6주차 발표 자료

이지환

Baseline code review

- 부동산 가격 예측 Project in kaggle

```
df.shape
```

```
(1460, 81)
```

```
df.columns
```

```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

Baseline code review

1. 데이터 전처리
- 결측치가 있는 컬럼 모두 삭제
- 좀 더 나은 방법이 있을 것
- 남은 컬럼 개수: 62개

```
[11]: df.isnull().sum()

[11]: Id               0
      MSSubClass       0
      MSZoning         0
      LotFrontage    259
      LotArea          0
                     ...
      MoSold           0
      YrSold           0
      SaleType         0
      SaleCondition    0
      SalePrice        0
      Length: 81, dtype: int64

[12]: df = df.dropna(axis=1, how='any')

[13]: df.isnull().sum()

[13]: Id               0
      MSSubClass       0
      MSZoning         0
      LotArea          0
      Street           0
                      ..
      MoSold           0
      YrSold           0
      SaleType         0
      SaleCondition    0
      SalePrice        0
      Length: 62, dtype: int64
```

Baseline code review

1. 데이터 전처리
- EDA 후 예측에 필요하지 않은 컬럼 선택
- 해당 컬럼 (24개) 삭제
- 정량적 분석과 병행했으면 더 좋았을듯
- 남은 컬럼 38개

```
column_are_not_helpful_in_prediction=ptr_df[['Street','LandContour','Utilities','LandSlope',
column_are_not_helpful_in_prediction
```

	Street	LandContour	Utilities	LandSlope	Condition1	Condition2	RoofMatl	ExterCond	Bsm
0	Pave	Lvl	AllPub	Gtl	Norm	Norm	CompShg	TA	
1	Pave	Lvl	AllPub	Gtl	Feedr	Norm	CompShg	TA	
2	Pave	Lvl	AllPub	Gtl	Norm	Norm	CompShg	TA	
3	Pave	Lvl	AllPub	Gtl	Norm	Norm	CompShg	TA	
4	Pave	Lvl	AllPub	Gtl	Norm	Norm	CompShg	TA	
...	...	...	...	...	...	...	...	...	
1455	Pave	Lvl	AllPub	Gtl	Norm	Norm	CompShg	TA	
1456	Pave	Lvl	AllPub	Gtl	Norm	Norm	CompShg	TA	
1457	Pave	Lvl	AllPub	Gtl	Norm	Norm	CompShg	Gd	
1458	Pave	Lvl	AllPub	Gtl	Norm	Norm	CompShg	TA	
1459	Pave	Lvl	AllPub	Gtl	Norm	Norm	CompShg	TA	

1460 rows × 24 columns

```
ptr_df.drop(column_are_not_helpful_in_prediction, axis=1, inplace=True)
```

Baseline code review

1. 데이터 전처리
- EDA를 토대로 아웃라이어 제거가 필요한 컬럼 선정
- 각 컬럼마다 아웃라이어 계산 후 제거
- 너무 많은 데이터를 삭제하는 것이 아닌지?
- 남은 데이터(row) 1014개

[23]:
```
def remove_outliers(df, columns, threshold=1.5):
    for column in columns:
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - threshold * IQR
        upper_bound = Q3 + threshold * IQR
        df = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
    return df

columns_to_remove_outliers = ['MSSubClass','OverallQual','OverallCond','BsmtFinSF1','

removed_outlier = remove_outliers(ptr_df, columns_to_remove_outliers)

removed_outlier
```

[23]:

	Id	MSSubClass	MSZoning	LotArea	LotShape	LotConfig	Neighborhood	HouseSt
0	1	60	RL	8450	Reg	Inside	CollgCr	2St
2	3	60	RL	11250	IR1	Inside	CollgCr	2St

Baseline code review

1. 데이터 전처리
- 범주형 컬럼을 숫자형으로 Encoding
- ordinal보다 One hot이 더 좋을지도?

