In [1]:
```
import sys

sys.path.append('C:\\Users\\Nikhil\\Data_Science_Projects')
from common_ds_modules import missing_values, data_manipulation, modeling
import os
import pandas as pd
import numpy as np
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
import seaborn as sns
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import Lasso
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeRegressor
from xgboost import XGBRegressor
import pickle
from sklearn.linear_model import Lasso, Ridge, ElasticNet
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import StackingRegressor

from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline

from sklearn.preprocessing import FunctionTransformer
log_transformer = FunctionTransformer(np.log1p)

from sklearn.compose import make_column_selector as selector
from scipy.stats import skew
```

```
C:\Users\Nikhil\anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarning: A Num
Py version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected versi
on 1.26.0
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [2]:
```
MAX_MODELS = 50
```

Utility Functions

In [3]:
```
def get_column(df, query=''):
    for c in df.columns:
        if query.lower() in c.lower():
            print(f'Column: {c}')
```

In [4]:
```
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')
```

```
train_df['MSZoning'] = train_df['MSZoning'].apply(str)
train_df['MSZoning'] = train_df['MSZoning'].replace({'C (all)': 'C'})

train_df['MSSubClass'] = train_df['MSSubClass'].apply(str)

test_df['MSZoning'] = test_df['MSZoning'].apply(str)
test_df['MSZoning'] = test_df['MSZoning'].replace({'C (all)': 'C'})
```

In [5]:
```
train_df['MSSubClass'] = train_df['MSSubClass'].apply(lambda x: str(x))
test_df['MSSubClass'] = test_df['MSSubClass'].apply(lambda x: str(x))
train_df['LogSalePrice'] = np.log1p(train_df['SalePrice'])
```

Fill in Missing Values

Getting rid of all columns that have more than 30% missing values. I chose 30 as a rather arbitrary number, I thought it was just right, if it is more than 30, then you will corrupt the variable distribution with the fill in value.

For categorical variables I fill in missing values by using the mode, and for numerical variables I use the mean.

In [6]:
```
train_drop = missing_values.get_high_missing_value_columns(train_df, 30)
test_drop = missing_values.get_high_missing_value_columns(test_df, 30)


train_low_missing_vals_df = train_df.drop(train_drop, axis='columns')
test_low_missing_vals_df = test_df.drop(test_drop, axis='columns')
```

This function gets the numerical and categorical variables, and fills in missing values as well.

In [7]:
```
numerical_variables, categorical_variables = data_manipulation.get_numerical_categoric


```

Sanity check to make sure that training and testing dataset don't have any columns with missing values after fill in.

In [8]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import make_column_transformer
```

In [9]:
```
def plot_variable_dist(df, variables):
    for c in variables:
        plt.hist(train_df[c])
        plt.title(f'Distribution for {c}')
        plt.show()
```

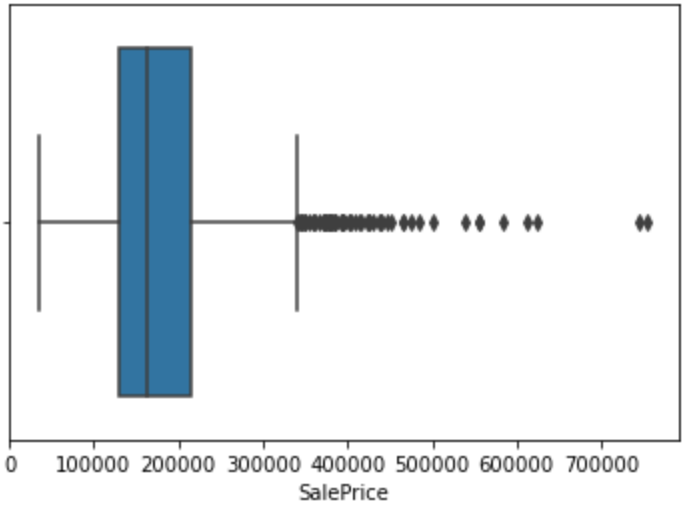
Understanding the target variable

For this problem I am trying to predict the SalePrice, so that is the target. In this section I will create a few plots to examine the SalePrice variable

In [10]:
```
target = 'SalePrice'
```

In [11]:
```
sns.boxplot(train_df[target])
plt.show()
```

```
C:\Users\Nikhil\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only valid po
sitional argument will be `data`, and passing other arguments without an explicit key
word will result in an error or misinterpretation.
  warnings.warn(
```

