회귀분석을 활용한 주택가격 예측

목차

01
데이터 전처리

- 왜도
- 이상치 제거
- 결측값 처리

02
교차검증 및 모델셋업

- 교차 검증
- 모델 셋업

03
스태킹 및 모델 평가

- Stacking
- Score Function
- Evaluation
- Fit Function

04
블렌딩 및 최종 제출

- 앙상블 기법
- 블렌딩
- RMSE값 비교
- 최종 제출

Intro.

Evaluation

Goal

It is your job to predict the sales price for each house. For each Id in the test set, you must predict the value of the SalePrice variable.

Metric

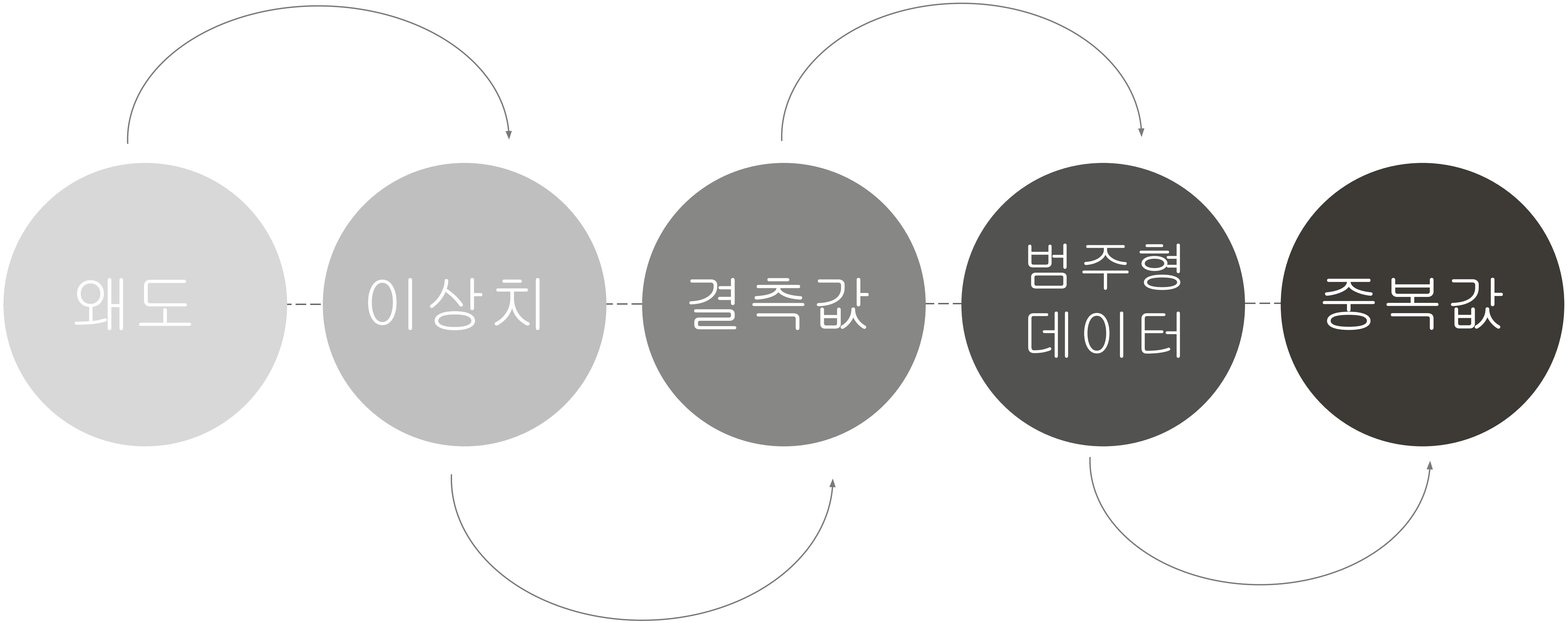
Submissions are evaluated on Root-Mean-Squared-Error (RMSE) between the logarithm of the predicted value and the logarithm of the observed sales price. (Taking logs means that errors in predicting expensive houses and cheap houses will affect the result equally.)

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

$$\mathrm{RMSE} = \sqrt{MSE} = \sqrt{\frac{\Sigma(\hat{y}-y)^2}{n}}$$

Part 1 데이터 전처리

데이터 전처리 과정



시작 전

```python
# Essentials

import numpy as np
import pandas as pd
import datetime
import random #파이썬 랜덤함수

# Plots

import seaborn as sns
import matplotlib.pyplot as plt

# Models

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor, BaggingRegressor
from sklearn.kernel_ridge import KernelRidge #커널 릿지 회귀 분석
from sklearn.linear_model import Ridge, RidgeCV
from sklearn.linear_model import ElasticNet, ElasticNetCV #엘라스틱넷
from sklearn.svm import SVR
from mlxtend.regressor import StackingCVRegressor
import lightgbm as lgb
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor
```