```python
def factorize_categorical_columns(column):
    if column.dtype == 'object':
        column_encoded, _ = pd.factorize(column)
        return column_encoded
    return column

# Apply factorize only to categorical columns
df_encoded = removed_outlier.apply(factorize_categorical_columns)

# 'df_encoded' now contains the encoded values for categorical columns
```

```python
df_encoded
```

	Id	MSSubClass	MSZoning	LotArea	LotShape	LotConfig	Neighborhood
0	1	60	0	8450	0	0	0
2	3	60	0	11250	1	0	0
3	4	70	0	9550	1	1	1
5	6	50	0	14115	1	0	2
6	7	20	0	10084	0	0	3

Baseline code review

1. 데이터 전처리
- 훈련에 사용할 컬럼을 지정
- 어떤 기준으로 정하였는지?

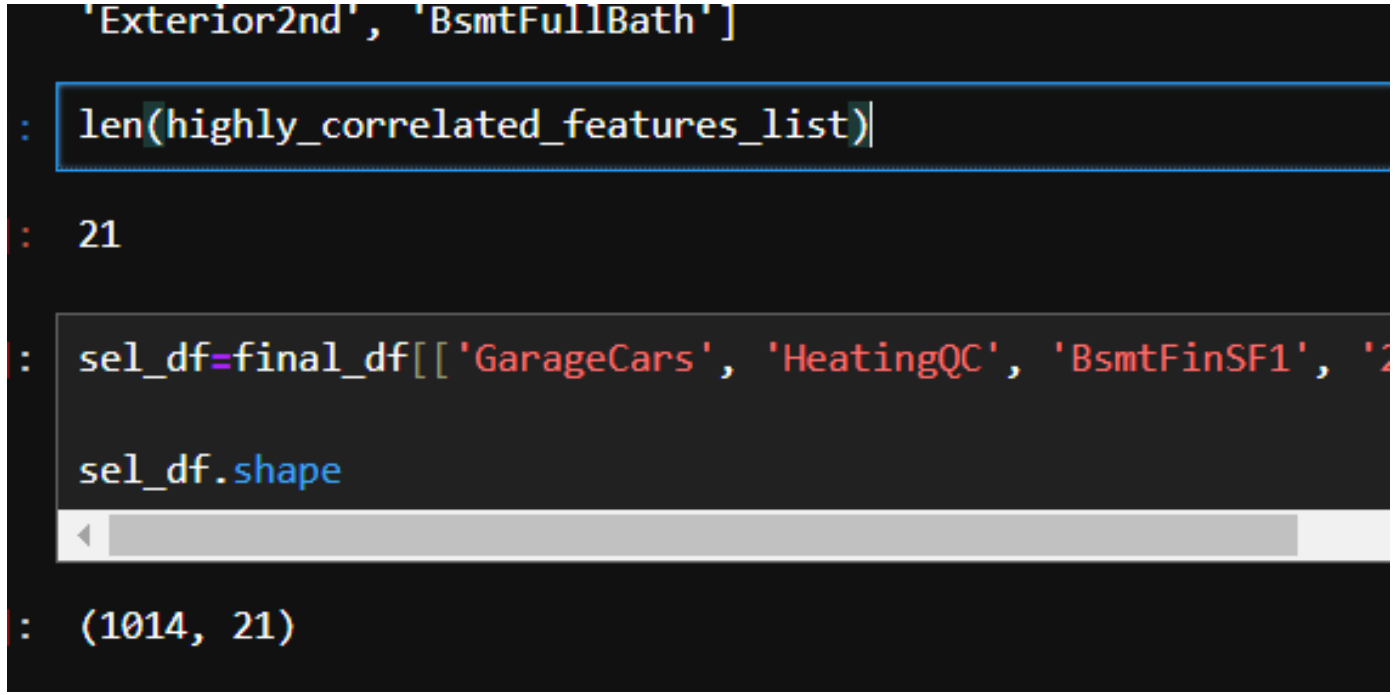
```
final_df=df_encoded[['MSSubClass', 'MSZoning', 'LotArea', 'LotShape', 'LotConfig',
       'Neighborhood', 'HouseStyle', 'OverallQual', 'OverallCond', 'RoofStyle', 'Exterior
       'Foundation', 'BsmtFinSF1', 'BsmtUnfSF', 'TotalBsmtSF', 'HeatingQC',
       '1stFlrSF', '2ndFlrSF', 'GrLivArea', 'BsmtFullBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenQual', 'TotRmsAbvGrd', 'Fireplaces',
       'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF', 'SalePrice']]
```

