유형별 임대주택 설계 시
단지 내 적정 주차 수요 예측

Playdata AI 22기 2nd project

2팀 고영현, 노민경, 전유진

목차

1. 문제 정의

2. 탐색적 데이터 분석

3. 데이터 전처리

4. 베이스라인 모델

5. 성능 평가

6. 성능 올리기

7. 최종 결론

문제 정의

주차장 수요 vs 공급 '엇박자' 심각

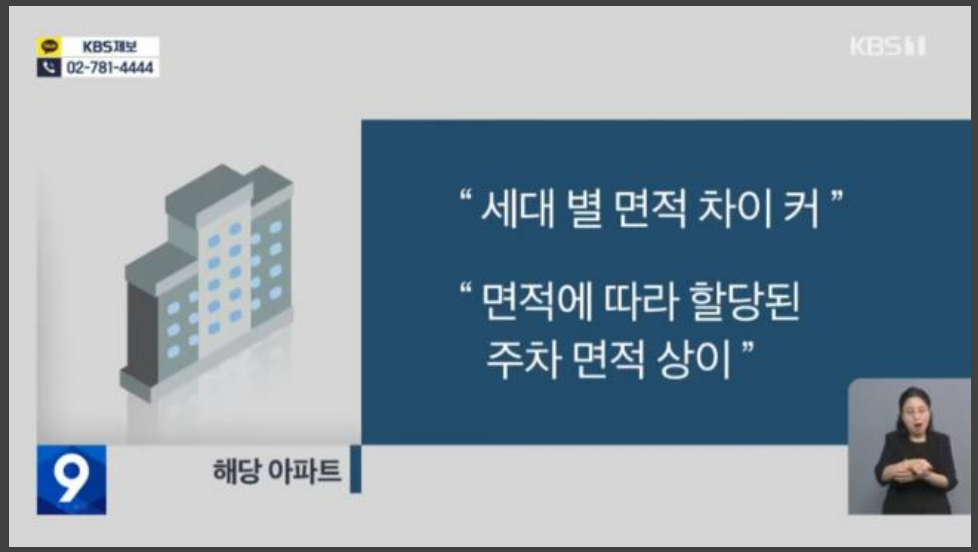
홍수영 기자 | 승인 2019.02.13 19:43 | 댓글 0

"주차장 부족 문제...종교시설·학교·상가와 공유 경제로 해결"

대전시, 부설주차장 개방사업 추진...최대 2500만원 지원

등록 2022-03-10 오전 10:34:57
수정 2022-03-10 오전 10:34:57

가 가



1인당 차량 보유수가 늘어남으로써,

주차장 수요와 공급의 불균형 심화

세대 내 무료 주차 불가능한 상황 발생

문제 정의

아파트 단지 내 필요한 주차 대수 산정법

①법정주차대수 ②장래주차수요 중 큰 값에 따라 결정

현재, ②장래주차수요 조사 방법

‘주차원단위’ + ‘건축연면적’

- ‘주차원단위’

신규 건축예정 부지 인근의 유사 단지를 피크 시간대 방문하여, 주차된 차량대수를 세는 방법으로

사람이 직접 조사

인력조사로 인한 오차발생, 현장조사 시점과 실제 건축시점과의 시간차 등의 문제로

과대 또는 과소 산정의 가능성 존재

문제 정의

❖ 문제 해결 목적

🏠 유형별 임대주택 설계 시 단지 내 적정 🅿 주차 수요를 예측

❖ 평가 성능지표 : MAE (Dacon 대회 규정)

❖ 주최 : 한국토지주택공사

❖ 주관 : 데이콘

1. 문제 정의

2. 탐색적 데이터 분석

3. 데이터 전처리

4. 베이스라인 모델

5. 성능 평가

6. 성능 올리기

7. 최종 결론

탐색적 데이터 분석

데이터 불러오기

```
test = pd.read_csv('./dataset/test.csv')
train = pd.read_csv('./dataset/train.csv')
age_gender = pd.read_csv('./dataset/age_gender_info.csv')
```

중복데이터 제거

```
train = train.drop_duplicates()
```

train.shape → (2632, 15)

탐색적 데이터 분석

데이터 훑어보기

train.head()

train.shape

test.head()

test.shape

age_gender.head()

age_gender.shape

	단지코드	총세대수	임대건물구분	지역	공급유형	전용면적	전용면적별세대수	공가수	자격유형	임대보증금	임대료	도보 10분거리 내 지하철역 수(환승노선 수 반영)	도보 10분거리 내 버스정류장 수	단지내주차면수	등록차량수
0	C2483	900	아파트	경상북도	국민임대	39.72	134	38.0	A	15667000	103680	0.0	3.0	1425.0	1015.0
1	C2483	900	아파트	경상북도	국민임대	39.72	15	38.0	A	15667000	103680	0.0	3.0	1425.0	1015.0
2	C2483	900	아파트	경상북도	국민임대	51.93	385	38.0	A	27304000	184330	0.0	3.0	1425.0	1015.0

	단지코드	총세대수	임대건물구분	지역	공급유형	전용면적	전용면적별세대수	공가수	자격유형	임대보증금	임대료	도보 10분거리 내 지하철역 수(환승노선 수 반영)	도보 10분거리 내 버스정류장 수	단지내주차면수
0	C1072	754	아파트	경기도	국민임대	39.79	116	14.0	H	22830000	189840	0.0	2.0	683.0
1	C1072	754	아파트	경기도	국민임대	46.81	30	14.0	A	36048000	249930	0.0	2.0	683.0
2	C1072	754	아파트	경기도	국민임대	46.90	112	14.0	H	36048000	249930	0.0	2.0	683.0

