

# 모여봐요\_나뭇잎숲

주제: DACON\_주차수요 예측 AI경진대회

## 🌲 모여봐요\_나뭇잎숲 팀원 소개

Aa 이름	☰ 태그	🔗 열 1
김남은(팀장)	<a href="https://github.com/DestinyKim">https://github.com/DestinyKim</a>	
김현준(부팀장)	<a href="https://github.com/hyunjun33">https://github.com/hyunjun33</a>	
박성준(팀원)	<a href="https://github.com/Junnewbe">https://github.com/Junnewbe</a>	
최아름(팀원)	<a href="https://github.com/areummy">https://github.com/areummy</a>	

## 🚩 목차

주제: DACON\_주차수요 예측 AI경진대회

🌲 모여봐요\_나뭇잎숲 팀원 소개

🚩 목차

🗨️ 베이스 라인 데이터 셋 & 릿지 모델 선정 이유

베이스 라인 데이터 셋 구축 과정에서 발견한 포인트

릿지 알고리즘을 활용하게 된 이유

🏠 Catboost 하이퍼 파라미터 튜닝

🏠 임대보증금

임대보증금 히트맵

🚗 추후 개선 사항

👏 결론 및 소감

## 📄 사용 Module

Aa Module	☰ Version	☰ 기능
<a href="#">pandas</a>	1.1.3	데이터 전처리
<a href="#">matplotlib</a>	3.3.2	다양한 Plotting 지원

Aa Module	≡ Version	≡ 기능
<u>seaborn</u>	0.11.0	matplotlib 기반, 고급 Plotting 지원
<u>lasso</u>		선형회귀모델
<u>ridge</u>		선형회귀모델
<u>catboost</u>		
<u>pycaret</u>		여러가지 머신러닝 라이브러리를 ML High-Level API로 제작한 라이브러리
제목 없음		

## 🤔 베이스 라인 데이터 셋 & 릿지 모델 선정 이유

### 베이스 라인 데이터 셋 구축 과정에서 발견한 포인트

- 트레인 셋의 행의 개수 2952개 & 단지코드 기준으로 분류하면 423개 단지 → 개별 행의 독립성

```
In [13]: print(train.shape[0])
          print(len(train["등록차량수"].unique()))

2952
354
```

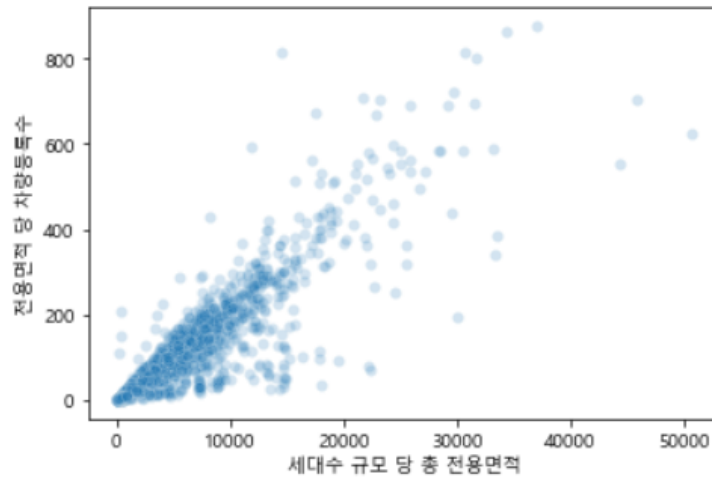
- 각 행의 데이터 개체의 정보를 최대한 활용하는 방향으로 데이터 셋 구축  
ex) 전용면적, 전용면적 별 세대수 특성 VS 나머지 공통된 특성들
- 파생변수 생성 (세대수 별 주차면수, 세대수 규모 당 총 전용면적, 단지별 총 전용면적)
- 다중공선성으로 예상되는 피쳐들 제외 (단지내주차면수, 총세대수 등등)
- 타깃 레이블 변환 (총 등록차량 수 → 전용면적 당 규모별 등록차량수)

```
# 세대수 별 주차면수 피쳐 생성
train_df["세대수비율주차면수"] = (train_df["전용면적별세대수"] / train_df["실거주세대수"]) * train_df["단지내주차면수"]
test_df["세대수비율주차면수"] = (test_df["전용면적별세대수"] / test_df["실거주세대수"]) * test_df["단지내주차면수"]

# 세대수 규모 당 총 전용 면적 피쳐 생성
train_df["세대수 규모 당 총 전용면적"] = train_df["전용면적"] * train_df["전용면적별세대수"]
test_df["세대수 규모 당 총 전용면적"] = test_df["전용면적"] * test_df["전용면적별세대수"]

# 전용면적 당 차량등록수 피쳐 생성
train_df["전용면적 당 차량등록수"] = (train_df["세대수 규모 당 총 전용면적"] / train_df["단지별 총 전용면적"]) * train_df["등록차량수"]
```