```
final_df
```

	MSSubClass	MSZoning	LotArea	LotShape	LotConfig	Neighborhood	HouseStyle	Overall
0	60	0	8450	0	0	0	0	
2	60	0	11250	1	0	0	0	
3	70	0	9550	1	1	1	0	
5	50	0	14115	1	0	2	1	
6	20	0	10084	0	0	3	2	
...	...	...	...	...	...	...	...	
1454	20	3	7500	0	0	3	2	

Baseline code review

1. 데이터 전처리
- 서로 상관관계가 높은 (>0.5) 컬럼 선정 후 필터링
- 다중공선성 문제로, 서로 상관관계가 높은 (0.8~0.9) 컬럼 중 하나를 삭제하는 게 맞지 않는지?



Baseline code review

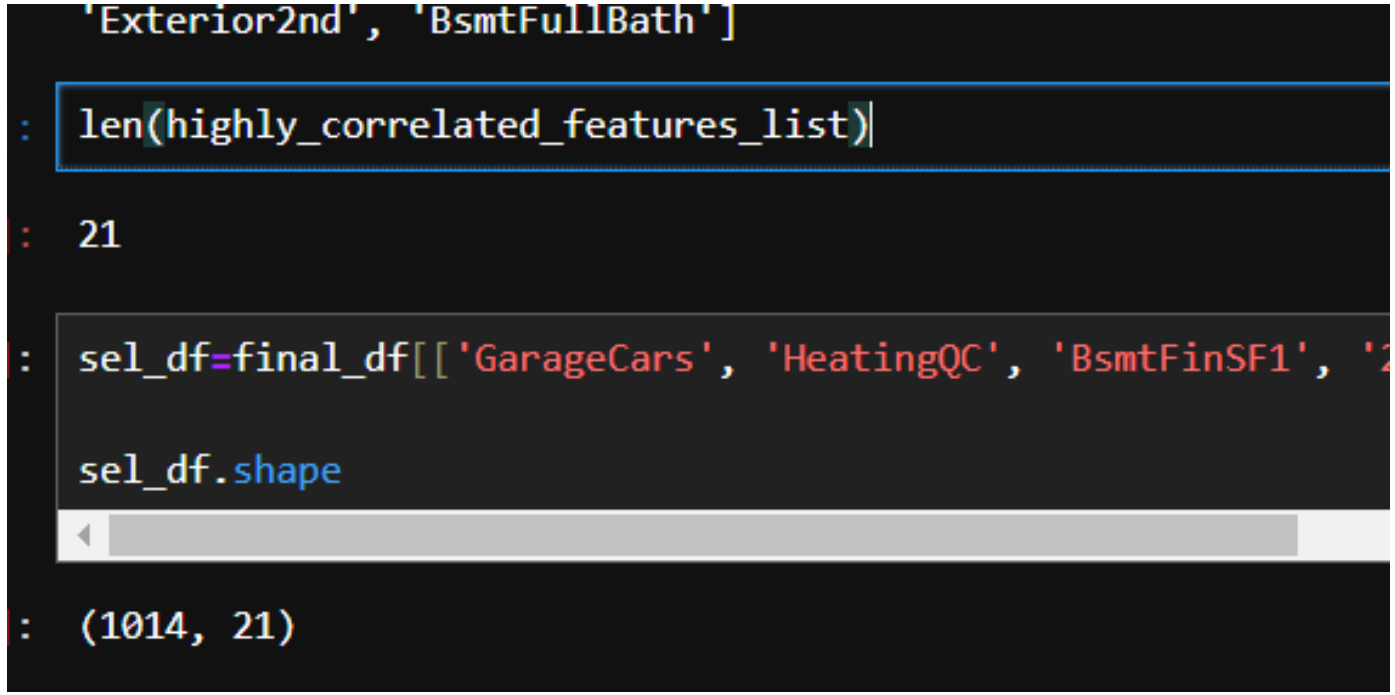
1. 데이터 전처리
- X와 Y로 split
- df_test의 결측치 중, 숫자형 컬럼은 중앙값, 범주형 컬럼은 최빈값으로 처리
- df와 결측치 처리 방식이 다름

```
def pre_process_test(df_test_sel):
    n_df_t=df_test_sel.select_dtypes(include='number')
    n_cols_t=n_df_t.columns
    for col in n_cols_t:
        df_test_sel.loc[:, col] = df_test_sel[col].fillna(df_test_sel[col].median())
    s_df_t=df_test_sel.select_dtypes(include='object')
    s_cols_t=s_df_t.columns
    for col in s_cols_t:
        df_test_sel.loc[:, col] = df_test_sel[col].fillna(df_test_sel[col].value_counts().idxmax())
    for col in s_cols_t:
        df_test_sel.loc[:,col]=pd.factorize(df_test_sel[col])[0]
    return df_test_sel

ptr_df_t=pre_process_test(df_test_sel)
ptr_df_t.isnull().sum()
```

Baseline code review

1. 데이터 전처리
- X와 Y로 split
- df_test의 결측치 중, 숫자형 컬럼은 중앙값, 범주형 컬럼은 최빈값으로 처리
- df와 결측치 처리 방식이 다름



Baseline code review

2. 모델링
- 선형회귀모델로 학습 후 X_test에 대해 예측
- 다른 모델로도 확인해볼 수 있음