	지역	10대미만(여자)	10대미만(남자)	10대(여자)	10대(남자)	20대(여자)	20대(남자)	30대(여자)	30대(남자)	40대(여자)	...	60대(여자)	60대(남자)	70대(여자)	70대(남자)	80대(여자)	80대(남자)
0	경상북도	0.030158	0.033195	0.056346	0.061360	0.060096	0.067859	0.053433	0.049572	0.083660	...	0.082684	0.063889	0.047717	0.030172	0.029361	0.011211
1	경상남도	0.027400	0.026902	0.053257	0.055568	0.064920	0.070618	0.056414	0.057550	0.077092	...	0.087201	0.069562	0.048357	0.033277	0.027361	0.011295
2	대전광역시	0.028197	0.029092	0.040490	0.042793	0.060834	0.064247	0.068654	0.066848	0.074667	...	0.088468	0.070261	0.051010	0.037143	0.032455	0.013751

탐색적 데이터 분석

데이터 훑어보기

train.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2952 entries, 0 to 2951
Data columns (total 15 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   단지코드                            2952 non-null   object
 1   총세대수                            2952 non-null   int64
 2   임대건물구분                          2952 non-null   object
 3   지역                            2952 non-null   object
 4   공급유형                            2952 non-null   object
 5   전용면적                            2952 non-null   float64
 6   전용면적별세대수                        2952 non-null   int64
 7   공가수                             2952 non-null   float64
 8   자격유형                            2952 non-null   object
 9   임대보증금                           2383 non-null   object
 10  임대료                             2383 non-null   object
 11  도보 10분거리 내 지하철역 수(환승노선 수 반영)    2741 non-null   float64
 12  도보 10분거리 내 버스정류장 수              2948 non-null   float64
 13  단지내주차면수                         2952 non-null   float64
 14  등록차량수                           2952 non-null   float64
dtypes: float64(6), int64(2), object(7)
memory usage: 346.1+ KB
```

test.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1022 entries, 0 to 1021
Data columns (total 14 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   단지코드                            1022 non-null   object
 1   총세대수                            1022 non-null   int64
 2   임대건물구분                          1022 non-null   object
 3   지역                            1022 non-null   object
 4   공급유형                            1022 non-null   object
 5   전용면적                            1022 non-null   float64
 6   전용면적별세대수                        1022 non-null   int64
 7   공가수                             1022 non-null   float64
 8   자격유형                            1020 non-null   object
 9   임대보증금                           842 non-null    object
 10  임대료                             842 non-null    object
 11  도보 10분거리 내 지하철역 수(환승노선 수 반영)    980 non-null    float64
 12  도보 10분거리 내 버스정류장 수              1022 non-null   float64
 13  단지내주차면수                         1022 non-null   float64
dtypes: float64(5), int64(2), object(7)
memory usage: 111.9+ KB
```

1. 문제 정의
2. 탐색적 데이터 분석
3. 데이터 전처리
4. 베이스라인 모델
5. 성능 평가
6. 성능 올리기
7. 최종 결론

데이터 전처리

❖ NULL 값 처리

1. 임대료, 임대보증금 평균값으로 처리
2. 지하철역 수, 버스정거장 수 0으로 처리
3. Test 데이터 [자격유형] 동일한 단지코드 내 자격 유형으로 처리

❖ 필요 없는 데이터 삭제

❖ 임대건물 구분 숫자 데이터로 변환

❖ 데이터 가공

1. age_gender 데이터 內 20대 이상 총합 수 구한 뒤 train & test 데이터로 병합

데이터 전처리

Train Data

NULL 값 확인

(train['임대보증금'].isnull()).sum() → 292

(train['임대료'].isnull()).sum() → 292

(train['도보 10분거리 내 지하철역 수(환승노선 수 반영)'].isnull()).sum() → 155

(train['도보 10분거리 내 버스정류장 수'].isnull()).sum() → 4

임대보증금 → 평균값으로 대체

1. '-' 값을 NaN 값으로 변환

```
train[train['임대보증금']=='-'] = train[train['임대보증금']=='-'].replace('-',np.NAN)
```

2. 특성 파악을 위해 float 형태로 변환

```
train['임대보증금'] = train['임대보증금'].astype(float)
```

3. 임대보증금 평균값 확인

```
mean = train['임대보증금'].mean()
mean = np.round(mean)
```

4. NaN 값을 평균값으로 변경

```
train['임대보증금'] = train['임대보증금'].fillna(mean)
```

Mean 값 : 26,826,528.0

Train Data

임대료 → 평균값으로 대체

1. '-' 값을 NaN 값으로 변환

```
train[train['임대료']=='-'] = train[train['임대료']=='-'].replace('-',np.NAN)
```

2. 특성 파악을 위해 float 형태로 변환

```
train['임대료'] = train['임대료'].astype(float)
```

3. 임대보증금 평균값 확인

```
mean = train['임대료'].mean()
mean = np.round(mean)
```

4. NaN 값을 평균값으로 변경

```
train['임대료'] = train['임대료'].fillna(mean)
```

Mean 값 : 1,995,598.0

데이터 전처리

Train Data

지하철 역 개수 -> 0으로 처리

1. 도보 10분 거리 내 지하철 역의 최빈값 확인

```
train['도보 10분거리 내 지하철역 수(환승노선 수 반영)'].value_counts()
```

최빈값 : 0

2. 단지코드가 서로 같을 경우 지하철 역수 확인

```
Code=['C1616','C1875','C2258','C1874','C1004','C1350','C2583','C1983','C2100','C1068','C2644','C1649','C13
12','C1704','C1005','C2156','C1175','C1424','C2216','C2520']
for i in code:
    print(train[train['단지코드'] == 'i']['도보 10분거리 내 지하철역 수(환승노선 수 반영)'])
```