- 피쳐들과 타깃 레이블간의 상관관계 분석 및 선형성 검증



## 릿지 알고리즘을 활용하게 된 이유

- 설계한 핵심 피쳐들의 선형성을 검증 → 단순 선형 회귀 모델의 적합도 가능성 판단
- 작은 데이터 셋의 규모 → GBDT 계열 알고리즘의 과대적합성이 높을 것으로 판단
- 데이터 셋의 규모와 피쳐 다양성을 고려 → 데이터 셋의 모든 정보를 활용하기 위해 "릿지" 적합

```
# 랜덤포레스트 모델 예측 성능 평가 -> MAE 14.1685

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error

rf_reg = RandomForestRegressor(n_jobs=-1, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X_train_df_log, y_train_df_log, test_size=0.25, random_state=42)

rf_reg.fit(X_train, y_train)
pred = rf_reg.predict(X_test)

mean_absolute_error(np.expm1(y_test), np.expm1(pred))
```

```
# 라쏘 모델 예측 성능 평가 -> MAE 16.8551

lasso_model = Lasso(alpha=0.01)
X_train, X_test, y_train, y_test = train_test_split(X_train_df_log, y_train_df_log, test_size=0.25, random_state=42)

lasso_model.fit(X_train, y_train)
pred = lasso_model.predict(X_test)

mean_absolute_error(np.expm1(y_test), np.expm1(pred))
```

```
# 릿지 모델 예측 성능 평가 -> MAE 14.9751

ridge_model = Ridge(alpha=20)
X_train, X_test, y_train, y_test = train_test_split(X_train_df_log, y_train_df_log, test_size=0.25, random_state=42)

ridge_model.fit(X_train, y_train)
pred = ridge_model.predict(X_test)

mean_absolute_error(np.expm1(y_test), np.expm1(pred))
```

- train\_test\_split 및 gridsearchcv를 이용해 최적 파라미터를 찾고, MAE 예측 성능 평가

```
# 선형회귀 모델 최적의 alpha 값 탐색
```

```
params = {"alpha": [0.01, 0.1, 0.3, 0.5, 1, 3, 5, 10, 20]}
```

```
elastic_params = {"alpha": [0.01, 0.1, 0.3, 0.5, 1, 3, 5, 10, 20],  
                  "l1_ratio": [0.1, 0.3, 0.5, 0.7, 1]}
```

```
ridge = Ridge()  
lasso = Lasso()  
elastic = ElasticNet()
```

