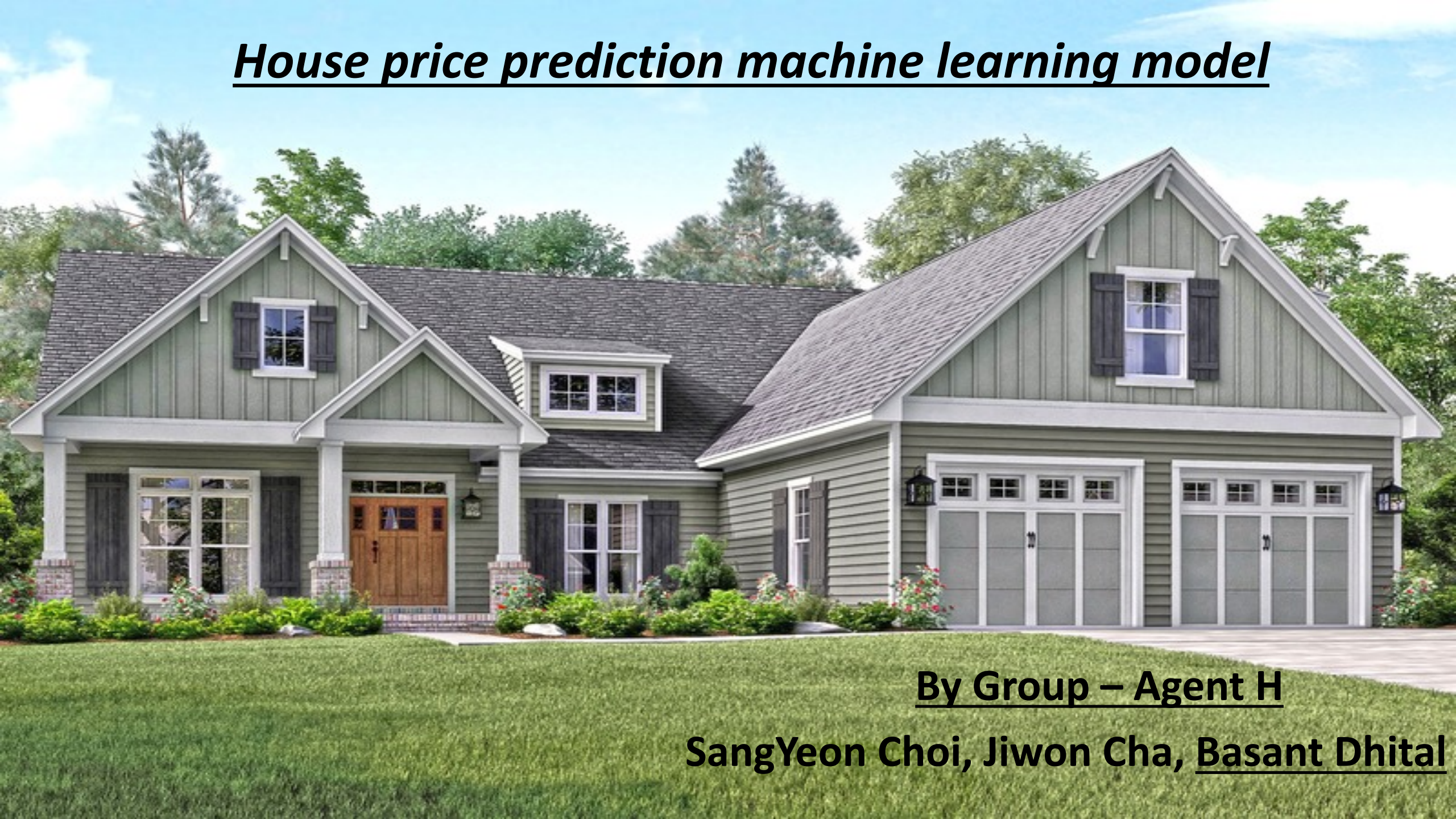
House price prediction machine learning model



By Group – Agent H

SangYeon Choi, Jiwon Cha, Basant Dhital

Project outline:

- Why this project?
- Project structure
- EDA / Feature Engineering
- How we built our House model ?
- Hyperparameter tuning
- Conclusion and future direction

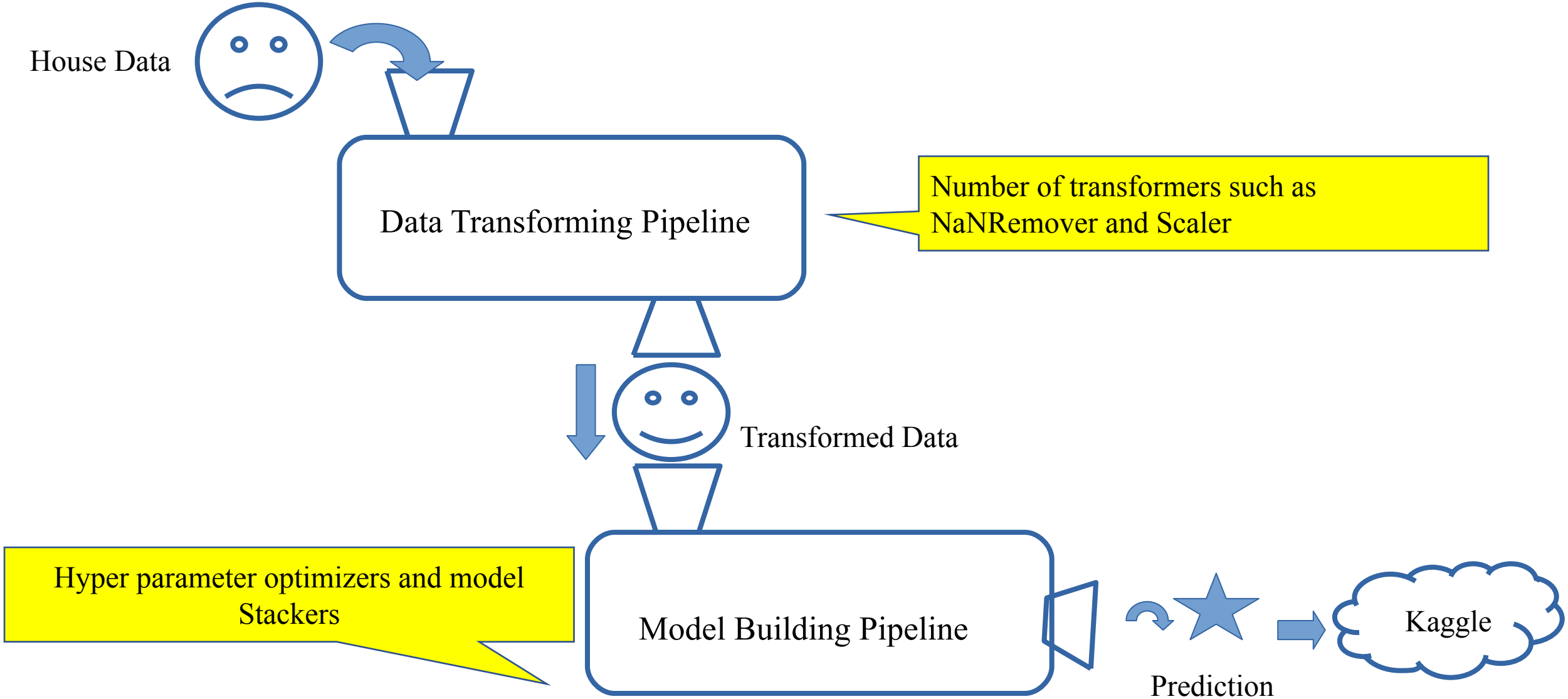
Why this project?

