Project 1 part 1 -report

Read in data, delet columns with almost one value, take log of sales price since it's skewed, drop 5 outliers.

In [4]:

```
data = pd.read_csv('Ames_data.csv')
data = data.drop(columns = ['Condition_2','Utilities','Longitude','Latitude'])
data = data.drop(index = data.index[data['Gr_Liv_Area']>4000])
```

Encode ordered categorical variable if the levels contains useful infomation for sales price. I provide 5 levels and each represent a scale number.

In [5]:

```
excellent=5;good=4;;average=3;fair=2;poor=1;No=0
dic_overall = {'Very_Excellent':excellent,'Excellent':excellent,'Very_Good':good,'Good':good,'Above_Ave
rage':average,
                 'Average':average,'Below_Average':fair,'Fair':fair,'Poor':poor,'Very_Poor':poor}
data['Overall_Qual'] = data['Overall_Qual'].map(dic_overall)
data['Overall_Cond'] = data['Overall_Cond'].map(dic_overall)
dic_exter = {'Excellent':excellent,'Good':good,'Typical':average, 'Fair':fair,'Poor':poor}
data['Exter_Qual'] = data['Exter_Qual'].map(dic_exter)
data['Exter_Cond'] = data['Exter_Cond'].map(dic_exter)
dic_bsmt = {'Excellent':excellent,'Good':good,'Typical':average,'Fair':fair,'Poor':poor,'No_Basement':N
o}
data['Bsmt_Qual'] = data['Bsmt_Qual'].map(dic_bsmt)
data['Bsmt_Cond'] = data['Bsmt_Cond'].map(dic_bsmt)
data = data.replace({'Bsmt_Exposure':{"No" : No, "Mn" : poor, "Av": fair,"Gd" : average,"No_Basement":N
o},
                'BsmtFin_Type_1':{"No_Basement" : No, "Unf" : poor, "LwQ": fair, "Rec" : average,"BLQ" :
good,
                                    "ALQ" : excellent, "GLQ" : excellent},
                'BsmtFin_Type_2':{"No_Basement" : No, "Unf" : poor, "LwQ": fair, "Rec" : average,"BLQ" :
good,
                                    "ALQ" : excellent, "GLQ" : excellent},
                'Fireplace_Qu':{'Excellent':excellent,'Good':good,'Typical':average,'Fair':fair,'Poor':po
or,'No_Fireplace':No},
                'Functional':{"Sal" : poor, "Sev" : poor, "Maj2" : poor, "Maj1" : poor,"Mod": poor, "Min2
" : poor, "Min1" : poor, "Typ" : fair},
               'Garage_Cond':{'Excellent':excellent,'Good':good,'Typical':average, 'Fair':fair,'Poor':poo
r,'No_Garage':No},
               'Garage_Qual':{'Excellent':excellent,'Good':good,'Typical':average,'Fair':fair,'Poor':poor
,'No_Garage':No},
               'Heating_QC': {'Excellent':excellent,'Good':good,'Typical':average,'Fair':fair,'Poor':poor
},
                'Kitchen_Qual': {'Excellent':excellent,'Good':good,'Typical':average, 'Fair':fair,'Poor':
poor},
               "Land_Slope" : {"Sev" : poor, "Mod" : fair, "Gtl" : average},
               "Paved_Drive" : {"Paved" : fair, "Dirt_Gravel" : poor,"Partial_Pavement" : No},
               'Pool_QC':{'Excellent':excellent,'Good':good,'Typical':average,'Fair':fair,'Poor':poor,'No
_Pool':No},
                "Street" : {"Grvl" : poor, "Pave" : fair},
                "Alley" : {"Gravel" : poor, "Paved" : fair,'No_Alley_Access':No},
                "Lot_Shape" : {"Irregular" : poor, "Moderately_Irregular" : fair, "Slightly_Irregular" :
average, "Regular" : good},
                'Fence':{'Good_Privacy':excellent,'Minimum_Privacy':good,'Good_Wood':average,'Minimum_Woo
d_Wire':fair,'No_Fence':poor},
                "Garage_Finish" : {"Fin" : excellent, "RFn" : average, "Unf" : poor, 'No_Garage':No}})
```

Adding more features, combine useful features together for example total squarfoot, total number of bath, total number of Porch, the sold year, remodeling year