```
]: from sklearn.tree import DecisionTreeClassifier
   from sklearn.ensemble import GradientBoostingRegressor
   from sklearn.metrics import mean_squared_error, r2_score
   from sklearn.linear_model import LinearRegression

   model = LinearRegression()
   model.fit(X_train, y_train)
```

```
]: ▾ LinearRegression
   LinearRegression()
```

Export CSV

```
]: test_pred=model.predict(X_test)
   end_pred=pd.DataFrame(test_pred,index=df_test.index)
   end_pred.columns=['SalePrice']
   end_pred.to_csv('submission.csv',sep=',')
   end_pred.head()
```

성능 개선

1. 데이터 전처리
- 결측치가 존재하는 컬럼을 세 가지로 나눔

1) 삭제할 컬럼
➔ 결측치가 유추 불가능하고 그 수가 많으므로 삭제

2) 결측치가 존재하는 row만 삭제할 컬럼
➔ 결측치가 유추 불가능하지만 그 수가 소수
➔ 해당 컬럼을 아예 삭제하진 않고 결측치 데이터만 삭제

3) 결측치를 다른 값으로 대체할 컬럼
➔ 결측치가 어떤 값인지 유추 가능
➔ 대부분 None, 0으로 대체 가능

+ reset_index로 인덱스 초기화

+missing value

삭제할 컬럼 (결측치 유추 불가 & 결측치 다수)

- LotFrontage

Null값 row을 삭제할 컬럼 (결측치 유추 불가, 결측치 소수)

- MasVnrArea
- Electrical

fillna로 채울 컬럼 (결측치 유추 가능)

- Alley -> 'None'
- MasVnrType -> 'None'
- BsmtQual -> 'None'
- BsmtCond -> 'None'
- BsmtExposure -> 'None'
- BsmtFinType1 -> 'None'
- BsmtFinType2 -> 'None'
- FireplaceQu -> 'None'
- GarageType -> 'None'
- GarageYrBlt -> 'None'
- GarageFinish -> 'None'
- GarageQual -> 'None'
- GarageCond -> 'None'
- PoolQC -> 'None'
- Fence -> 'None'
- MiscFeature -> 'None'

성능 개선

1. 데이터 전처리
- z-normalization
- 이후 df_test에도 적용

z-normalization

```
# training dataset에 normalize하는 함수 생성(validation/ test에 사용할 cache 저장)
def z_normalize(df_, columns):
    if sum(df_.loc[:, columns].std() == 0) != 0: # 표준편차가 0인 (즉, 분산이 0인) 열이 존재한다면
        print('하나의 값만 존재하는 컬럼이 있음') # 다음과 같은 내용을 출력한다.
        return
    cache = {}
    cache['mean'] = df_.loc[:, columns].mean(axis = 0)
    cache['std'] = df_.loc[:, columns].std(axis = 0)

    return (df_.loc[:, columns] - df_.loc[:, columns].mean(axis=0)) / df_.loc[:, columns].std(axis = 0), cache

# 2. validation set과 test set을 normalize하는 함수 생성 및 적용
def z_normalize_val(df_, columns, cache):
    return (df_.loc[:, columns] - cache['mean']) / cache['std']
```

```
X_norm = X.copy()
X_norm.loc[:,X_norm.columns], cache = z_normalize(X_norm, X_norm.columns)
X = X_norm.copy()
```

성능 개선

+) OLS summary 비교

Baseline code

OLS Regression Results

Dep. Variable:	SalePrice	**R-squared (uncentered):**	0.982
Model:	OLS	**Adj. R-squared (uncentered):**	0.981
Method:	Least Squares	**F-statistic:**	2653.
Date:	Wed, 08 Nov 2023	**Prob (F-statistic):**	0.00
Time:	11:02:55	**Log-Likelihood:**	-11740.
No. Observations:	1014	**AIC:**	2.352e+04
Df Residuals:	994	**BIC:**	2.362e+04
Df Model:	20		
Covariance Type:	nonrobust		

My code

OLS Regression Results

Dep. Variable:	SalePrice	**R-squared (uncentered):**	0.989
Model:	OLS	**Adj. R-squared (uncentered):**	0.988
Method:	Least Squares	**F-statistic:**	1065.
Date:	Wed, 08 Nov 2023	**Prob (F-statistic):**	0.00
Time:	13:55:35	**Log-Likelihood:**	-11414.
No. Observations:	1007	**AIC:**	2.298e+04
Df Residuals:	931	**BIC:**	2.335e+04
Df Model:	76		
Covariance Type:	nonrobust		

성능 개선

+) r2, adj_r2 비교

Baseline code