모두 NaN 값으로 확인

지하철 역 개수 -> 0으로 처리

3. 지하철역 수의 최빈값은 0이므로, NaN 값을 모두 0으로 처리

```
train['도보 10분거리 내 지하철역 수(환승노선 수 반영)'] = train['도보 10분거리 내 지하철역 수(환승노선 수 반영)'].fillna(0)
```

데이터 전처리

Train Data

버스정거장 개수 -> 0으로 처리

1. 도보 10분 거리 내 지하철 역의 최빈값 확인

```
train['도보 10분거리 내 버스정류장 수'].value_counts()
```

최빈값 : 2.0

2. NaN 값을 지하철과 동일하게 0으로 변경

```
train['도보 10분거리 내 버스정류장 수'] = train['도보 10분거리 내 버스정류장 수'].fillna(0)
```

최빈값이 2로 나왔지만, 처리해야할 데이터는 4건

따라서, NaN 값을 지하철과 동일하게 0으로 변경

데이터 전처리

Test Data

NULL 값 확인

(test['자격유형'].isnull()).sum() → 2

(test['임대보증금'].isnull()).sum() → 180

(test['임대료'].isnull()).sum() → 180

(test['도보 10분거리 내 지하철역 수(환승노선 수 반영)'].isnull()).sum() → 42

데이터 전처리

Test Data

임대보증금 → 평균값으로 대체

1. '-' 값을 NaN 값으로 변환

```
test[test['임대보증금']=='-'] = test[test['임대보증금']=='-'].replace('-',np.NAN)
```

2. 특성 파악을 위해 float 형태로 변환

```
test['임대보증금'] = test['임대보증금'].astype(float)
```

3. 임대보증금 평균값 확인

```
mean = test['임대보증금'].mean()
mean = np.round(mean)
```

4. NaN 값을 평균값으로 변경

```
test['임대보증금'] = test['임대보증금'].fillna(mean)
```

Mean 값 : 25,050,940.0

데이터 전처리

Test Data

임대료 → 평균값으로 대체

1. '-' 값을 NaN 값으로 변환

```
test[test['임대료']=='-'] = test[test['임대료']=='-'].replace('-',np.NAN)
```

2. 특성 파악을 위해 float 형태로 변환

```
test['임대료'] = test['임대료'].astype(float)
```

3. 임대보증금 평균값 확인

```
mean = test['임대료'].mean()
mean = np.round(mean)
```

Mean 값 : 177,574.0

4. NaN 값을 평균값으로 변경

```
test['임대료'] = test['임대료'].fillna(mean)
```

Test Data

자격 유형

1. 자격유형에서 Null 값 확인

test[test['자격유형'].isnull()]

	단지코드	총세대수	임대건물구분	지역	공급유형	전용면적	전용면적별 세대수	공가수	자격유형	임대보증금	임대료	도보 10분거리 내 지하철역 수(환승노선 수 반영)	도보 10분거리 내 버스정류장 수	단지내주차면수
196	C2411	962	아파트	경상남도	국민임대	46.90	240	25.0	NaN	71950000.0	37470.0	0.0	2.0	840.0
258	C2253	1161	아파트	강원도	영구임대	26.37	745	0.0	NaN	2249000.0	44770.0	0.0	2.0	173.0

2. 단지코드가 서로 같을 경우 해당하는 자격유형으로 변경

test.loc[(test.단지코드=='C2411')&test.자격유형.isnull(), '자격유형'] = 'A'

test.loc[(test.단지코드=='C2253')&test.자격유형.isnull(), '자격유형'] = 'C'

Test Data

지하철 역 개수 -> 0으로 처리

Train 데이터와 마찬가지로 0값으로 처리

```
test['도보 10분거리 내 지하철역 수(환승노선 수 반영)'] = test['도보 10분거리 내 지하철역 수(환승노선 수 반영)'].fillna(0)
```

Age_gender Data

20대 이상 합 구하기

1. 20대 미만 수 삭제

```
age_gender_20=age_gender.drop(['10대미만(여자)','10대미만(남자)','10대(여자)','10대(남자)'],axis=1)
```

2. 컬럼 추가

```
age_gender['20대이상총합'] = age_gender_20.sum(axis=1)
```

3. Train & Test 데이터에 Join

```
train = pd.merge(train,age_gender[['지역','20대이상총합']],how='left',left_on='지역',right_on='지역')
test = pd.merge(test,age_gender[['지역','20대이상총합']],how='left',left_on='지역',right_on='지역')
```

데이터 전처리

상관관계 확인

train.corr()['등록차량수'].sort_values()

```
도보 10분거리 내 지하철역 수(환승노선 수 반영)   -0.063716
20대이상총합                          -0.032255
전용면적                             0.081118
공가수                              0.095454
도보 10분거리 내 버스정류장 수              0.117849
전용면적별세대수                         0.188906
임대보증금                            0.283975
임대료                              0.373893
총세대수                             0.423656
단지내주차면수                          0.851280
등록차량수                            1.000000
Name: 등록차량수, dtype: float64
```

train.corr()['단지내주차면수'].sort_values()

```
도보 10분거리 내 지하철역 수(환승노선 수 반영)   -0.028426
20대이상총합                          0.019532
전용면적                             0.035756
도보 10분거리 내 버스정류장 수              0.093102
전용면적별세대수                         0.231223
임대보증금                            0.251715
임대료                              0.255493
공가수                              0.292526
총세대수                             0.598347
등록차량수                            0.851280
단지내주차면수                          1.000000
Name: 단지내주차면수, dtype: float64
```

