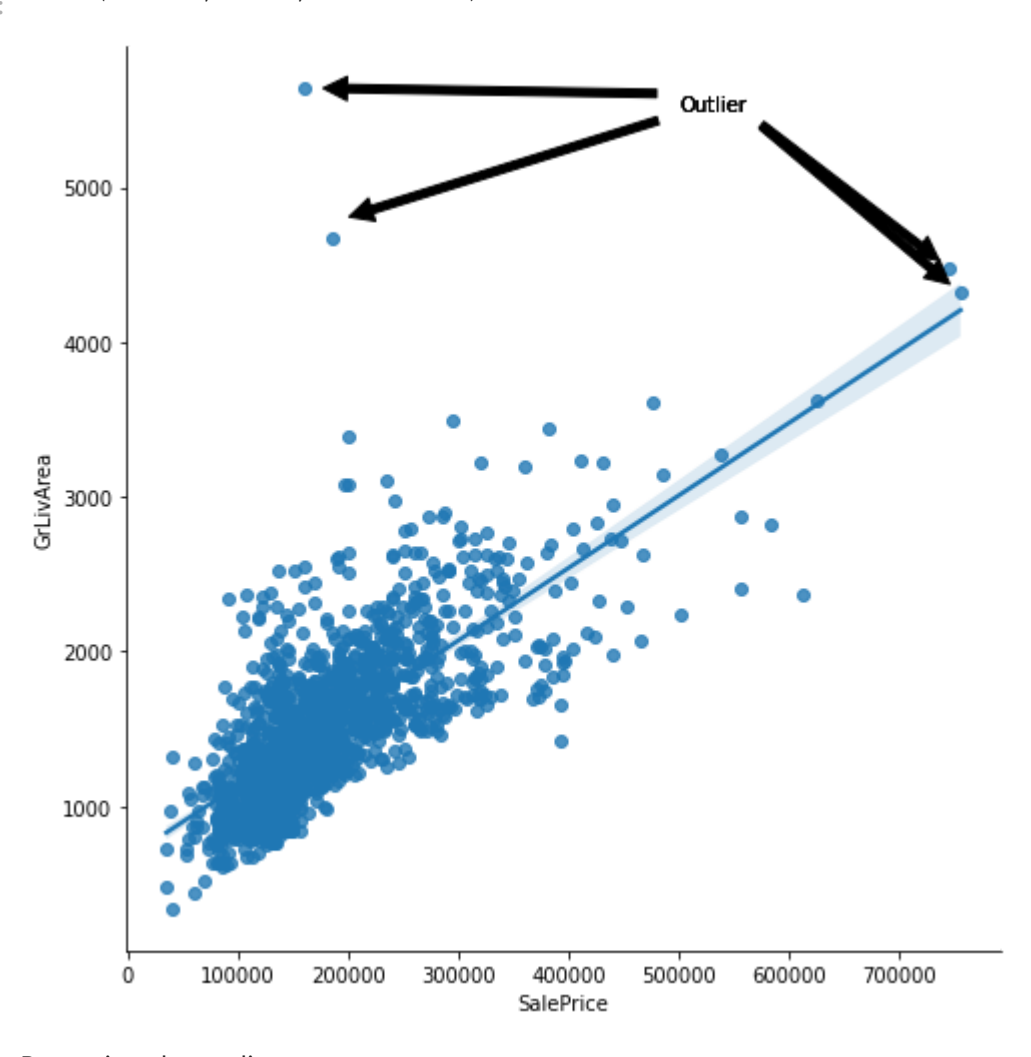
In [10]:
```python
# MasVnrType and Area
data["MasVnrType"] = data["MasVnrType"].fillna("None")
data["MasVnrArea"] = data["MasVnrArea"].fillna(0)
for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2'):
    data[col] = data[col].fillna('None')
data['Electrical'] = data['Electrical'].fillna(data['Electrical'].mode()[0])

# FireplaceQu
data["FireplaceQu"] = data["FireplaceQu"].fillna("None")
for col in ('GarageType', 'GarageFinish', 'GarageQual', 'GarageCond'):
    data[col] = data[col].fillna('None')

# GarageYrBlt
data['GarageYrBlt'] = data['GarageYrBlt'].fillna(0)
```

Treatment of outliers in GrLivArea columns due to documentation advice

In [11]:
```python
sns.lmplot('SalePrice','GrLivArea',size = 7,data = data)
plt.annotate('Outlier', xy=(160000,5642), xytext=(500000, 5500),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )
plt.annotate('Outlier', xy=(184750,4774), xytext=(500000, 5500),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )
plt.annotate('Outlier', xy=(755000,4316), xytext=(500000, 5500),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )
plt.annotate('Outlier', xy=(745000,4476), xytext=(500000, 5500),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )
```

```
C:\Users\ocsas\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variabl
es as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing oth
er arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
C:\Users\ocsas\anaconda3\lib\site-packages\seaborn\regression.py:581: UserWarning: The `size` parameter has bee
n renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```

Out[11]:
```
Text(500000, 5500, 'Outlier')
```



Removing the outliers

In [12]:
```python
data = data[data['GrLivArea'] < 4000]
data.head()
```

Out[12]:

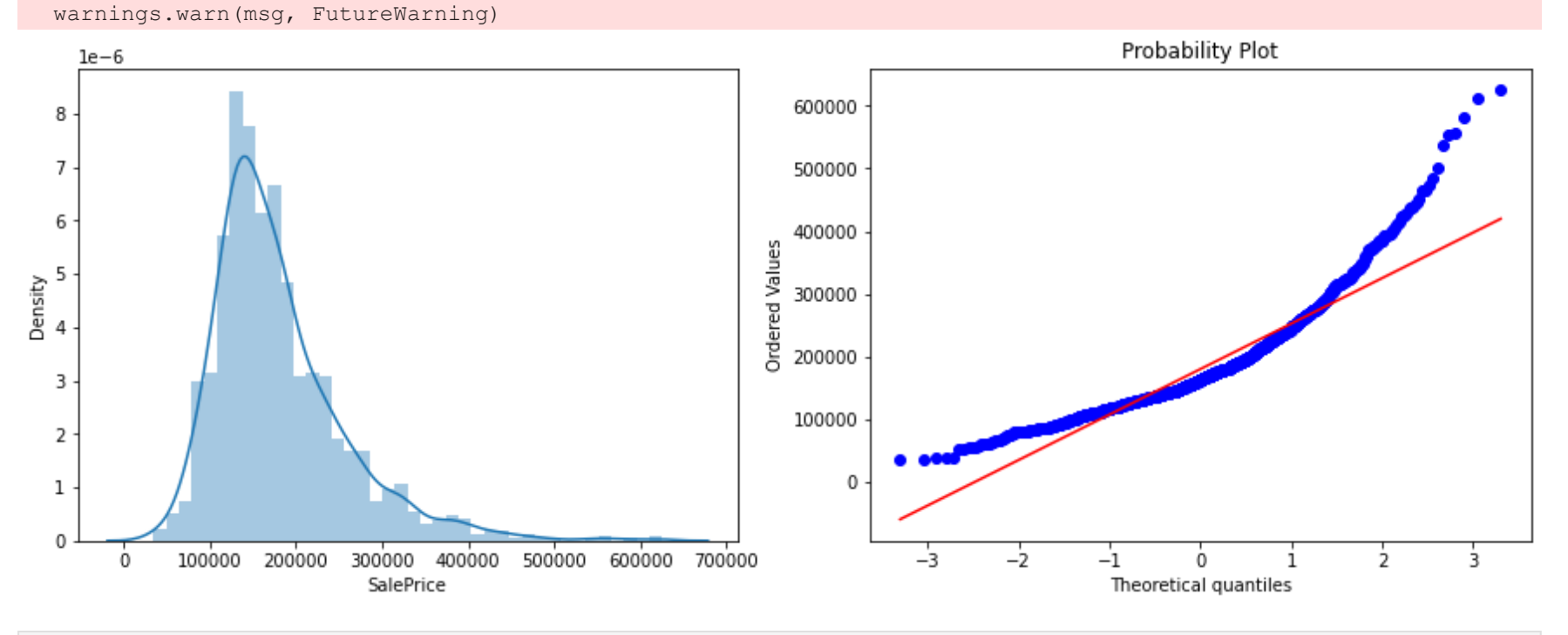
	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	...	EnclosedPorch	3SsnPorc
0	60	RL	65.0	8450	Pave	Reg	Lvl	AllPub	Inside	Gtl	...	0	
1	20	RL	80.0	9600	Pave	Reg	Lvl	AllPub	FR2	Gtl	...	0	
2	60	RL	68.0	11250	Pave	IR1	Lvl	AllPub	Inside	Gtl	...	0	
3	70	RL	60.0	9550	Pave	IR1	Lvl	AllPub	Corner	Gtl	...	272	
4	60	RL	84.0	14260	Pave	IR1	Lvl	AllPub	FR2	Gtl	...	0	

5 rows × 76 columns

How is the sales price distributed

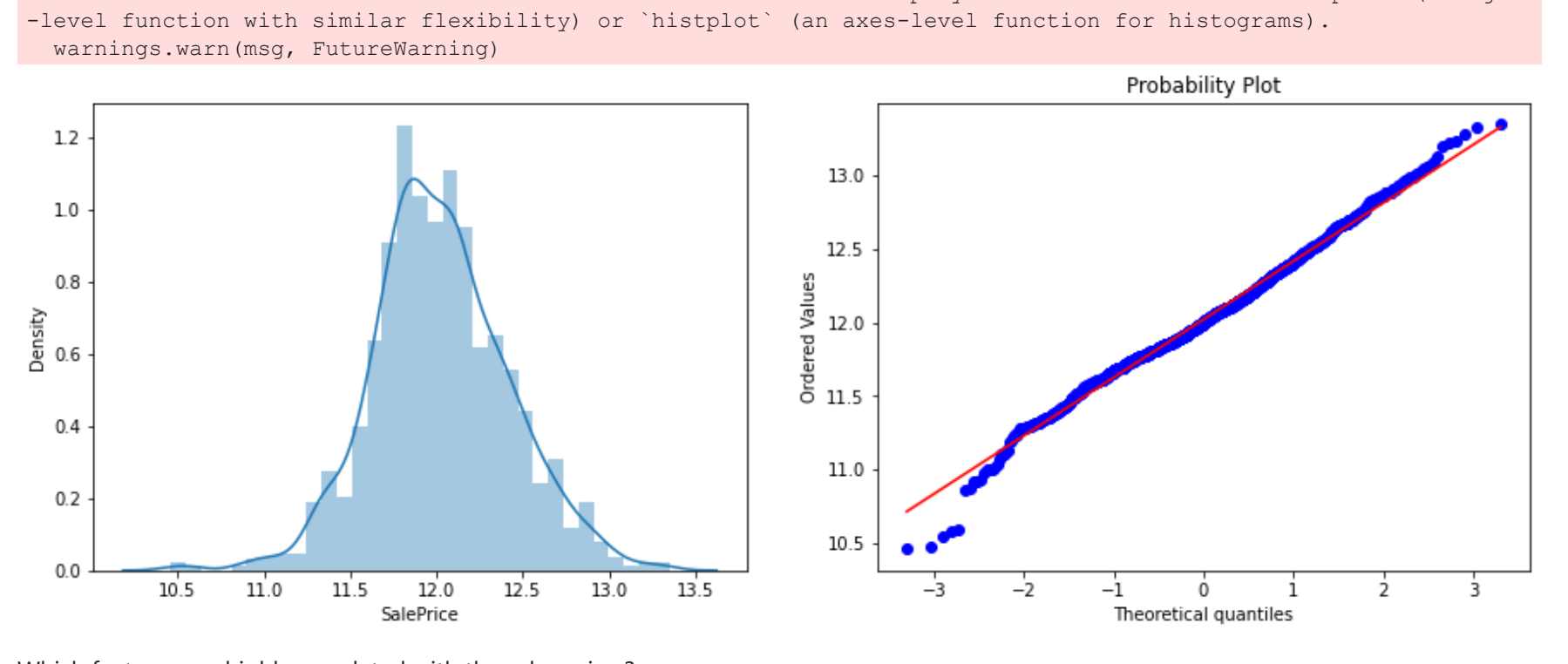
In [13]:
```python
#Data Exploration
#
data['SalePrice'].describe()
fig,(ax1, ax2) = plt.subplots(1, 2,figsize=(15, 5))
_ = sns.distplot(data['SalePrice'],ax = ax1)
_ = stats.probplot(data['SalePrice'], plot=plt)
```

```
C:\Users\ocsas\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprec
ated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [14]:
```python
data['SalePrice'] = np.log(data['SalePrice'])
fig,(ax1, ax2) = plt.subplots(1, 2,figsize=(15, 5))
_ = sns.distplot(data['SalePrice'],ax = ax1)
_ = stats.probplot(data['SalePrice'],plot=plt)
```

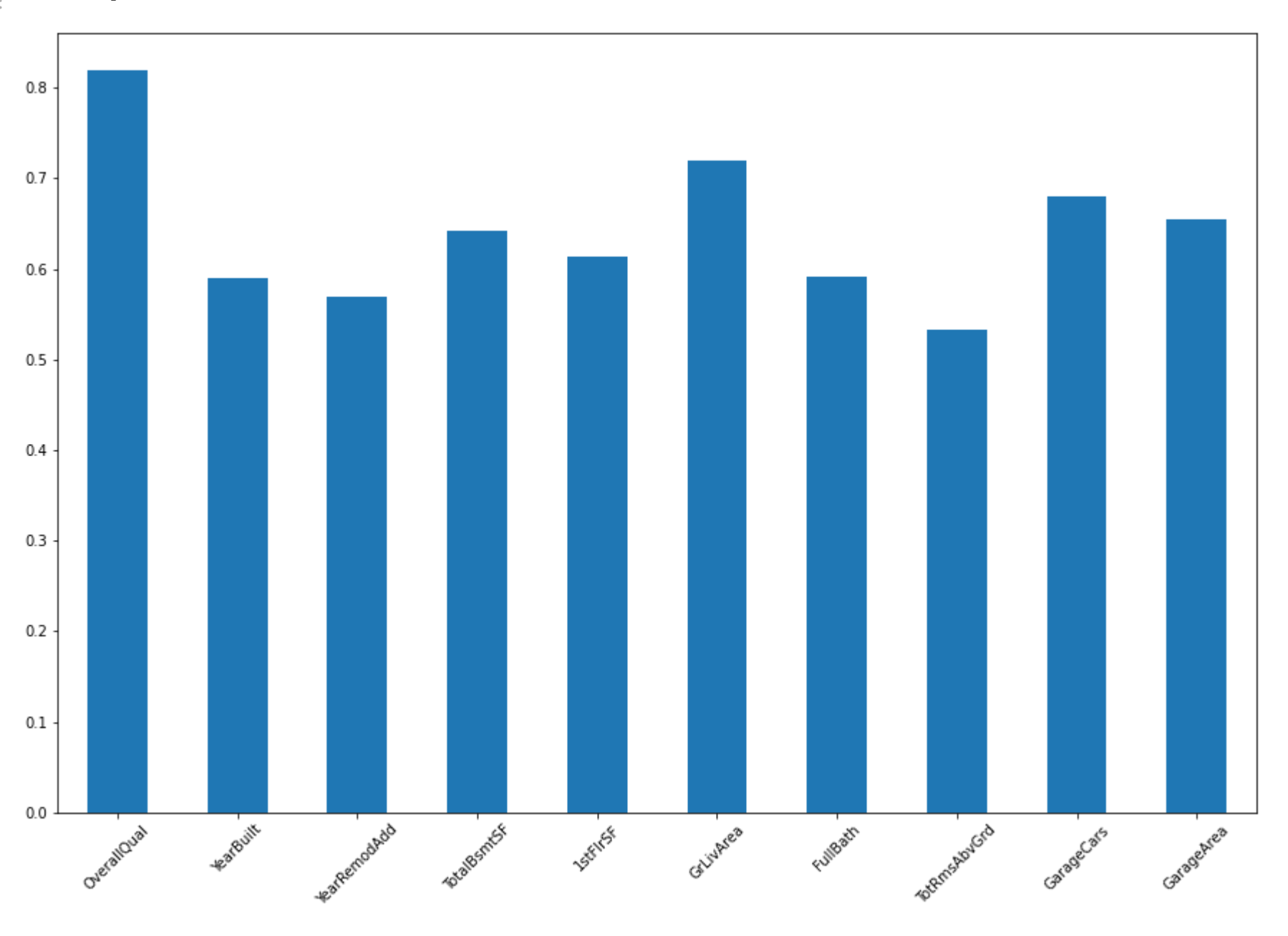
```
C:\Users\ocsas\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprec
ated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



Which features are highly correlated with the sales price ?

In [15]:
```python
#Relationship between sale price and other numerical columns

num = data.select_dtypes(include = ['int64','float64'])
corr = num.corr().loc['SalePrice']
corr[corr >= 0.5][:-1].plot(kind = 'bar',figsize = (15,10), rot = 45)
```

Out[15]:
```
<AxesSubplot:>
```



In [16]:
```python
num = num[['YearBuilt','YearRemodAdd','TotalBsmtSF', '1stFlrSF', 'GrLivArea',
          'FullBath','TotRmsAbvGrd','GarageCars', 'GarageArea','SalePrice']]
num.head()
```

Out[16]:

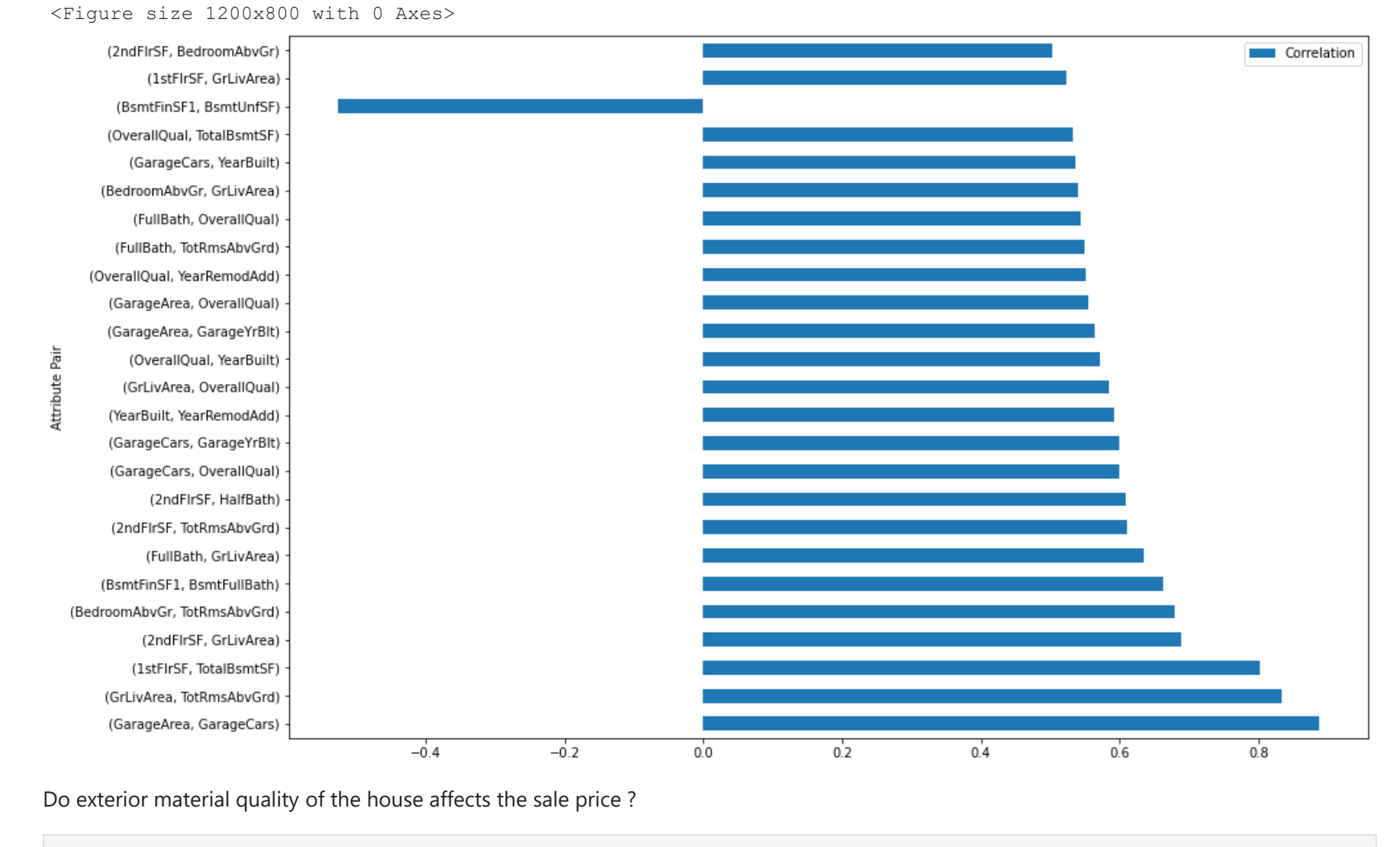
	YearBuilt	YearRemodAdd	TotalBsmtSF	1stFlrSF	GrLivArea	FullBath	TotRmsAbvGrd	GarageCars	GarageArea	SalePrice
0	2003	2003	856	856	1710	2	8	2	548	12.247694
1	1976	1976	1262	1262	1262	2	6	2	460	12.109011
2	2001	2002	920	920	1786	2	6	2	608	12.317167
3	1915	1970	756	961	1717	1	7	3	642	11.849398
4	2000	2000	1145	1145	2198	2	9	3	836	12.429216

- Relationship of variables with each other

Does the variable shows multicollinearity ?

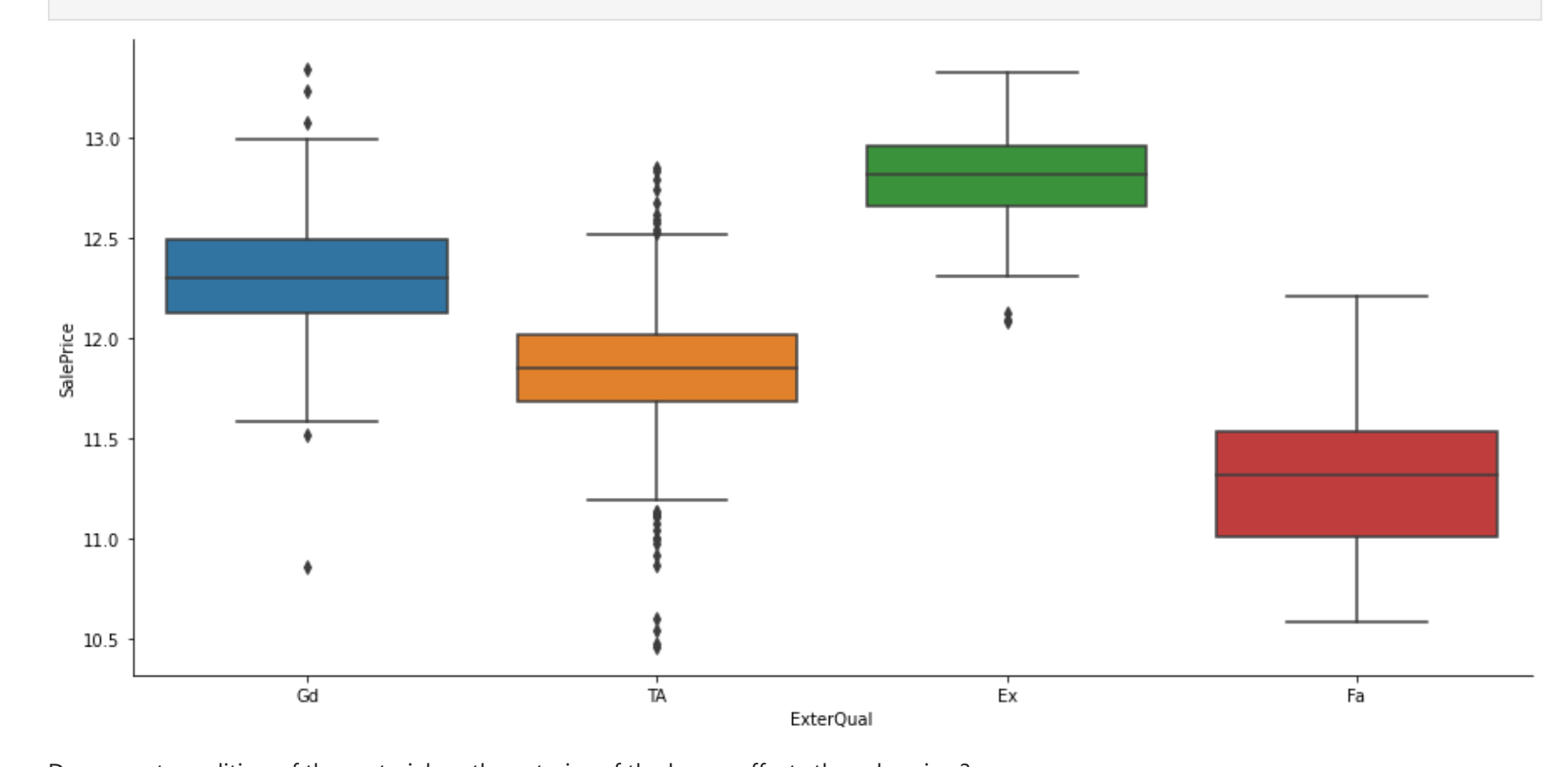
In [17]:
```python
correlations=data.corr()
attrs = correlations.iloc[:-1,:-1] # all except target

threshold = 0.5
important_corrs = (attrs[abs(attrs) > threshold][attrs != 1.0]) \
    .unstack().dropna().to_dict()

unique_important_corrs = pd.DataFrame(
    list(set([(tuple(sorted(key)), important_corrs[key]) \
    for key in important_corrs])),
        columns=['Attribute Pair', 'Correlation'])

    # sorted by absolute value
unique_important_corrs = unique_important_corrs.loc[
    abs(unique_important_corrs['Correlation']).argsort()[::-1]]
plt.figure(dpi = 200)
unique_important_corrs.plot(x = 'Attribute Pair', y = 'Correlation',kind = 'barh',figsize =(15,10))
```

Out[17]:
```
<AxesSubplot:ylabel='Attribute Pair'>
<Figure size 1200x800 with 0 Axes>
```



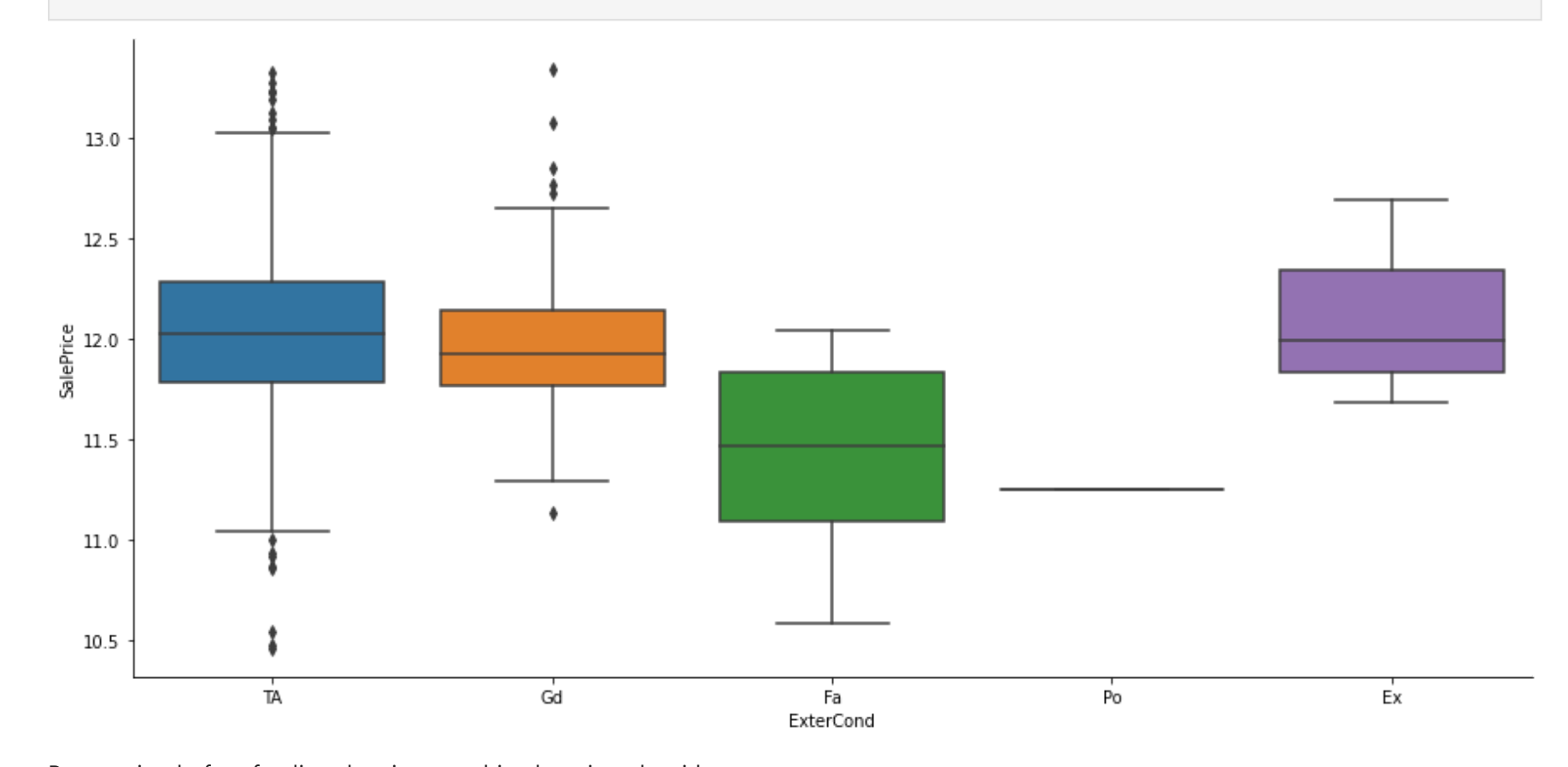
Do exterior material quality of the house affects the sale price ?

In [18]:
```python
obj = data.select_dtypes(include = ['object'])
obj_2 = data.select_dtypes(include = ['object'])
data_object = pd.concat([num['SalePrice'],obj['ExterQual']],axis = 1)
plt.figure(figsize = (15,7))
sns.boxplot(x = 'ExterQual', y = 'SalePrice',data = data)
sns.despine()
```



Do present condition of the material on the exterior of the house affects the sale price ?

In [19]:
```python
data_object = pd.concat([num['SalePrice'],obj['ExterCond']],axis = 1)
plt.figure(figsize = (15,7))
sns.boxplot(x = 'ExterCond', y = 'SalePrice',data = data)
sns.despine()
```



Preparation before feeding data into machine learning algorithm

In [20]:
```python
#Check for skewness in numerical data
target = data['SalePrice']
data = data.iloc[:,:-1]

skewness = data.select_dtypes(include = ['int64','float64']).columns
skewed_feats = data[skewness].apply(lambda x: skew(x.dropna())).sort_values(ascending=False)
skewness = pd.DataFrame({'Skew' :skewed_feats})
skewness.head(10)
```

Out[20]:

	Skew
MiscVal	24.418175
PoolArea	17.504556
LotArea	12.574590
3SsnPorch	10.279262
LowQualFinSF	8.989291
KitchenAbvGr	4.476748
BsmtFinSF2	4.244209
BsmtHalfBath	4.124712
ScreenPorch	4.111400
EnclosedPorch	3.081275

In [21]:
```python
skewness = skewness[abs(skewness) > 0.75]
print("There are {} skewed numerical features to Box Cox transformed".format(skewness.shape[0]))

from scipy.special import boxcox1p
skewed_features = skewness.index
lam = 0.15
for feat in skewed_features:
    #all_data[feat] += 1
    data[feat] = boxcox1p(data[feat], lam)
```

```
There are 35 skewed numerical features to Box Cox transformed
```

In [22]:
```python
#Getting dummy for categorical features
le = LabelEncoder()
obj = obj.apply(le.fit_transform)
data = pd.get_dummies(data)
data.shape


from sklearn.model_selection import train_test_split
X_train,X_test,y_train, y_test = train_test_split(data,target,random_state = 1)
```

In [23]:
```python
#Importing libraries
from sklearn.linear_model import ElasticNet, Lasso,  BayesianRidge, LassoLarsIC
from sklearn.ensemble import RandomForestRegressor,  GradientBoostingRegressor
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import KFold, cross_val_score, train_test_split,GridSearchCV,ShuffleSplit,learning
from sklearn.metrics import mean_squared_error
```

In [24]:
```python
#Validation Function
n_folds = 5

def rmsle_cv(model):
    kf = KFold(n_folds, shuffle=True, random_state=42).get_n_splits(X_train.values)
    rmse= np.sqrt(-cross_val_score(model, X_train.values, y_train, scoring="neg_mean_squared_error", cv = kf))
    return(rmse)
```

In [25]:
```python
#Initialize all base models
lasso = make_pipeline(RobustScaler(), Lasso(random_state = 3))
ENet = make_pipeline(RobustScaler(),ElasticNet(random_state=3))
GBoost = GradientBoostingRegressor(random_state =3)
```

In [26]:
```python
#Results
score = rmsle_cv(lasso)
print("Lasso score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
score = rmsle_cv(ENet)
print("Elastic Net score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
score = rmsle_cv(GBoost)
print("Gradient Boosting score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
```

```
Lasso score: 0.3662 (0.0108)

Elastic Net score: 0.3657 (0.0110)

Gradient Boosting score: 0.1268 (0.0116)
```

Can we improve the algorithm ?

Grid Search for Lasso

In [27]:
```python
clf = Lasso(random_state = 3)
grid_values = {'alpha': [0.001, 0.01, 0.05, 0.1, 1, 10, 100]}

# default metric to optimize over grid parameters: accuracy
grid_clf_acc = GridSearchCV(clf,scoring = 'neg_mean_squared_error', cv = 5,param_grid = grid_values)
grid_clf_acc.fit(X_train, y_train)

print('Grid best parameter (Min. Mean Squared Error): ', grid_clf_acc.best_params_)
print('Grid best score (Mean Squared Error): ', grid_clf_acc.best_score_)
```

```
Grid best parameter (Min. Mean Squared Error):  {'alpha': 0.001}
Grid best score (Mean Squared Error):  -0.012640104655994803
```

Grid Search for Elastic Net

In [28]:
```python
clf = ElasticNet(random_state = 3)
grid_values = {'alpha': [0.001, 0.01, 0.05, 0.1, 1, 10, 100],
               'l1_ratio': [0.1,0.3,0.5,0.7,0.9]}

# default metric to optimize over grid parameters: accuracy
grid_clf_acc = GridSearchCV(clf,scoring = 'neg_mean_squared_error', cv = 5,param_grid = grid_values)
grid_clf_acc.fit(X_train, y_train)

print('Grid best parameter (Min. Mean Squared Error): ', grid_clf_acc.best_params_)
print('Grid best score (Mean Squared Error): ', grid_clf_acc.best_score_)
```

```
Grid best parameter (Min. Mean Squared Error):  {'alpha': 0.001, 'l1_ratio': 0.5}
Grid best score (Mean Squared Error):  -0.012024791050936933
```

Grid Search for Gradient Boosting

In [34]:
```python
clf = GradientBoostingRegressor(warm_start = True,random_state = 3)
grid_values = {'loss': ['ls','huber'],
               'learning_rate': [0.1,0.5],
               'max_depth': [3,5],
               'n_estimators': [100,300]}

# default metric to optimize over grid parameters: accuracy
grid_clf_acc = GridSearchCV(clf,scoring = 'neg_mean_squared_error', cv = 2,param_grid = grid_values)
grid_clf_acc.fit(X_train, y_train)

print('Grid best parameter (Min. Mean Squared Error): ', grid_clf_acc.best_params_)
print('Grid best score (Mean Squared Error): ', grid_clf_acc.best_score_)
```

