In [1]:

```
import os
import pandas as pd
import numpy as np

from scipy.stats import skew,norm

import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style='darkgrid', palette='tab10')

%matplotlib inline
%config InlineBackend.figure format = 'retina'
```

In [2]:

```
train = pd.read_csv("/media/curtis0982/70607E83607E5038/python_code_E/data/classification/ames_house_prices/train.csv")
test = pd.read_csv("/media/curtis0982/70607E83607E5038/python_code_E/data/classification/ames house prices/test.csv")
```

In [3]:

```
train.shape
```

Out[3]:

```
(1460, 81)
```

In [4]:

```
test.shape
```

Out[4]:

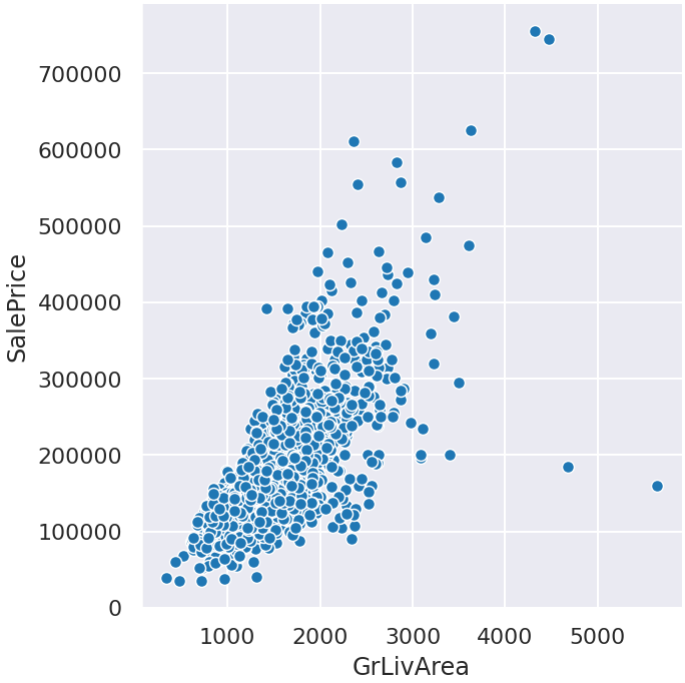
```
(1459, 80)
```

In [5]:

```
train_id = train['Id']
test_id = test['Id']
train = train.drop('Id',axis=1)
test = test.drop('Id',axis=1)
```

In [6]:

```
sns.relplot(y='SalePrice', x='GrLivArea', data=train);
```

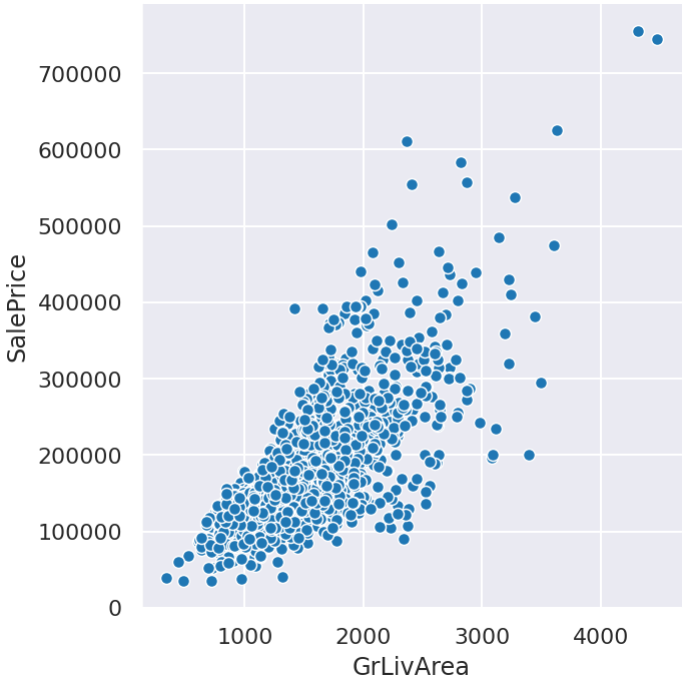


In [7]:

```
train = train.drop(train[(train['GrLivArea']>4000) & (train['SalePrice']<200000)].index)
```