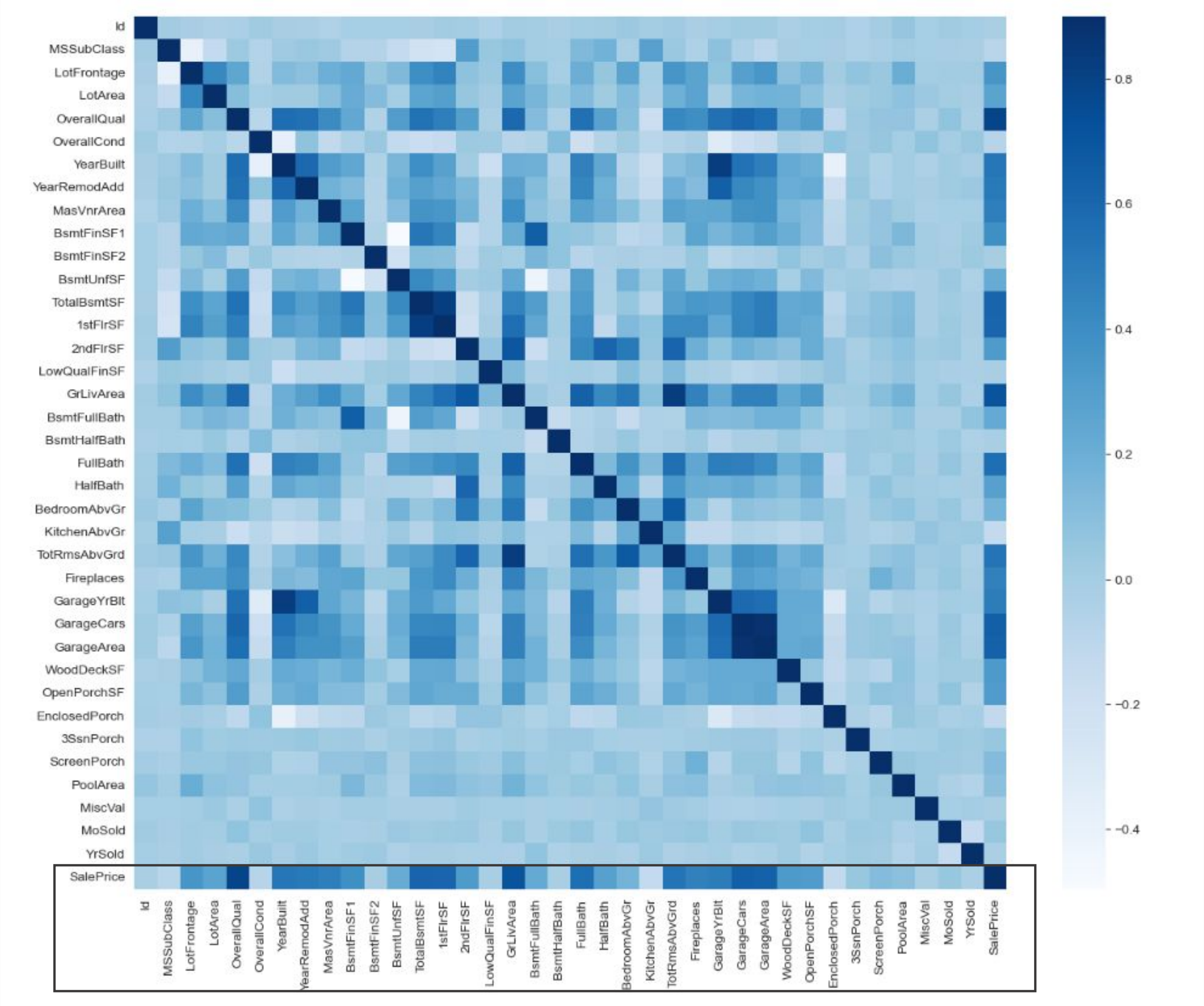
```python
# Stats

from scipy.stats import skew, norm
from scipy.special import boxcox1p
from scipy.stats import boxcox_normmax

# Misc

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold, cross_val_score #교차검증
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import OneHotEncoder #원핫인코딩
from sklearn.preprocessing import LabelEncoder #라벨 인코딩
from sklearn.pipeline import make_pipeline #파이프라인 = 데이터 전처리에서 모델 학습까지 이어주는 함수
from sklearn.preprocessing import scale #스케일링 함수
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.decomposition import PCA
```

라이브러리 불러오기

```python
# Read in the dataset as a dataframe

train = pd.read_csv(r"C:₩Users₩USER₩.kaggle₩train.csv")
test = pd.read_csv(r"C:₩Users₩USER₩.kaggle₩test.csv")
train.shape, test.shape
```

