In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.model_selection import train_test_split
from scipy import stats
from scipy.stats import norm, skew
from scipy.special import boxcox1p
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVR
import xgboost as xgb
from sklearn.base import BaseEstimator, TransformerMixin, RegressorMixin, clon
e
from sklearn.model_selection import train_test_split
from sklearn.preprocessing  import StandardScaler, LabelEncoder, MinMaxScaler,
RobustScaler
from xgboost import XGBRegressor
from sklearn.linear_model import ElasticNet, Lasso
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

In [2]:
```
warnings.filterwarnings("ignore")
```

In [3]:
```
df = pd.read_csv("dataset.csv")
pd.set_option('max_columns',81)
```

In [4]:
```
df.shape
```

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

In [5]:
```
y = df.SalePrice
X = df.drop(['SalePrice'], axis=1)
```

In [6]:
```
xtrain, xtest , ytrain, ytest = train_test_split(X, y, test_size=0.2, random_s
tate=0)
```

Removing Missing Values

In [7]:
```
#Variables that contain more than 80 percent of null values are dropped
df = df.drop(['Id','PoolQC','MiscFeature','Alley','Fence'], axis=1)
```

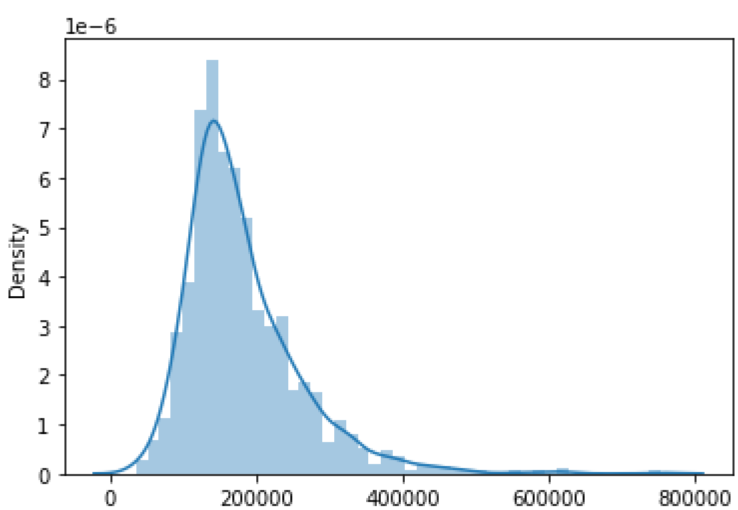
In [8]:
```
#to replace the following column with missing values with most occuring value
 i.e. Mode of column
missing_val = ['GarageCond','GarageType','GarageYrBlt','GarageFinish','GarageQ
ual','BsmtExposure',
               'BsmtFinType2','BsmtFinType1','BsmtCond','BsmtQual','MasVnrAre
a','MasVnrType','Electrical']
```

In [9]:
```python
def impute_nan(DataFrame,ColName):
    Mode_Category = DataFrame[ColName].mode()[0]
    DataFrame[ColName].fillna(Mode_Category,inplace=True)

for Columns in missing_val:
    impute_nan(xtrain,Columns)
    impute_nan(xtest,Columns)
```

In [10]:
```python
#Imputing missing values with a None Category
xtrain['FireplaceQu'].fillna('None',inplace=True)
xtest['FireplaceQu'].fillna('None',inplace=True)
#Imputing missing values with a Mean Value
xtrain['LotFrontage'].fillna(int(xtrain['LotFrontage'].mean()),inplace=True)
xtest['LotFrontage'].fillna(int(xtest['LotFrontage'].mean()),inplace=True)
```