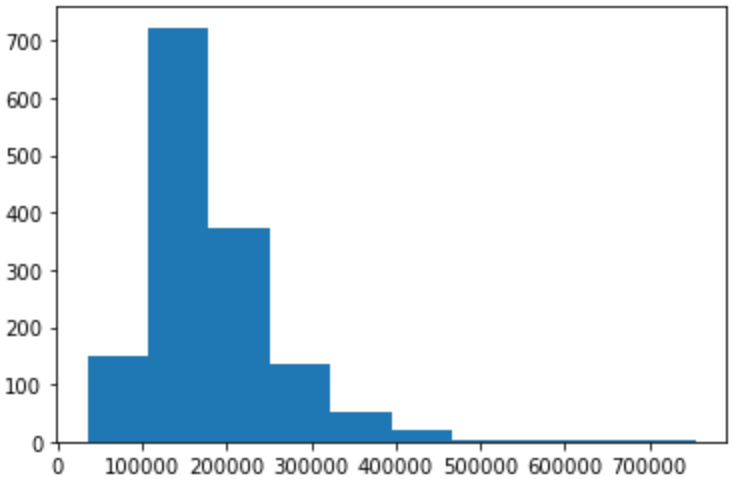


In [12]:
```
q1 = np.percentile(train_df[target], 25)
q3 = np.percentile(train_df[target], 75)
iqr = q3 - q1
max_outlier = q3+1.5*iqr
print(f'Max outlier: {max_outlier}')
num_outliers = train_df[train_df['SalePrice'] > max_outlier].shape[0]
print(f'Percentage of outliers: {100*round(num_outliers/train_df.shape[0], 2)}')
```

```
Max outlier: 340037.5
Percentage of outliers: 4.0
```

Histogram of Sale Price

In [13]:
```
plt.hist(train_df['SalePrice'])
plt.show()
```



In [14]:
```
label = 'SalePrice'
```

Correlation Matrix for all numerical variables

In [15]:
```
corr = train_df.corr()[label].sort_values()
valid_correlations = corr[(corr > 0.4) & (corr < 1)].index.tolist()
train_df[valid_correlations].corr()
```

Out[15]:

	Fireplaces	MasVnrArea	GarageYrBlt	YearRemodAdd	YearBuilt	TotRmsAbvGrd	FullB
Fireplaces	1.000000	0.249070	0.046822	0.112581	0.147716	0.326114	0.243
MasVnrArea	0.249070	1.000000	0.252691	0.179618	0.315707	0.280682	0.276
GarageYrBlt	0.046822	0.252691	1.000000	0.642277	0.825667	0.148112	0.484
YearRemodAdd	0.112581	0.179618	0.642277	1.000000	0.592855	0.191740	0.439
YearBuilt	0.147716	0.315707	0.825667	0.592855	1.000000	0.095589	0.468
TotRmsAbvGrd	0.326114	0.280682	0.148112	0.191740	0.095589	1.000000	0.554
FullBath	0.243671	0.276833	0.484557	0.439046	0.468271	0.554784	1.000
1stFlrSF	0.410531	0.344501	0.233449	0.240379	0.281986	0.409516	0.380
TotalBsmtSF	0.339519	0.363936	0.322445	0.291066	0.391452	0.285573	0.323
GarageArea	0.269141	0.373066	0.564567	0.371600	0.478954	0.337822	0.405
GarageCars	0.300789	0.364204	0.588920	0.420622	0.537850	0.362289	0.469
GrLivArea	0.461679	0.390857	0.231197	0.287389	0.199010	0.825489	0.630
OverallQual	0.396765	0.411876	0.547766	0.550684	0.572323	0.427452	0.550
LogSalePrice	0.489450	0.430809	0.541073	0.565608	0.586570	0.534422	0.594

In [16]:
```
# compartmentalize in another program
```

In [17]:
```
y_train = train_low_missing_vals_df['LogSalePrice']
train_model_data, test_model_data = data_manipulation.get_train_test_model_data(train_
                                                                                 numeri
```

```
                                                                                'SaleP

train_drop_columns = list(set(train_model_data.columns) - set(test_model_data.columns)
train_model_data_final = train_model_data.drop(train_drop_columns, axis='columns')
```