데이터셋 불러오기

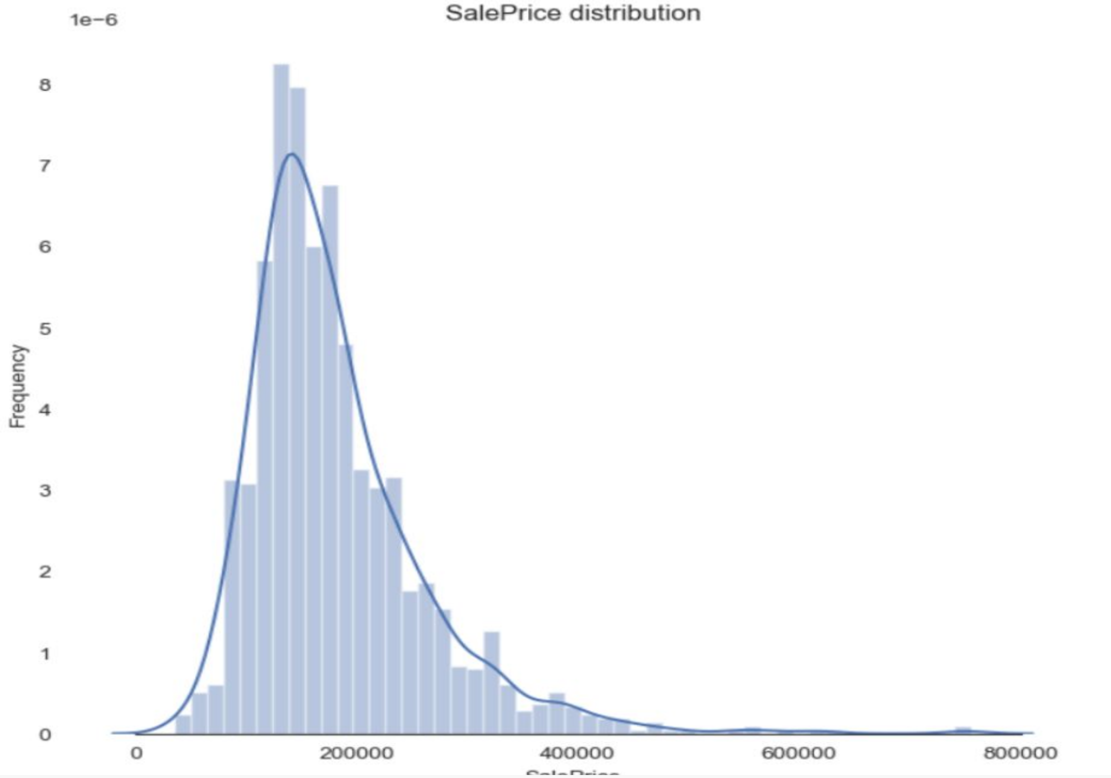
시작 전



수치형 데이터들의 상관관계 시각화

집 값에 품질평가와 면적이 가장 큰 영향을 주는 것을 확인 할 수 있음

왜도



△ 집 값 분포 그래프

왜도

데이터 분포의 비대칭 정도

첨도

데이터 분포의 뾰족한 정도

왜도를 줄여야 하는 이유

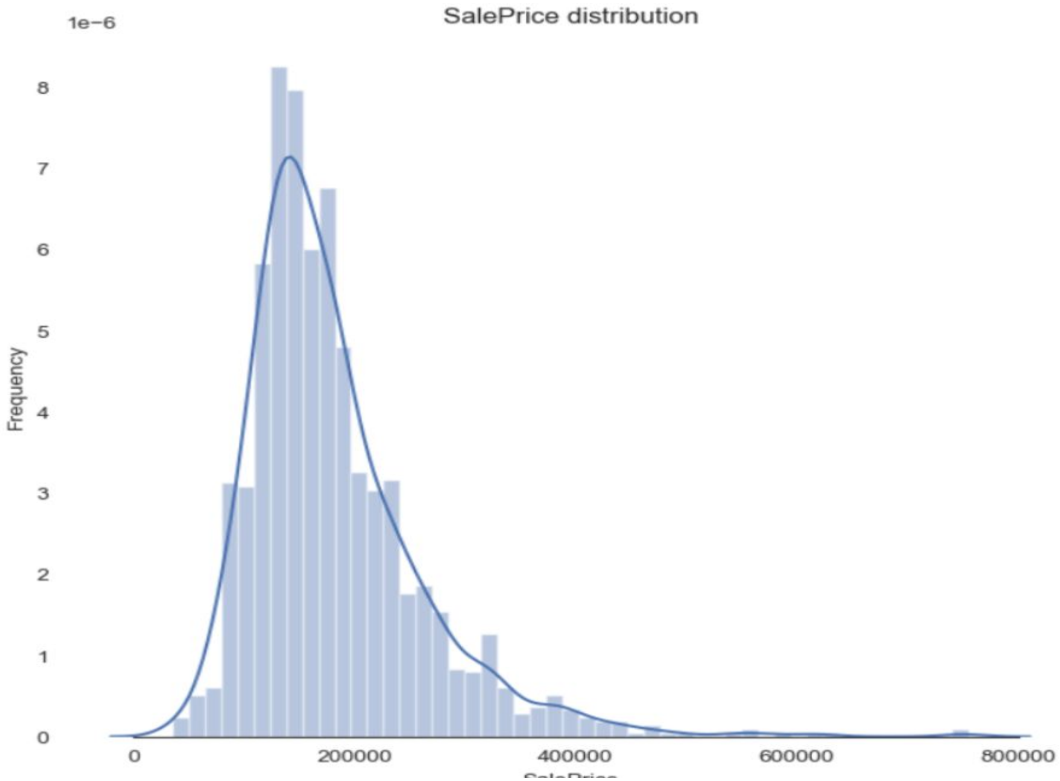
머신러닝은 정규분포 모양의 데이터를
학습하는데에 특화되어있음

왜도 해결 방법

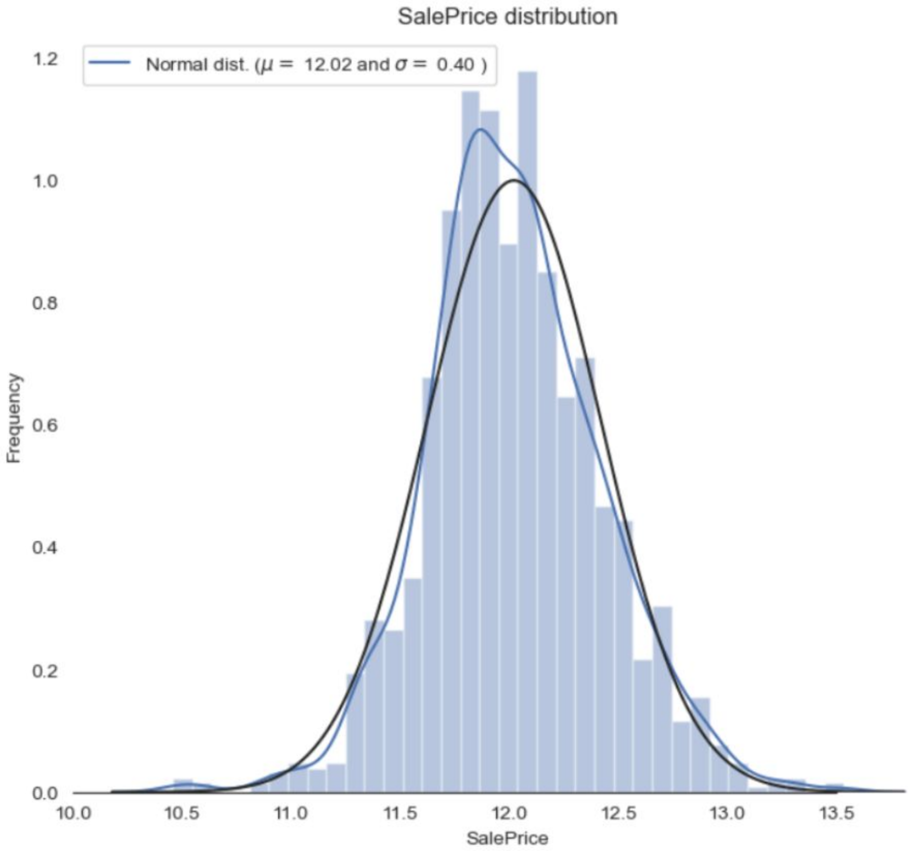
- 로그 변환

```
#log(1+x) 변환 => 왜도를 없애 정규분포로 만들기

train["SalePrice"] = np.log1p(train["SalePrice"])
```