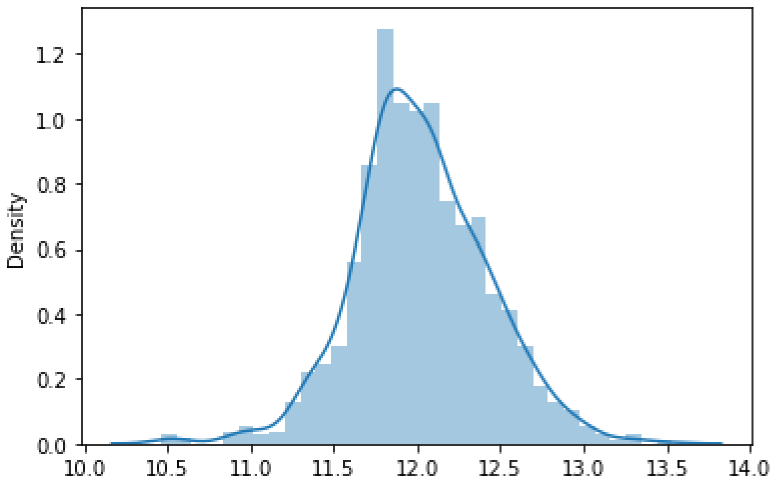
Checking for Skewness in training set

In [11]:
```python
try:
    sns.distplot(ytrain , fit='norm');
    (mu, sigma) = norm.fit(ytrain)
    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(ytrain, plot=plt)
    plt.show()
except AttributeError:
    pass
```



In [12]:

```
try :
    #We use the numpy fuction log1p which  applies log(1+x) to all elements of the column
    ytrain = np.log1p(ytrain)
    sns.distplot(ytrain , fit='norm');
    (mu, sigma) = norm.fit(ytrain)
    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')
except AttributeError:
    pass
```



In [13]:

```
num = xtrain.dtypes[xtrain.dtypes != "object"].index
num = num.drop(['YearBuilt','YearRemodAdd','YrSold'])
# Check the skew of all numerical features
skewed_feats = xtrain[num].apply(lambda x: skew(x)).sort_values(ascending=False)
print("\nSkew in numeric features: \n")
skewness = pd.DataFrame({'Skew' :skewed_feats})
skewness
```

Skew in numeric features:

Out[13]:

	Skew
MiscVal	22.309998
PoolArea	17.469394
LotArea	12.124482
3SsnPorch	10.584917
LowQualFinSF	8.595711
KitchenAbvGr	4.643574
BsmtFinSF2	4.179755
ScreenPorch	4.125879
BsmtHalfBath	4.118557
EnclosedPorch	3.062687
MasVnrArea	2.718080
OpenPorchSF	2.401261
LotFrontage	1.912608
WoodDeckSF	1.514965
MSSubClass	1.422439
GrLivArea	1.075380
1stFlrSF	0.960357
BsmtUnfSF	0.880931
2ndFlrSF	0.780461
BsmtFinSF1	0.763211
OverallCond	0.679231
TotRmsAbvGrd	0.622816
Fireplaces	0.621503
HalfBath	0.600492
TotalBsmtSF	0.590725
BsmtFullBath	0.579140
BedroomAbvGr	0.234383
MoSold	0.233001
GarageArea	0.174087
OverallQual	0.169675
FullBath	0.032919
Id	-0.015361
GarageCars	-0.358061
GarageYrBlt	-0.705524

In [14]:

```
skewness = skewness[abs(skewness) > 0.75]
print("There are {} skewed features for Box Cox transformation".format(skewness.shape[0]))
skewed_features = skewness.index
lam = 0.15
for i in skewed_features:
    xtrain[i] = boxcox1p(xtrain[i], lam)
xtrain[skewed_features] = np.log1p(xtrain[skewed_features])
```

```
There are 34 skewed features for Box Cox transformation
```