- Buying a house is lifetime investment. (House price according to features)
- The real world problem are put in Kaggle, Zigbang, etc. to get solution using machine learning. These skills are in high demand in the market
- The main goal of this project was to predict house price using machine learning algorithm. Our goal was to develop automated process for this

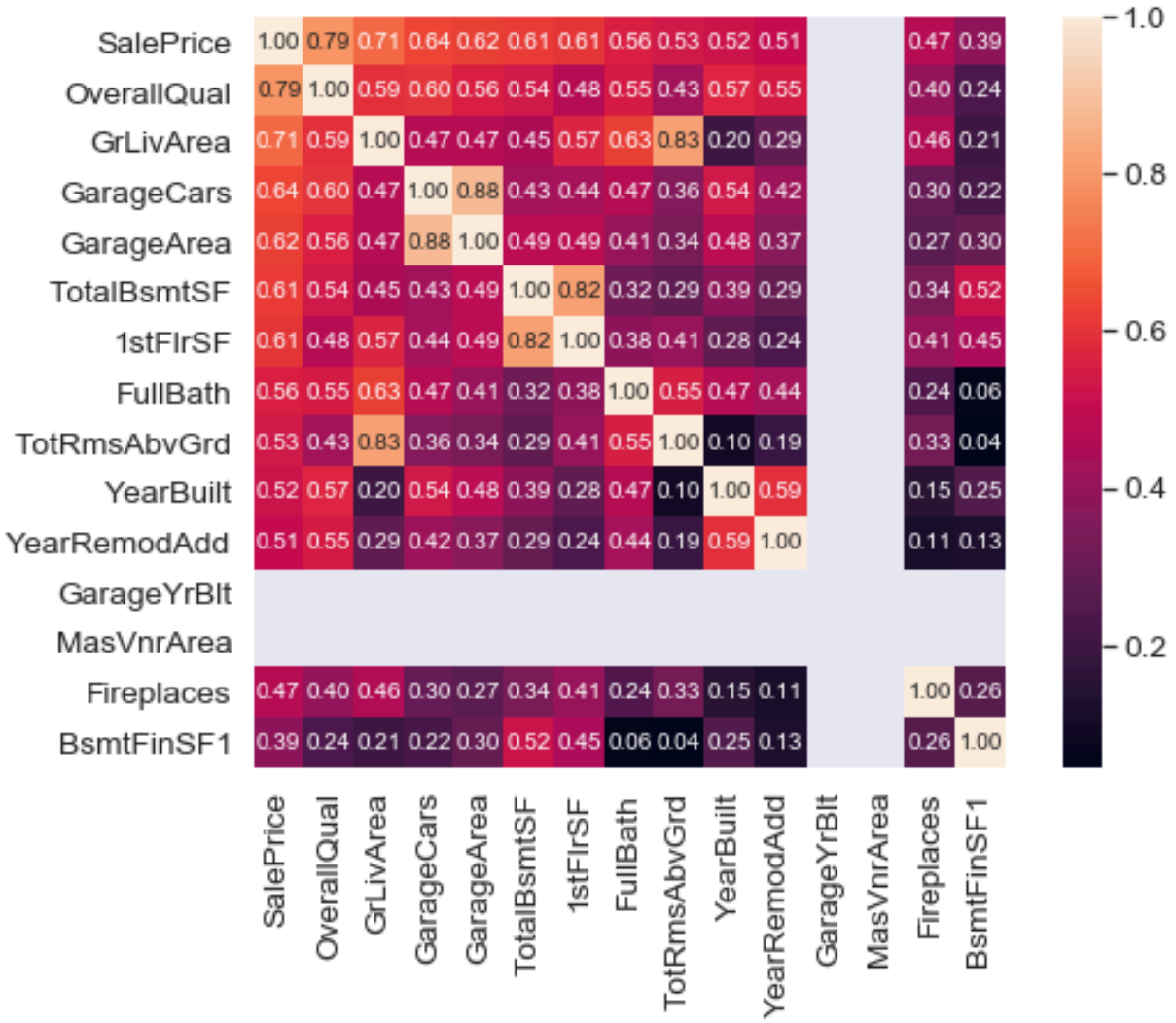
Project structure :

Simple but efficient co-op environment

Pipelines for Fast iteration:

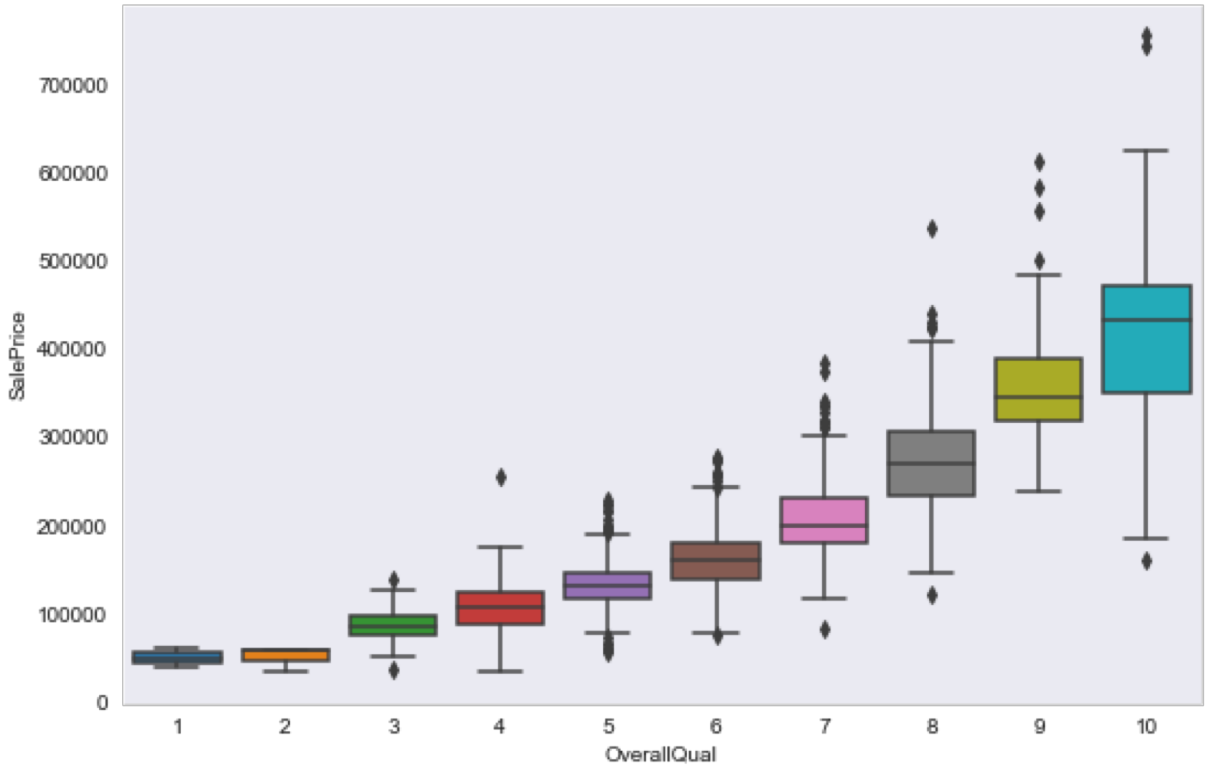


Some EDA visualization:

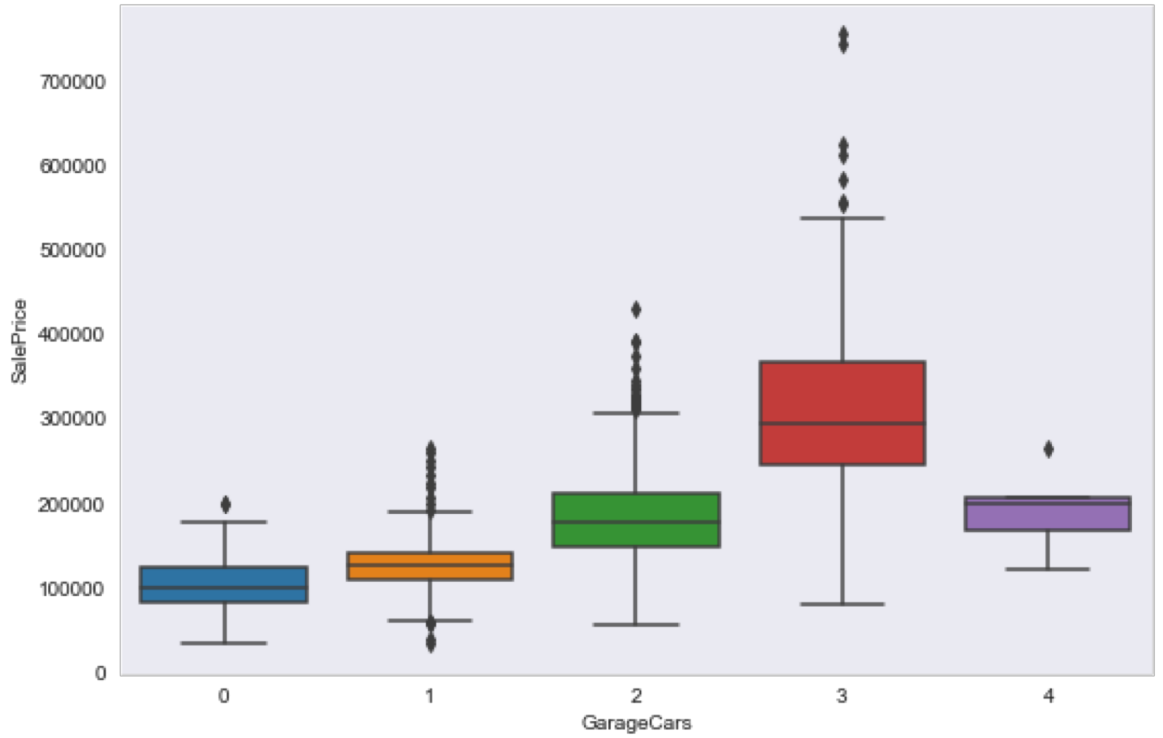


correlation between various features

contd……



Box plot of overallQual vs salesprice



Box plot of salesprice vs GarageCars

Automatic data transformer for outliers:

The Z-score of an observation is defined as

$$Z_i = \frac{Y_i - \bar{Y}}{s}$$

with $\bar{Y}$ and s denoting the sample mean and sample standard deviation, respectively. In other words, data is given in units of how many standard deviations it is from the mean.

Z score

```
z = np.abs(stats.zscore(tmpDf[numeric_columns]))
row, col = np.where(z > 5)

rows_count = df.groupby(['row']).count()
rows_count[rows_count.col>2]
```

Created effective strategy

```
    ('OutlierRemover', OutlierRemover()),
    ('NaNRemover', NaNRemover()),
    ('AdditionalFeatureGenerator', AdditionalF
    ('TypeTransformer', TypeTransformer()),
    ('ErrorImputer', ErrorImputer()),
    ('SkewFixer', SkewFixer()),
    ('Scaler', Scaler()),
    ('FeatureDropper', FeatureDropper()),
    ('Dummyfier', Dummyfier()),
])
```

made a pipeline element

Data Transformations:

Used data_descriptions.txt and common sense to impute NAN data

	GarageFinish	GarageQual	GarageCond	GarageYrBlt	GarageType	GarageCars	GarageArea	OverallCond	YearBuilt
666	NaN	NaN	NaN	NaN	Detchd	1.0	360.0	8	1910
1116	NaN	NaN	NaN	NaN	Detchd	NaN	NaN	6	19

Imputed NaN for garage related data by assuming that have similar value with those have same overallcond

	col	sum
72	PoolQC	2909
74	MiscFeature	2814
6	Alley	2721
73	Fence	2348
80	SalePrice	1459
57	FireplaceQu	1420
3	LotFrontage	486
60	GarageFinish	159
63	GarageQual	159
64	GarageCond	159
59	GarageYrBlt	159
58	GarageType	157
32	BsmtExposure	82
31	BsmtCond	82
30	BsmtQual	81
35	BsmtFinType2	80
33	BsmtFinType1	79
25	MasVnrType	24
26	MasVnrArea	23
2	MSZoning	4
55	Functional	2
48	BsmtHalfBath	2

List of NAN containing columns

31 out of 35 NAN containing columns are taken care of by manual NAN Imputer

This made big difference in kaggle score

How we built our House model ?