In [6]:

```
data['Total_SF'] = data['Total_Bsmt_SF'] + data['First_Flr_SF'] + data['Second_Flr_SF']
data['Total_Bath'] = data['Bsmt_Full_Bath']+0.5*data['Bsmt_Half_Bath']+data['Full_Bath'] + 0.5*data['Ha
lf_Bath']
data["All_Porch"] = data["Open_Porch_SF"] + data["Enclosed_Porch"] + data["Three_season_porch"] + data[
```

```
"Screen_Porch"]
data['Sold_Build_Year'] = data['Year_Sold'] - data['Year_Built']
data['Rm_Build_Year'] = data['Year_Remod_Add'] - data['Year_Built']
```

Divide continuous and object data, fill in nan with mean and 'not applied' respectively, first divide continuous and object data, then remove variable where the value occur the most takes up to 99%of total

In [7]:

```
#
categorical_features = data.select_dtypes(include = ["object"]).columns
numerical_features = data.select_dtypes(exclude = ["object"]).columns
numerical_features = numerical_features.drop("Sale_Price")
train_num = data[numerical_features]
train_num = train_num.fillna(train_num.mean())
train_cat = data[categorical_features]
train_cat = train_cat.apply(lambda x:x.fillna('not_applied'))
train_cat = pd.get_dummies(train_cat)
all_data = pd.concat([train_num,train_cat,data['Sale_Price']],axis=1)
#
total = all_data.shape[0]
to_rm = []
for i in all_data:
    if (all_data[i].value_counts().max()/total)>=0.99:
        to_rm.append(i)
data_model = all_data.drop(columns=to_rm)
train = data_model[:ntrain]
test = data_model[ntrain:]
ytrain = np.log(train['Sale_Price'])
xtrain = train.drop(columns=['Sale_Price'])
ytest = test['Sale_Price']
xtest = test.drop(columns = ['Sale_Price'])
```

For xgb model, choose the best parameter of the results of random cross validation search. Present the model as the final model.

In [15]:

```
import xgboost as xgb
random_grid = {'n_estimators': [1000,1500,2000,2500],'colsample_bytree': [0.2,0.4,0.6,0.8],
               'max_depth': [1,2,3,4,5,6,10,14,18,23],'min_child_weight':[1,1.2,1.4,1.6,1.8],
               'reg_alpha':[0.3,0.4,0.5],'reg_lambda':[0.6,0.7,0.8,0.9],'subsample':[0.4,0.5,0.6]}
model_xgb = xgb.XGBRegressor()
xgb_random = RandomizedSearchCV(estimator = model_xgb, param_distributions = random_grid, n_iter = 100,
cv = 3,
                                verbose=2, random_state=42, n_jobs = -1)
xgb_random.fit(xtrain,ytrain)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

```
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:  3.3min
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed: 17.0min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 31.1min finished
```

Out[15]:

```
RandomizedSearchCV(cv=3, error_score='raise',
          estimator=XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1),
          fit_params=None, iid=True, n_iter=100, n_jobs=-1,
          param_distributions={'subsample': [0.4, 0.5, 0.6], 'reg_lambda': [0.6, 0.7, 0.8, 0.9], 'min_c
hild_weight': [1, 1.2, 1.4, 1.6, 1.8], 'max_depth': [1, 2, 3, 4, 5, 6, 10, 14, 18, 23], 'reg_alpha': [0
.3, 0.4, 0.5], 'colsample_bytree': [0.2, 0.4, 0.6, 0.8], 'n_estimators': [1000, 1500, 2000, 2500]},
          pre_dispatch='2*n_jobs', random_state=42, refit=True,
          return_train_score='warn', scoring=None, verbose=2)
```

Then another model was trained using the same approach, the final best models are shown below

In [ ]:

```
model_xgb1 = xgb.XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
```

```
        colsample_bytree=0.2, gamma=0, learning_rate=0.1, max_delta_step=0,
        max_depth=1, min_child_weight=1.2, missing=None, n_estimators=1500,
        n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
        reg_alpha=0.3, reg_lambda=0.9, scale_pos_weight=1, seed=None,
        silent=True, subsample=0.4)

model_xgb2 = xgb.XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
        colsample_bytree=0.2, gamma=0, learning_rate=0.1, max_delta_step=0,
        max_depth=2, min_child_weight=1, missing=None, n_estimators=1500,
        n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
        reg_alpha=0.4, reg_lambda=0.6, scale_pos_weight=1, seed=None,
        silent=True, subsample=0.5)
```

Overall, my laptop is a Lenovo X1 with 2.40GHz, 8GB memory, the running time for grid searching took about 50 mins to finish. Once the models have been built, it took 10.5 seconds to run, the average RMSE is 0.118 when using the ten train/test split