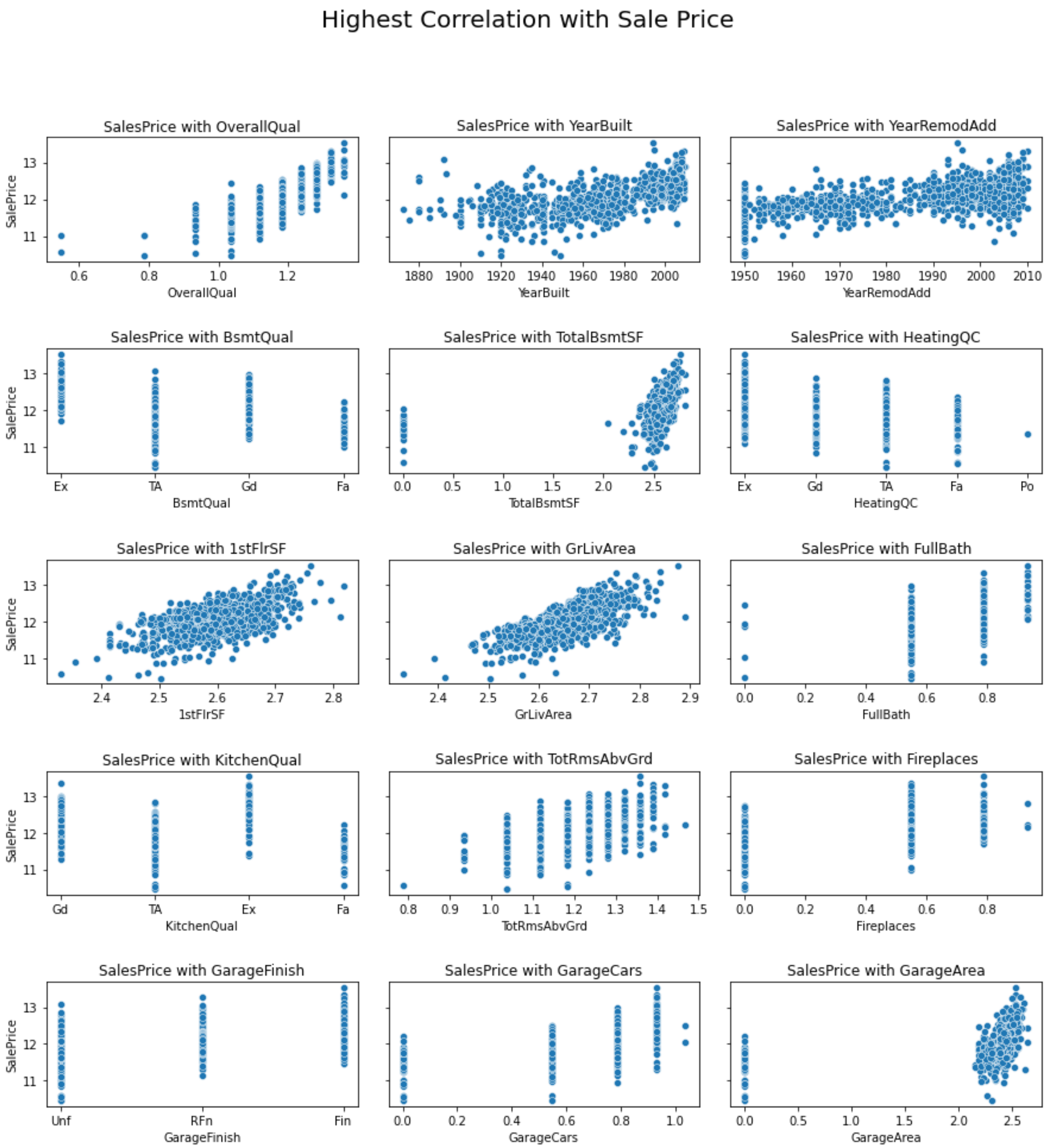
In [15]:

```
high_correlated_var= ['OverallQual','GrLivArea','GarageCars','YearBuilt','GarageArea','FullBath','TotalBsmtSF','1stFlrSF',
                      'YearRemodAdd','TotRmsAbvGrd','Fireplaces','GarageFinish','BsmtQual','KitchenQual','HeatingQC']

for i in xtrain.columns:
    if i not in high_correlated_var:
        xtrain = xtrain.drop([i],axis=1)
        xtest = xtest.drop([i],axis=1)
```

Taking highly correlated variables found during EDA from the initial contribution keeping the threshold for positive correlation to be 0.5 and negative correlation to be -0.4. The correlation has been calculated using the filter method. Filter methods are much faster compared to wrapper methods as they do not involve training the models. On the other hand, wrapper methods are computationally costly, and in the case of massive datasets, wrapper methods are not the most effective feature selection method to consider.