'단지내주차면수' 와 '등록차량수' 가 높은 상관관계를 갖고 있다는 점 확인

데이터 삭제

삭제 데이터 목록

- 지역 : 20대 이상의 비율과 동일함
- 공급유형: 임대건물 구분의 하위데이터 개념
- 공가수 : 상관관계가 매우 작음
- 자격유형: 차량에 대한 규제는 보유 여부가 아닌 액수 관련

```
train = train.drop(['지역','공급유형','공가수','자격유형'],axis=1)
test = test.drop(['지역','공급유형','공가수','자격유형'],axis=1)
```

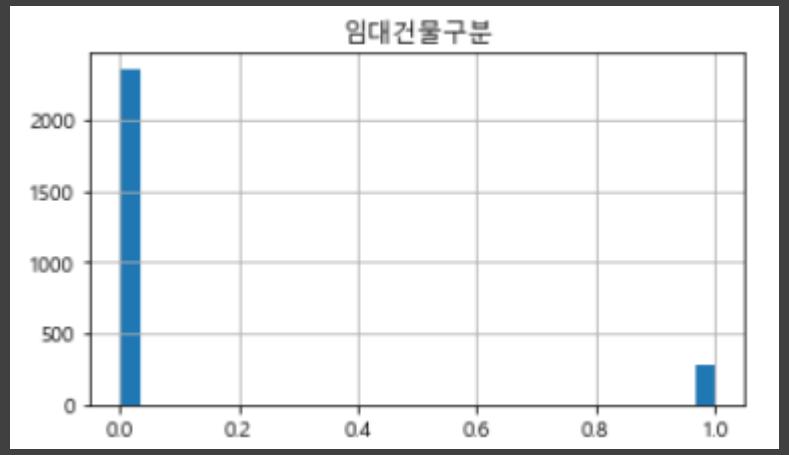
임대건물 구분 숫자데이터 변환

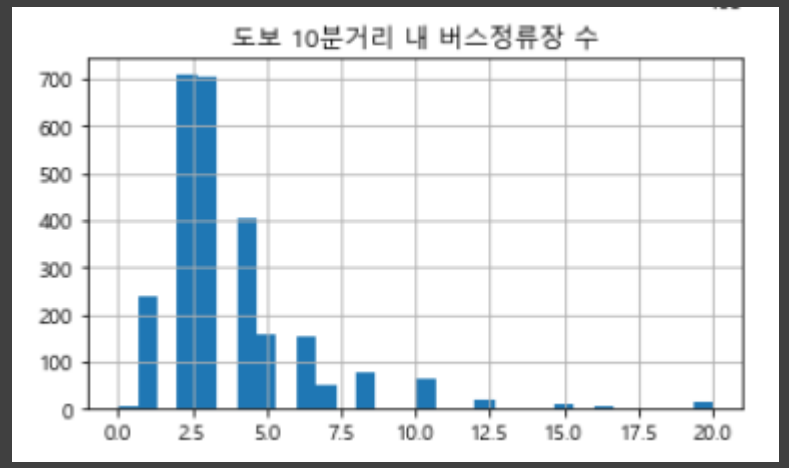
1. 아파트는 0, 상가는 1 로 변환

```
train['임대건물구분'] = train['임대건물구분'].map({'아파트':0, '상가':1})
test['임대건물구분'] = test['임대건물구분'].map({'아파트':0, '상가':1})
```

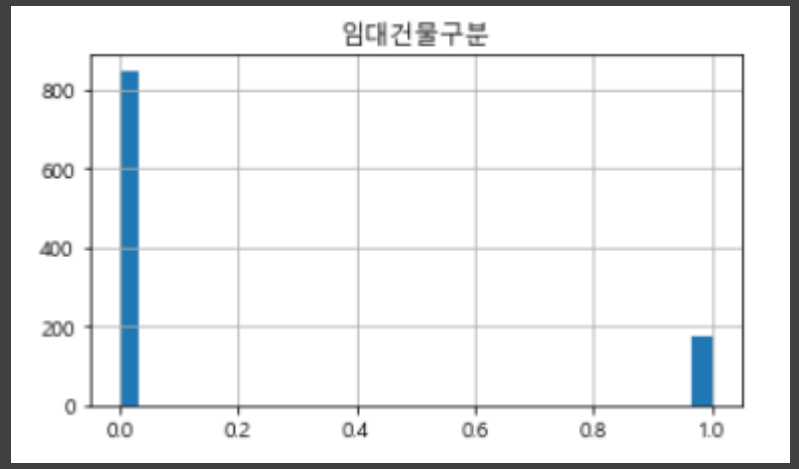
히스토그램

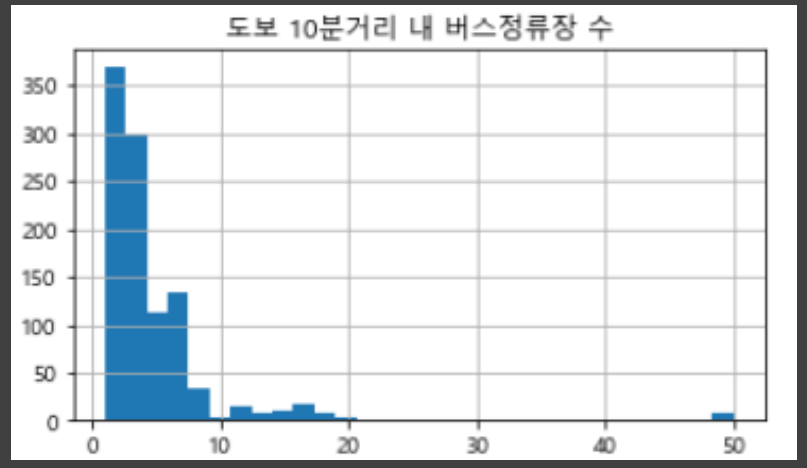
h = train.hist(bins=30,figsize=(20,15))





