Housing Project.

In [1]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import skew,norm
from scipy.stats.stats import pearsonr
```

In [2]:
```python
train = pd.read_csv("train.csv")
test = pd.read_csv("Test.csv")
```

In [3]:
```python
train.head()
```

Out[3]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape
0	127	120	RL	NaN	4928	Pave	NaN	IR1
1	889	20	RL	95.0	15865	Pave	NaN	IR1
2	793	60	RL	92.0	9920	Pave	NaN	IR1
3	110	20	RL	105.0	11751	Pave	NaN	IR1
4	422	20	RL	NaN	16635	Pave	NaN	IR1

5 rows × 81 columns

In [4]:
```python
test.head()
```

Out[4]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape
0	337	20	RL	86.0	14157	Pave	NaN	IR1
1	1018	120	RL	NaN	5814	Pave	NaN	IR1
2	929	20	RL	NaN	11838	Pave	NaN	Reg
3	1148	70	RL	75.0	12000	Pave	NaN	Reg
4	1227	60	RL	86.0	14598	Pave	NaN	IR1

5 rows × 80 columns

In [5]:
```python
print ("Train data: \n")
print ("Number of columns: " + str (train.shape[1]))
print ("number of rows: " + str (train.shape[0]))

print('\nTest data: \n')
print ("number of columns:" + str (test.shape[1]))
print ("Number of columns:" +  str (test.shape[0]))
```

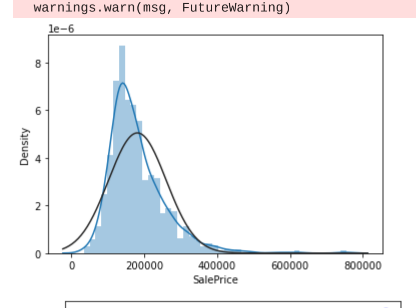
```
Train data:

Number of columns: 81
number of rows: 1168

Test data:

number of columns:80
Number of columns:292
```

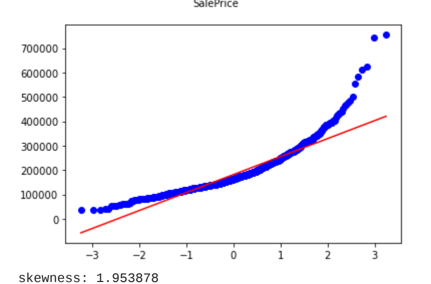
In [6]:
```python
#descriptive statistics summary
train['SalePrice'].describe()
```

Out[6]:
```
count      1168.000000
mean     181477.005993
std       79105.586863
min       34900.000000
25%      130375.000000
50%      163995.000000
75%      215000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

In [7]:
```python
# kernel density plot
sns.distplot(train.SalePrice,fit=norm);
plt.ylabel =('Frequency')
plt.title = ('SalePrice Distribution');
#Get the fitted parameters used by the function
(mu, sigma) = norm.fit(train['SalePrice']);
#QQ plot
fig = plt.figure()
res = stats.probplot(train['SalePrice'], plot=plt)
plt.show()
print("skewness: %f" % train['SalePrice'].skew())
print("kurtosis: %f" % train ['SalePrice'].kurt())
```

```
C:\Users\kisho\anaconda3\lib\site-packages\seaborn\distribution
s.py:2619: FutureWarning: `distplot` is a deprecated function a
nd will be removed in a future version. Please adapt your code
to use either `displot` (a figure-level function with similar f
lexibility) or `histplot` (an axes-level function for histogram
s).
  warnings.warn(msg, FutureWarning)
```