In [16]:

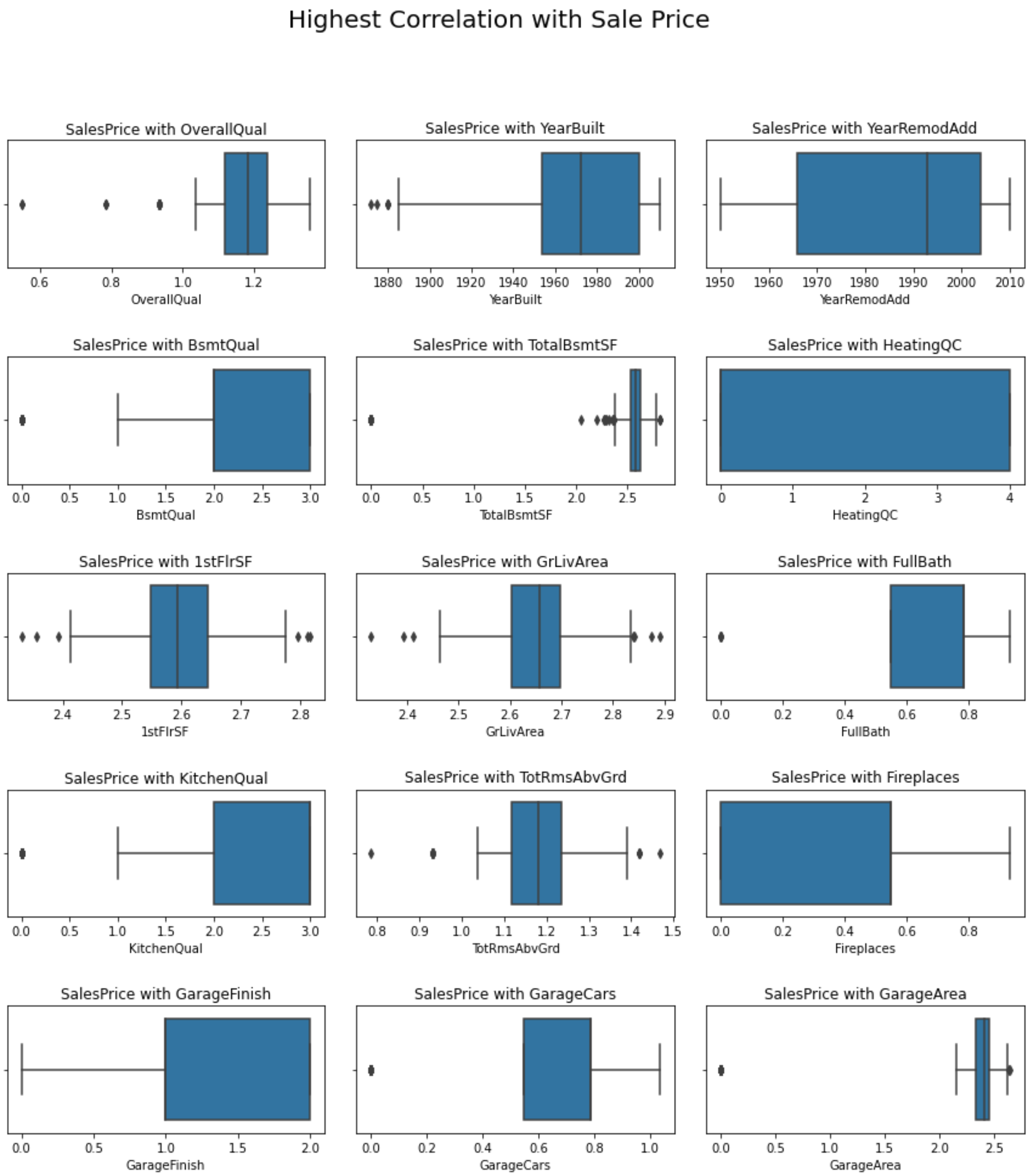
```
fig, axes = plt.subplots(5,3, figsize=(15, 15), sharey=True);
plt.subplots_adjust(hspace = 0.7, wspace=0.1)
fig.suptitle('Highest Correlation with Sale Price', fontsize=20);

for i,col in zip(range(15),xtrain.columns):
    sns.scatterplot(y=ytrain, x=xtrain[col], ax=axes[i//3][i%3])
    axes[i//3][i%3].set_title('SalesPrice with '+col)
```