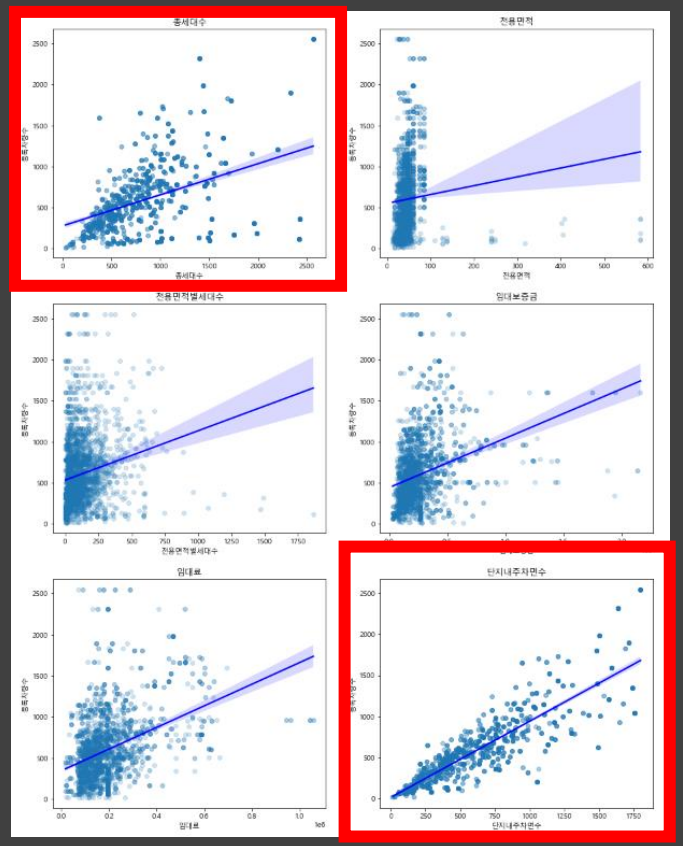
h = test.hist(bins=30,figsize=(20,15))





데이터 시각화

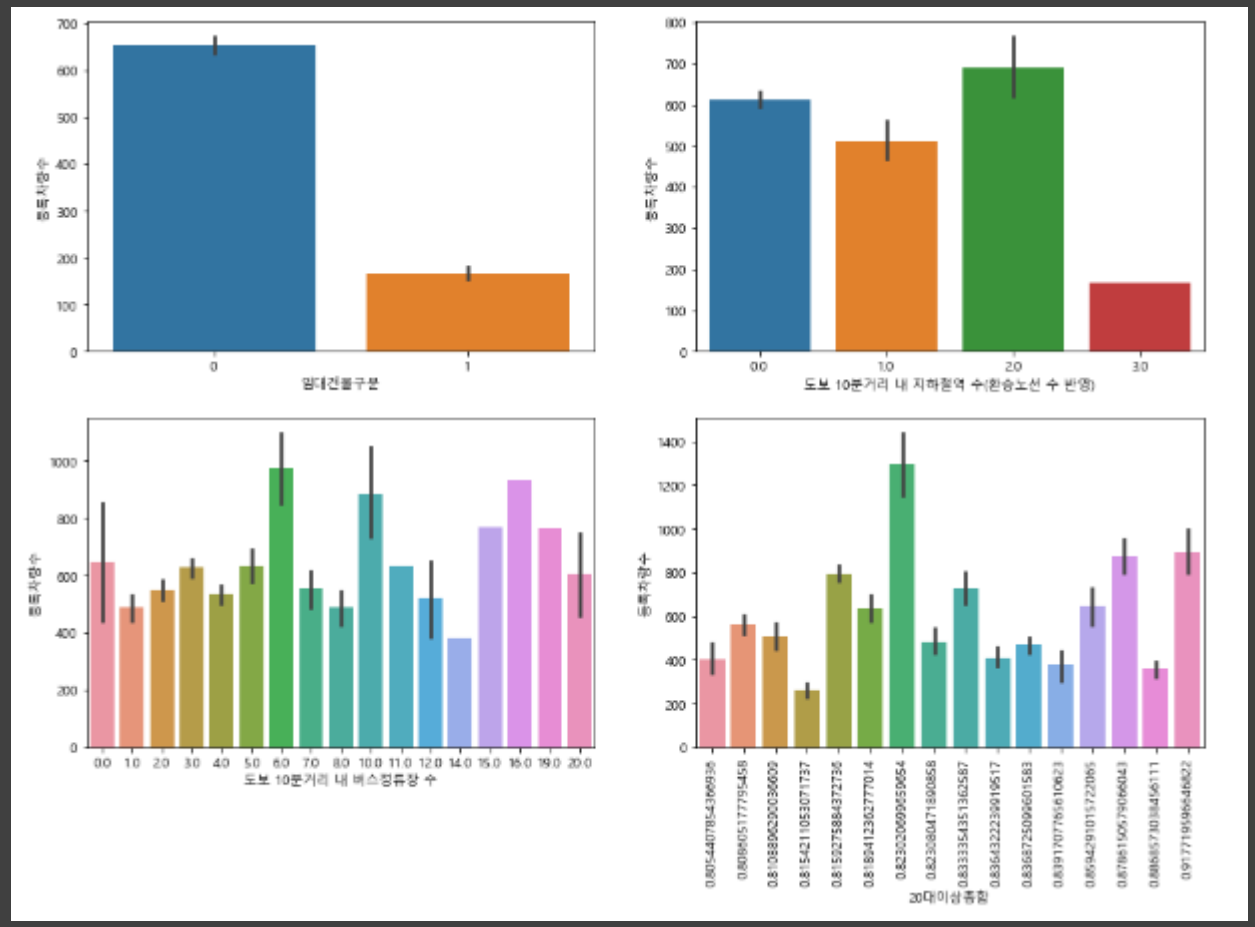
산점도 (연속형 데이터)



등록차량수와 상관관계 高

총 세대수, 단지내 주차면수

Bar 그래프 (범주형 데이터)



1. 문제 정의
2. 탐색적 데이터 분석
3. 데이터 전처리

4. 베이스라인 모델

5. 성능 평가
6. 성능 올리기
7. 최종 결론

베이스라인 모델

```
# 단지코드 값이 문자열데이터이기 때문에 이를 제외하고 실시
x_train2 = x_train.drop(['단지코드'],axis=1)
```

```
# MAE (Mean Absolute Error) 평가
```

7개 모델
선형회귀
SGD Regressor
다항회귀(LinearRegression)
다항회귀(SGD)
릿지모델
베깅
랜덤포레스트

베이스라인 모델

선형회귀 (LinearRegression)

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
score = cross_val_score(lin_reg,x_train2,y_train,cv=5,scoring='neg_mean_absolute_error')
np.mean(-score)
```

166.69

베이스라인 모델

선형회귀 (Kford)

```
from sklearn.model_selection import KFold
mae_lis = []
lin_reg = LinearRegression()
kfold = KFold(n_splits=5)
for train_index, valid_index in kfold.split(x_train2):
    # 훈련(학습)
    x_train_2, y_train_2 = x_train2.iloc[train_index], y_train.iloc[train_index] # 4/5
    x_valid, y_valid = x_train2.iloc[valid_index], y_train.iloc[valid_index] # 1/5
    lin_reg.fit(x_train_2, y_train_2)
    # 예측과 평가(정확도)
    pred = lin_reg.predict(x_valid)
    mae = mean_absolute_error(y_valid, pred)
    mae_lis.append(mae)
print("평균 mae: ", np.mean(mae_lis))
```

166.69

베이스라인 모델

SGD Regressor

```
from sklearn.linear_model import SGDRegressor

sgd_reg = SGDRegressor(penalty='None', random_state=42)

# 정규화
from sklearn.preprocessing import StandardScaler
std_scaler = StandardScaler()
x_train_scaled = std_scaler.fit_transform(x_train2)
scores = cross_val_score(sgd_reg, x_train_scaled, y_train, scoring="neg_mean_absolute_error", cv=5)
np.mean(-scores)
```

168.05

베이스라인 모델

다항회귀(LinearRegression)

```
from sklearn.preprocessing import PolynomialFeatures
poly_feature = PolynomialFeatures(degree = 2 , include_bias = False)
x_train_poly = poly_feature.fit_transform(x_train2)
```

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
score = cross_val_score(lin_reg,x_train_poly,y_train,cv=5,scoring='neg_mean_absolute_error')
np.mean(-score)
```

316.14

베이스라인 모델

다항회귀(SGD)

```
from sklearn.linear_model import SGDRegressor
from sklearn.preprocessing import PolynomialFeatures
poly_feature = PolynomialFeatures(degree = 2, include_bias = False)
x_train_poly = poly_feature.fit_transform(x_train2)
from sklearn.linear_model import SGDRegressor
sgd_reg = SGDRegressor(penalty='None', random_state=42)

# STD Scale (표준화)
from sklearn.preprocessing import StandardScaler
std_scaler = StandardScaler()
x_train_poly_scaled = std_scaler.fit_transform(x_train_poly)
scores = cross_val_score(sgd_reg, x_train_poly_scaled, y_train, scoring="neg_mean_absolute_error", cv=5)
np.mean(-scores)
```

201.68

베이스라인 모델

릿지

```
from sklearn.linear_model import Ridge

rid_params = {'alpha':[1,5,7,10,13]}
ridge = Ridge()
gridsearch_ridge = GridSearchCV(ridge,rid_params,scoring='neg_mean_absolute_error'
                                ,cv= 5,n_jobs=-1)
gridsearch_ridge.fit(x_train2,y_train)

cvres = gridsearch_ridge.cv_results_

for mean_score, params in zip(cvres['mean_test_score'], cvres['params']):
    print(-mean_score, params) # rmsle와 그 때의 하이퍼 파라미터
```

166.14

베이스라인 모델

베깅

```
from sklearn.ensemble import BaggingRegressor

bag = BaggingRegressor(LinearRegression())
bag_params = {'n_estimators':[300,500,700,1000],'max_samples':[100,200,300]}
gridsearch_bag = GridSearchCV(bag,bag_params,scoring='neg_mean_absolute_error'
                              ,cv= 5,n_jobs=-1)
gridsearch_bag.fit(x_train2,y_train)

cvres = gridsearch_bag.cv_results_

for mean_score, params in zip(cvres['mean_test_score'], cvres['params']):
    print(-mean_score, params) # rmsle와 그 때의 하이퍼 파라미터
```

169.57

베이스라인 모델

랜덤포레스트

```
# 여러번 시도 후 뽑아낸것(100~1,000) => 5,200이 최고성능

from sklearn.ensemble import RandomForestRegressor
f_params = {'n_estimators':[200,500,800],'max_depth':[3,5,7,10,30,50]},

forest = RandomForestRegressor(random_state=42)
gridsearch_forest = GridSearchCV(forest,f_params,scoring='neg_mean_absolute_error' ,cv= 5,n_jobs=-1)
gridsearch_forest.fit(x_train2,y_train)

cvres = gridsearch_forest.cv_results_

for mean_score, params in zip(cvres['mean_test_score'], cvres['params']):
    print(-mean_score, params) # rmsle와 그 때의 하이퍼 파라미터
```

167.78

베이스라인 모델

결과

모델	MAE 값
선형회귀	166.69
SGD Regressor	168.05
다항회귀(LinearRegression)	316.14
다항회귀(SGD)	201.68
릿지모델	166.14
베깅	169.57
랜덤포레스트	167.78

① 선형회귀, ② 랜덤포레스트, ③ 릿지모델을 기본 모델로 선정

베이스라인 모델

결과

cross_val_predict를 이용해 답이 잘 추출되는 지 확인

x_train을 LinearRegression에 적용

```
from sklearn.model_selection import cross_val_predict
cross_val_predict(lin_reg,   x_train2,y_train,cv=5)
```

```
array([1203.52077513, 1199.1594247 , 1286.53885636, ...,  208.49741102,
        207.8877332 ,  213.75492185])
```

```
y_train # 비교모델(진짜 답)
```

```
0       1015.0
1       1015.0
2       1015.0
3       1015.0
4       1015.0
         ...
2627     146.0
2628     146.0
2629     146.0
2630     146.0
2631     146.0
Name: 등록차량수, Length: 2632, dtype: float64
```

단지코드가 같아도 같은 답이 나오지 않음 → 단지코드 데이터 활용

1. 문제 정의
2. 탐색적 데이터 분석
3. 데이터 전처리
4. 베이스라인 모델

5. 성능 평가

6. 성능 올리기
7. 최종 결론

1. 단지 코드 ONE HOT ENCODING

단지 코드 → ONE HOT 인코딩

```
from sklearn.preprocessing import OneHotEncoder
onehot = OneHotEncoder(sparse=False)
x_train_code=onehot.fit_transform(x_train[['단지코드']])
x_train_code
onehot.categories_
```

데이터의 컬럼수가 달라짐

단지코드로 그룹화 후 진행

2. 단지코드 그룹화

```
# y값도 그룹화 하기 위해 x, y분리 전 데이터를 이용

train_group = pd.DataFrame(train.groupby(['단지코드']).mean())

x_train_group = train_group.drop(['등록차량수'],axis=1)
y_train_group = train_group['등록차량수']
```

2. 단지코드 그룹화

그룹화한 데이터에 모델 적용하기

1. 선형회귀

```
lin_reg = LinearRegression()
score =
cross_val_score(lin_reg,x_train_group,y_train_group,cv=5,scoring='neg_mean_absolute_error')
np.mean(-score)
```

130.96

2. 단지코드 그룹화

그룹화한 데이터에 모델 적용하기

2. 랜덤포레스트

```
score =
cross_val_score(best_forest,x_train_group,y_train_group,cv=5,scoring='neg_mean_absolute_error')
np.mean(-score)
```

137.88

성능 평가

2. 단지코드 그룹화

그룹화한 데이터에 모델 적용하기

3. 릿지

```
score =
cross_val_score(best_ridge,x_train_group,y_train_group,cv=5,scoring='neg_mean_absolute_error')
np.mean(-score)
```

BEST

130.78

3. 단지코드와 임대건물 구분 후 그룹화

```
# y값도 그룹화 하기 위해 x, y분리 전 데이터를 이용
train_group2 = pd.DataFrame(train.groupby(['단지코드','임대건물구분'],as_index=False).mean())

x_train_group2 = train_group2.drop(['등록차량수'],axis=1)
y_train_group2 = train_group2['등록차량수']

# 단지코드 이용을 위해 ONE HOT ENCODING
code2 = onehot.fit_transform(x_train_group2[['단지코드']])
code_data = pd.DataFrame(code2,columns=onehot.categories_)
x_train_pcode2 = pd.concat([code_data,x_train_group2],axis = 1)
```

3. 단지코드와 임대건물 구분 후 그룹화

그룹화한 데이터에 모델 적용하기

1. 선형회귀

```
lin_reg = LinearRegression()
score =
cross_val_score(lin_reg,x_train_pcode2,y_train_group2,cv=5,scoring='neg_mean_absolute_error')
np.mean(-score)
```

138.02

3. 단지코드와 임대건물 구분 후 그룹화

그룹화한 데이터에 모델 적용하기

2. 랜덤포레스트

```
score =
cross_val_score(best_forest,x_train_pcode2,y_train_group2,cv=5,scoring='neg_mean_absolute_error')
np.mean(-score)
```

131.93

3. 단지코드와 임대건물 구분 후 그룹화

그룹화한 데이터에 모델 적용하기

3. 릿지

```
score =
cross_val_score(best_ridge,x_train_pcode2,y_train_group2,cv=5,scoring='neg_mean_absolute_error')
np.mean(-score)
```

BEST

129.99

성능 평가

TEST 값 추출

```
# test 데이터 가공
test_group_1 = pd.DataFrame(test.groupby(['단지코드']).mean())

lin_reg.fit(x_train_group,y_train_group)
y_pred = lin_reg.predict(test_group_1)

# y_pred를 다시 test데이터의 단지코드에 맞춰 분산시켜줘야함
# 이를 위해 test데이터와 y_pred데이터를 병합
test_group_1['y_pred'] = y_pred
```

단지코드	총세대수	임대건물구분	전용면적
C1003	480.0	0.000000	32.136250
C1006	1505.0	0.740741	60.894074
C1016	643.0	0.000000	48.142500
C1019	321.0	0.000000	44.875000
C1030	75.0	0.000000	26.338333
...	...	...	...