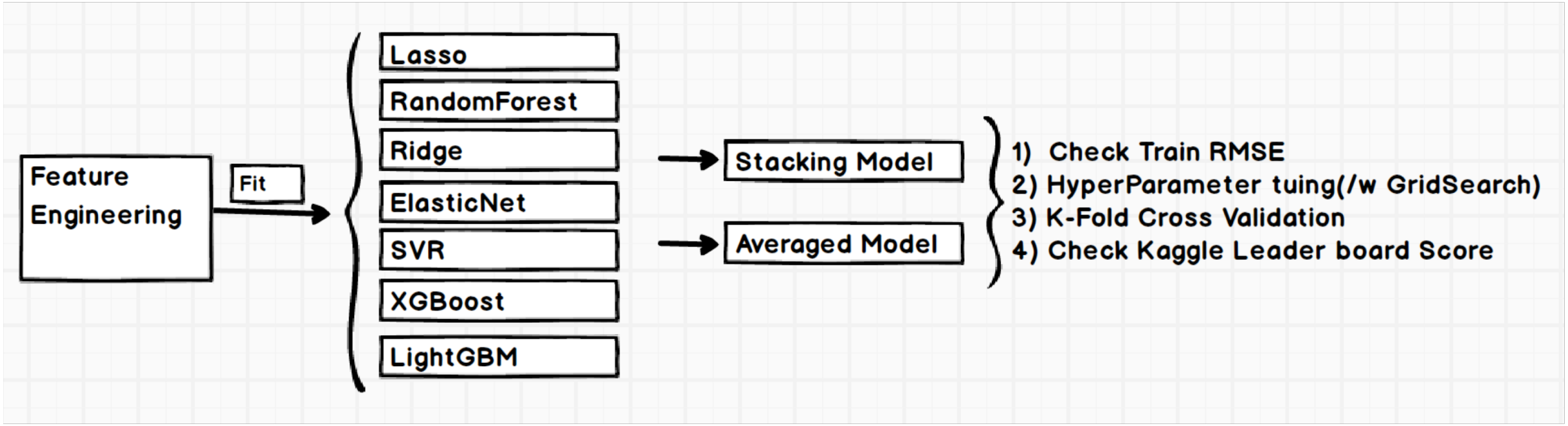
1. Start with Lasso (Test / Feature Selection)
    - Fast and Good result
    - A few Parameter

2. Add Supervised Learning ML
    - Ridge / RandomForest / Boost ..
    - Comparison of results between various algorithms

3. Make Stacking / Average Model
    - To reduce residual, Make a Integrated model

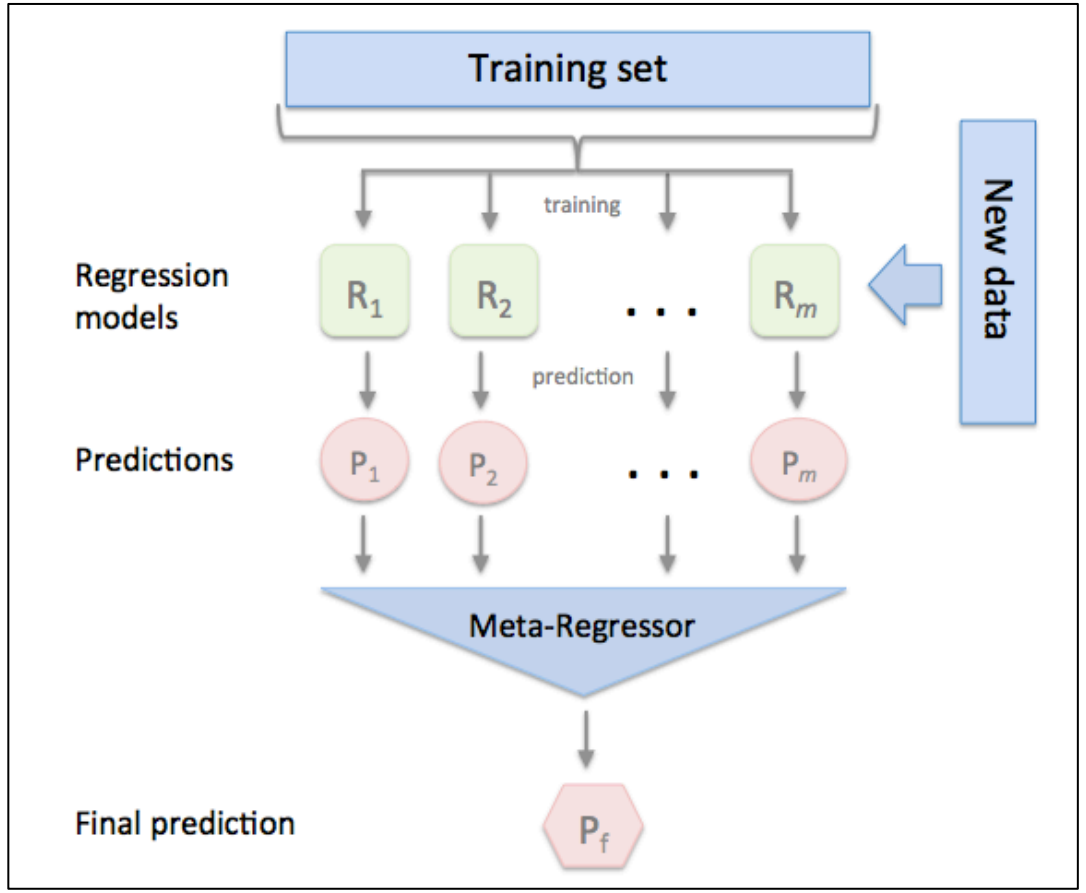
It was possible to test all these models using pipelines that we built

Final house model:



- Workflow for Model selection
  - Used Various Algorithm
  - GridSearch for finding Hyperparameter
  - Made a Model (Stacking / Averaged)

Stacking Regression: Two heads are better than one

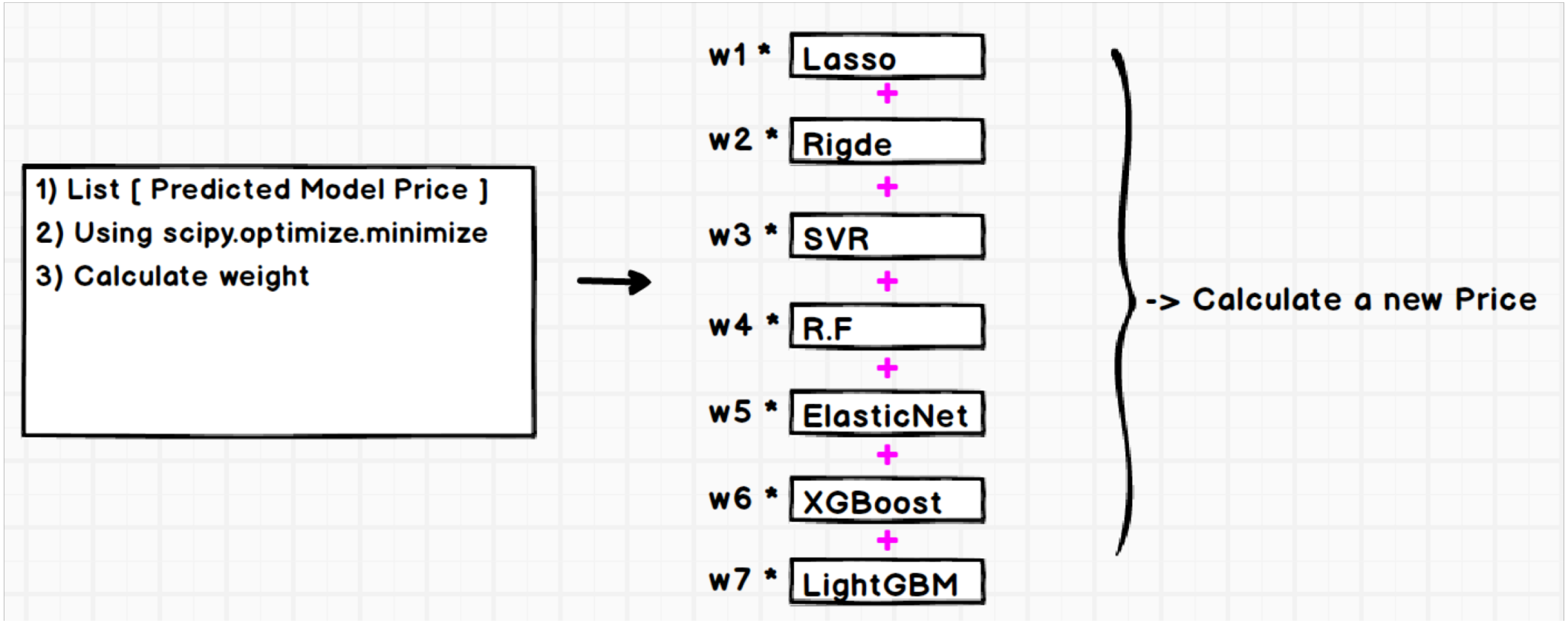


<http://rasbt.github.io/mlxtend/user_guide/regressor/StackingRegressor/>

- Stacking is a method that combines predictions of several different models
- Various ML Algorithms can be mixed
- Meta Regressor
  - An algorithm that combines different models

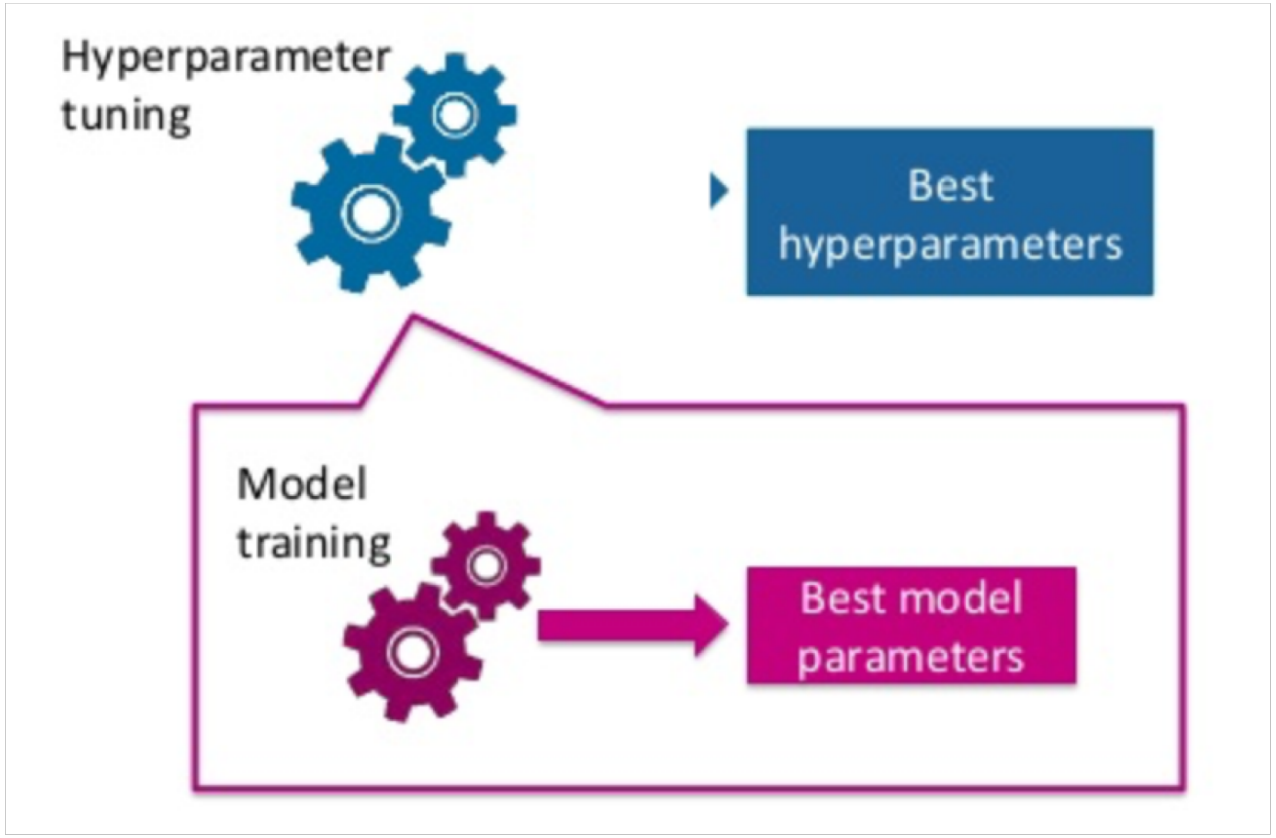
```
model = StackingRegressor(
    regressors=[rf, elnet, lso, rdg, svr,gbm,lbm],
    meta_regressor=Lasso(alpha=0.0005)
)
```