Checking for Outliers

In [17]:

```
xt = xtrain.copy()
for column in xt.columns:
    if column in xtrain.dtypes[df.dtypes == "object"].index:
        labelencoder = LabelEncoder()
        xt[column] = labelencoder.fit_transform(xt[column])

fig, axes = plt.subplots(5,3, figsize=(15, 15), sharey=True);
plt.subplots_adjust(hspace = 0.7, wspace=0.1)
fig.suptitle('Highest Correlation with Sale Price', fontsize=20);

for i,col in zip(range(15),xtrain.columns):
    sns.boxplot(x=xt[col], ax=axes[i//3][i%3])
    axes[i//3][i%3].set_title('SalesPrice with '+col)
```



In [18]:
```
drop_index = xt[ (xt['OverallQual']<1) | (xt['YearBuilt']<1880) | (xt['BsmtQua
l']<0.5) | (xt['TotalBsmtSF']<2.3) |
                (xt['1stFlrSF']<2.4) | (xt['1stFlrSF']>2.8) | (xt['GrLivArea']
<2.45) | (xt['GrLivArea']>2.85) |
                (xt['FullBath']<0.1) | (xt['KitchenQual']<0.5) | (xt['TotRmsAb
vGrd']<1) | (xt['TotRmsAbvGrd']>1.4) |
                (xt['GarageCars']==0.0) | (xt['GarageArea']<0.1)].index
```

In [19]:
```
#drop_index = drop_index.tolist()
```

In [20]:
```
xtrain = xtrain.drop(index = drop_index)
ytrain = ytrain.drop(index = drop_index)
```

In [21]:
```
print(xtest.shape,xtrain.shape)
```

```
(292, 15) (934, 15)
```

Encoding data using One hot Encoder

In [22]:
```
def encoding(dataframe):
    category_df =pd.DataFrame()
    categorical_cols = ['YearBuilt', 'YearRemodAdd' ,'BsmtQual', 'HeatingQC',
'KitchenQual', 'GarageFinish']
    for i in categorical_cols:
        category_df[i]= dataframe[i]
    category_df['YearBuilt'] = category_df['YearBuilt'].astype(str)
    category_df['YearRemodAdd'] = category_df['YearRemodAdd'].astype(str)
    category_df = pd.get_dummies(category_df)
    dataframe =  dataframe.drop(['YearBuilt', 'YearRemodAdd' ,'BsmtQual', 'Hea
tingQC', 'KitchenQual', 'GarageFinish'],axis=1)
    dataframe = pd.concat([dataframe,category_df], axis=1)
    return dataframe
```

In [23]:
```
def normalise(dataframe):
    dataframe = np.array(dataframe)
    norm = MinMaxScaler().fit(dataframe)
    dataframe = norm.transform(dataframe)
    return dataframe, norm
```

In [24]:
```
def denormalise(dataframe,norm):
    dataframe = norm.inverse_transform(dataframe)
    return dataframe
```

In [25]:
```
xtrain = encoding(xtrain)
```

In [26]:
```
xtest = encoding(xtest)
```

In [27]:
```
print(xtest.shape,xtrain.shape)
```

```
(292, 165) (934, 190)
```

In [28]:
```
#Adding the left over one hot encoded columns to the train and test set respec
tively
a = xtest.columns
b = xtrain.columns
print(len(a),len(b))
for i in b:
    if i not in a:
        xtest[i]= 0
for i in a:
     if i not in b:
        xtrain[i]=0
```

```
165 190
```