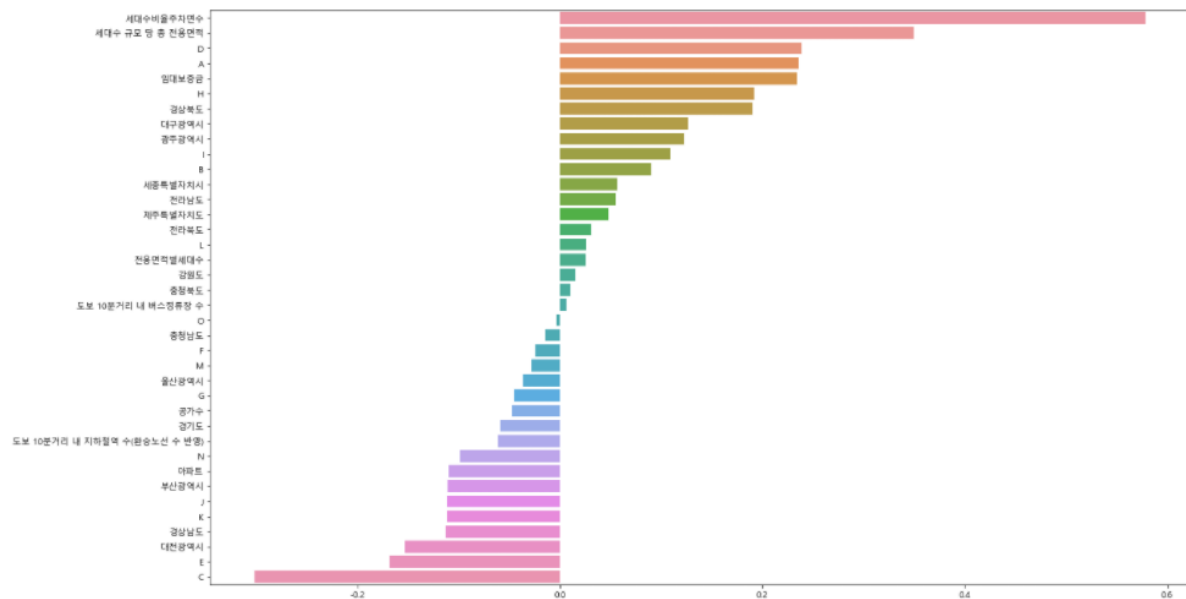
```
grid_ridge = GridSearchCV(ridge, param_grid=params, cv=5, scoring="neg_mean_absolute_error")  
grid_lasso = GridSearchCV(lasso, param_grid=params, cv=5, scoring="neg_mean_absolute_error")  
grid_elastic = GridSearchCV(elastic, param_grid=elastic_params, cv=5, scoring="neg_mean_absolute_error")
```

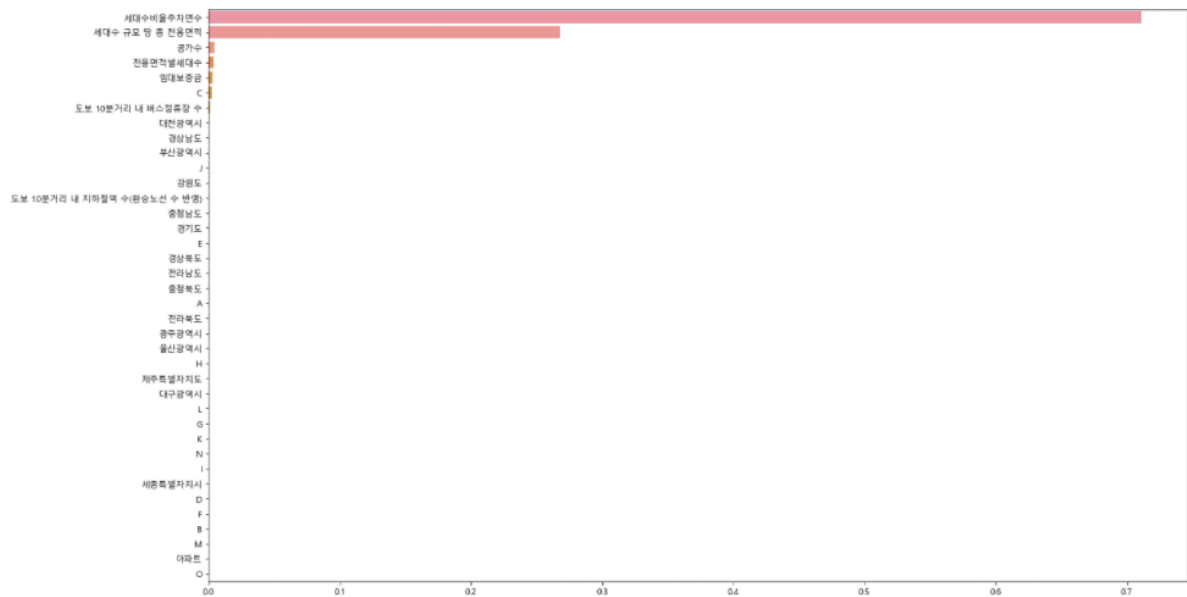
```
grid_ridge.fit(X_train_df_log, y_train_df_log)  
grid_lasso.fit(X_train_df_log, y_train_df_log)  
grid_elastic.fit(X_train_df_log, y_train_df_log)
```

```
grid_ridge.best_params_
```

```
importance = ridge_model.coef_  
feature_importance = pd.Series(data=importance, index=X_train_df_log.columns)  
feature_importance = feature_importance.sort_values(ascending=False)
```

```
plt.figure(figsize=(20, 12))  
sns.barplot(x=feature_importance, y=feature_importance.index)  
plt.show()
```





