Introduction

A house value is simply more than location and square footage. Like the features that make up a person, an educated party would want to know all aspects that give a house its value.

We are going to take advantage of all of the feature variables available to use and use it to analyze and predict house prices.

We are going to break everything into logical steps that allow us to ensure the cleanest, most realistic data for our model to make accurate predictions from.

1.Load Data and Packages 2.Analyzing the Test Variable (Sale Price) 3.Multivariable Analysis 4.Impute Missing Data and Clean Data 5.Feature Transformation/Engineering 6.Modeling and Predictions

1. Loading Data and Packages

import pandas as pd import numpy as np import seaborn as sns import matplotlib import matplotlib.pyplot as plt import warnings import xgboost as xgb import lightgbm as lgb from scipy.stats import skew from scipy import stats from scipy.stats.stats import pearsonr from scipy.stats import norm from collections import Counter from sklearn.linear_model import LinearRegression,LassoCV, Ridge, LassoLarsCV,ElasticNetCV from sklearn.model_selection import GridSearchCV, cross_val_score, learning_curve from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, ExtraTreesRegressor, GradientBoostingRegressor from sklearn.preprocessing import StandardScaler, Normalizer, RobustScaler warnings.filterwarnings('ignore') sns.set(style='white', context='notebook', palette='deep') %config InlineBackend.figure_format = 'retina' #set 'png' here when working on notebook %matplotlib inline

In [3]:

```
# Load train and Test set
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
```

In [4]:

```
# Check the numbers of samples and features
print("The train data size before dropping Id feature is : {} ".format(train.shape))
print("The test data size before dropping Id feature is : {} ".format(test.shape))

# Save the 'Id' column
train_ID = train['Id']
test_ID = test['Id']

# Now drop the 'Id' column since it's unnecessary for the prediction process.
train.drop("Id", axis = 1, inplace = True)
test.drop("Id", axis = 1, inplace = True)

# Check data size after dropping the 'Id' variable
print("\nThe train data size after dropping Id feature is : {} ".format(train.shape))
print("The test data size after dropping Id feature is : {} ".format(test.shape))
```

```
The train data size before dropping Id feature is : (1460, 81)
The test data size before dropping Id feature is : (1459, 80)

The train data size after dropping Id feature is : (1460, 80)
The test data size after dropping Id feature is : (1459, 79)
```

In [5]:

```
train.head()
```

Out[5]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Util
0	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	Al
1	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	Al
2	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	Al
3	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	Al
4	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	Al

5 rows × 80 columns

In [6]:

```
test.head()
```

Out[6]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Util
0	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	Al
1	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	Al
2	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	Al
3	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	Al
4	120	RL	43.0	5005	Pave	NaN	IR1	HLS	Al

5 rows × 79 columns

In [ ]:

```
From looking at the head of both sets, we can see that the only difference in featur
es is "Sale Price". This makes sense because we are trying to predict it!
```

2. Analyzing the Test Variable (Sale Price)

In [7]:

```
#Let's check out the most interesting feature in this study: Sale Price. Important Not
e: This data is from Ames, Iowa. The location is extremely correlated with Sale Price.
 (I had to take a double-take at a point, since I consider myself a house-browsing enth
usiast)
# Getting Description
train['SalePrice'].describe()
```

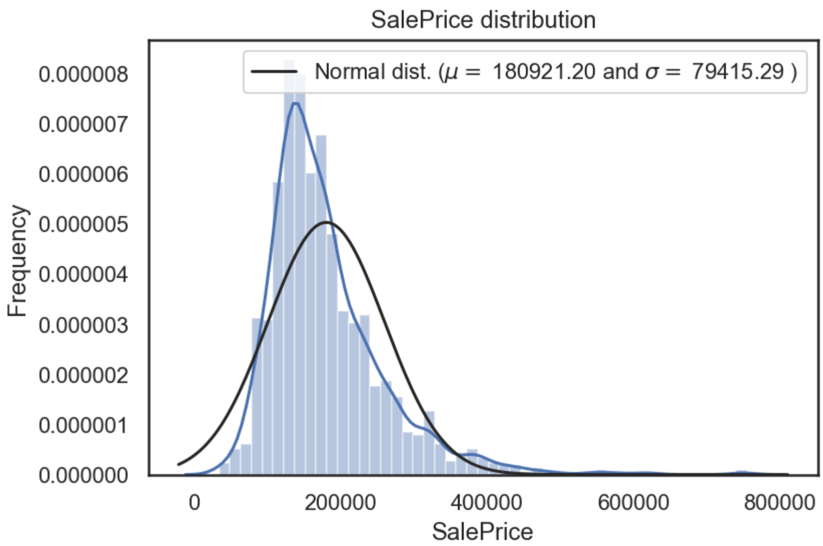
Out[7]:

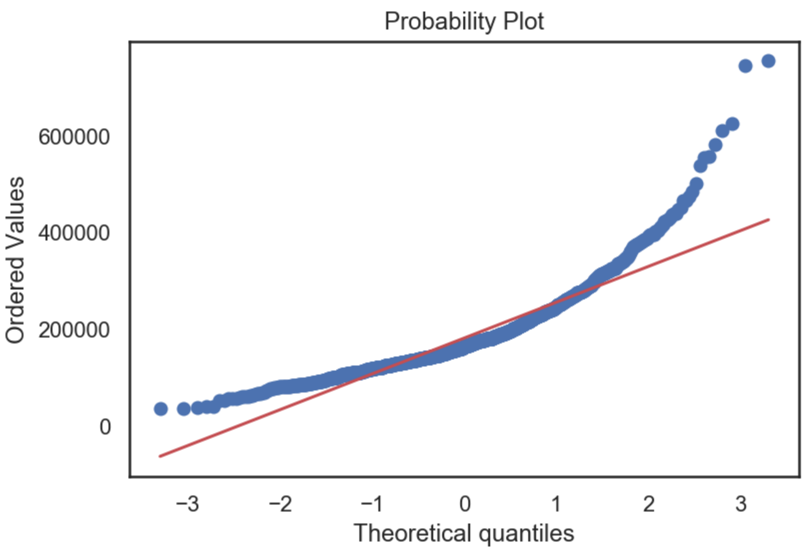
```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

In [8]:

```
# Plot Histogram
sns.distplot(train['SalePrice'] , fit=norm);

# Get the fitted parameters used by the function
(mu, sigma) = norm.fit(train['SalePrice'])
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)],
            loc='best')
plt.ylabel('Frequency')
plt.title('SalePrice distribution')

fig = plt.figure()
res = stats.probplot(train['SalePrice'], plot=plt)
plt.show()

print("Skewness: %f" % train['SalePrice'].skew())
print("Kurtosis: %f" % train['SalePrice'].kurt())
```

```
 mu = 180921.20 and sigma = 79415.29
```






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

3. Multivariable Analvsis

In [9]:

```
# Checking Categorical Data
train.select_dtypes(include=['object']).columns
```

Out[9]:

```
Index(['MSZoning', 'Street', 'Alley', 'LotShape', 'LandContour', 'Utilitie
s',
       'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition
2',
       'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st',
       'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond', 'Foundatio
n',
       'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinTyp
e2',
       'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual',
       'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQ
ual',
       'GarageCond', 'PavedDrive', 'PoolQC', 'Fence', 'MiscFeature',
       'SaleType', 'SaleCondition'],
      dtype='object')
```

In [10]:

```
# Checking Numerical Data
train.select_dtypes(include=['int64','float64']).columns
```

Out[10]:

```
Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCon
d',
       'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF
2',
       'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
       'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBat
h',
       'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces',
       'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorch
SF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',
       'MoSold', 'YrSold', 'SalePrice'],
      dtype='object')
```

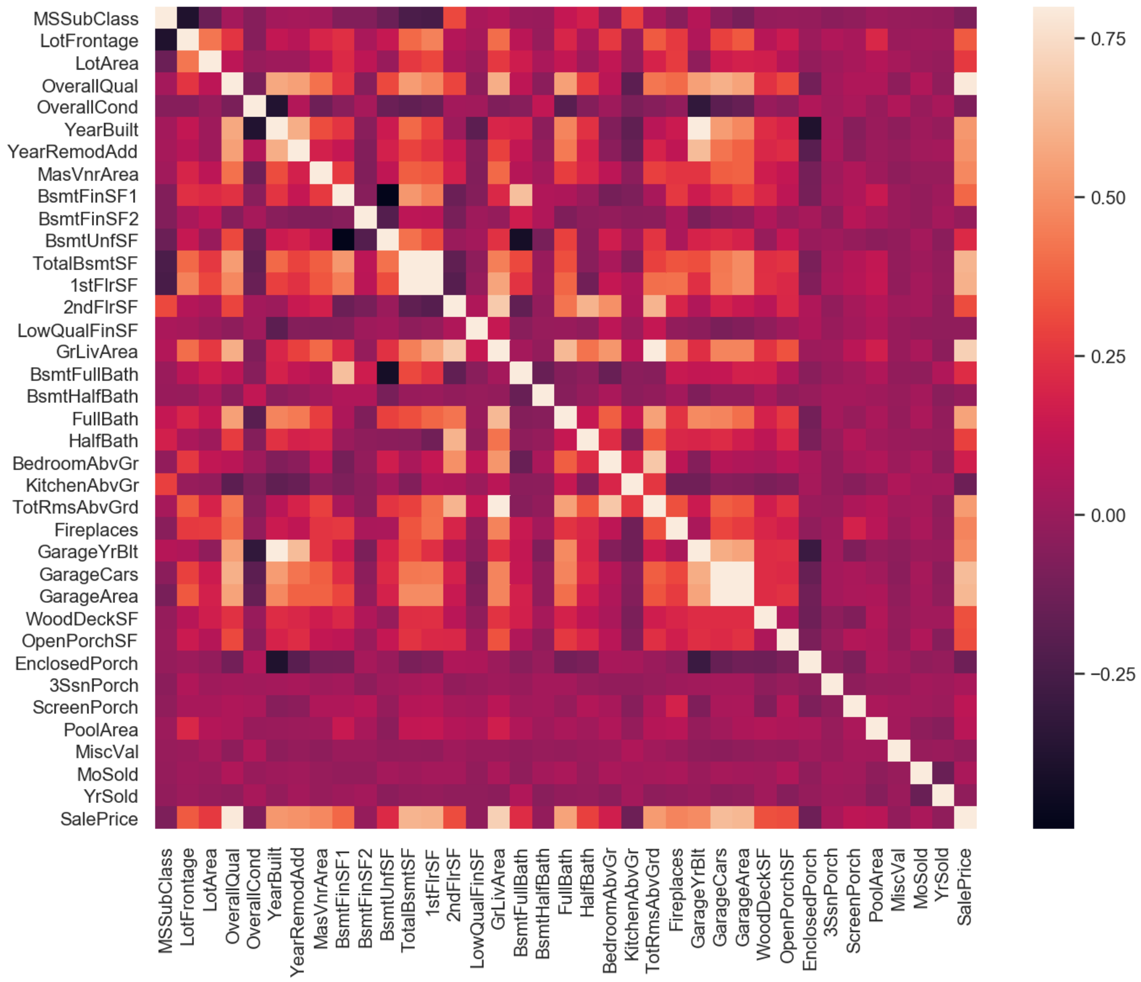
In [11]:

```
cat = len(train.select_dtypes(include=['object']).columns)
num = len(train.select_dtypes(include=['int64','float64']).columns)
print('Total Features: ', cat, 'categorical', '+',
      num, 'numerical', '=', cat+num, 'features')
```

```
Total Features:  43 categorical + 37 numerical = 80 features
```

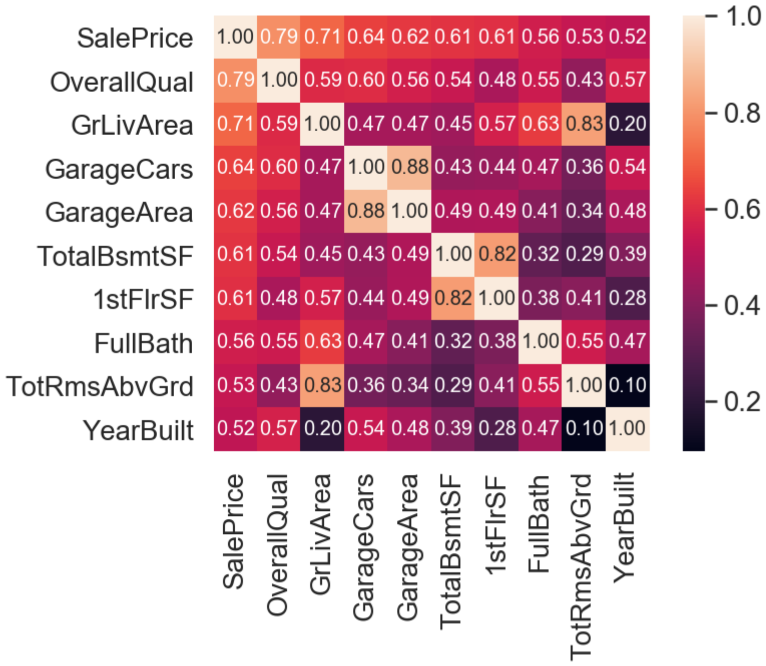
In [12]:

```
# Correlation Matrix Heatmap
corrmat = train.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True);
```



In [13]:

```
# Top 10 Heatmap
k = 10 #number of variables for heatmap
cols = corrmat.nlargest(k, 'SalePrice')['SalePrice'].index
cm = np.corrcoef(train[cols].values.T)
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size':
10}, yticklabels=cols.values, xticklabels=cols.values)
plt.show()
```



In [14]:

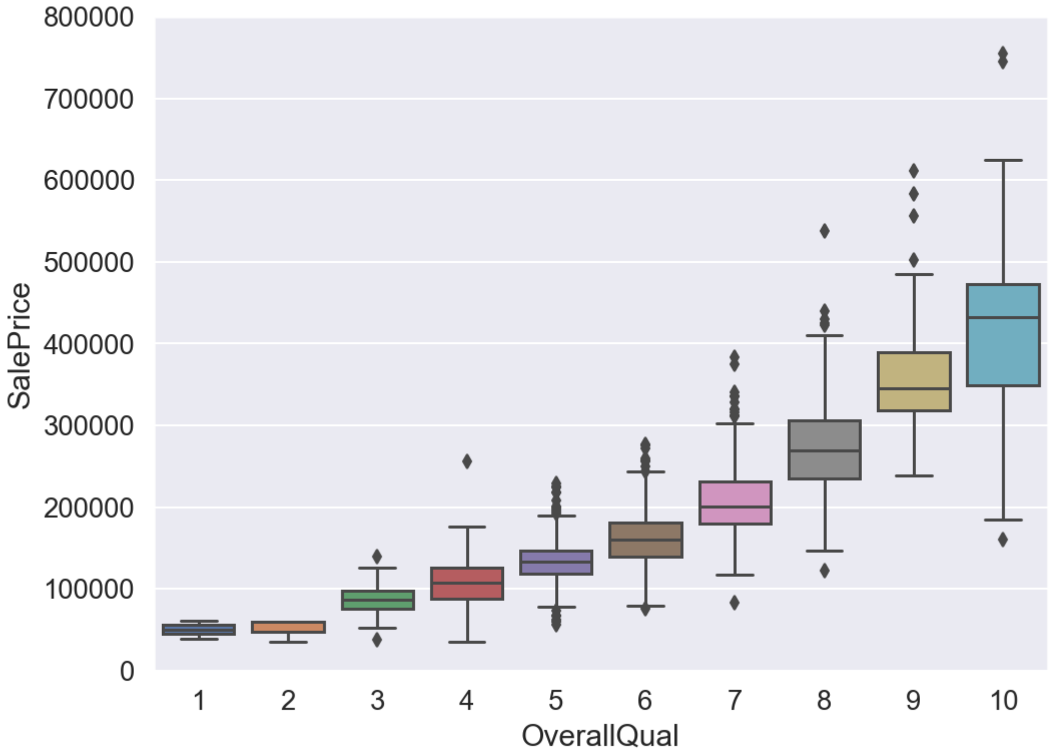
```
most_corr = pd.DataFrame(cols)
most_corr.columns = ['Most Correlated Features']
most_corr
```

Out[14]:

	Most Correlated Features
0	SalePrice
1	OverallQual
2	GrLivArea
3	GarageCars
4	GarageArea
5	TotalBsmtSF
6	1stFlrSF
7	FullBath
8	TotRmsAbvGrd
9	YearBuilt

In [15]:

```
# Overall Quality vs Sale Price
var = 'OverallQual'
data = pd.concat([train['SalePrice'], 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);
```

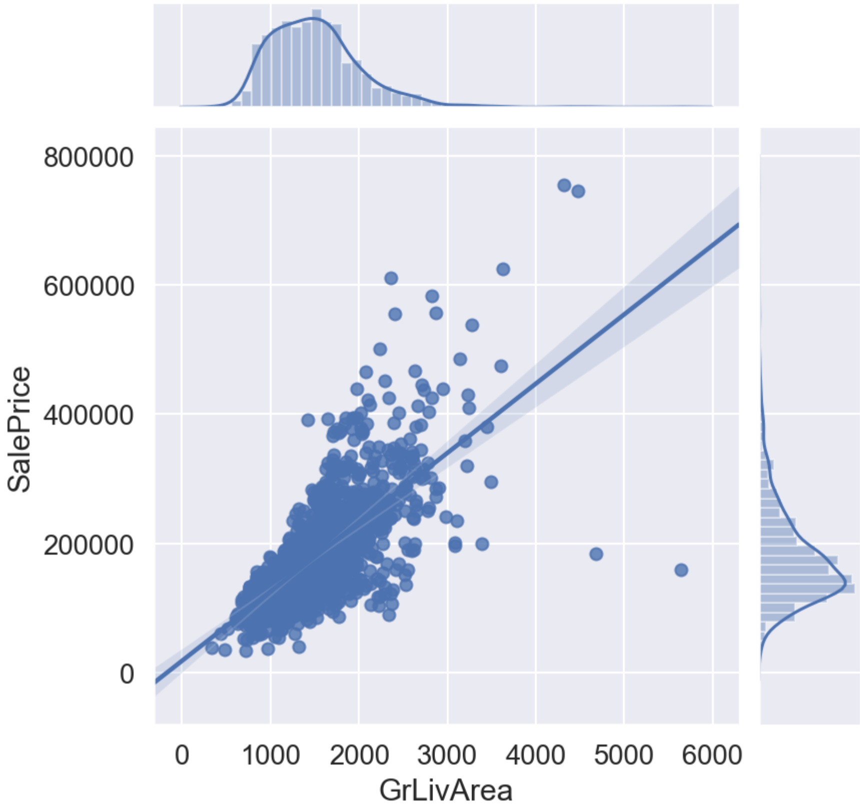


In [16]:

```
# Living Area vs Sale Price
sns.jointplot(x=train['GrLivArea'], y=train['SalePrice'], kind='reg')
```

Out[16]:

```
<seaborn.axisgrid.JointGrid at 0x2d31c1f1ef0>
```



In [17]:

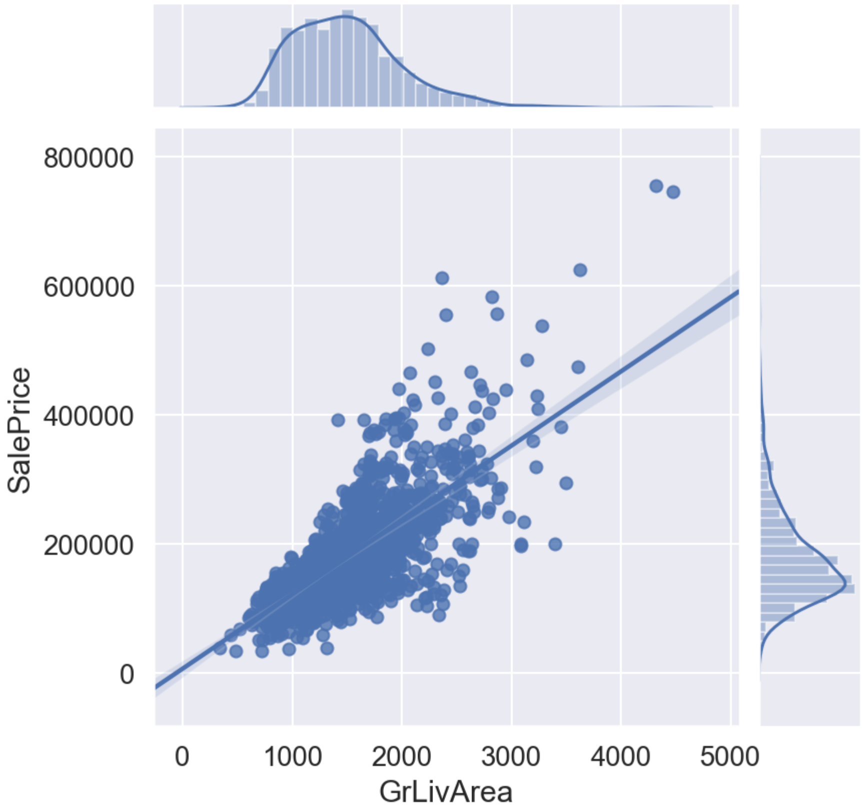
```
# Removing outliers manually (Two points in the bottom right)
train = train.drop(train[(train['GrLivArea']>4000)
                         & (train['SalePrice']<300000)].index).reset_index(drop=True)
```

In [18]:

```
# Living Area vs Sale Price
sns.jointplot(x=train['GrLivArea'], y=train['SalePrice'], kind='reg')
```

Out[18]:

```
<seaborn.axisgrid.JointGrid at 0x2d31c312860>
```

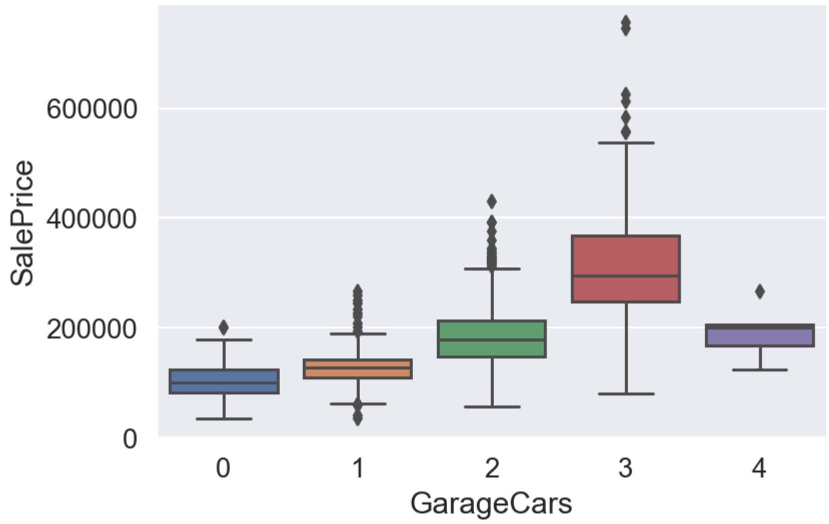


In [19]:

```
# Garage Area vs Sale Price
sns.boxplot(x=train['GarageCars'], y=train['SalePrice'])
```

Out[19]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2d31cdfebe0>
```



In [20]:

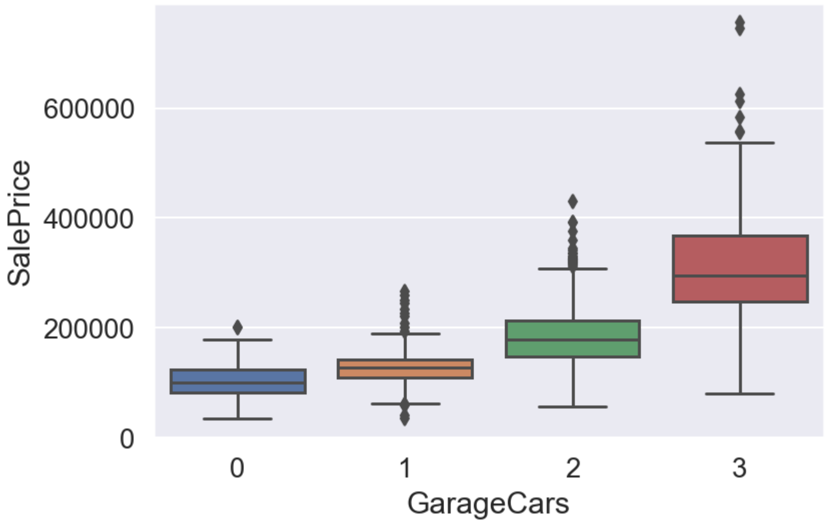
```
# Removing outliers manually (More than 4-cars, less than $300k)
train = train.drop(train[(train['GarageCars']>3)
                         & (train['SalePrice']<300000)].index).reset_index(drop=True)
```

In [21]:

```
# Garage Area vs Sale Price
sns.boxplot(x=train['GarageCars'], y=train['SalePrice'])
```

Out[21]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2d31b5fbcc0>
```

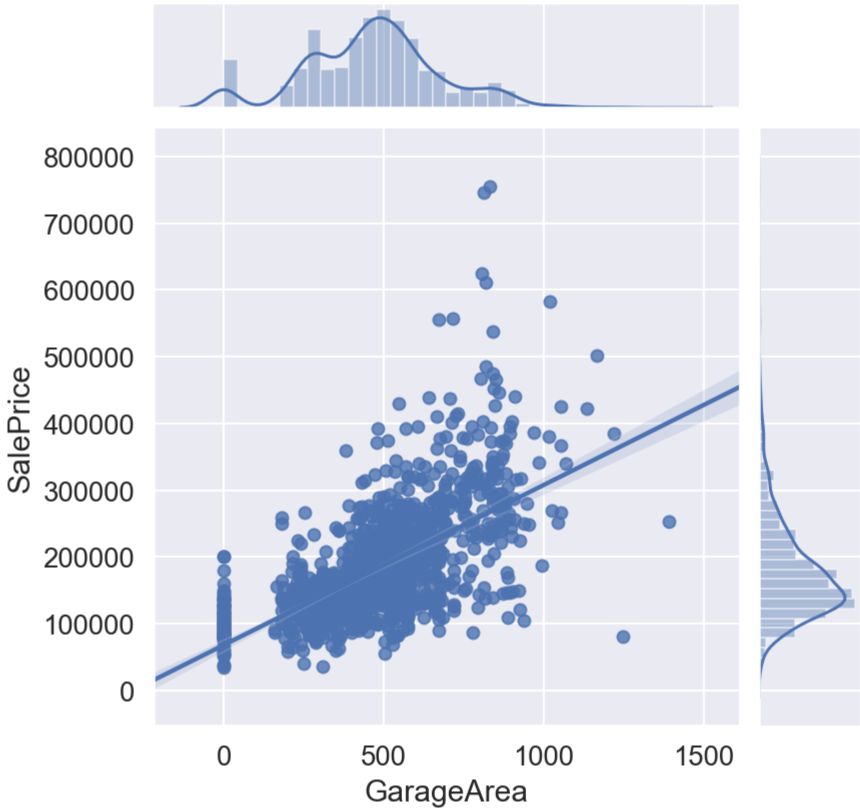


In [22]:

```
# Garage Area vs Sale Price
sns.jointplot(x=train['GarageArea'], y=train['SalePrice'], kind='reg')
```

Out[22]:

```
<seaborn.axisgrid.JointGrid at 0x2d31bdd9668>
```



Again with the bottom two data-points. Let's remove those outliers.

In [23]:

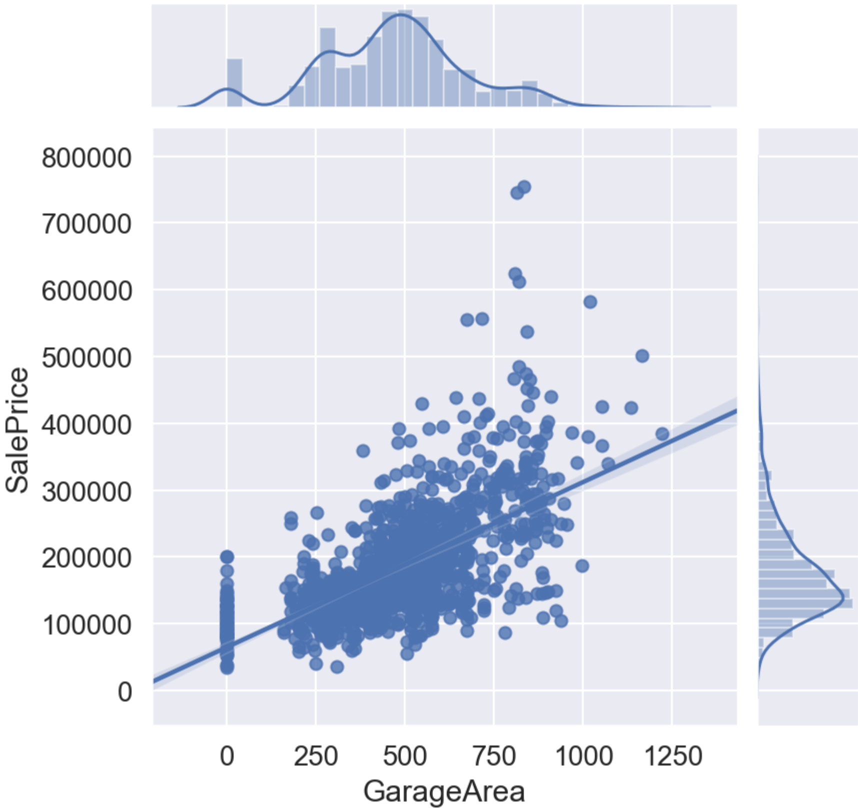
```
# Removing outliers manually (More than 1000 sqft, less than $300k)
train = train.drop(train[(train['GarageArea']>1000)
                         & (train['SalePrice']<300000)].index).reset_index(drop=True)
```

In [24]:

```
# Garage Area vs Sale Price
sns.jointplot(x=train['GarageArea'], y=train['SalePrice'], kind='reg')
```

Out[24]:

```
<seaborn.axisgrid.JointGrid at 0x2d31d195eb8>
```



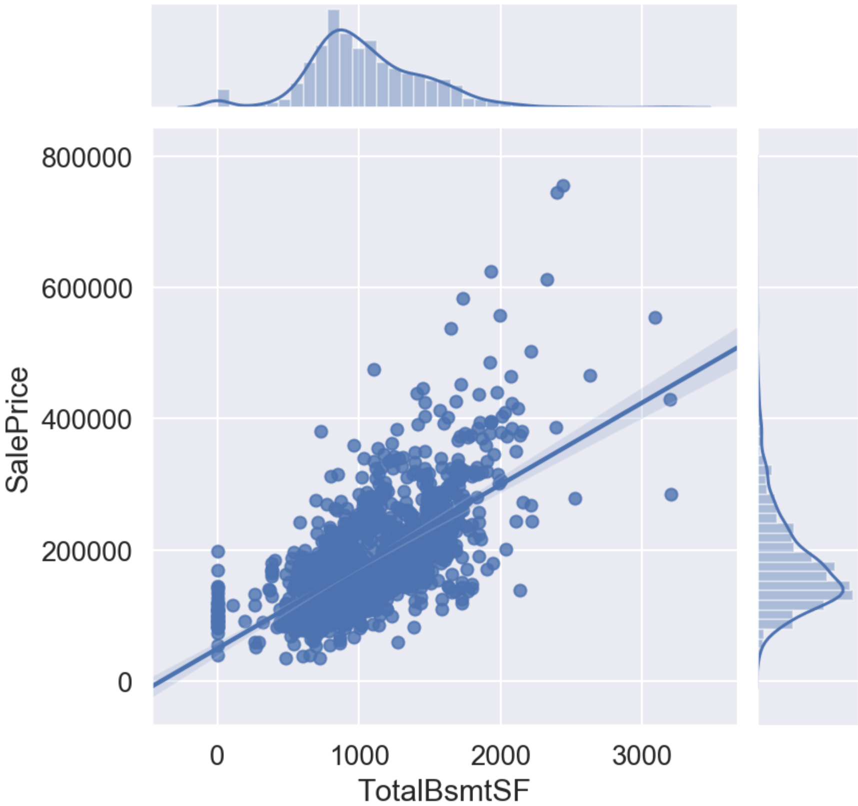
Only 0.01 point Pearson-R Score increase, but looks much better!

In [25]:

```
# Basement Area vs Sale Price
sns.jointplot(x=train['TotalBsmtSF'], y=train['SalePrice'], kind='reg')
```

Out[25]:

```
<seaborn.axisgrid.JointGrid at 0x2d31d5fb860>
```

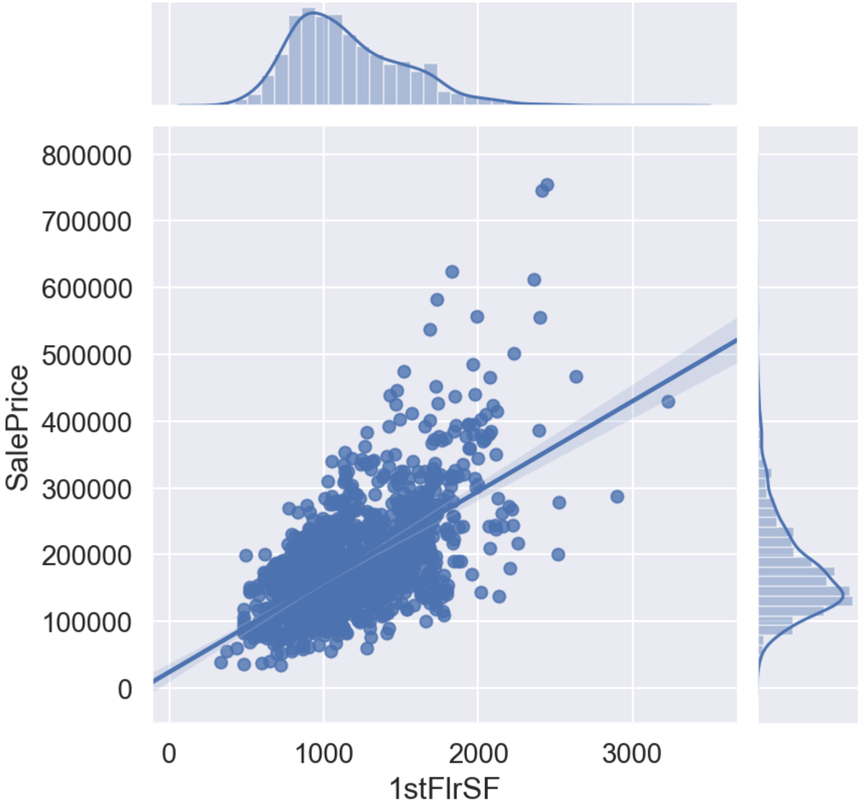


In [26]:

```
# First Floor Area vs Sale Price
sns.jointplot(x=train['1stFlrSF'], y=train['SalePrice'], kind='reg')
```

Out[26]:

```
<seaborn.axisgrid.JointGrid at 0x2d31d748c50>
```

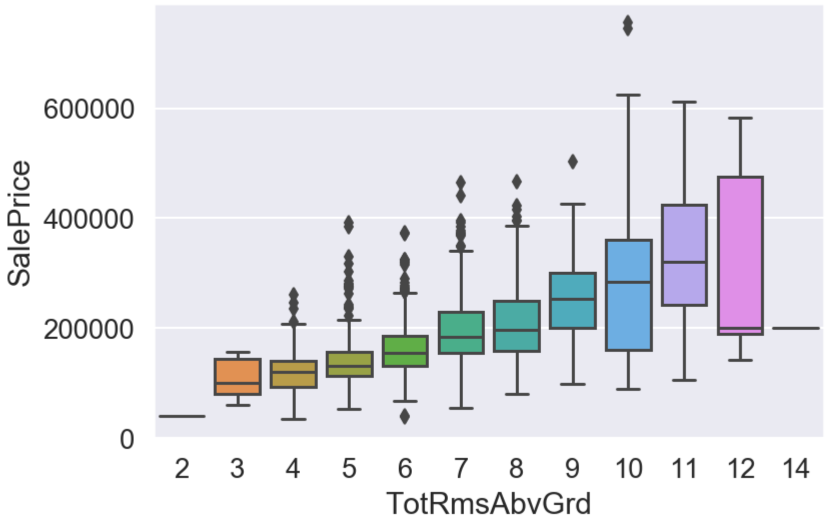


In [27]:

```
# Total Rooms vs Sale Price
sns.boxplot(x=train['TotRmsAbvGrd'], y=train['SalePrice'])
```

Out[27]:

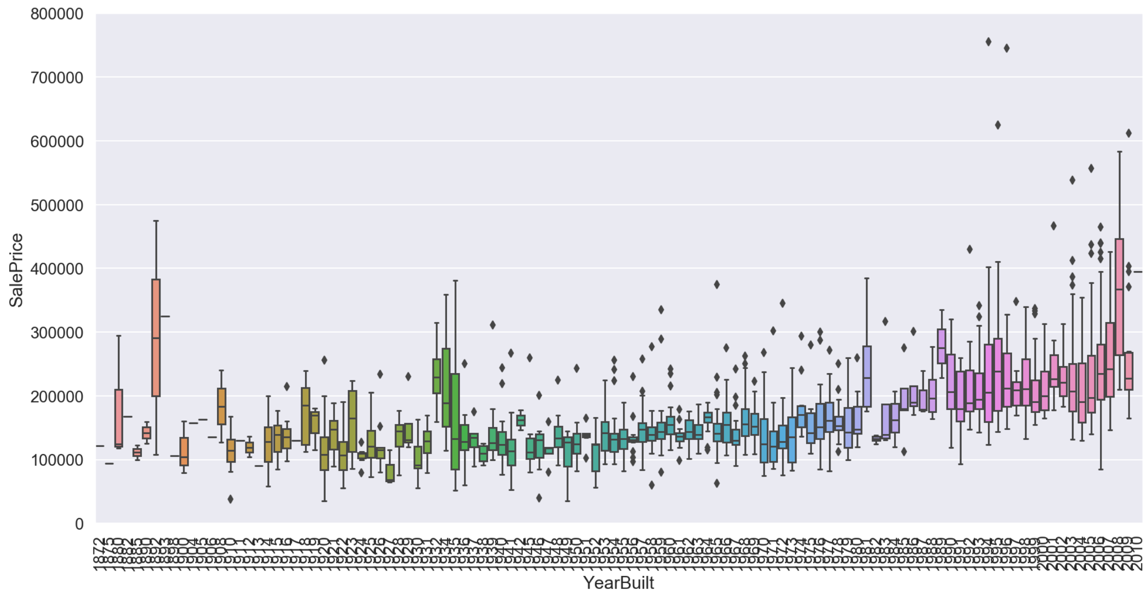
```
<matplotlib.axes._subplots.AxesSubplot at 0x2d31f5220b8>
```



It seems like houses with more than 11 rooms come with a $100k off coupon. It looks like an outlier but I'll let it slide.

In [28]:

```
# Total Rooms vs Sale Price
var = 'YearBuilt'
data = pd.concat([train['SalePrice'], 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);
```



Although it seems like house prices decrease with age, we can't be entirely sure. Is it because of inflation or stock market crashes,Let's leave the years alone.

4. Impute Missing Data and Clean Data

In [29]:

```
# Combining Datasets
ntrain = train.shape[0]
ntest = test.shape[0]
y_train = train.SalePrice.values
all_data = pd.concat((train, test)).reset_index(drop=True)
all_data.drop(['SalePrice'], axis=1, inplace=True)
print("Train data size is : {}".format(train.shape))
print("Test data size is : {}".format(test.shape))
print("Combined dataset size is : {}".format(all_data.shape))
```

```
Train data size is : (1448, 80)
Test data size is : (1459, 79)
Combined dataset size is : (2907, 79)
```

In [30]:

```
# Find Missing Ratio of Dataset
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(ascendi
ng=False)[:30]
missing_data = pd.DataFrame({'Missing Ratio' :all_data_na})
missing_data
```

Out[30]:

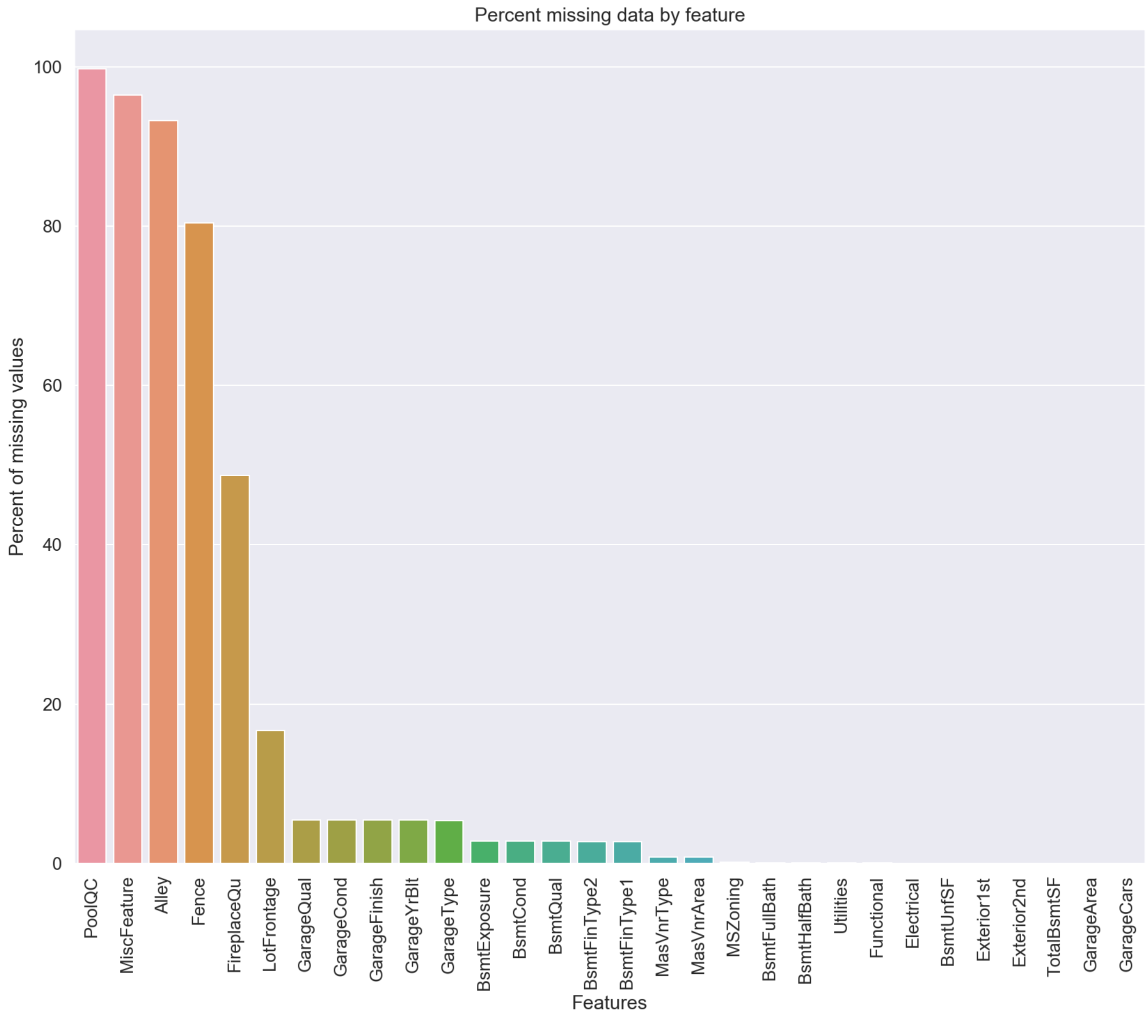
	Missing Ratio
PoolQC	99.690402
MiscFeature	96.422429
Alley	93.223254
Fence	80.392157
FireplaceQu	48.710010
LotFrontage	16.683867
GarageQual	5.469556
GarageCond	5.469556
GarageFinish	5.469556
GarageYrBlt	5.469556
GarageType	5.400757
BsmtExposure	2.820777
BsmtCond	2.820777
BsmtQual	2.786378
BsmtFinType2	2.751978
BsmtFinType1	2.717578
MasVnrType	0.825593
MasVnrArea	0.791194
MSZoning	0.137599
BsmtFullBath	0.068799
BsmtHalfBath	0.068799
Utilities	0.068799
Functional	0.068799
Electrical	0.034400
BsmtUnfSF	0.034400
Exterior1st	0.034400
Exterior2nd	0.034400
TotalBsmtSF	0.034400
GarageArea	0.034400
GarageCars	0.034400

In [31]:

```
# Percent missing data by feature
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[31]:

```
Text(0.5, 1.0, 'Percent missing data by feature')
```



In [32]:

```
#Imputing Missing Values
all_data["PoolQC"] = all_data["PoolQC"].fillna("None")
all_data["MiscFeature"] = all_data["MiscFeature"].fillna("None")
all_data["Alley"] = all_data["Alley"].fillna("None")
all_data["Fence"] = all_data["Fence"].fillna("None")
all_data["FireplaceQu"] = all_data["FireplaceQu"].fillna("None")
all_data["LotFrontage"] = all_data.groupby("Neighborhood")["LotFrontage"].transform(lam
bda x: x.fillna(x.median()))
for col in ('GarageType', 'GarageFinish', 'GarageQual', 'GarageCond'):
    all_data[col] = all_data[col].fillna('None')
for col in ('GarageYrBlt', 'GarageArea', 'GarageCars'):
    all_data[col] = all_data[col].fillna(0)
for col in ('BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF','TotalBsmtSF', 'BsmtFullBath', 'Bsm
tHalfBath'):
    all_data[col] = all_data[col].fillna(0)
for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2'):
    all_data[col] = all_data[col].fillna('None')
all_data["MasVnrType"] = all_data["MasVnrType"].fillna("None")
all_data["MasVnrArea"] = all_data["MasVnrArea"].fillna(0)
all_data['MSZoning'] = all_data['MSZoning'].fillna(all_data['MSZoning'].mode()[0])
all_data = all_data.drop(['Utilities'], axis=1)
all_data["Functional"] = all_data["Functional"].fillna("Typ")
all_data['Electrical'] = all_data['Electrical'].fillna(all_data['Electrical'].mode()[0
])
all_data['KitchenQual'] = all_data['KitchenQual'].fillna(all_data['KitchenQual'].mode()
[0])
all_data['Exterior1st'] = all_data['Exterior1st'].fillna(all_data['Exterior1st'].mode()
[0])
all_data['Exterior2nd'] = all_data['Exterior2nd'].fillna(all_data['Exterior2nd'].mode()
[0])
all_data['SaleType'] = all_data['SaleType'].fillna(all_data['SaleType'].mode()[0])
all_data['MSSubClass'] = all_data['MSSubClass'].fillna("None")
```

In [33]:

```
# Check if there are any missing values left
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(ascendi
ng=False)
missing_data = pd.DataFrame({'Missing Ratio' :all_data_na})
missing_data.head()
```

Out[33]:

Missing Ratio

5. Feature Transformation/Engineering

In [34]:

```
all_data['MSSubClass'].describe()
```

Out[34]:

```
count    2907.000000
mean       57.094943
std        42.510238
min        20.000000
25%        20.000000
50%        50.000000
75%        70.000000
max       190.000000
Name: MSSubClass, dtype: float64
```

In [35]:

```
#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)
```

In [36]:

```
all_data['KitchenQual'].unique()
```

Out[36]:

```
array(['Gd', 'TA', 'Ex', 'Fa'], dtype=object)
```

Here, data_description.txt comes to the rescue again!

Kitchen Quality:

Ex: Excellent Gd: Good TA: Typical/Average Fa: Fair Po: Poor Is a score of "Gd" better than "TA" but worse than "Ex"? I think so, let's encode these labels to give meaning to their specific orders.

In [37]:

```
from sklearn.preprocessing import LabelEncoder
cols = ('FireplaceQu', 'BsmtQual', 'BsmtCond', 'GarageQual', 'GarageCond',
        'ExterQual', 'ExterCond','HeatingQC', 'PoolQC', 'KitchenQual', 'BsmtFinType1',
        'BsmtFinType2', 'Functional', 'Fence', 'BsmtExposure', 'GarageFinish', 'LandSlo
pe',
        'LotShape', 'PavedDrive', 'Street', 'Alley', 'CentralAir', 'MSSubClass', 'Overa
llCond',
        'YrSold', 'MoSold')
# Process columns and 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))

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

```
Shape all_data: (2907, 78)
```

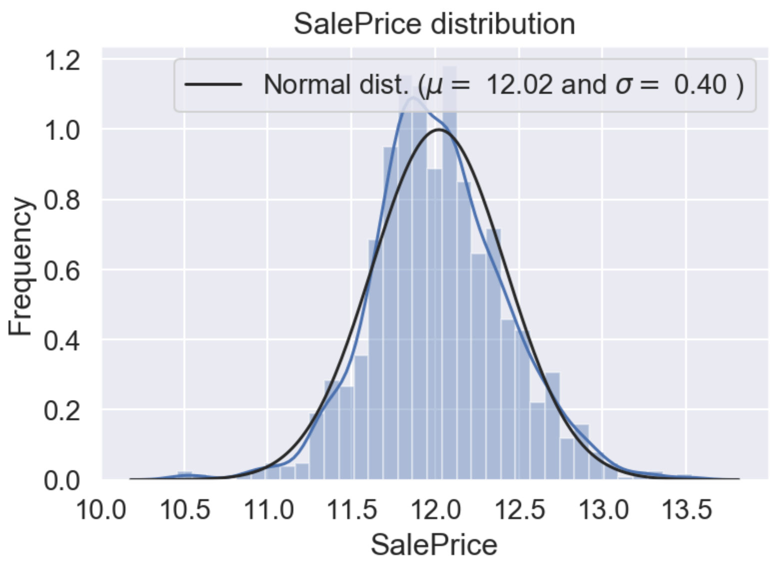
In [38]:

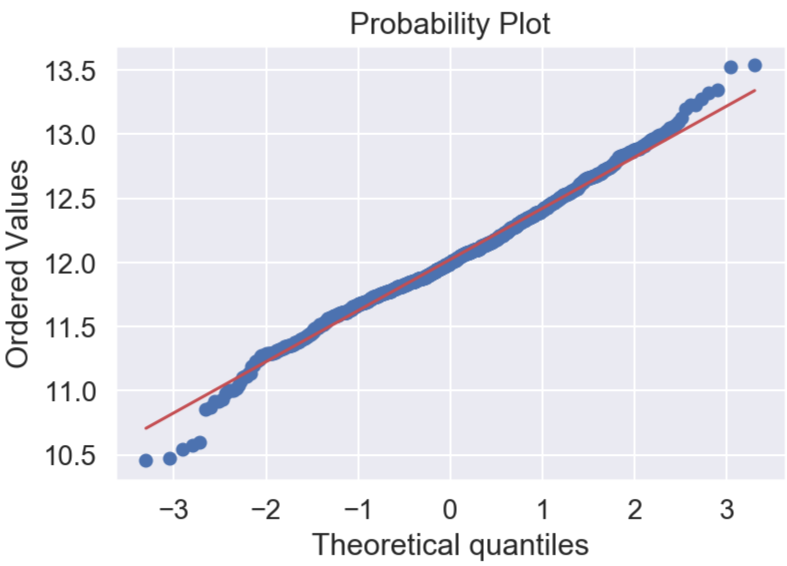
```
#Let's engineer one feature to combine square footage, this may be useful later on.
# Adding Total Square Feet feature
all_data['TotalSF'] = all_data['TotalBsmtSF'] + all_data['1stFlrSF'] + all_data['2ndFlr
SF']
```

In [39]:

```
#Fixing "skewed" features.
#Here, we fix all of the skewed data to be more normal so that our models will be more
 accurate when making predictions.
# We use the numpy fuction log1p which  applies log(1+x) to all elements of the column
train["SalePrice"] = np.log1p(train["SalePrice"])

#Check the new distribution
sns.distplot(train['SalePrice'] , fit=norm);

# Get the fitted parameters used by the function
(mu, sigma) = norm.fit(train['SalePrice'])
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)],
            loc='best')
plt.ylabel('Frequency')
plt.title('SalePrice distribution')

fig = plt.figure()
res = stats.probplot(train['SalePrice'], plot=plt)
plt.show()
y_train = train.SalePrice.values

print("Skewness: %f" % train['SalePrice'].skew())
print("Kurtosis: %f" % train['SalePrice'].kurt())
```

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






```
Skewness: 0.130172
Kurtosis: 0.822862
```

In [40]:

```
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_values(as
cending=False)
skewness = pd.DataFrame({'Skewed Features' :skewed_feats})
skewness.head()
```

Out[40]:

	Skewed Features
MiscVal	21.911765
PoolArea	17.658029
LotArea	13.147728
LowQualFinSF	12.063406
3SsnPorch	11.352135

In [41]:

```
skewness = skewness[abs(skewness) > 0.75]
print("There are {} skewed numerical features to Box Cox transform".format(skewness.sha
pe[0]))

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

```
There are 59 skewed numerical features to Box Cox transform
```

In [42]:

```
all_data = pd.get_dummies(all_data)
print(all_data.shape)
```

```
(2907, 220)
```