In [29]:
```
print(xtest.shape,xtrain.shape)
```

```
(292, 194) (934, 194)
```

In [30]:
```
xtrain , norm_train = normalise(xtrain)
```

In [31]:
```
xtest , norm_test = normalise(xtest)
```

Applying Machine Learning Models to the data

In [32]:
```
#creating model pipelines
base_models = {"Elastic Net":make_pipeline(RobustScaler(),ElasticNet(alpha=0.0
005,l1_ratio=0.9)),

               "Lasso" : make_pipeline(RobustScaler(), Lasso(alpha =0.0005,ran
dom_state=1)),

               "Random Forest": RandomForestRegressor(n_estimators=300),
               "SVM": SVR(),
               "XGBoost": XGBRegressor(),
               "Gradient Boosting":make_pipeline(StandardScaler(),GradientBoos
tingRegressor(n_estimators=3000,
                                                 learning_rate=0.005,max_depth
=4, max_features='sqrt',
                                                 min_samples_leaf=15, min_samp
les_split=10,loss='huber', random_state =5))}
```

In [33]:
```
models_data = {'R^2':{'Training':{},'Testing':{}},
               'MAE':{'Training':{},'Testing':{}},
               'MSE':{'Training':{},'Testing':{}},
               'RMSE':{'Training':{},'Testing':{}}}
```

In [40]:

```
for name in base_models:
    #fitting the model
    model = base_models[name].fit(xtrain, ytrain)
    #make predictions with train and test datasets
    y_pred_train = model.predict(xtrain)
    y_pred_test = model.predict(xtest)

    #calculate the error for training and testing

    mae_train, mae_test = mean_absolute_error(ytrain, y_pred_train), mean_squa
red_error(np.log1p(ytest), y_pred_test)
    models_data['MAE']['Training'][name], models_data['MAE']['Testing'][name]
= mae_train, mae_test

    mse_train, mse_test = mean_squared_error(ytrain, y_pred_train), mean_squar
ed_error(np.log1p(ytest), y_pred_test)
    models_data['MSE']['Training'][name], models_data['MSE']['Testing'][name]
= mse_train, mse_test

    rmse_train, rmse_test = np.sqrt(mse_train), np.sqrt(mse_test)
    models_data['RMSE']['Training'][name], models_data['RMSE']['Testing'][name
] = rmse_train, rmse_test

    print('\n========================={}========================='.format(name
))
    print('**********Training**********************Testing********')
    print('MAE    : ',mae_train,' '*(25-len(str(mae_train))),mae_test)
    print('MSE    : ',mse_train,' '*(25-len(str(mse_train))),mse_test)
    print('RMSE   : ',rmse_train,' '*(25-len(str(rmse_train))),rmse_test)
```

```
=========================Elastic Net=========================
**********Training**********************Testing********
MAE    :  0.09536441852511089        0.19671621946451287
MSE    :  0.016650390115091496       0.19671621946451287
RMSE   :  0.12903639066205896        0.44352702224837764

=========================Lasso=========================
**********Training**********************Testing********
MAE    :  0.09580612943140099        0.19631180524663489
MSE    :  0.01686669361714626        0.19631180524663489
RMSE   :  0.1298718353498797         0.44307088061238564

=========================Random Forest=========================
**********Training**********************Testing********
MAE    :  0.038337429339717115       0.17575906685132706
MSE    :  0.0030789444431446713      0.17575906685132706
RMSE   :  0.055488236979964245       0.4192362899980476

=========================SVM=========================
**********Training**********************Testing********
MAE    :  0.0752888369056795         0.10097539202931448
MSE    :  0.008817298696220912       0.10097539202931448
RMSE   :  0.09390047228965843        0.3177662537610224

=========================XGBoost=========================
**********Training**********************Testing********
MAE    :  0.023410412001879707       0.13190947739770784
MSE    :  0.0010415417156598961      0.13190947739770784
RMSE   :  0.032272925427669186       0.36319344349493404

=========================Gradient Boosting=========================
**********Training**********************Testing********
MAE    :  0.08291753797984634        0.11415398137461771
MSE    :  0.01399132928736621        0.11415398137461771
RMSE   :  0.11828494953867212        0.3378668101110521
```

We can improve upon Model performance by using more data from the original dataset in the future and make new features on the basis of correlation with each other. Cross validation can also be used to analyse the performance of the model. We can also use machine learning models for feature selection in future to get more insights.

According to the above summary we observe SVM regressor performed quite well for the test dataset with error values significantly lower than the other models.

In [ ]: