회귀분석 프로젝트
최종발표
(1조)

임현수
최민권

목차

데이터 소개

데이터 Load & 확인

```
usedcar_origin = pd.read_csv("datas/train-data.csv", index_col=0)
usedcar_origin.head()
```

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	Price
0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg	998 CC	58.16 bhp	5.0	NaN	1.75
1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	5.0	NaN	12.50
2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2 kmpl	1199 CC	88.7 bhp	5.0	8.61 Lakh	4.50
3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77 kmpl	1248 CC	88.76 bhp	7.0	NaN	6.00
4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	NaN	17.74

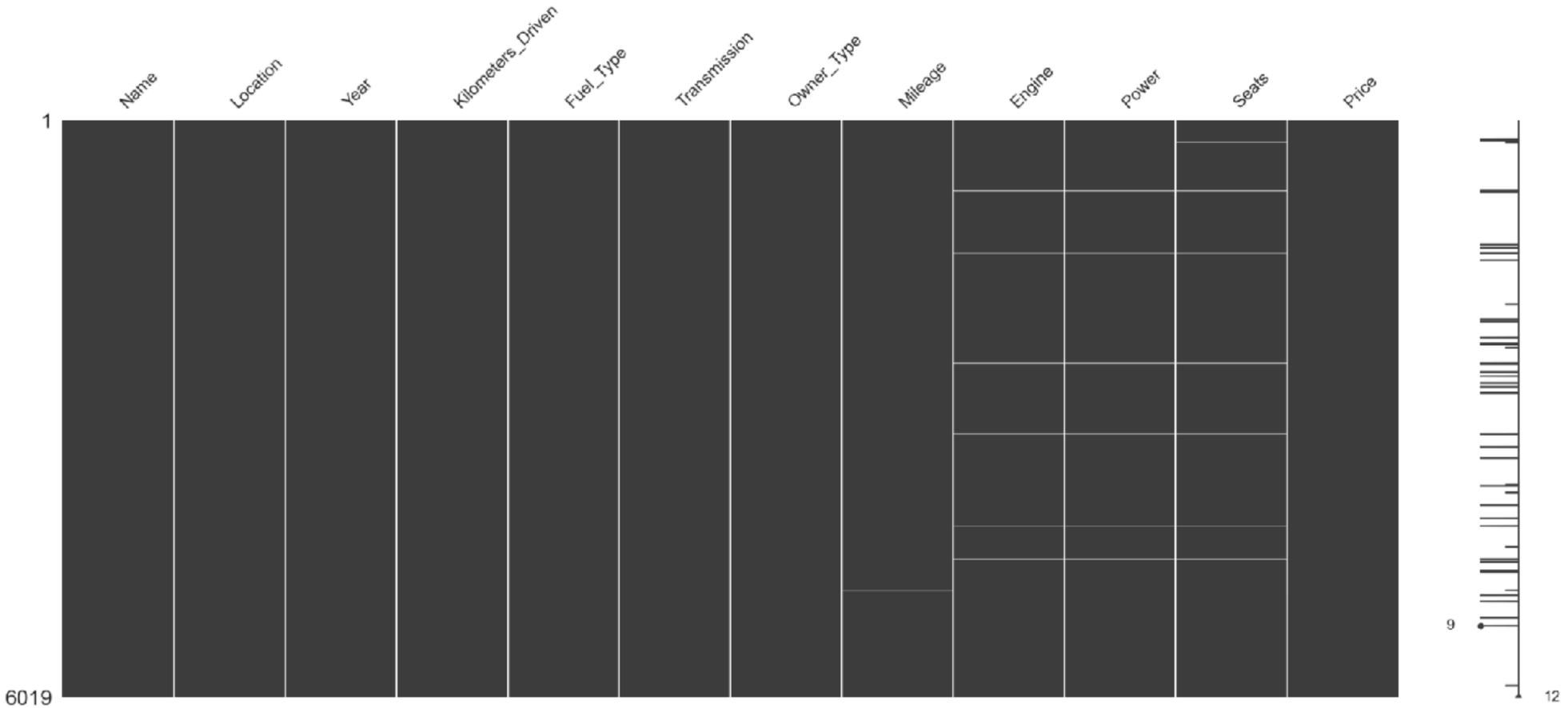
```
usedcar_origin.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6019 entries, 0 to 6018
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Name               6019 non-null   object 
 1   Location           6019 non-null   object 
 2   Year               6019 non-null   int64  
 3   Kilometers_Driven  6019 non-null   int64  
 4   Fuel_Type          6019 non-null   object 
 5   Transmission       6019 non-null   object 
 6   Owner_Type         6019 non-null   object 
 7   Mileage            6017 non-null   object 
 8   Engine             5983 non-null   object 
 9   Power              5983 non-null   object 
 10  Seats              5977 non-null   float64
 11  New_Price          824 non-null    object 
 12  Price              6019 non-null   float64
dtypes: float64(2), int64(2), object(9)
memory usage: 658.3+ KB
```

전처리 & 시각화

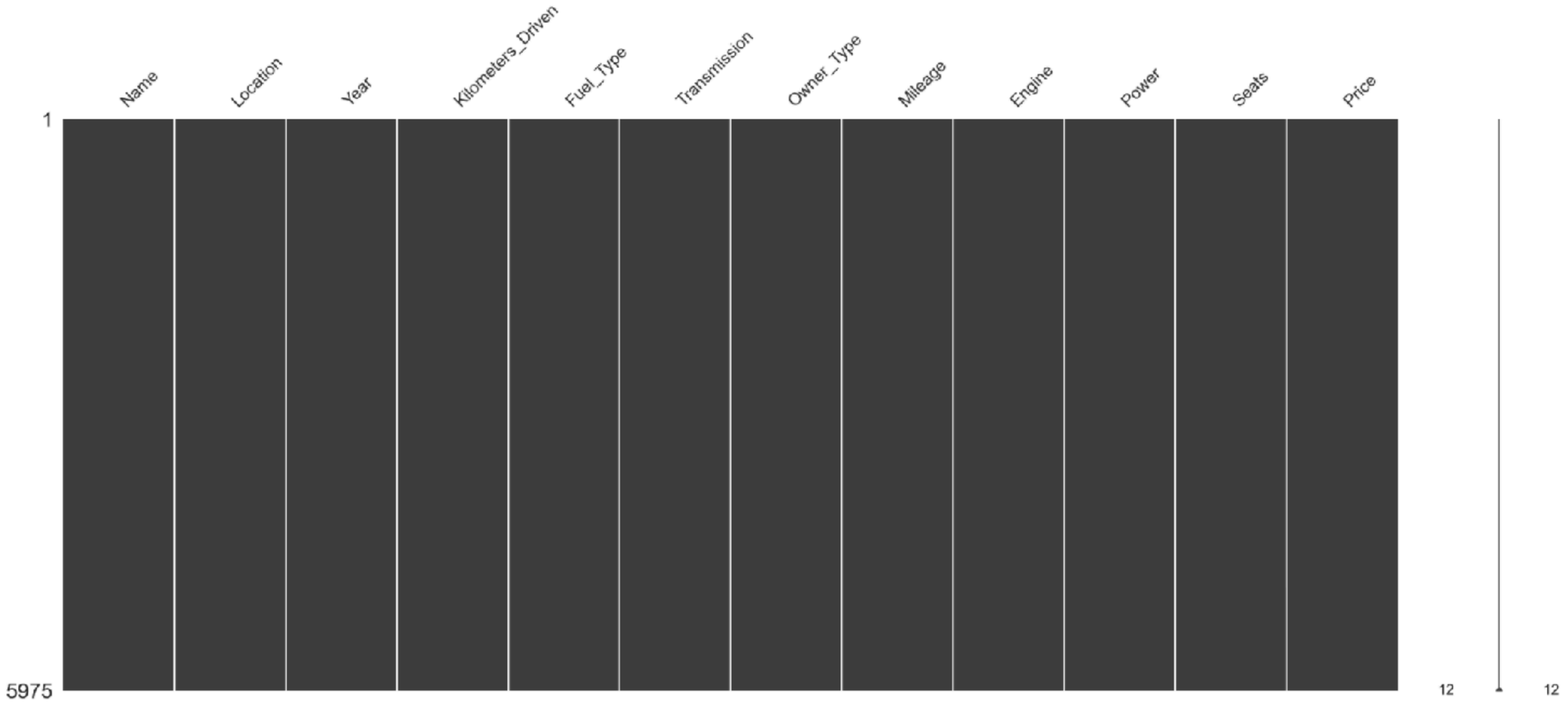
결측치 확인

```
# New_price는 drop
usedcar_origin.drop('New_Price', axis=1, inplace=True)
msno.matrix(usedcar_origin);
```