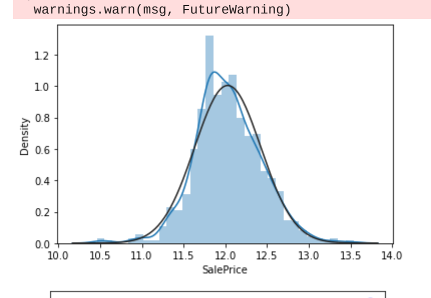


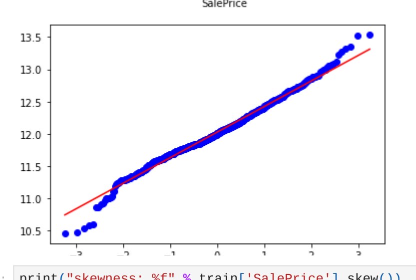


```
skewness: 1.953878
```

In [8]:
```python
#log transform the target
train["SalePrice"] = np.log1p(train["SalePrice"])

#Kernel Density plot
sns.distplot(train.SalePrice,fit=norm);
plt.ylabel=('Frequency')
plt.title=('SalePrice distribution');
#Get the fitted parameters used by the function
(mu,sigma)= norm.fit(train['SalePrice']);
#QQ plot
fig =plt.figure()
res =stats. probplot(train['SalePrice'], plot=plt)
plt.show()
```

```
C:\Users\kisho\anaconda3\lib\site-packages\seaborn\distribution
s.py:2619: FutureWarning: `distplot` is a deprecated function a
nd will be removed in a future version. Please adapt your code
to use either `displot` (a figure-level function with similar f
lexibility) or `histplot` (an axes-level function for histogram
s).
  warnings.warn(msg, FutureWarning)
```

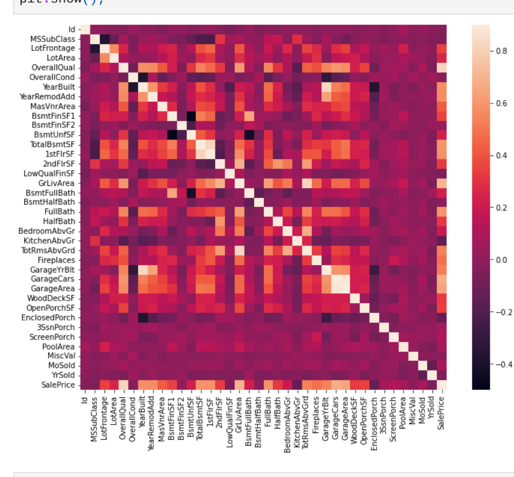




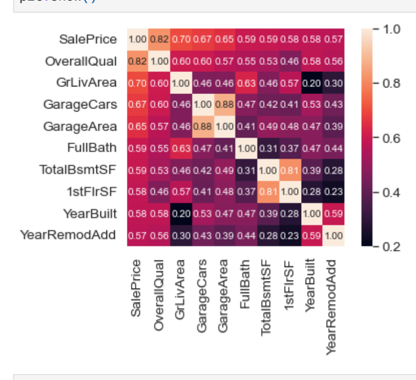
In [9]:
```python
print("skewness: %f" % train['SalePrice'].skew())
print("kurtosis: %f" % train ['SalePrice'].kurt())
```

```
skewness: 0.073610
kurtosis: 0.995996
```

In [10]:
```python
#correration matrix
corrmat = train.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat,vmax=0.9, square=True)
plt.show();
```

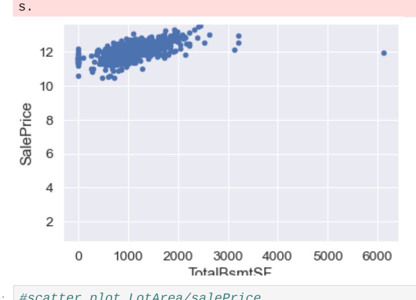


In [11]:
```python
cols = corrmat.nlargest(10, 'SalePrice')['SalePrice'].index
cm = np.corrcoef(train[cols].values.T)
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2
                 xticklabels=cols.values)
plt.yticks(rotation=0)
plt.xticks(rotation=90)
plt.show()
```



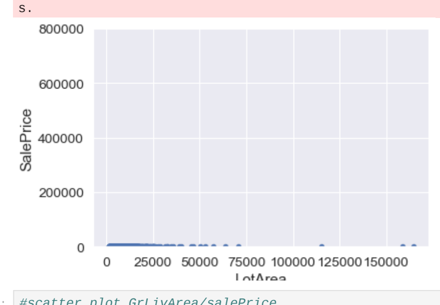
In [12]:
```python
var ='TotalBsmtSF'
data = pd.concat([train['SalePrice'],train[var]],axis=1)
data.plot.scatter(x=var, y='SalePrice', ylim =0.800000);
plt.show()
```