Averaged Model:



Tuning Hyper Parameter:

To make a better model is a very important process



- We can adopt three different ways:
  - GridSearch
  - RandomSearch
  - Bayesian Optimization

- And we choose GridSearch. Why?
  a. Many references available
  b. Easier to use and small data size

Used K-fold Cross validation = 5:

```
def get_best_estimator(train_data, y_train_values, estimator=None,
    name = estimator.__class__.__name__
    pipeline = Pipeline(steps=[
        (name, estimator),
    ])

    params = [
        {
            name+"__"+k: v for k, v in params.items()
        }
    ]
    from sklearn.model_selection import cross_val_score, KFold
    scorer = make_scorer(mean_squared_error, greater_is_better=Fals
    kf = KFold(cv, shuffle=True, random_state=42).get_n_splits(trai
    grid_search = GridSearchCV(pipeline, param_grid=params, scoring
    grid_search.fit(train_data, y_train_values)

    print("Estimator: {} score: ({}) best params: {}".format(name,
    print(grid_search.best_estimator_)
    return grid_search.best_estimator_
```

```
rf_param = {
    'max_depth': [3, 4, 5],
    'min_samples_leaf': [3, 4, 5],
    'n_estimators': [5, 7, 10]
}
ls_param = {'alpha': [0.0003, 0.0004, 0.0005,
                      0.0006, 0.0007, 0.0008],
            'max_iter': [10000], "normalize": [False]}

elnet_param = {'alpha': [0.0003, 0.0004, 0.0005],
               'l1_ratio': [0.9, 0.95, 0.99, 1],
               'max_iter': [10000]}

ridge_param = {'alpha': [10, 10.1, 10.2, 10.3, 10.4, 10.5]}

svr_param = {'gamma': [1e-08, 1e-09],
             'C': [100000, 110000],
             'epsilon': [1, 0.1, 0.01]
             }
```

Find best parameter:

```
Estimator: ElasticNet score: (0.109056891298853322) best params: {'ElasticNet__alpha': 0.0005,
 'ElasticNet__l1_ratio': 1, 'ElasticNet__max_iter': 10000}
Pipeline(memory=None,
     steps=[('ElasticNet', ElasticNet(alpha=0.0005, copy_X=True, fit_intercept=True, l1_ratio=1,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False))])
Fitting 5 folds for each of 6 candidates, totalling 30 fits
[Parallel(n_jobs=4)]: Done  30 out of  30 | elapsed:    0.4s finished
Estimator: Lasso score: (0.109056891298853322) best params: {'Lasso__alpha': 0.0005, 'Lasso__max_iter':
 10000, 'Lasso__normalize': False}
Pipeline(memory=None,
     steps=[('Lasso', Lasso(alpha=0.0005, copy_X=True, fit_intercept=True, max_iter=10000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False))])
Fitting 5 folds for each of 6 candidates, totalling 30 fits
```

Result:

Simple model

Model	Train RMSE	Kaggle Score
ElasticNet	0.10879657	0.11684
Lasso	0.10869445	0.11691
Ridge	0.11043459	0.11562
SVR	0.11368618	0.12019
Randomforest	0.15893652	0.17354
Xgboost*	0.12115257	0.12965
lightGBM*	0.13731836	0.14344