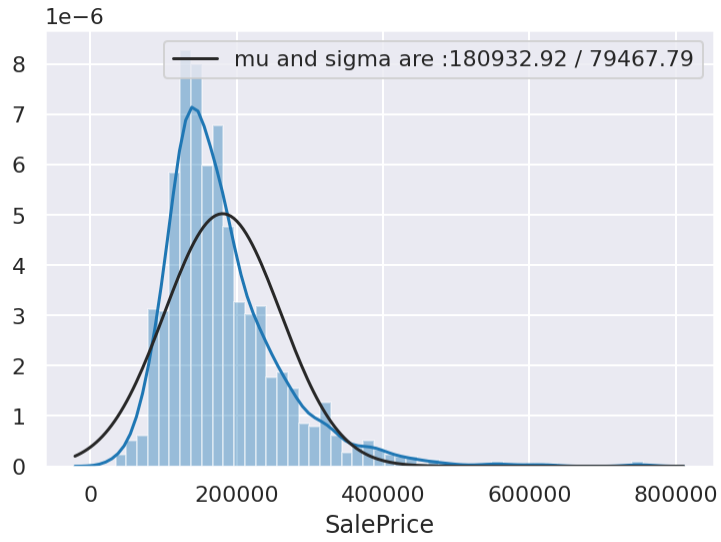
In [8]:

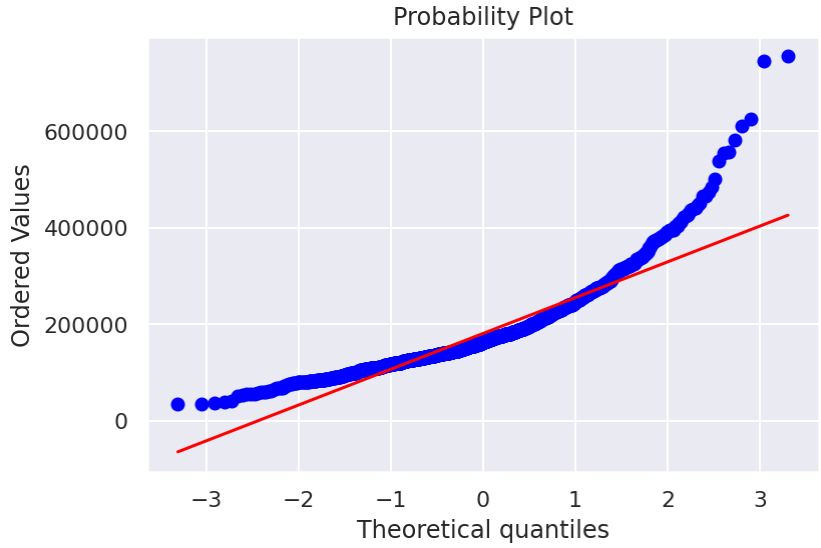
```
sns.relplot(y='SalePrice', x='GrLivArea', data=train);
```



In [9]:

```
#distplot
sns.distplot(train['SalePrice'],fit=norm)
(miu_s, sigma_s) = norm.fit(train['SalePrice'])
#legend記得要用bracket包
plt.legend(['mu and sigma are :{:.2f} / {:.2f}'.format(miu_s, sigma_s )])
plt.show()


#qq-plot
from scipy import stats
plt.figure()
stats.probplot(train['SalePrice'], plot=plt)
plt.show()
```



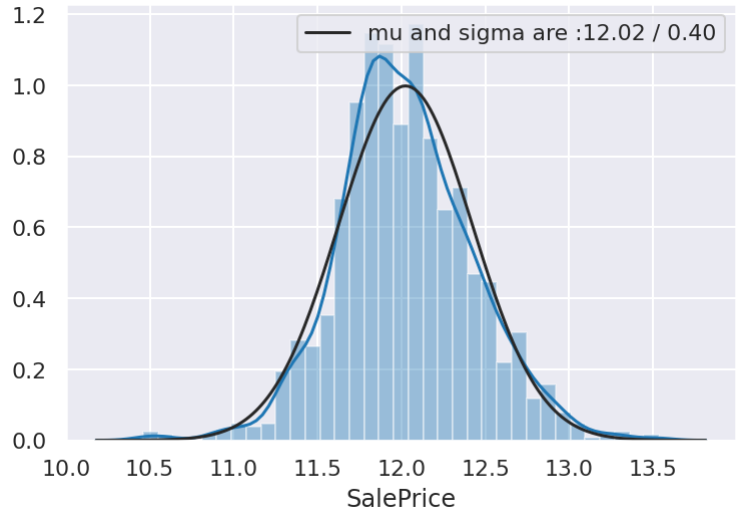


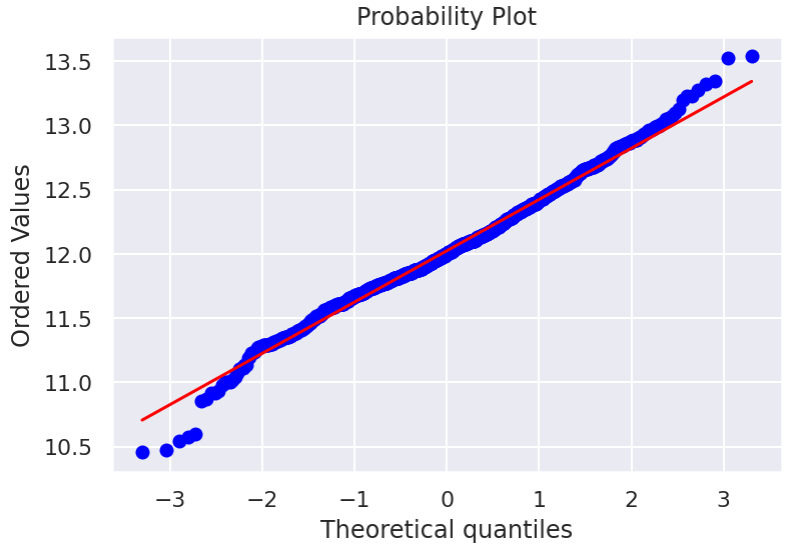
In [10]:

```
train['SalePrice'] = np.log1p(train['SalePrice'])
n_train = train.shape[0]
n_test = test.shape[0]
```

In [11]:

```
#distplot
sns.distplot(train['SalePrice'],fit=norm)
##記得要用tuple包 (mu,sigma)
(miu_s, sigma_s) = norm.fit(train['SalePrice'])
##legend記得要用bracket包
plt.legend(['mu and sigma are :{:.2f} / {:.2f}'.format(miu_s, sigma_s )])
plt.show()


#qq-plot
from scipy import stats
plt.figure()
stats.probplot(train['SalePrice'], plot=plt)
plt.show()
```





In [12]:

```
y_train = train['SalePrice']
train = train.drop('SalePrice',axis=1)
```

In [13]:

```
#記得要用([])list or tuple包
df = pd.concat([train, test]).reset_index(drop=True)
```

In [14]:

```
df.shape
```

Out[14]:

```
(2917, 79)
```

In [15]:

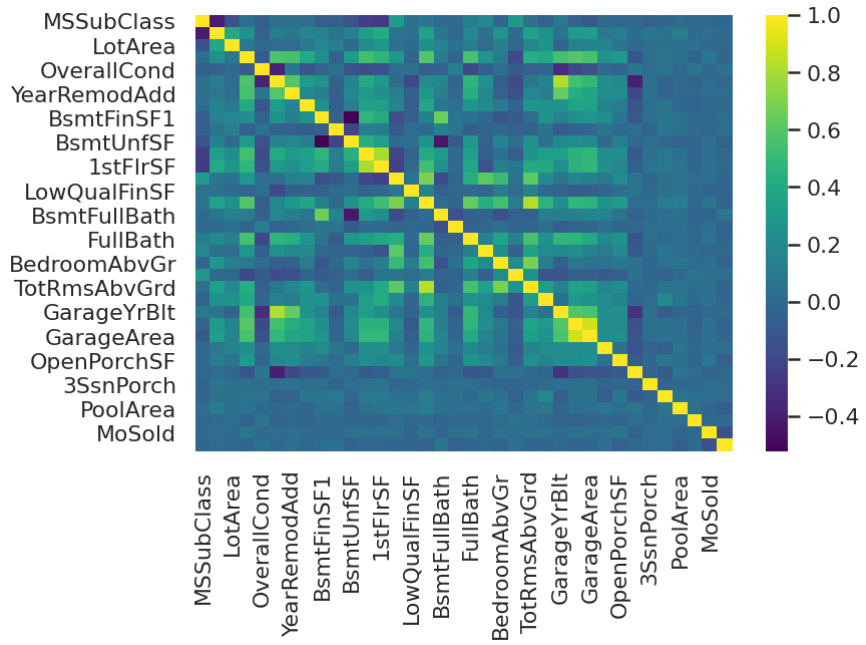
```
#check corr with y
##sort_values 有底線
corr_y = train.corrwith(y_train).sort_values(ascending=False)
corr_y[abs(corr_y)>0.3]
```

Out[15]:

```
OverallQual     0.821405
GrLivArea       0.725211
GarageCars      0.681033
GarageArea      0.656129
TotalBsmtSF     0.647563
1stFlrSF        0.620500
FullBath        0.595899
YearBuilt       0.587043
YearRemodAdd    0.565992
GarageYrBlt     0.541638
TotRmsAbvGrd    0.537702
Fireplaces      0.491998
MasVnrArea      0.434621
BsmtFinSF1      0.392283
LotFrontage     0.372900
WoodDeckSF      0.334251
OpenPorchSF     0.325215
2ndFlrSF        0.319953
HalfBath        0.314186
dtype: float64
```

In [16]:

```
corr_mat = train.corr()
sns.heatmap(corr_mat, cmap='viridis');
```



imputing

In [17]:

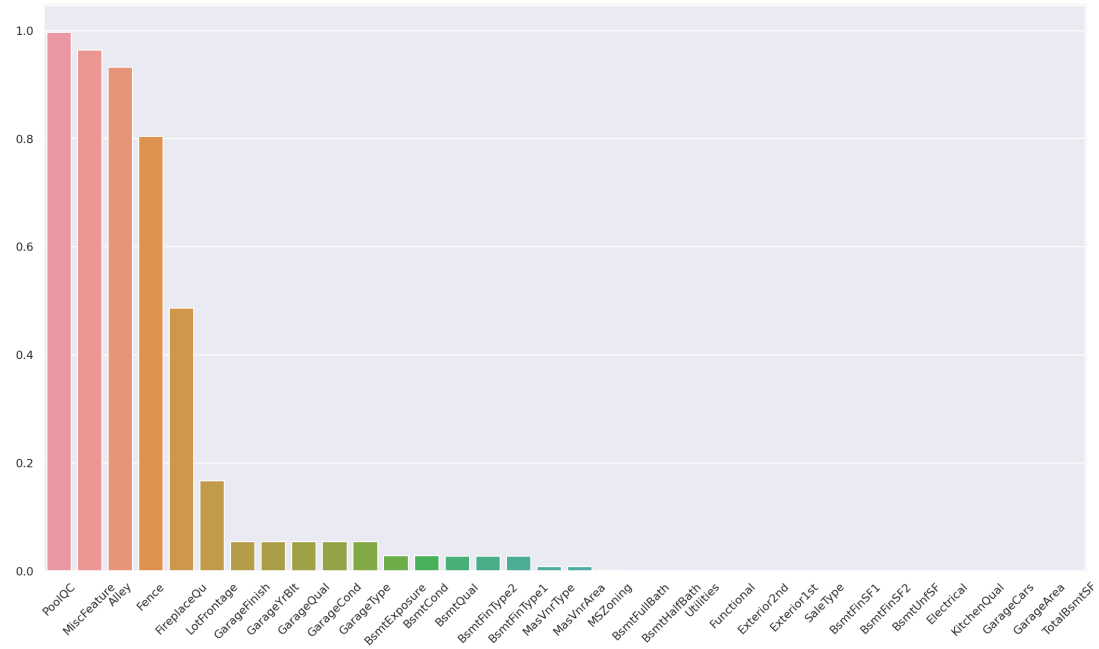
```
#補值前要要先看missing ratio

df_missing = df.isnull().sum()/len(df)
df_missing = df_missing[~(df_missing==0)].sort_values(ascending=False)
df_missing
```

Out[17]:

```
PoolQC          0.996915
MiscFeature     0.964004
Alley           0.932122
Fence           0.804251
FireplaceQu     0.486802
LotFrontage     0.166610
GarageFinish    0.054508
GarageYrBlt     0.054508
GarageQual      0.054508
GarageCond      0.054508
GarageType      0.053822
BsmtExposure    0.028111
BsmtCond        0.028111
BsmtQual        0.027768
BsmtFinType2    0.027425
BsmtFinType1    0.027083
MasVnrType      0.008228
MasVnrArea      0.007885
MSZoning        0.001371
BsmtFullBath    0.000686
BsmtHalfBath    0.000686
Utilities       0.000686
Functional      0.000686
Exterior2nd     0.000343
Exterior1st     0.000343
SaleType        0.000343
BsmtFinSF1      0.000343
BsmtFinSF2      0.000343
BsmtUnfSF       0.000343
Electrical      0.000343
KitchenQual     0.000343
GarageCars      0.000343
GarageArea      0.000343
TotalBsmtSF     0.000343
dtype: float64
```

In [18]:

```
#figsize 注意中間沒有底線
plt.figure(figsize=(18,10))
plt.xticks(rotation=45)
sns.barplot(x = df_missing.index ,y=df_missing);
```



In [19]:

```
df = df.drop(['PoolQC','MiscFeature','Alley'], axis=1)
```

In [20]:

```
#超過5成沒有的給None
df['Fence'] = df['Fence'].fillna('None')
df['FireplaceQu'] = df['FireplaceQu'].fillna('None')

#補garage相關
for col in ('GarageType', 'GarageFinish', 'GarageQual', 'GarageCond'):
    df[col] = df[col].fillna('None')
##no garage=>Garage面積為0
for col in ('GarageYrBlt', 'GarageArea', 'GarageCars'):
    df[col] = df[col].fillna(0)

#補basement相關
##no basement=>basement面積為0
for col in ('BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF','TotalBsmtSF', 'BsmtFullB
ath', 'BsmtHalfBath'):
    df[col] = df[col].fillna(0)
for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinT
ype2'):
    df[col] = df[col].fillna('None')


df['MasVnrType'] = df['MasVnrType'].fillna('None')
df['MasVnrArea'] = df['MasVnrArea'].fillna(0)

#按領域取
##假設zone可以用Neighborhood切分下的眾數
df['MSZoning'] = df.groupby('Neighborhood')['MSZoning'].transform(lambda x: x
.fillna(x.mode()[0]))
df['LotFrontage'] = df.groupby('Neighborhood')['LotFrontage'].transform(lambd
a x: x.fillna(x.median()))

#不重要的給none
for col in ['Utilities','Functional','Exterior2nd','Exterior1st','SaleType',\
            'Electrical','KitchenQual']:
    df[col] = df[col].fillna('None')

for col in ('GarageCars','GarageArea','TotalBsmtSF'):
    df[col] = df[col].fillna(0)
```

In [21]:

```
df_missing = df.isnull().sum()/len(df)
df_missing = df_missing[~(df_missing==0)].sort_values(ascending=False)
df_missing
```

Out[21]:

```
Series([], dtype: float64)
```

In [22]:

```
for col in ['MSSubClass','OverallQual','OverallCond','MoSold','YrSold']:
    df[col] = df[col].astype(str)
```

label transform

In [23]:

```
from sklearn.preprocessing import LabelEncoder

category_mask = df.dtypes == 'object'
category_col = df.columns[category_mask].tolist()
for col in category_col:
    df[col] = df[col].astype(str)
category_col[0:5]
```

Out[23]:

```
['MSSubClass', 'MSZoning', 'Street', 'LotShape', 'LandContou
r']
```

In [24]:

```
le = LabelEncoder()
df[category_col] = df[category_col].apply(lambda x : le.fit_transform(x))
```

In [25]:

```
df.columns[df.dtypes == 'string']
```

Out[25]:

```
Index([], dtype='object')
```

In [26]:

```
#加上估價重要變數: 總坪數
df['TotalSF'] = df['TotalBsmtSF'] + df['1stFlrSF'] + df['2ndFlrSF']
```

transform that are too skew

In [27]:

```
numeric_col = [i for i in df.columns if i not in category_col]
numeric_col
```

Out[27]:

```
['LotFrontage',
 'LotArea',
 'YearBuilt',
 'YearRemodAdd',
 'MasVnrArea',
 'BsmtFinSF1',
 'BsmtFinSF2',
 'BsmtUnfSF',
 'TotalBsmtSF',
 '1stFlrSF',
 '2ndFlrSF',
 'LowQualFinSF',
 'GrLivArea',
 'BsmtFullBath',
 'BsmtHalfBath',
 'FullBath',
 'HalfBath',
 'BedroomAbvGr',
 'KitchenAbvGr',
 'TotRmsAbvGrd',
 'Fireplaces',
 'GarageYrBlt',
 'GarageCars',
 'GarageArea',
 'WoodDeckSF',
 'OpenPorchSF',
 'EnclosedPorch',
 '3SsnPorch',
 'ScreenPorch',
 'PoolArea',
 'MiscVal',
 'TotalSF']
```

In [28]:

```
from scipy.stats import skew
from scipy.special import boxcox1p
skewness = df[numeric_col].apply(lambda x:skew(x.dropna())).sort_values(ascen
ding=False)
skewness.head(10)
skewness_df = pd.DataFrame({'skewness':skewness})
skewness_df

#series也可按條件提取
skewness_too = skewness[abs(skewness)>0.75].index
skewness too
```

Out[28]:

```
Index(['MiscVal', 'PoolArea', 'LotArea', 'LowQualFinSF', '3Ssn
Porch',
       'KitchenAbvGr', 'BsmtFinSF2', 'EnclosedPorch', 'ScreenP
orch',
       'BsmtHalfBath', 'MasVnrArea', 'OpenPorchSF', 'WoodDeckS
F', '1stFlrSF',
       'LotFrontage', 'GrLivArea', 'TotalSF', 'BsmtFinSF1', 'B
smtUnfSF',
       '2ndFlrSF', 'GarageYrBlt'],
      dtype='object')
```

In [29]:

```
skewness[abs(skewness)>0.75]
```

Out[29]:

```
MiscVal          21.939672
PoolArea         17.688664
LotArea          13.109495
LowQualFinSF     12.084539
3SsnPorch        11.372080
KitchenAbvGr      4.300550
BsmtFinSF2        4.144503
EnclosedPorch     4.002344
ScreenPorch       3.945101
BsmtHalfBath      3.929996
MasVnrArea        2.621719
OpenPorchSF       2.529358
WoodDeckSF        1.844792
1stFlrSF          1.257286
LotFrontage       1.103039
GrLivArea         1.068750
TotalSF           1.009157
BsmtFinSF1        0.980645
BsmtUnfSF         0.919688
2ndFlrSF          0.861556
GarageYrBlt      -3.904632
dtype: float64
```

In [30]:

```
lam = 0.15
df[skewness too] = df[skewness too].apply(lambda x:boxcox1p(x,lam))
```

one hot encoding

In [31]:

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
import sklearn
```

In [32]:

```
cltran = ColumnTransformer([('new_col', OneHotEncoder(),category_col)], remai
nder='passthrough')
df ohc = cltran.fit transform(df)
```

In [33]:

```
x_train =  df_ohc[:n_train]
x test = df ohc[n train:]
```

In [34]:

```
y train = y train.values
```

In [35]:

```
x train.shape
```

Out[35]:

```
(1458, 344)
```

In [36]:

```
y train.shape
```

Out[36]:

```
(1458,)
```

In [37]:

```
from sklearn.model_selection  import GridSearchCV,cross_val_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import SGDRegressor
from sklearn.ensemble import GradientBoostingRegressor,AdaBoostRegressor
```

In [48]:

```
# prepare configuration for cross validation test harness
# prepare models
models = []
models.append(('DecisionTreeRegressor', DecisionTreeRegressor()))
models.append(('SGDRegressor', SGDRegressor()))
models.append(('GradientBoostingRegressor', GradientBoostingRegressor()))
models.append(('AdaBoostRegressor', AdaBoostRegressor()))

# evaluate each model in turn
results = []
names = []
scoring = 'neg_mean_squared_error'
scores={}
for name, model in models:
    cv_results = cross_val_score(model, x_train, y_train, cv=4, scoring=scori
ng)
    results.append(cv_results)
    #scores[name]= name)
    scores[name] = np.mean(cv_results)
    print("model: {},score: {:.4f}".format(name,np.mean(cv_results)))
#print(scores)
```

```
model: DecisionTreeRegressor,score: -0.0411
model: SGDRegressor,score: -822813007500992716590532263936O.00
00
model: GradientBoostingRegressor,score: -0.0161
model: AdaBoostRegressor,score: -0.0320
```

In [ ]:

```
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(scores)
ax.set_xticklabels(names)
plt.show()
```

In [50]:

```
gbr = GradientBoostingRegressor()
gbr.fit(x_train,y_train)
y_test =gbr.predict(x_test)
```

```
---------------------------------------------------------------
-------------
NameError                                 Traceback (most rece
nt call last)
<ipython-input-50-2bab22e7b021> in <module>
      2 gbr.fit(x_train,y_train)
      3 y_test =gbr.predict(x_test)
----> 4 y_test=numpy.expm1(y_test)

NameError: name 'numpy' is not defined
```

In [51]:

```
y_test=np.expm1(y_test)
```

In [53]:

```
sub = pd.DataFrame()
sub['Id'] = test_id
sub['SalePrice'] = y_test
```

In [54]:

```
sub
```

Out[54]:

	Id	SalePrice
0	1461	122311.234883
1	1462	156474.925833
2	1463	197005.690460
3	1464	197529.397533
4	1465	190324.703725
...	...	...
1454	2915	86651.117965
1455	2916	88827.949815
1456	2917	146184.675060
1457	2918	118431.422209
1458	2919	215448.168692

1459 rows × 2 columns

In [ ]:

```

```