```
r2=reg.score(X,y)
r2
```

```
0.8460515711954664
```

```
X.shape
```

```
(1014, 20)
```

```
r2=reg.score(X, y)
n=X.shape[0]
p=X.shape[1]
adj_r2=1-(1-r2)*(n-1)/(n p-1)
adj_r2
```

```
0.8429508979063519
```

My code

```
r2=reg.score(X,y)
r2
```

```
0.8968911520521112
```

```
X.shape
```

```
(1007, 78)
```

```
r2=reg.score(X, y)
n=X.shape[0]
p=X.shape[1]
adj_r2=1-(1-r2)*(n-1)/(n- -1)
adj_r2
```

```
0.8882246756082154
```

성능 개선

1. 데이터 전처리
- 훈련 데이터와 동일한 방법으로 결측치 처리

```
# 삭제할 컬럼
df_test1 = df_test.copy()
df_test1 = df_test1.drop('LotFrontage',axis=1)
```

```
# Null 값 row를 삭제할 컬럼
df_test2 = df_test1.copy()
df_test2 = df_test2.drop(index = df2[df2.MasVnrArea.isnull()].index)
df_test2 = df_test2.drop(index = df2[df2.Electrical.isnull()].index)
```

```
# fillna로 채울 컬럼
df_test3 = df_test2.copy()
num_col = df_test3.select_dtypes(include=np.number).columns
cat_col = df_test3.select_dtypes(exclude=np.number).columns

df_test3[num_col] = df_test3[num_col].fillna(0)
df_test3[cat_col] = df_test3[cat_col].fillna('None')
# df3 = df3.fillna('None')
```

```
# 결측치 없음
df3.isnull().sum().sum()
```

```
0
```

```
# 인덱스 초기화
df_test3 = df_test3.reset_index(drop=True)
```

성능 개선

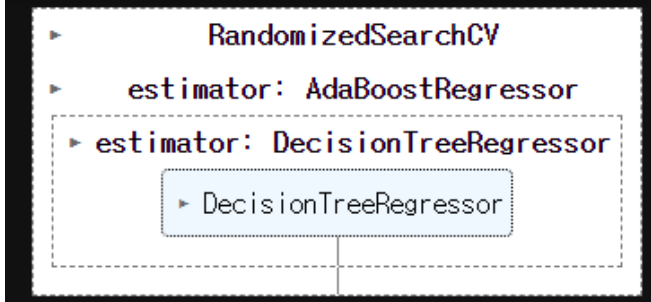
2. 모델링
- 선형회귀 모델 대신 Decision Tree 기반 Adaboost 모델 사용
- RandomizedSearchCV로 하이퍼파라미터 튜닝
- 오른쪽과 같이 결과 확인

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
from scipy.stats import randint, reciprocal

params = {'n_estimators': randint(150,250), 'learning_rate':reciprocal(0.5,2.0),
          'estimator__max_leaf_nodes':[None,2,3], 'estimator__min_samples_split':[2,3,4]}

dtr = DecisionTreeRegressor()
# adbr = AdaBoostRegressor(DecisionTreeRegressor(max_leaf_nodes=2, min_samples_split=3), n_
adbr = AdaBoostRegressor(dtr, random_state=7)

model = RandomizedSearchCV(adbr, params)
model.fit(X_train, y_train)
```




```
model.best_params_
```

```
{'estimator__max_leaf_nodes': None,
 'estimator__min_samples_split': 4,
 'learning_rate': 1.257348371896528,
 'n_estimators': 207}
```

성능 개선

3. 예측 및 결과

개선 방안

1. 컬럼 축소
- 도메인 지식을 수집 및 활용하여 부동산 가격 예측에 필요하지 않은 컬럼을 삭제
- PCA 등을 통해 차원 축소

2. 다양한 모델 비교
- Adaboost뿐만 아니라 여러 ML/DL 모델을 활용한 결과를 비교하여 최선의 결과 창출

3. 범주형 컬럼 인코딩
- Ordinal한 방식이 아닌, One-Hot 또는 다른 인코딩 방식을 선택하여 예측 결과를 확인