왜도


SalePrice distribution
1e−6
Frequency
0
1
2
3
4
5
6
7
8
0
200000
400000
600000
800000

로그 변환


SalePrice distribution
Normal dist. ($\mu =$ 12.02 and $\sigma =$ 0.40)
Frequency
0.0
0.2
0.4
0.6
0.8
1.0
1.2
10.0
10.5
11.0
11.5
12.0
12.5
13.0
13.5
SalePrice

이상치 제거

```
# 이상치 제거=> drop 함수 (위의 조건 이용)

train.drop(train[(train['OverallQual']<5) & (train['SalePrice']>200000)].index, inplace=True)
train.drop(train[(train['GrLivArea']>4500) & (train['SalePrice']<300000)].index, inplace=True)
train.reset_index(drop=True, inplace=True)
```

이상치 제거

drop() 함수 사용
범위를 정하여 이상치
제거

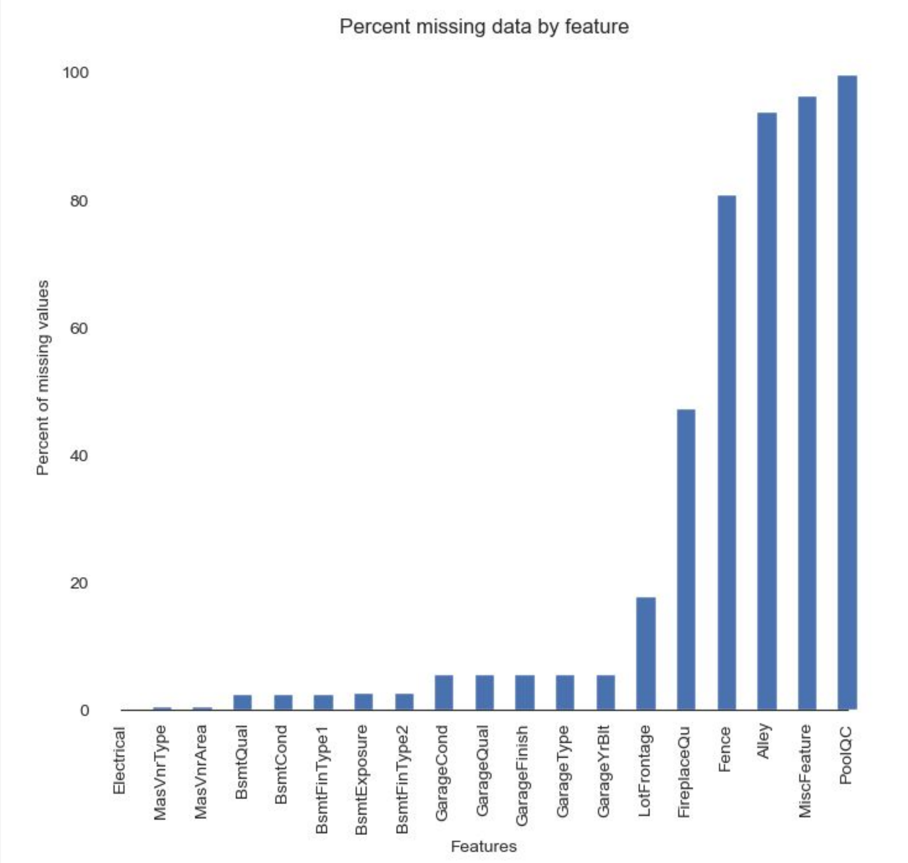
결측값 처리

결측값 파악

```python
# 결측값 파악하기
#mean을 활용해 10개의 결측값 확인하기

def percent_missing(df):
    data = pd.DataFrame(df)
    df_cols = list(pd.DataFrame(data))
    dict_x = {}
    for i in range(0, len(df_cols)):
        dict_x.update({df_cols[i]: round(data[df_cols[i]].isnull().mean()*100,2)})

    return dict_x

missing = percent_missing(all_features)
df_miss = sorted(missing.items(), key=lambda x: x[1], reverse=True)
print('Percent of missing data')
df_miss[0:10]
```

```
Percent of missing data
[('PoolQC', 99.69),
 ('MiscFeature', 96.4),
 ('Alley', 93.21),
 ('Fence', 80.43),
 ('FireplaceQu', 48.68),
 ('LotFrontage', 16.66),
 ('GarageYrBlt', 5.45),
 ('GarageFinish', 5.45),
 ('GarageQual', 5.45),
 ('GarageCond', 5.45)]
```