In our Case,

1. XG / light was not a good score.
   - Boost Algorithm
   - Need to tune the parameter

2. Better linear than tree
   - Example(lasso > R.F)

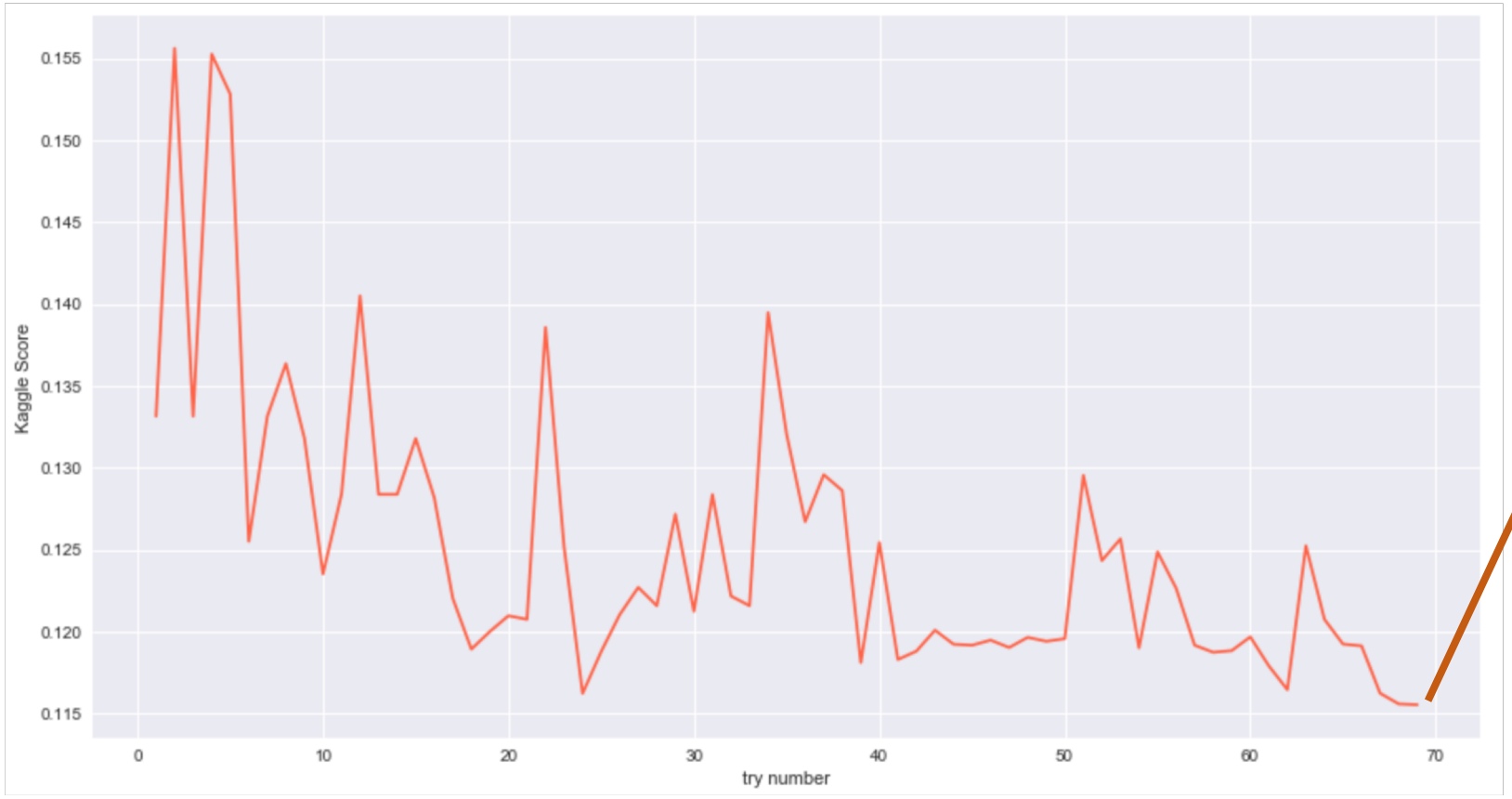
House model

Model	Train RMSE	Kaggle Score
StackingRegressor	0.11069786	**0.11554**
AveragingRegressor	0.12458452	0.12066

- StackingRegressor is better than AveragingRegressor
- *did not use in our house model

Kaggle result:

Top 8% (11/17/2018)



Final Kaggle Score :
0.11554

Conclusion:

- All of the parameters are important
  a. Skewness , Z score and many variables to select
  b. Tuning model hyperparameter

- Feature engineering was most important to predict house price
  a. FillNa (0 , Mode , median)
  b. binning
  c. Using domain knowledge

- GBM/Xgboost powerful to give good benchmark solutions but not always best choice

Future directions:

- Tune parameter for Xgboost, lightGBM model in future and predict better accuracy
- Apply clustering analysis to create new features
- Investigate more feature engineering
- It would be nice to get time series event data and study the effects of 2008 recession on house sale price and predict its effect in case of recession in future

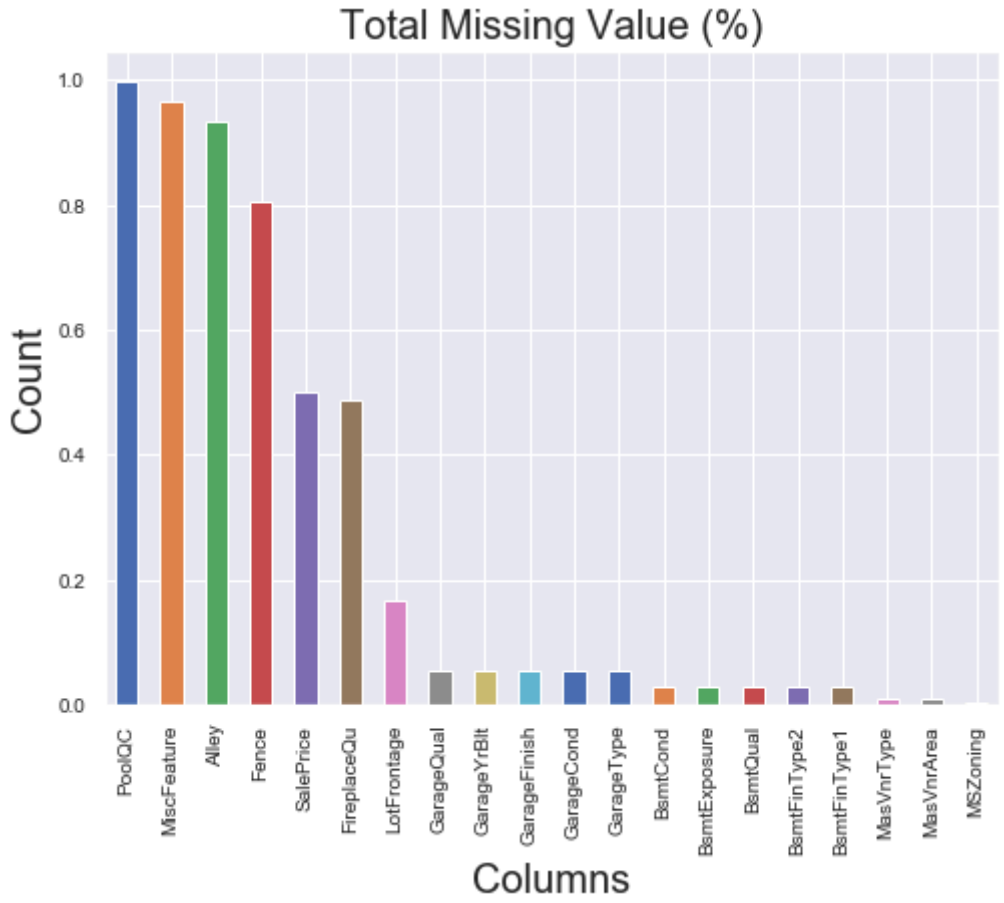
Thank you
very much!!!!