성능 평가

TEST 값 추출

제출결과

113점

```
# merge를 위해 단지코드와 y_pred로 이루어진 데이터프레임을 생성
test_group_code_y = pd.DataFrame({'num':test_group_1['y_pred'].values,'단지코드
':test_group_1.index})
test_group_code_y

# 최종 테스트 데이터 작성
test_pryd_group1
= pd.merge(test, test_group_code_y,
                    how='left', left_on='단지코드', right_on='단지코드')
```

	num	단지코드
0	285.579942	C1003
1	352.222801	C1006
2	642.279935	C1016
3	292.872666	C1019
4	45.386937	C1030
...	...	...

1. 문제 정의
2. 탐색적 데이터 분석
3. 데이터 전처리
4. 베이스라인 모델
5. 성능 평가

6. 성능 올리기

7. 최종 결론

성능 올리기

2차 시도

```
# test 데이터에 버스정류장이 50개인 말도 안되는 값이 존재

# 버스정류장 수의 평균값인 4로 변경


test_group_2 = pd.DataFrame(test.groupby(['단지코드']).mean())

test_group_2[test_group_2['도보 10분거리 내 버스정류장 수']!=50]['도보 10분거리 내
버스정류장 수'].mean()


test_group_2[test_group_2['도보 10분거리 내 버스정류장 수']==50] = 4.0
```

성능 올리기

2차 시도: 제출용 샘플

제출결과

149점

```
sample = pd.read_csv('D:₩data/sample_submission.csv')

sample=sample.drop(['num'],axis=1)


# 최종 테스트 데이터 작성

samples = pd.merge(sample,test_group_code_y2,how='left',left_on='code',right_on='단지코드')

samples = samples.drop(['단지코드'], axis=1)

samples.to_csv('2번시도.csv', index=False)
```

성능 올리기

3차 시도

제출결과

112점

단지코드와 임대건물구분의 값 이용하기

단지코드의 값을 숫자형 자료로 바꾸기

코드 앞 C값을 제거 후 숫자형 자료로 변경

모델	값
선형회귀	128.15
랜덤 포레스트	131.92
릿지	129.99

성능 올리기

4차 시도

제출결과

120점

전용면적이 넓은 집일수록 가족구성원의 수가 늘어 더 많은 차량을 소유 할 확률이 높다고 판단

전용면적을 cut을 이용해 그룹으로 나누고, 해당 그룹에 속하는 세대 수의 개수를 구해서 컬럼 추가

모델	값
선형회귀	130.57
랜덤 포레스트	135.53
릿지	130.84

성능 올리기

5차 시도

제출결과

113점

전용면적별 세대수는 총세대수와 연관되어 있음 → 전용면적별세대수 삭제

임대보증금은 임대료와 상관관계가 높음 → 임대보증금 삭제

모델	값
선형회귀	127.72
랜덤 포레스트	131.36
릿지	127.42

1. 문제 정의
2. 탐색적 데이터 분석
3. 데이터 전처리
4. 베이스라인 모델
5. 성능 평가
6. 성능 올리기

7. 최종 결론

최종 결과

DACON 제출 결과

	기준	제출 결과
1차 시도	**단지코드 그룹화**	113점
2차 시도	**일부 데이터 값 수정** 버스정류장 수 → 평균값으로 수정	149점
3차 시도	**단지코드와 임대건물구분의 값 이용** 그룹화+int자료형	112점
4차 시도	**전용면적 및 전용면적별 세대 수 활용**	120점
5차 시도	**3차 시도 일부 데이터 값 수정** 전용면적별 세대수, 임대보증금 삭제	113점

최종 결과

1차 시도

	A	B
1	code	num
2	C1072	713.5358
3	C1128	1142.774
4	C1456	575.4207
5	C1840	505.6778
6	C1332	1074.535
7	C1563	1586.651
8	C1794	975.596
9	C1640	471.4512
10	C1377	359.2176
11	C2072	249.8292
12	C1472	424.044
13	C1006	352.2228
14	C1083	423.8607
15	C1311	271.3407

3차 시도

	A	B
1	code	num
2	C1072	711.4473
3	C1128	1137.415
4	C1456	581.2697
5	C1840	504.5002
6	C1332	1070.049
7	C1563	1576.622
8	C1794	964.1518
9	C1640	457.7658
10	C1377	353.2559
11	C2072	247.6431
12	C1472	418.3434
13	C1006	326.7992
14	C1006	333.0287
15	C1083	418.6248

5차 시도

	A	B
1	code	num
2	C1072	702.5406
3	C1128	1130.293
4	C1456	564.4784
5	C1840	508.0273
6	C1332	1062.347
7	C1563	1563.069
8	C1794	966.7082
9	C1640	448.2436
10	C1377	341.2957
11	C2072	245.7852
12	C1472	418.5887
13	C1006	313.151
14	C1006	329.0587
15	C1083	409.9358

1등 데이터

	A	B
1	code	num
2	C1072	715.4763
3	C1128	1213.099
4	C1456	503.1091
5	C1840	541.2139
6	C1332	1152.392
7	C1563	1871.871
8	C1794	1075.322
9	C1640	531.5664
10	C1377	371.9539
11	C2072	272.0441
12	C1472	458.0953
13	C1006	296.5733
14	C1083	436.775
15	C1311	263.3441

단지코드

C1072

713대

711대

702대

715대


P
감사합니다