Correcting Data for Skew

Pipelines are done sequentially, column transformer runs in parallel

In [18]:
```
missing_value_pct_threshold = 30
train_drop = missing_values.get_high_missing_value_columns(train_df, missing_value_pct
test_drop = missing_values.get_high_missing_value_columns(test_df, missing_value_pct_t

train_low_missing_vals_df = train_df.drop(train_drop, axis='columns')
test_low_missing_vals_df = test_df.drop(test_drop, axis='columns')
```

In [19]:
```
ohe = OneHotEncoder(handle_unknown='ignore')
numerical_imputer = SimpleImputer(strategy='mean')
categorical_imputer = SimpleImputer(strategy='most_frequent')
ordinal_imputer = SimpleImputer(strategy='most_frequent')
```

In [20]:
```
categorical_variables, discrete_numerical_variables,\
continuous_numerical_variables = data_manipulation.get_variables(train_low_missing_val
discrete_numerical_variables = list(set(discrete_numerical_variables) - set(['PoolArea
ind_continuous_variables = list(set(continuous_numerical_variables) - set(['SalePrice'
numerical_variables = continuous_numerical_variables + discrete_numerical_variables
```

In [21]:
```
numerical_columns_selector = selector(dtype_exclude=object)

numerical_columns = numerical_columns_selector(train_low_missing_vals_df)
numerical_columns = list(set(numerical_columns) - set(['SalePrice', 'LogSalePrice', 'I
skewed_feats = train_low_missing_vals_df[numerical_columns].apply(lambda x: skew(x.dro
skewed_feats = skewed_feats[skewed_feats > 0.75].index
```

In [22]:
```
skewed_continuous_variables = list(set(skewed_feats).intersection(set(ind_continuous_v
cont_variables = list(set(ind_continuous_variables) - set(skewed_continuous_variables)
```

In [23]:
```
skewed_discrete_variables = list(set(skewed_feats).intersection(set(discrete_numerical
discrete_variables2 = list(set(discrete_numerical_variables) - set(skewed_discrete_var
```

In [24]:
```
categorical_pipeline = make_pipeline(categorical_imputer, ohe)
skewed_numerical_pipeline = make_pipeline(log_transformer, numerical_imputer)
skewed_discrete_pipeline = make_pipeline(log_transformer, categorical_imputer)
```

In [25]:
```
preprocessor2 = make_column_transformer((categorical_pipeline, categorical_variables),
                                        (skewed_numerical_pipeline, skewed_continuous_v
                                        (skewed_discrete_pipeline, skewed_discrete_vari
                                        (numerical_imputer, cont_variables),
                                        (categorical_imputer, discrete_variables2),
                                       )
```

In [26]:
```
lr_pipe = make_pipeline(preprocessor2, LinearRegression())
lr_score = -cross_val_score(lr_pipe, train_low_missing_vals_df, y_train, cv=5, scoring
lr_score
```

Out[26]:
```
0.14061181475376594
```

Why is it better to cross validate a pipeline, rather than just the preprocessed data and the model, because it doesnt simulate reality,

You can determine things like which imputation technique is better, data transformation technique is better because that is now part of the pipeline

Data Preprocessing Step

Now do entire modeling step all over again

In [27]:
```
## Modeling Phase 2 <a id='modeling2'></a>
```

Decision Tree Regressor

In [28]:
```
dt_pipe = make_pipeline(preprocessor2, DecisionTreeRegressor())
```

In [29]:
```
%%time
dt_pipe = make_pipeline(preprocessor2, DecisionTreeRegressor())

dt_param_grid = {
    'decisiontreeregressor__max_depth':[2,3],
    'decisiontreeregressor__min_samples_split':[50, 30],
    'decisiontreeregressor__min_samples_leaf':[50, 30],
    'decisiontreeregressor__max_features' : [30, 40],
    'decisiontreeregressor__min_impurity_decrease': [i/100 for i in range(1, 3)],
    'decisiontreeregressor__ccp_alpha': [i/10 for i in range(1, 3)]
}

dt_random_search = RandomizedSearchCV(dt_pipe, dt_param_grid, scoring='neg_root_mean_s
                                      n_iter=MAX_MODELS, verbose=0)

dt_random_search = dt_random_search.fit(train_low_missing_vals_df, y_train)

dt_best_params = { k.split('__')[1]:v for (k,v) in zip(list(dt_random_search.best_para
dt_final_pipe = make_pipeline(preprocessor2, DecisionTreeRegressor(**dt_best_params))
dt_score = -cross_val_score(dt_final_pipe, train_low_missing_vals_df, y_train, cv=5, s
dt_score
```

Out[29]:
```
CPU times: total: 14.8 s
Wall time: 29.9 s
0.3992256860383698
```

In [30]:
```
%%time

xgb_param_grid = {
    'xgbregressor__n_estimators': [30, 40],
    'xgbregressor__max_depth':[2,3],
    'xgbregressor__max_leaves':[20, 30],
    'xgbregressor__reg_alpha': [i/100 for i in range(1,3)],
    'xgbregressor__reg_lambda': [i/100 for i in range(1,3)],
    'xgbregressor__colsample_bytree': [i/10 for i in range(1, 3)],
    'xgbregressor__min_child_weight': [10*i for i in range(1,3)],
```

```
    'xgbregressor__learning_rate': [i/100 for i in range(1,3)]
}

xgb_pipe = make_pipeline(preprocessor2, XGBRegressor())


xgb_random_search = RandomizedSearchCV(xgb_pipe, xgb_param_grid, scoring='neg_root_mea
                                n_iter=MAX_MODELS, verbose=0)

xgb_random_search = xgb_random_search.fit(train_low_missing_vals_df, y_train)

xgb_best_params = { k.split('__')[1]:v for (k,v) in zip(list(xgb_random_search.best_pa

xgb_final_pipe = make_pipeline(preprocessor2, XGBRegressor(**xgb_best_params))

xgb_score = -cross_val_score(xgb_final_pipe, train_low_missing_vals_df, y_train, cv=5,
xgb_score
```

```
CPU times: total: 6min 33s
Wall time: 1min 7s
```

Out[30]:
```
5.142034522057028
```

Changed max leaves 3-5, LR: 0.03-0.05

In [31]:
```
%%time

xgb_param_grid = {
    'xgbregressor__n_estimators': [110, 120],
    'xgbregressor__max_depth':[5,6],
    'xgbregressor__max_leaves':[40, 30, 50],
    'xgbregressor__reg_alpha': [i/100 for i in range(1,3)],
    'xgbregressor__reg_lambda': [i/100 for i in range(1,3)],
    'xgbregressor__colsample_bytree': [i/10 for i in range(2, 5)],
    'xgbregressor__learning_rate': [i/100 for i in range(3,6)],
    'xgbregressor__gamma': [i/100 for i in range(1, 3)]
}

xgb_pipe = make_pipeline(preprocessor2, XGBRegressor())


xgb_random_search = RandomizedSearchCV(xgb_pipe, xgb_param_grid, scoring='neg_root_mea
                                n_iter=MAX_MODELS, verbose=0)

xgb_random_search = xgb_random_search.fit(train_low_missing_vals_df, y_train)

xgb_best_params = { k.split('__')[1]:v for (k,v) in zip(list(xgb_random_search.best_pa

xgb_final_pipe = make_pipeline(preprocessor2, XGBRegressor(**xgb_best_params))

xgb_score = -cross_val_score(xgb_final_pipe, train_low_missing_vals_df, y_train, cv=5,
xgb_score
```

```
CPU times: total: 15min 26s
Wall time: 6min 10s
```

Out[31]:
```
0.1260244258022003
```

Lasso Regression

In [32]:
```
scaled_train_data = StandardScaler().fit_transform(train_model_data_final)
```

In [33]:
```
lasso_param_grid = {
    'lasso__alpha':[0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 75, 100, 200, 300, 400, 500, 6
}

ohe2 = OneHotEncoder(handle_unknown='ignore', sparse=False)
categorical_pipeline2 = make_pipeline(categorical_imputer, ohe2)


preprocessor3 = make_column_transformer((categorical_pipeline2, categorical_variables)
                                        (skewed_numerical_pipeline, skewed_continuous_v
                                        (skewed_discrete_pipeline, skewed_discrete_vari
                                        (numerical_imputer, cont_variables),
                                        (categorical_imputer, discrete_variables2),
                                    )

lasso_pipe = make_pipeline(preprocessor3, StandardScaler(), Lasso())


lasso_random_search = RandomizedSearchCV(lasso_pipe, lasso_param_grid, scoring='neg_rc
                                n_iter=MAX_MODELS, verbose=0)

lasso_random_search = lasso_random_search.fit(train_low_missing_vals_df, y_train)

lasso_best_params = { k.split('__')[1]:v for (k,v) in zip(list(lasso_random_search.bes

lasso_final_pipe = make_pipeline(preprocessor3, StandardScaler(), Lasso(**lasso_best_p

lasso_score = -cross_val_score(lasso_final_pipe, train_low_missing_vals_df, y_train, c
lasso_score
```

```
C:\Users\Nikhil\anaconda3\lib\site-packages\sklearn\model_selection\_search.py:306: U
serWarning: The total space of parameters 19 is smaller than n_iter=50. Running 19 it
erations. For exhaustive searches, use GridSearchCV.
  warnings.warn(
```

Out[33]:
```
0.12843928473790503
```

Ridge Regression

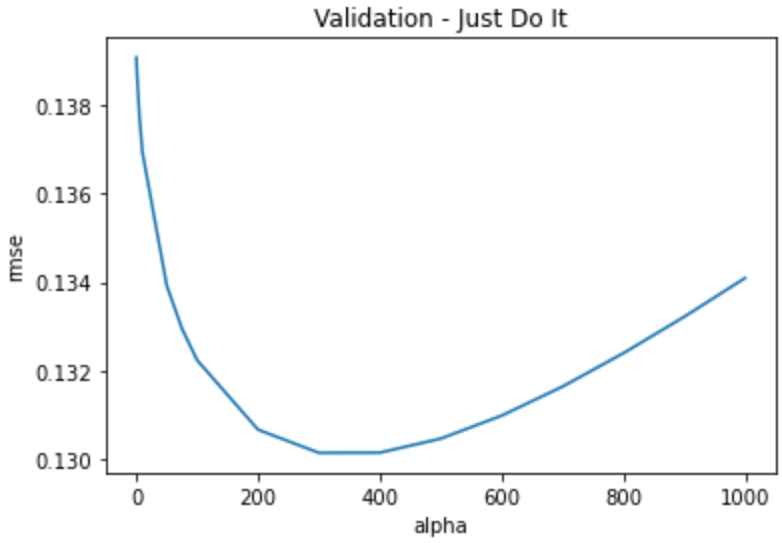
In [34]:
```
ridge_param_grid = {
    'ridge__alpha':[0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 75, 100, 200, 300, 400, 500, 6
}

ohe2 = OneHotEncoder(handle_unknown='ignore', sparse=False)
categorical_pipeline2 = make_pipeline(categorical_imputer, ohe2)


preprocessor3 = make_column_transformer((categorical_pipeline2, categorical_variables)
                                        (skewed_numerical_pipeline, skewed_continuous_v
                                        (skewed_discrete_pipeline, skewed_discrete_vari
                                        (numerical_imputer, cont_variables),
                                        (categorical_imputer, discrete_variables2),
                                    )

ridge_pipe = make_pipeline(preprocessor3, StandardScaler(), Ridge())
```

```
ridge_random_search = RandomizedSearchCV(ridge_pipe, ridge_param_grid, scoring='neg_rc
                                n_iter=MAX_MODELS, verbose=0)

ridge_random_search = ridge_random_search.fit(train_low_missing_vals_df, y_train)

ridge_best_params = { k.split('__')[1]:v for (k,v) in zip(list(ridge_random_search.bes

ridge_final_pipe = make_pipeline(preprocessor3, StandardScaler(), Lasso(**ridge_best_p

ridge_score = -cross_val_score(ridge_final_pipe, train_low_missing_vals_df, y_train, c
ridge_score, ridge_best_params
```

```
C:\Users\Nikhil\anaconda3\lib\site-packages\sklearn\model_selection\_search.py:306: U
serWarning: The total space of parameters 19 is smaller than n_iter=50. Running 19 it
erations. For exhaustive searches, use GridSearchCV.
  warnings.warn(
```

Out[34]:
```
(0.39922568603836983, {'alpha': 300})
```

In [35]:
```
alphas = [0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 75, 100, 200, 300, 400, 500, 600, 700, 8
ohe2 = OneHotEncoder(handle_unknown='ignore', sparse=False)
categorical_pipeline2 = make_pipeline(categorical_imputer, ohe2)
preprocessor3 = make_column_transformer((categorical_pipeline2, categorical_variables)
                                        (skewed_numerical_pipeline, skewed_continuous_v
                                        (skewed_discrete_pipeline, skewed_discrete_vari
                                        (numerical_imputer, cont_variables),
                                        (categorical_imputer, discrete_variables2),
                                       )
cv_ridge = [-cross_val_score(make_pipeline(preprocessor3, StandardScaler(), Ridge(alph
            for alpha in alphas]
cv_ridge = pd.Series(cv_ridge, index = alphas)
cv_ridge.plot(title = "Validation - Just Do It")
plt.xlabel("alpha")
plt.ylabel("rmse")
plt.show()
```



In [36]:
```
(cv_ridge == cv_ridge.min())
```

Out[36]:
```
0.01       False
0.05       False
0.10       False
0.50       False
1.00       False
5.00       False
10.00      False
50.00      False
75.00      False
100.00     False
200.00     False
300.00      True
400.00     False
500.00     False
600.00     False
700.00     False
800.00     False
900.00     False
1000.00    False
dtype: bool
```

Elastic Net Regressor

In [37]:
```
elastic_net_param_grid = {
    'elasticnet__alpha': [0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 75, 100, 200, 300, 400,
    'elasticnet__l1_ratio':[0.01, 0.05, 0.1, 0.5, 1]
}

ohe2 = OneHotEncoder(handle_unknown='ignore', sparse=False)
categorical_pipeline2 = make_pipeline(categorical_imputer, ohe2)


preprocessor3 = make_column_transformer((categorical_pipeline2, categorical_variables)
                                        (skewed_numerical_pipeline, skewed_continuous_v
                                        (skewed_discrete_pipeline, skewed_discrete_vari
                                        (numerical_imputer, cont_variables),
                                        (categorical_imputer, discrete_variables2),
                                    )

elastic_net_pipe = make_pipeline(preprocessor3, StandardScaler(), ElasticNet())


elastic_net_random_search = RandomizedSearchCV(elastic_net_pipe, elastic_net_param_gri
                                n_iter=MAX_MODELS, verbose=0)

elastic_net_random_search = elastic_net_random_search.fit(train_low_missing_vals_df, y

elastic_net_best_params = { k.split('__')[1]:v for (k,v) in zip(list(elastic_net_randc

elastic_net_final_pipe = make_pipeline(preprocessor3, StandardScaler(), ElasticNet(**e

elastic_net_score = -cross_val_score(elastic_net_final_pipe, train_low_missing_vals_df
elastic_net_score, elastic_net_best_params
```

Out[37]:
```
(0.12395987205101548, {'l1_ratio': 0.1, 'alpha': 0.05})
```

KNN Regressor

In [38]:
```
%%time

knn_param_grid = {
    'kneighborsregressor__weights' : ['uniform'],
    'kneighborsregressor__algorithm' : ['auto'],
    'kneighborsregressor__n_neighbors' : [5,10,15, 20, 25, 30],
    'kneighborsregressor__leaf_size': [10],
    'kneighborsregressor__p': [1],
}


knn_pipe = make_pipeline(preprocessor2, KNeighborsRegressor())


knn_random_search = RandomizedSearchCV(knn_pipe, knn_param_grid, scoring='neg_root_mea
                                n_iter=MAX_MODELS, verbose=0)

knn_random_search = knn_random_search.fit(train_low_missing_vals_df, y_train)

knn_best_params = { k.split('__')[1]:v for (k,v) in zip(list(knn_random_search.best_pa

knn_final_pipe = make_pipeline(preprocessor2, KNeighborsRegressor(**knn_best_params))

knn_score = -cross_val_score(knn_final_pipe, train_low_missing_vals_df, y_train, cv=5,
knn_score, knn_best_params
```

```
C:\Users\Nikhil\anaconda3\lib\site-packages\sklearn\model_selection\_search.py:306: U
serWarning: The total space of parameters 6 is smaller than n_iter=50. Running 6 iter
ations. For exhaustive searches, use GridSearchCV.
  warnings.warn(
CPU times: total: 25.4 s
Wall time: 7.76 s
```

Out[38]:
```
(0.2257985937954227,
 {'weights': 'uniform',
  'p': 1,
  'n_neighbors': 10,
  'leaf_size': 10,
  'algorithm': 'auto'})
```

Random Forest Stacking Regressor

In [39]:
```
%%time
estimators = [
 ('KNN', knn_final_pipe),
    ('Lasso', lasso_final_pipe),
    ('Ridge', ridge_final_pipe),
    ('Elastic', elastic_net_final_pipe),
    ('XGBRegressor', xgb_final_pipe)
]


final_estimator=RandomForestRegressor()

rf_stacking_regressor = StackingRegressor(
        estimators=estimators,
        final_estimator=final_estimator,
    )
```

```
-cross_val_score(rf_stacking_regressor, train_low_missing_vals_df, y_train, cv=5, scor
```

```
CPU times: total: 2min 20s
Wall time: 38.2 s
```

Out[39]:
```
0.12759110990383354
```