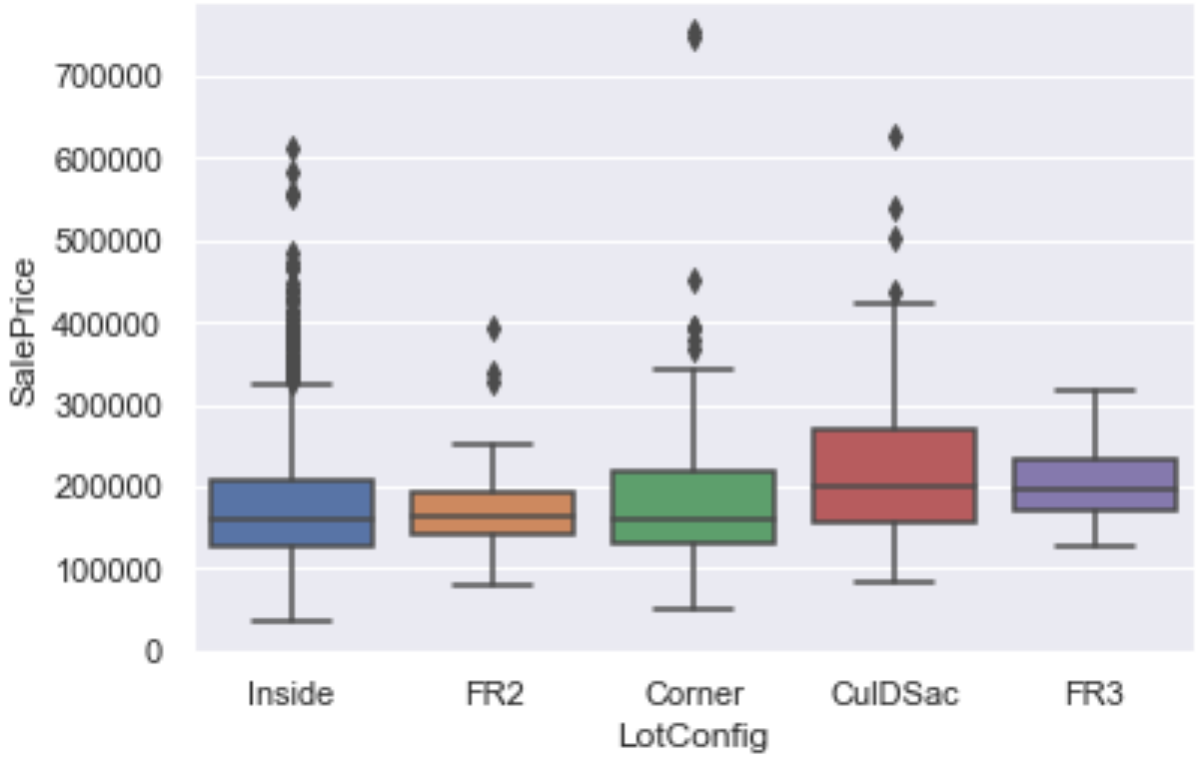
Project Result:

Achieved Rank 500 by moving fast, failing fast

EDA contd…

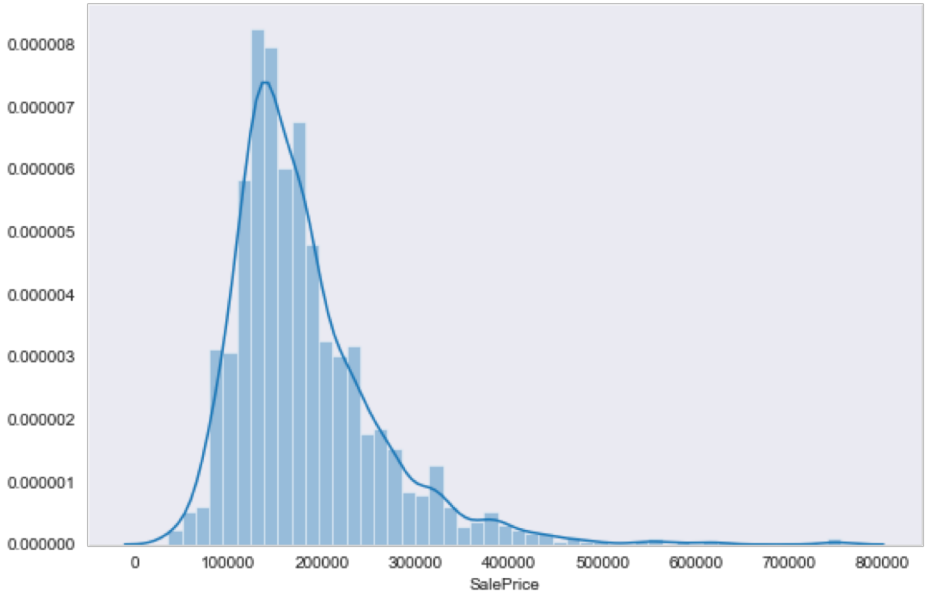
Fixing NA's and outliers



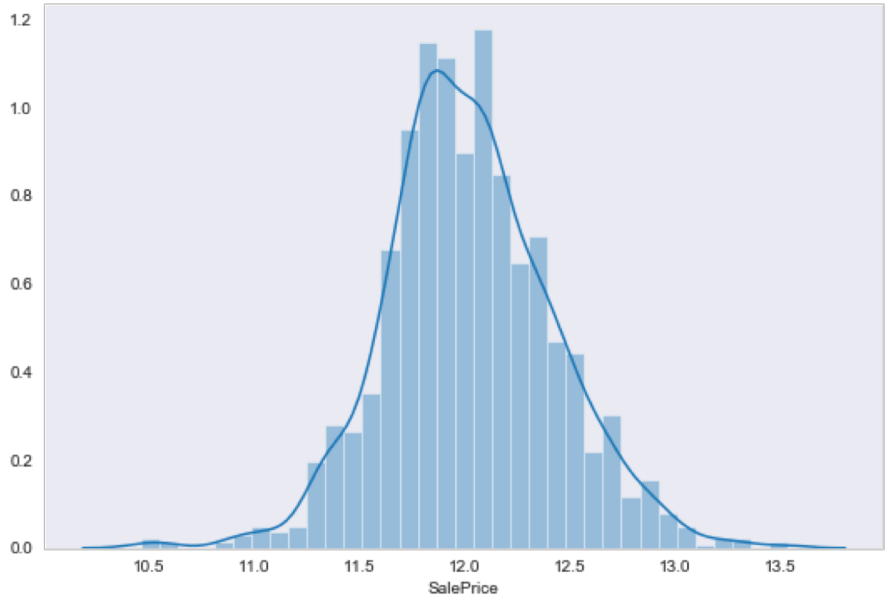


EDA processing:

Checking normality of the data



Initial distribution of data



Distribution of data after log transformation

Data Transformations:

But most of null imputation is done manually
since Auto imputer performed poorly

```
transform_pipeline = Pipeline(steps=[
    ('OutlierRemover', OutlierRemover()),
    ('NaNImputer', NaNImputer()),
    ('NaNRemover', NaNRemover()),
    ('AdditionalFeatureGenerator', AdditionalF
    ('TypeTransformer', TypeTransformer()),
    ('ErrorImputer', ErrorImputer()),
    ('SkewFixer', SkewFixer()),
    ('Scaler', Scaler()),
    ('FeatureDropper', FeatureDropper()),
    ('Dummyfier', Dummyfier()),
])
```