In [43]:

```
train = all_data[:ntrain]
test = all_data[ntrain:]
```

6. Modeling and Predictions

In [44]:

```
from sklearn.linear_model import ElasticNet, Lasso,  BayesianRidge, LassoLarsIC
from sklearn.ensemble import RandomForestRegressor,  GradientBoostingRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import RobustScaler
from sklearn.base import BaseEstimator, TransformerMixin, RegressorMixin, clone
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.metrics import mean_squared_error
import xgboost as xgb
import lightgbm as lgb
```

In [45]:

```
# Cross-validation with k-folds
n_folds = 5

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

In [46]:

```
lasso = make_pipeline(RobustScaler(), Lasso(alpha =0.0005, random_state=1))
ENet = make_pipeline(RobustScaler(), ElasticNet(alpha=0.0005, l1_ratio=.9, random_state
=3))
KRR = KernelRidge(alpha=0.6, kernel='polynomial', degree=2, coef0=2.5)
GBoost = GradientBoostingRegressor(n_estimators=3000, learning_rate=0.05,
                                   max_depth=4, max_features='sqrt',
                                   min_samples_leaf=15, min_samples_split=10,
                                   loss='huber', random_state =5)
model_xgb = xgb.XGBRegressor(colsample_bytree=0.2, gamma=0.0,
                             learning_rate=0.05, max_depth=6,
                             min_child_weight=1.5, n_estimators=7200,
                             reg_alpha=0.9, reg_lambda=0.6,
                             subsample=0.2,seed=42, silent=1,
                             random_state =7)
model_lgb = lgb.LGBMRegressor(objective='regression',num_leaves=5,
                              learning_rate=0.05, n_estimators=720,
                              max_bin = 55, bagging_fraction = 0.8,
                              bagging_freq = 5, feature_fraction = 0.2319,
                              feature_fraction_seed=9, bagging_seed=9,
                              min_data_in_leaf =6, min_sum_hessian_in_leaf = 11)
```

In [47]:

```
#Checking performance of base models by evaluating the cross-validation RMSLE error.
score = rmsle_cv(lasso)
print("\nLasso score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
score = rmsle_cv(ENet)
print("ElasticNet score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
score = rmsle_cv(KRR)
print("Kernel Ridge 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()))
score = rmsle_cv(model_xgb)
print("Xgboost score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
score = rmsle_cv(model_lgb)
print("LGBM score: {:.4f} ({:.4f})\n" .format(score.mean(), score.std()))
```

```
Lasso score: 0.1111 (0.0071)

ElasticNet score: 0.1111 (0.0072)

Kernel Ridge score: 0.1148 (0.0075)

Gradient Boosting score: 0.1173 (0.0079)

Xgboost score: 0.1180 (0.0067)

LGBM score: 0.1149 (0.0069)
```

In [48]:

```
#Here, we stack the models to average their scores.
class AveragingModels(BaseEstimator, RegressorMixin, TransformerMixin):
    def __init__(self, models):
        self.models = models

    # we define clones of the original models to fit the data in
    def fit(self, X, y):
        self.models_ = [clone(x) for x in self.models]

        # Train cloned base models
        for model in self.models_:
            model.fit(X, y)

        return self

    #Now we do the predictions for cloned models and average them
    def predict(self, X):
        predictions = np.column_stack([
            model.predict(X) for model in self.models_
        ])
        return np.mean(predictions, axis=1)
```

In [49]:

```
#Here we average ENet, GBoost, KRR, and lasso. We'll add in XGBoost and LightGBM later
averaged_models = AveragingModels(models = (ENet, GBoost, KRR, lasso))

score = rmsle_cv(averaged_models)
print("Averaged base models score: {:.4f} ({:.4f})\n".format(score.mean(), score.std
()))
```

```
Averaged base models score: 0.1085 (0.0073)
```

In [50]:

```
class StackingAveragedModels(BaseEstimator, RegressorMixin, TransformerMixin):
    def __init__(self, base_models, meta_model, n_folds=5):
        self.base_models = base_models
        self.meta_model = meta_model
        self.n_folds = n_folds

    # We again fit the data on clones of the original models
    def fit(self, X, y):
        self.base_models_ = [list() for x in self.base_models]
        self.meta_model_ = clone(self.meta_model)
        kfold = KFold(n_splits=self.n_folds, shuffle=True)

        # Train cloned base models then create out-of-fold predictions
        # that are needed to train the cloned meta-model
        out_of_fold_predictions = np.zeros((X.shape[0], len(self.base_models)))
        for i, clf in enumerate(self.base_models):
            for train_index, holdout_index in kfold.split(X, y):
                instance = clone(clf)
                self.base_models_[i].append(instance)
                instance.fit(X[train_index], y[train_index])
                y_pred = instance.predict(X[holdout_index])
                out_of_fold_predictions[holdout_index, i] = y_pred

        # Now train the cloned  meta-model using the out-of-fold predictions
        self.meta_model_.fit(out_of_fold_predictions, y)
        return self

    def predict(self, X):
        meta_features = np.column_stack([
            np.column_stack([model.predict(X) for model in base_models]).mean(axis=1)
            for base_models in self.base_models_ ])
        return self.meta_model_.predict(meta_features)
```

In [51]:

```
stacked_averaged_models = StackingAveragedModels(base_models = (ENet, GBoost, KRR),
                                                 meta_model = lasso)

score = rmsle_cv(stacked_averaged_models)
print("Stacking Averaged models score: {:.4f} ({:.4f})".format(score.mean(), score.std
()))
```

```
Stacking Averaged models score: 0.1077 (0.0070)
```

In [52]:

```
def rmsle(y, y_pred):
    return np.sqrt(mean_squared_error(y, y_pred))
```

In [53]:

```
#Stacked models
stacked_averaged_models.fit(train.values, y_train)
stacked_train_pred = stacked_averaged_models.predict(train.values)
stacked_pred = np.expm1(stacked_averaged_models.predict(test.values))
print(rmsle(y_train, stacked_train_pred))
```

```
0.07728267431091478
```

In [54]:

```
#XGBOOST
model_xgb.fit(train, y_train)
xgb_train_pred = model_xgb.predict(train)
xgb_pred = np.expm1(model_xgb.predict(test))
print(rmsle(y_train, xgb_train_pred))
```

```
0.04199482385381862
```

In [55]:

```
#LightGBM
model_lgb.fit(train, y_train)
lgb_train_pred = model_lgb.predict(train)
lgb_pred = np.expm1(model_lgb.predict(test.values))
print(rmsle(y_train, lgb_train_pred))
```

```
0.07127683177891941
```

In [56]:

```
'''RMSE on the entire Train data when averaging'''

print('RMSLE score on train data:')
print(rmsle(y_train,stacked_train_pred*0.70 +
               xgb_train_pred*0.10 + lgb_train_pred*0.20 ))
```

```
RMSLE score on train data:
0.07069658201525272
```

Ensemble Prediction Note: To get our weights for each model, we'll take the inverse of each regressor and average it out of 100%

In [57]:

```
# Example
Stacked = 1/(0.1077)
XGBoost = 1/(0.1177)
LGBM = 1/(0.1159)
Sum = Stacked + XGBoost + LGBM
Stacked = Stacked/Sum
XGBoost = XGBoost/Sum
LGBM = LGBM/Sum
print(Stacked, XGBoost, LGBM)
```

```
0.35158188821434966 0.32171086967447293 0.3267072421111774
```

In [58]:

```
'''RMSE on the entire Train data when averaging'''

print('RMSLE score on train data:')
print(rmsle(y_train,stacked_train_pred*Stacked +
               xgb_train_pred*XGBoost + lgb_train_pred*LGBM))
```

```
RMSLE score on train data:
0.061173602032788627
```

In [59]:

```
ensemble = stacked_pred*Stacked + xgb_pred*XGBoost + lgb_pred*LGBM
```

In [60]:

```
#SUBMISSION
sub = pd.DataFrame()
sub['Id'] = test_ID
sub['SalePrice'] = ensemble
sub.to_csv('submission.csv',index=False)
```