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



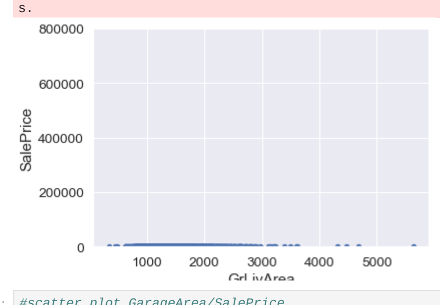
In [13]:
```python
#scatter plot LotArea/salePrice
var = 'LotArea'
data = pd.concat([train['SalePrice'], train[var]], axis=1)
data.plot.scatter(x= var, y='SalePrice', ylim =(0,800000));
plt.show();
```

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



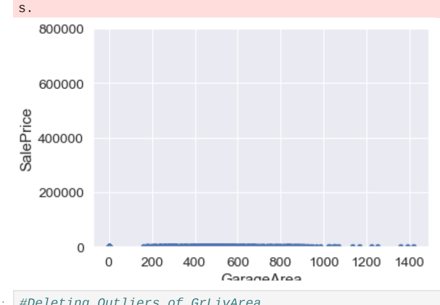
In [14]:
```python
#scatter plot GrLivArea/salePrice
var ='GrLivArea'
data =pd.concat([train['SalePrice'], train[var]], axis=1)
data.plot.scatter(x=var, y='SalePrice',ylim=(0,800000));
plt.show()
```

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



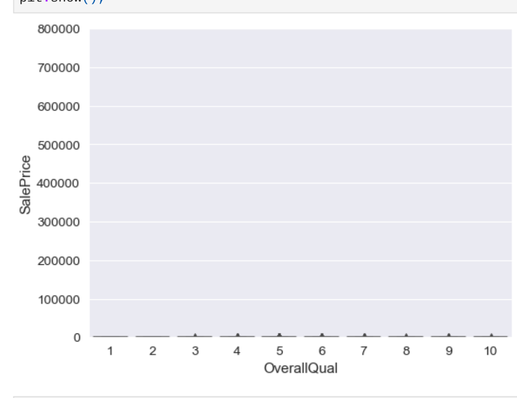
In [15]:
```python
#scatter plot GarageArea/SalePrice
var = 'GarageArea'
data =pd.concat([train['SalePrice'], train[var]], axis=1)
data.plot.scatter(x=var,y='SalePrice', ylim= (0,800000));
plt.show()
```

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

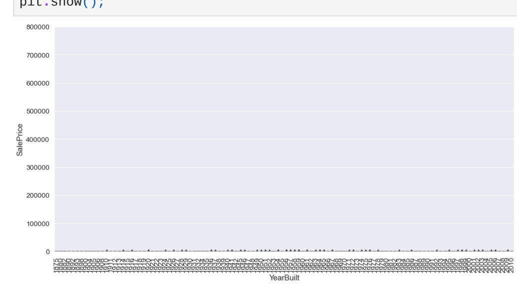


In [16]:
```python
#Deleting Outliers of GrLivArea
train = train.drop(train[(train['GrLivArea']>4000) & (train['Sal
```

In [17]:
```python
#box plot overallqual/salePrice
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)
plt.show();
```



In [18]:
```python
#year built
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);
plt.show();
```



Inputing Missing Values

In [19]:
```python
train.head()
```

Out[19]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape
0	127	120	RL	NaN	4928	Pave	NaN	IR1
1	889	20	RL	95.0	15865	Pave	NaN	IR1
2	793	60	RL	92.0	9920	Pave	NaN	IR1
3	110	20	RL	105.0	11751	Pave	NaN	IR1
4	422	20	RL	NaN	16635	Pave	NaN	IR1

5 rows × 81 columns

In [20]:
```python
all_data = pd.concat((train.loc[:, 'MSSubClass': 'SaleCondition'
                      test.loc[:,'MSSubClass':'SaleCondition']))
print("all_data size is: {} ".format(all_data.shape))
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].ind
# [:30]
missing_data =pd.DataFrame({'Missing Raio':all_data_na})
missing_data.head(20)
```

```
all_data size is: (1456, 79)
```