## Catboost 하이퍼 파라미터 튜닝

### ▼ 1. 최적의 파라미터 값 찾기(GridSearchCV)

#### GridSearchCV 란?

하이퍼 파라미터 그리드에 기술된 모든 파라미터를 편리하게 찾게 해주지만 동시에 순차적으로 파라미터를 다양하게 테스트 하므로 수행시간이 상대적으로 오래걸립니다.

```
params = {'iterations': [200,300,400],
          'depth': [4,5,6],
          'learning_rate': [0.2,0.4,0.5],
          'l2_leaf_reg': [7,8,9]}

cat_reg = CatBoostRegressor(random_seed=42)
grid_model = GridSearchCV(cat_reg, param_grid=params, cv=3, scoring="neg_mean_absolute_error")
gridmodel.fit(X_train_df_log, y_train_df_log)

grid_model.best_params_
{'depth': 4, 'iterations': 200, 'l2_leaf_reg': 9, 'learning_rate': 0.2}

X_train, X_test, y_train, y_test = train_test_split(X_train_df_log, y_train_df_log, test_size=0.25,
                                                    random_state=42)

pred = cat_boost_best.predict(X_test)
mean_absolute_error(np.expm1(y_test), np.expm1(pred))
>>> 9.962978529209792
```

- 결과 : 104.6389221828

### ▼ 2. 최적의 파라미터 값 찾기



그리드 안에 다양한 수를 넣으며 비교 해본 결과,  
Best parameter를 찾기에 너무 오랜시간이 걸려 값을 한개씩 대입해 보았습니다.

```
params = {'iterations': [300],
          'depth': [3],
          'learning_rate': [0.1],
          'l2_leaf_reg': [1]}

cat_reg = CatBoostRegressor(random_seed=42)
grid_model = GridSearchCV(cat_reg, param_grid=params, cv=3, scoring="neg_mean_absolute_error")
grid_model.fit(X_train_df_log, y_train_df_log)

grid_model.best_params_
>>> {'depth': 3, 'iterations': 300, 'l2_leaf_reg': 1, 'learning_rate': 0.1}
```

```
X_train, X_test, y_train, y_test = train_test_split(X_train_df_log, y_train_df_log, test_size=0.25,
                                                    random_state=42)

pred = cat_boost_best.predict(X_test)
mean_absolute_error(np.exp1(y_test), np.exp1(pred))
>>> 10.731237926130666
```

- 결과: 92.0333993119

**MAE=10.95 > 88점**

**MAE=10.82 > 87점**

- MAE값을 10.82보다 미세하게 낮추기 위해 하이퍼 파라미터 조정

**MAE=10.79 > 90점**

**MAE=10.73 > 89점**

**MAE=10.42 > 89점**

**MAE=9.96 > 104점**

```
params = {'iterations': [100,200,300],
          'depth': [2,3,4],
          'learning_rate': [0.1,0.2,0.3],
          'l2_leaf_reg': [1,2,3]}

cat_reg = CatBoostRegressor(random_seed=42)
grid_model = GridSearchCV(cat_reg, param_grid=params, cv=3, scoring="neg_mean_absolute_error")
grid_model.fit(X_train_df_log, y_train_df_log)

grid_model.best_params_
>>> {'depth': 4, 'iterations': 100, 'l2_leaf_reg': 2, 'learning_rate': 0.2}
```

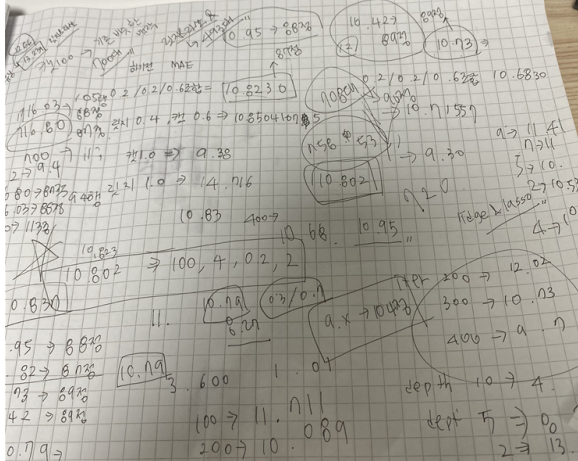
```
X_train, X_test, y_train, y_test = train_test_split(X_train_df_log, y_train_df_log, test_size=0.25,
                                                    random_state=42)

pred = cat_boost_best.predict(X_test)
mean_absolute_error(np.exp1(y_test), np.exp1(pred))
>>> 10.802095654274565
```

- 결과: 89.461210929



하이퍼 파라미터 튜닝을 하면 정확도가 오히려 낮아지기 때문에 기본 Catboost 모델을 제시하여 87점대로 낮출 수 있었습니다.



이 말고도 다양한 값을 대입하여 하이퍼 파라미터를 측정하였지만,,, 기록의 미숙함으로 다양한 값의 결과를 보여드리지 못한 점 죄송합니다...

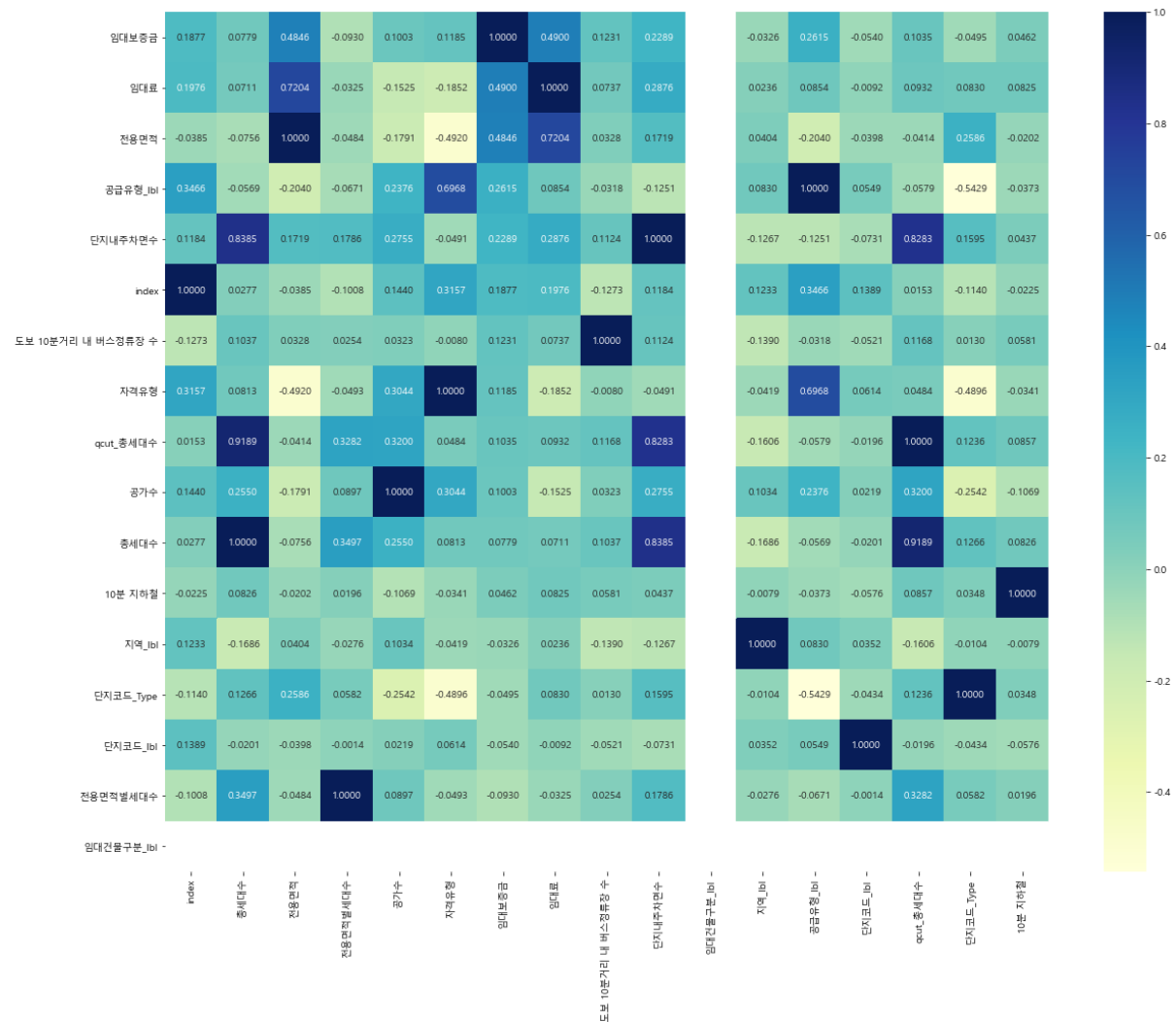
개선할점: 모든 시행 착오들을 기록해 가면서 어떠한 점이 부족했는지 상세히 기록해 놓아야 겠다고 다짐했습니다,,^^