```
Grid best parameter (Min. Mean Squared Error):  {'learning_rate': 0.1, 'loss': 'ls', 'max_depth': 3, 'n_estimat
ors': 300}
Grid best score (Mean Squared Error):  -0.017096473459364592
```

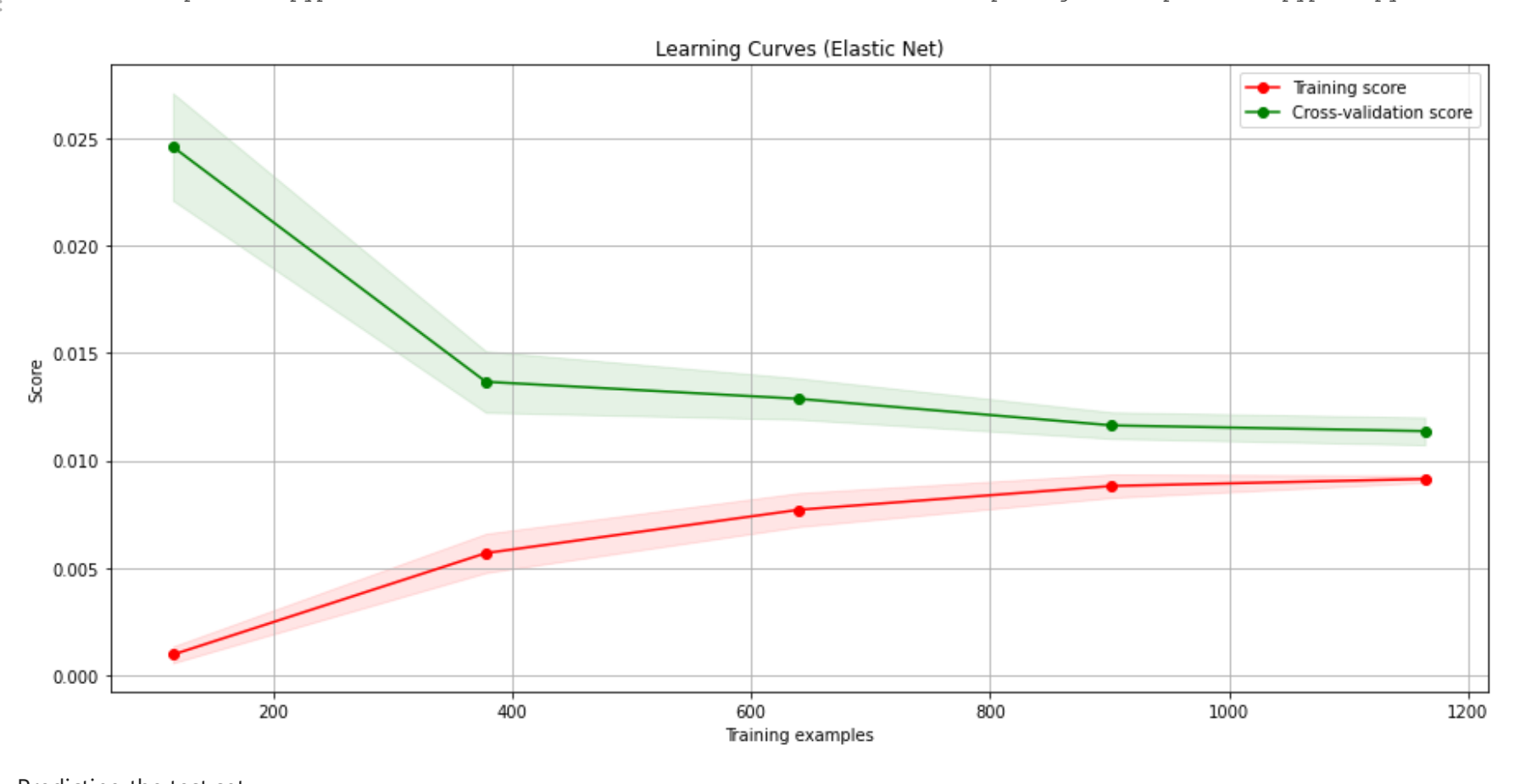
In [30]:
```python
#Initialize all tuned models
lasso = make_pipeline(RobustScaler(), Lasso(alpha = 0.001,random_state = 3))
ENet = make_pipeline(RobustScaler(),ElasticNet(alpha = 0.001, l1_ratio = 0.3,random_state=3))
GBoost = GradientBoostingRegressor(learning_rate = 0.1, loss = 'huber', max_depth = 3,n_estimators = 300,randon
score = rmsle_cv(lasso)
print("Lasso score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
score = rmsle_cv(ENet)
print("Elastic Net score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
score = rmsle_cv(GBoost)
print("Gradient Boosting score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
```

```
Lasso score: 0.1087 (0.0135)

Elastic Net score: 0.1074 (0.0145)

Gradient Boosting score: 0.1220 (0.0137)
```

In [35]:
```python
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure(figsize = (15,7))
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes,train_scores,test_scores = learning_curve(
        estimator, X, y,scoring = 'neg_mean_squared_error',cv=cv, n_jobs=n_jobs,
        train_sizes=train_sizes)
    train_scores_mean = abs(np.mean(train_scores, axis=1))
    train_scores_std = abs(np.std(train_scores, axis=1))
    test_scores_mean = abs(np.mean(test_scores, axis=1))
    test_scores_std = abs(np.std(test_scores, axis=1))
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt
```

In [32]:
```python
title = "Learning Curves (Elastic Net)"
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)

estimator = ENet
plot_learning_curve(estimator, title, data, target, cv=cv, n_jobs=4)
```

Out[32]:
```
<module 'matplotlib.pyplot' from 'C:\\Users\\ocsas\\anaconda3\\lib\\site-packages\\matplotlib\\pyplot.py'>
```



Predicting the test set

In [33]:
```python
ENet.fit(X_train,y_train)

y_pred = ENet.predict(X_test)
mse = mean_squared_error(y_test,y_pred)
rmse = np.sqrt(mse)
rmse
```

Out[33]:
```
0.11835193967533453
```