Out[20]:

	Missing Raio
PoolQC	99.656593
MiscFeature	96.291209
Alley	93.750000
Fence	80.769231
FireplaceQu	47.390110
LotFrontage	17.788462
GarageType	5.563187
GarageYrBlt	5.563187
GarageFinish	5.563187
GarageQual	5.563187
GarageCond	5.563187
BsmtExposure	2.609890
BsmtFinType2	2.609890
BsmtFinType1	2.541209
BsmtCond	2.541209
BsmtQual	2.541209
MasVnrArea	0.549451
MasVnrType	0.549451
Electrical	0.068681

inputing missing values

Based on feature description provide, A feature that has NA means it is absent

In [21]:
```python
for col in ('PoolQC','MiscFeature','GarageType','Alley','Fence',
            'GarageQual','GarageCond','MasVnrType','MSSubClass'):
    all_data[col] = all_data[col].fillna('None')
```

In [22]:
```python
#Replacing missing value with 0(since no garage = no cars in suc
for col in ('GarageYrBlt','GarageArea', 'GarageCars'):
    all_data[col] = all_data[col].fillna(0)
```

In [23]:
```python
#missing values are likely zero for no basement
for col in ('BsmtFinSF1','BsmtFinSF2', 'BsmtUnfSF','TotalBsmtSF'
            'BsmtHalfBath'):
    all_data[col] = all_data[col].fillna(0)

#
all_data["MasVnrArea"] = all_data["MasVnrArea"].fillna(0)

#for below categorical basement-related feature NaN means that t
for col in ('BsmtQual', 'BsmtCond','BsmtExposure','BsmtFinType1'
    all_data[col] = all_data[col].fillna('None')

#group by Neigborhood and fill missing value with median Lot fro
all_data["LotFrontage"] = all_data.groupby("Neighborhood")["LotF
lambda x: x.fillna(x.median()))
```

Setting Mode values for missing entries

In [24]:
```python
#msZoning classification: 'RL' is common
all_data ['MSZoning'] = all_data['MSZoning'].fillna(all_data['MS

#functional: NA is typical
all_data["Functional"] = all_data["Functional"].fillna('Typ')

#Electrical
all_data['Electrical'] = all_data['Electrical'].fillna(all_data[

#KitchenQual
all_data['KitchenQual'] =all_data['KitchenQual'].fillna(all_data

#Extrerior !st and Exterior 2nd
all_data ['Exterior1st']= all_data['Exterior1st'].fillna(all_dat
all_data['Exterior2nd']= all_data['Exterior2nd'].fillna(all_data

#sale type
all_data['SaleType'] = all_data['SaleType'].fillna(all_data['Sal
```

In [25]:
```python
#dropping as same value 'AllPub' for all records except 2NA and
all_data = all_data.drop(['Utilities'], axis=1)
```

In [26]:
```python
#Transforming required numerical features to categorical
all_data['MSSubClass']= all_data['MSSubClass'].apply(str)
all_data['OverallCond'] =all_data['OverallCond'].astype(str)
all_data['YrSold'] = all_data['YrSold'].astype(str)
all_data['MoSold'] = all_data['MoSold'].astype(str)
```

In [27]:
```python
#Label Encoding some categorical variables
#for information in their ordering set

from sklearn.preprocessing import LabelEncoder
cols = ('FireplaceQu', 'BsmtQual', 'BsmtCond', 'GarageQual', 'Ga
        'ExterQual', 'ExterCond','HeatingQC', 'PoolQC', 'Kitchen
        'BsmtFinType2', 'Functional', 'Fence', 'BsmtExposure', '
        'LotShape', 'PavedDrive', 'Street', 'Alley', 'CentralAir
        'YrSold', 'MoSold')
#apply LabelEncoder to categorical features
for c in cols:
    lbl = LabelEncoder()
    lbl.fit(list(all_data[c].values))
    all_data[c] = lbl.transform(list(all_data[c].values))
#shape
print('Shape all_data: {}'.format(all_data.shape))
```

```
Shape all_data: (1456, 78)
```