## 임대보증금

```
from sklearn.model_selection import train_test_split
sel = [ '전용면적', '자격유형', '임대료', '단지내주차면수', '공급유형_lbl',
        '지역_lbl', '도보 10분거리 내 버스정류장 수', 'qcut_총세대수' ]
x = all_df_1[sel]
y = all_df_1['임대보증금']

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0,)
```



## 임대보증금 히트맵

- 임대보증금과 최소한 0.1이상의 상관관계가 있는 Features 선택
- 임대건물구분\_lbi 오류? 발생

```
all_df["임대보증금"] = all_df["임대보증금"].fillna(0)
all_df.loc[all_df["임대보증금"] == "- ", "임대보증금"] = 0
all_df["임대보증금"] = all_df["임대보증금"].astype(int)

all_df_0=all_df.loc[all_df['임대보증금']==0]
all_df_1=all_df.loc[all_df['임대보증금']!=0]
all_df_1['임대건물구분_lbi'].value_counts()

1      3110
Name: 임대건물구분_lbi, dtype: int64
```



- 오류 이유는 임대보증금 NULL값(-)을 모두 0으로 변환후 0을 기준으로 나누는 과정에서 all\_df\_1['임대건물구분\_lbi'] 속에 value 종류가 1(아파트)만 존재! 단일값의 Feature는 다른 변수와 상관관계 분석 불가?



## 추후 개선 사항

- 원핫인코딩 처리 후, 많아진 데이터 피쳐들의 개수를 줄이되, 데이터 자체의 손실을 없애기 위해 상관관계가 높은 피쳐끼리 묶어 새로운 피쳐를 생성하여 모델 성능을 개선시켜 볼 수 있을 것 같다.
- 이처럼 다양한 데이터들을 활용하지 못 한 점이 한계점이다. 시간이 더 있다면 향후에는 지역별(수도권, 비수도권), 공휴일, 사고 발생 등과 같은 다른 데이터들을 활용하여 최적의 주차수요 예측을 위한 모델 개발을 고안해 볼 것이다.

## 결론 및 소감

 이름	 태그
<u>김남은</u>	
<u>김현준</u>	
<u>박성준</u>	프로젝트 중에 잠이 부족하다
<u>최아름</u>	