결측값 시각화



결측값 처리 방법

피쳐 특성에 알맞게 각각 적용

결측값 삭제

평균값으로 대체

중앙값으로 대체

최빈값으로 대체

‘0’ or ‘None’으로 대체

결측값 처리

- 최빈값으로 대체

```
#the data description states that NA refers to typical ('Typ') values
#'Functional'의 결측값을 Typ로 채우기(fillna = 채우기)
features['Functional'] = features['Functional'].fillna('Typ')

# Replace the missing values in each of the columns below with their mode
#위와 동일함

features['Electrical'] = features['Electrical'].fillna("SBrkr")
features['KitchenQual'] = features['KitchenQual'].fillna("TA")

#최빈값으로 채우기
features['Exterior1st'] = features['Exterior1st'].fillna(features['Exterior1st'].mode()[0])
features['Exterior2nd'] = features['Exterior2nd'].fillna(features['Exterior2nd'].mode()[0])
features['SaleType'] = features['SaleType'].fillna(features['SaleType'].mode()[0])
features['MSZoning'] = features.groupby('MSSubClass')['MSZoning'].transform(lambda x: x.fillna(x.mode()[0]))

#'MSSubClass'열을 기준으로 그룹화, 각 그룹 내에서 최빈값으로 대체
```

- 중앙값으로 대체

```
# Group the by neighborhoods, and fill in missing value by the median LotFrontage of the neighborhood
#결측치를 중앙값으로 대체하기(transform = 대체, median=중앙값)

features['LotFrontage'] = features.groupby('Neighborhood')['LotFrontage'].transform(lambda x: x.fillna(x.median()))
```

결측값 처리

- ‘0’ or ‘None’으로 대체

```
features["PoolQC"] = features["PoolQC"].fillna("None")

# Replacing the missing values with 0, since no garage = no cars in garage
#차고가 없으면 차가 없다고 가정함

for col in ('GarageYrBlt', 'GarageArea', 'GarageCars'):
    features[col] = features[col].fillna(0)

# Replacing the missing values with None
#차고가 없으면 각각의 품질 및 상태에 대한 정보가 없다고 가정

for col in ['GarageType', 'GarageFinish', 'GarageQual', 'GarageCond']:
    features[col] = features[col].fillna('None')

# NaN values for these categorical basement features, means there's no basement
#지하실이 없으면 각각의 품질, 상태 및 외부 노출 정보가 없다고 가정

for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2'):
    features[col] = features[col].fillna('None')
```

```
# We have no particular intuition around how to fill in the rest of the categorical features
# So we replace their missing values with None
# 나머지 범주형 특성들의 결측값은 None으로 대체하기('None')

objects = []
for i in features.columns:
    if features[i].dtype == object:
        objects.append(i)
features.update(features[objects].fillna('None'))

# And we do the same thing for numerical features, but this time with 0s
#수치형 변수들은 0으로 처리하기

numeric_dtypes = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
numeric = []
for i in features.columns:
    if features[i].dtype in numeric_dtypes:
        numeric.append(i)
features.update(features[numeric].fillna(0))
return features
```

결측값 처리

결측값 처리 전

```
Percent of missing data
[('PoolQC', 99.69),
 ('MiscFeature', 96.4),
 ('Alley', 93.21),
 ('Fence', 80.43),
 ('FireplaceQu', 48.68),
 ('LotFrontage', 16.66),
 ('GarageYrBlt', 5.45),
 ('GarageFinish', 5.45),
 ('GarageQual', 5.45),
 ('GarageCond', 5.45)]
```

→

결측값 처리 후

```
Percent of missing data
[('MSSubClass', 0.0),
 ('MSZoning', 0.0),
 ('LotFrontage', 0.0),
 ('LotArea', 0.0),
 ('Street', 0.0),
 ('Alley', 0.0),
 ('LotShape', 0.0),
 ('LandContour', 0.0),
 ('Utilities', 0.0),
 ('LotConfig', 0.0)]
```

전처리 마무리 작업

원 핫 인코딩 - get_dummies() 함수 사용

ID	과일
1	사과
2	바나나
3	체리

→

ID	사과	바나나	체리
1	1	0	0
2	0	1	0
3	0	0	1

```
all_features = pd.get_dummies(all_features).reset_index(drop=True)
all_features.shape
```

범주형 데이터 처리

duplicated()함수 사용

a	1
b	2
c	3
a	3
b	2
c	1

→

a	1
b	2
c	3
a	3
c	1

```
# Remove any duplicated column names
# duplicated()를 이용해 중복값 처리

all_features = all_features.loc[:,~all_features.columns.duplicated()]
```

중복값 제거

Part 2 교차검증 & 모델 셋업

교차 검증

K겹 교차검증(K-fold cross validation)

```
# 범주형데이터를 인코딩을 통해 수치형데이터로 변환.

kf = KFold(n_splits=12, random_state=42, shuffle=True)
```

- 과적합 방지 및 신뢰성 있는 모델 평가
- Fold를 교차하여 테스트 데이터로 사용함으로써 모델의 성능를 측정

교차 검증

K겹 교차검증(K-fold cross validation)

```
# 교차검증해주기
def rmsle(y, y_pred):
    return np.sqrt(mean_squared_error(y, y_pred))

def cv_rmse(model, X=X):
    rmse = np.sqrt(-cross_val_score(model, X, train_labels, scoring="neg_mean_squared_error", cv=kf))
    return (rmse)
```

- mean_squared_error () 로 도출 → **RMSLE**
- cross_val_scroe() 과 Kfold 교차검증 적용 → **RMSE**

모델 셋업

GBR

- 잔차 학습 트리 모델

```
gbr = GradientBoostingRegressor(n_estimators=6000,
                                learning_rate=0.01,
                                max_depth=4,
                                max_features='sqrt',
                                min_samples_leaf=15,
                                min_samples_split=10,
                                loss='huber',
                                random_state=42)
```

LightGBM

- Leaf-wise 분산

```
lightgbm = LGBMRegressor(objective='regression',
                         num_leaves=6,
                         learning_rate=0.01,
                         n_estimators=7000,
                         max_bin=200,
                         bagging_fraction=0.8,
                         bagging_freq=4,
                         bagging_seed=8,
                         feature_fraction=0.2,
                         feature_fraction_seed=8,
                         min_sum_hessian_in_leaf = 11,
                         verbose=-1,
                         random_state=42)
```

XGBoost

- Level-wise 분산

```
xgboost = XGBRegressor(learning_rate=0.01,
                       n_estimators=6000,
                       max_depth=4,
                       min_child_weight=0,
                       gamma=0.6,
                       subsample=0.7,
                       colsample_bytree=0.7,
                       objective='reg:squarederror',
                       nthread=-1,
                       scale_pos_weight=1,
                       seed=27,
                       reg_alpha=0.00006,
                       random_state=42)
```

모델 셋업

Ridge

```
ridge_alphas = [1e-15, 1e-10, 1e-8, 9e-4, 7e-4, 5e-4, 3e-4, 1e-4, 1e-3, 5e-2, 1e-2, 0.1, 0.3, 1, 3, 5, 10, 15, 18, 20, 30, 50, 75, 100]
ridge = make_pipeline(RobustScaler(), RidgeCV(alphas=ridge_alphas, cv=kf))
```

- 과적합 방지 위한 규제항 추가

→ Alpha value로 규제 강도 조정

RandomForest

```
rf = RandomForestRegressor(n_estimators=1200,
                           max_depth=15,
                           min_samples_split=5,
                           min_samples_leaf=5,
                           max_features=None,
                           oob_score=True,
                           random_state=42)
```

- 랜덤 추출 샘플

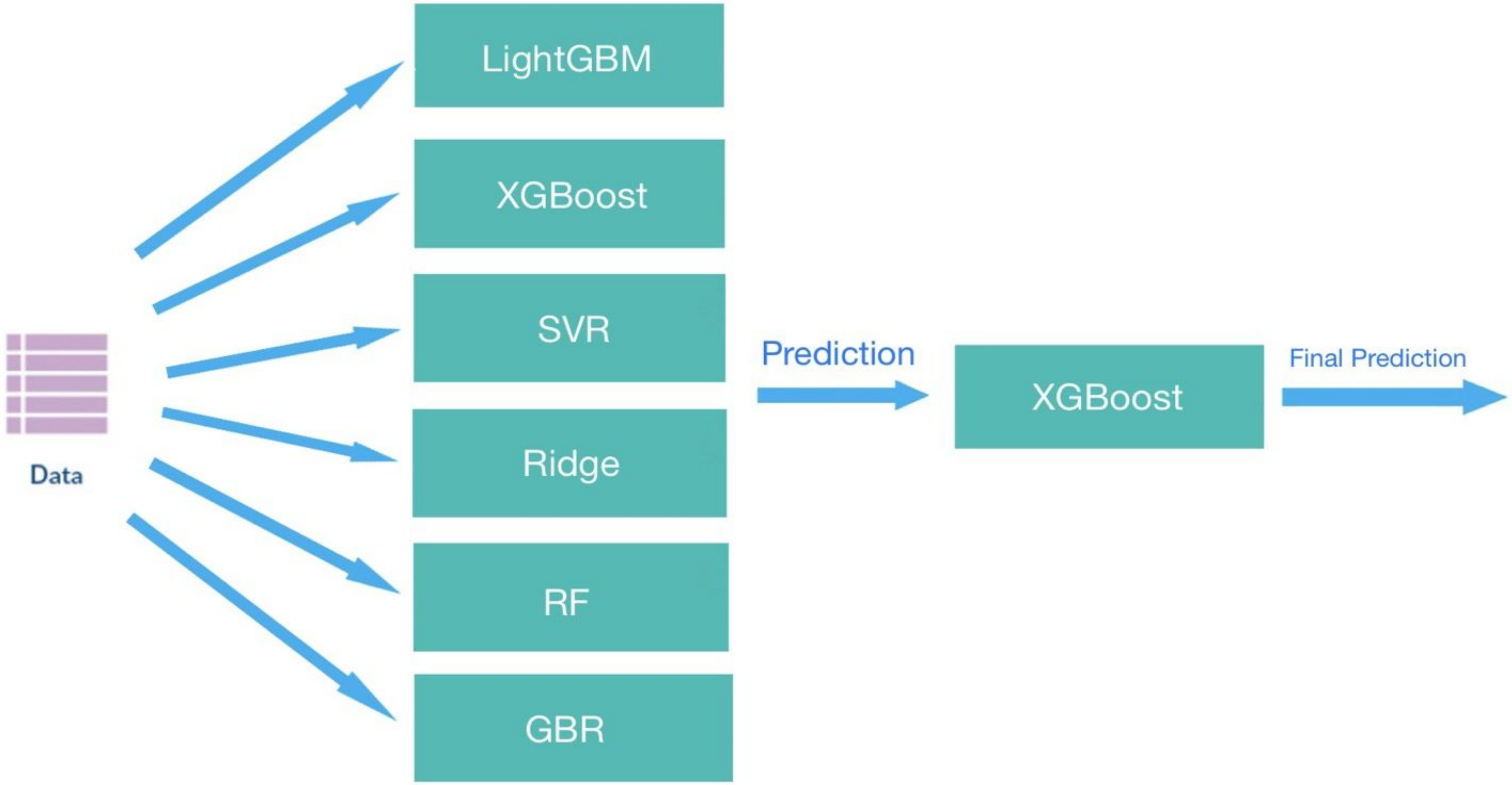
SVR

```
svr = make_pipeline(RobustScaler(), SVR(C= 20, epsilon= 0.008, gamma=0.0003))
```

- 비선형 데이터
- 학습 가능 모델

Part 3 스태킹 및 모델 평가

Stacking


LightGBM
XGBoost
SVR
Ridge
RF
GBR
Data
Prediction
XGBoost
Final Prediction

Score function

```
scores = {}

score = cv_rmse(lightgbm)
print("lightgbm: {:.4f} ({:.4f})".format(score.mean(), score.std()))
scores['lgb'] = (score.mean(), score.std())
```

lightgbm: 0.1159 (0.0177)

```
score = cv_rmse(xgboost)
print("xgboost: {:.4f} ({:.4f})".format(score.mean(), score.std()))
scores['xgb'] = (score.mean(), score.std())
```

xgboost: 0.1337 (0.0168)

```
score = cv_rmse(svr)
print("SVR: {:.4f} ({:.4f})".format(score.mean(), score.std()))
scores['svr'] = (score.mean(), score.std())
```

SVR: 0.1625 (0.0191)

```
score = cv_rmse(rf)
print("rf: {:.4f} ({:.4f})".format(score.mean(), score.std()))
scores['rf'] = (score.mean(), score.std())
```

rf: 0.1344 (0.0160)

```
score = cv_rmse(gbr)
print("gbr: {:.4f} ({:.4f})".format(score.mean(), score.std()))
scores['gbr'] = (score.mean(), score.std())
```

gbr: 0.1114 (0.0168)

```
score = cv_rmse(ridge)
print("ridge: {:.4f} ({:.4f})".format(score.mean(), score.std()))
scores['ridge'] = (score.mean(), score.std())
```

ridge: 0.1110 (0.0163)

Fit function

```
print('stack_gen')
stack_gen_model = stack_gen.fit(np.array(X), np.array(train_labels))
```

stack_gen

```
print('lightgbm')
lgb_model_full_data = lightgbm.fit(X, train_labels)
```

lightgbm

```
print('xgboost')
xgb_model_full_data = xgboost.fit(X, train_labels)
```

xgboost

```
print('Svr')
svr_model_full_data = svr.fit(X, train_labels)
```

Svr

```
print('Ridge')
ridge_model_full_data = ridge.fit(X, train_labels)
```

Ridge

```
print('RandomForest')
rf_model_full_data = rf.fit(X, train_labels)
```

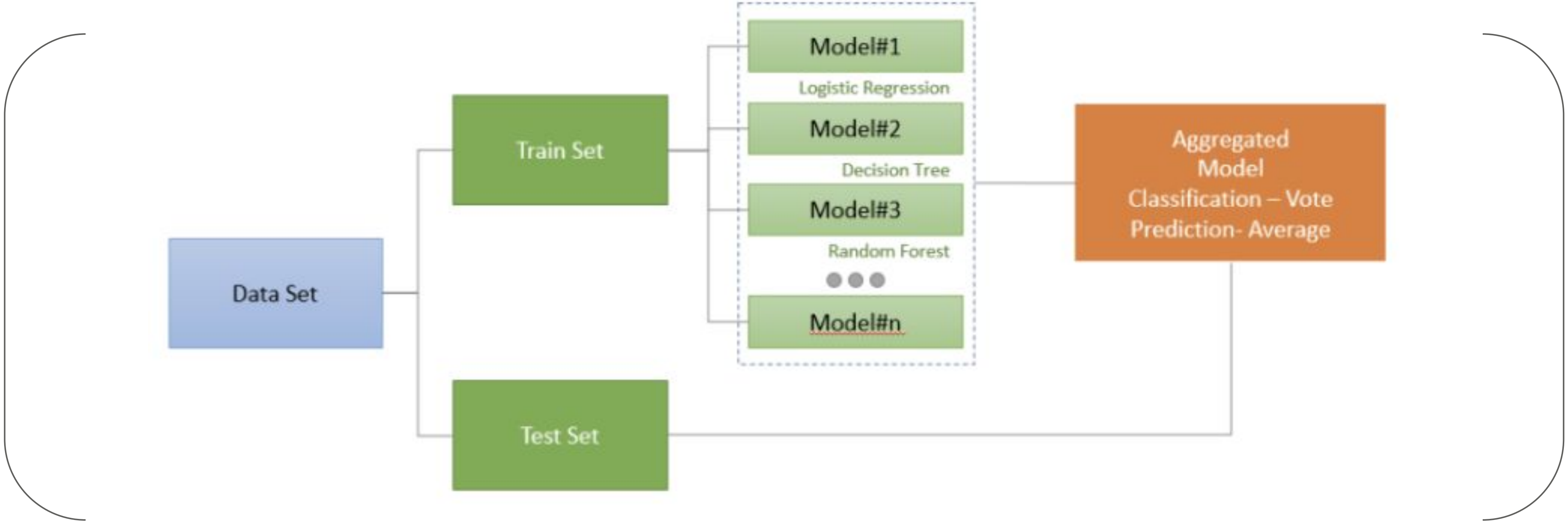
RandomForest

```
print('GradientBoosting')
gbr_model_full_data = gbr.fit(X, train_labels)
```

GradientBoosting

Part 4 블렌딩 및 최종 제출

앙상블 기법


Data Set
Train Set
Test Set
Model#1
Logistic Regression
Model#2
Decision Tree
Model#3
Random Forest
Model#n
Aggregated
Model
Classification – Vote
Prediction- Average

블렌딩

```
def blended_predictions(X):
    return ((0.1 * ridge_model_full_data.predict(X)) + ₩
            (0.2 * svr_model_full_data.predict(X)) + ₩
            (0.1 * gbr_model_full_data.predict(X)) + ₩
            (0.1 * xgb_model_full_data.predict(X)) + ₩
            (0.1 * lgb_model_full_data.predict(X)) + ₩
            (0.05 * rf_model_full_data.predict(X)) + ₩
            (0.35 * stack_gen_model.predict(np.array(X))))
```

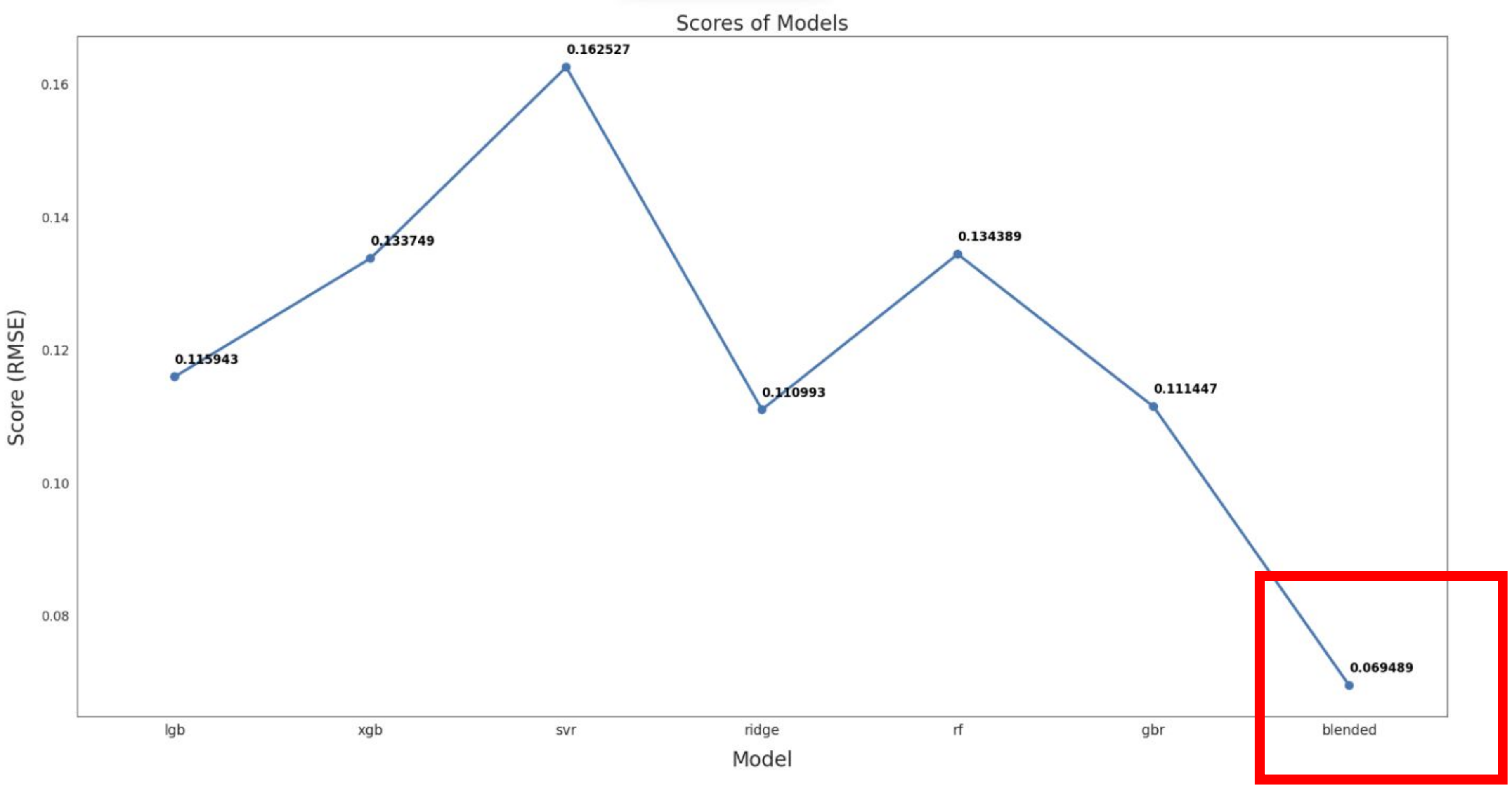
```
blended_score = rmsle(train_labels, blended_predictions(X))
scores['blended'] = (blended_score, 0)
print('RMSLE score on train data:')
print(blended_score)
```

```
RMSLE score on train data:
0.06948925951580026
```

RMSE값 비교

```
# Plot the predictions for each model
sns.set_style("white")
fig = plt.figure(figsize=(24, 12))

ax = sns.pointplot(x=list(scores.keys()), y=[score for score, _ in scores.values()], markers=['o'], linestyles=['-'])
for i, score in enumerate(scores.values()):
    ax.text(i, score[0] + 0.002, '{:.6f}'.format(score[0]), horizontalalignment='left', size='large', color='black', weight='semibold')

plt.ylabel('Score (RMSE)', size=20, labelpad=12.5)
plt.xlabel('Model', size=20, labelpad=12.5)
plt.tick_params(axis='x', labelsize=13.5)
plt.tick_params(axis='y', labelsize=12.5)

plt.title('Scores of Models', size=20)

plt.show()
```

RMSE값 비교


Scores of Models
Score (RMSE)
0.16
0.14
0.12
0.10
0.08
0.162527
0.133749
0.115943
0.110993
0.134389
0.111447
0.069489
lgb
xgb
svr
ridge
rf
gbr
blended
Model

최종 제출

```
preds = np.expm1(blended_predictions (X_test))
```

```
preds
```

```
array([129247.80019341, 162671.80206651, 184959.09624027, ...,
       165931.19146128, 124599.76923249, 218315.36061823])
```

```
sub_fin = pd.DataFrame({"id":test_ID, "SalePrice":preds})
sub_fin.to_csv("sub_fin.csv", index = False)
sub_fin = pd.read_csv("sub_fin.csv")
sub_fin.head()
```

	id	SalePrice
0	1461	129247.800193
1	1462	162671.802067
2	1463	184959.096240
3	1464	195962.186563
4	1465	191369.379068

sub_fin.csv
Complete · 24s ago

0.12412

Fin.