In [28]:
```python
#add total surface area as TotalSf = basement + firstflr + secon
all_data['TotalSF'] = all_data['TotalBsmtSF'] + all_data['1stFlr
```

In [29]:
```python
#log transform skewed numeric features
numeric_features = all_data.dtypes[all_data.dtypes != "object"].

skewed_features = all_data[numeric_features].apply(lambda x : sk
#compute skewness
print ("\skew in numerical features: \n")
skewness = pd.DataFrame({'Skew' :skewed_features})
skewness.head(7)
```

```
\skew in numerical features:
```

Out[29]:

	Skew
MiscVal	24.418175
PoolArea	17.504556
LotArea	12.574590
3SsnPorch	10.279262
LowQualFinSF	8.989291
LandSlope	4.801326
KitchenAbvGr	4.476748

Box cox transformation of highly skewed features

In [30]:
```python
skewness = skewness[abs(skewness) > 0.75]
print ("There are {} skewed numerical features to box cox transf

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)
```

```
There are 59 skewed numerical features to box cox transform
```

adding dummy categorical features

In [31]:
```python
all_data = pd.get_dummies(all_data)
print(all_data.shape)
```

```
(1456, 220)
```

In [32]:
```python
ntrain = train.shape[0]
ntest = test.shape[0]
y_train= train.SalePrice.values
train = pd.DataFrame(all_data[:ntrain])
test = pd.DataFrame(all_data[ntrain:])
```

Linear regression Modeling

I will be implementing

Lasso Regression Gradient Boosting Regression

In [33]:
```python
from sklearn.linear_model import Lasso
from sklearn.preprocessing import RobustScaler
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import KFold, cross_val_score, trai
from sklearn.metrics import mean_squared_error
from sklearn.base import BaseEstimator, TransformerMixin, Regres
```

In [34]:
```python
#validation function
n_folds = 5

def RMSLE_cv(model):
    kf = KFold(n_folds, shuffle=True, random_state=42).get_n_spl
    rmse= np.sqrt(-cross_val_score(model, train.values, y_train,
cv = kf))
    return(rmse)
```

In [35]:
```python
#lasso
lasso = make_pipeline(RobustScaler(), Lasso(alpha = 0.0005, rand

#Gradient Boosting Regression
GBoost = GradientBoostingRegressor(loss='huber', learning_rate=0
                                   min_samples_split=10, min_sam
                                   random_state=5,max_features='
```

scores from the above models

In [36]:
```python
#Lasso
score = RMSLE_cv(lasso)
print ("\n Lasso score: {:.4f} ({:.4f})\n".format(score.mean(),s

#Gradient Boosting Regression
score = RMSLE_cv(GBoost)
print ("\n GBoost score: {:.4f} ({:.4f})\n".format(score.mean(),
```

```
 Lasso score: 0.1167 (0.0114)


 GBoost score: 0.1232 (0.0149)
```

stacking the models

In [37]:
```python
class AveragingModels(BaseEstimator, RegressorMixin, Transformer
    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 the
    def predict(self, X):
        predictions = np.column_stack([
            model.predict(X) for model in self.models_
        ])
        return np.mean(predictions, axis=1)



# Averaged base models score

averaged_models = AveragingModels(models = (GBoost, lasso))

score = RMSLE_cv(averaged_models)
print(" Averaged base models score: {:.4f} ({:.4f})\n".format(sc
```

```
 Averaged base models score: 0.1144 (0.0142)
```

In [38]:
```python
#defining RMSLE evaluation function
def RMSLE (y, y_pred):
    return np.sqrt(mean_squared_error(y, y_pred))
```

In [39]:
```python
#final training and prediction of the stacked regressor

averaged_models.fit(train.values, y_train)
stacked_train_pred = averaged_models.predict(train.values)
stacked_pred = np.expm1(averaged_models.predict(test.values))
print("RMSLE score on the train data:")
print(RMSLE(y_train,stacked_train_pred))
print("Accuracy score:")
averaged_models.score(train.values, y_train)
```

```
RMSLE score on the train data:
0.06868888502115643
Accuracy score:
```

Out[39]:
```
0.9694556424772478
```