결측치 제거

```
# 44/6019 = 0.7%. 적은 부분이라 결측값 44개는 삭제 결정
usedcar_origin.dropna(inplace=True)
msno.matrix(usedcar_origin);
```



형 변환

```
usedcar_origin = pd.read_csv("datas/train-data.csv", index_col=0)
usedcar_origin.head()
```

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	Price
0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg	998 CC	58.16 bhp	5.0	NaN	1.75
1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	5.0	NaN	12.50
2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2 kmpl	1199 CC	88.7 bhp	5.0	8.61 Lakh	4.50
3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77 kmpl	1248 CC	88.76 bhp	7.0	NaN	6.00
4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	NaN	17.74

```
# Mileage, Engine, Power 숫자화
# 우선 2개 (Mileage, Engine)부터 숫자화
# Mileage
usedcar_origin["Mileage"] = usedcar_origin["Mileage"].str.split(" ", expand=True)[0].str.strip().astype(float)
# Engine
usedcar_origin["Engine"] = usedcar_origin["Engine"].str.split(" ", expand=True)[0].str.strip().astype(float)
```

형 변환

```
usedcar_origin["Power"].astype("float")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-1813-af984430e214> in <module>
----> 1 usedcar_origin["Power"].astype("float")

/opt/anaconda3/envs/fc14/lib/python3.7/site-packages/pandas/core/generic.py in astype(self, dtype, copy, errors)
   5870         else:
   5871             # else, only a single dtype is given
-> 5872             new_data = self._mgr.astype(dtype=dtype, copy=copy, errors=errors)
   5873             return self._constructor(new_data).__finalize__(self)
   5874

/opt/anaconda3/envs/fc14/lib/python3.7/site-packages/pandas/core/internals/managers.py in astype(self, dtype, copy, errors)
    629         self, dtype, copy: bool = False, errors: str = "raise"
    630     ) -> "BlockManager":
--> 631         return self.apply("astype", dtype=dtype, copy=copy, errors=errors)
    632
    633     def convert(

/opt/anaconda3/envs/fc14/lib/python3.7/site-packages/pandas/core/internals/managers.py in apply(self, f, filter, **kwargs)
    425                     applied = b.apply(f, **kwargs)
    426                 else:
--> 427                     applied = getattr(b, f)(**kwargs)
    428             except (TypeError, NotImplementedError):
    429                 if not ignore_failures:

/opt/anaconda3/envs/fc14/lib/python3.7/site-packages/pandas/core/internals/blocks.py in astype(self, dtype, copy, errors)
    671             vals1d = values.ravel()
    672             try:
--> 673                 values = astype_nansafe(vals1d, dtype, copy=True)
    674             except (ValueError, TypeError):
    675                 # e.g. astype_nansafe can fail on object

/opt/anaconda3/envs/fc14/lib/python3.7/site-packages/pandas/core/dtypes/cast.py in astype_nansafe(arr, dtype, copy, skipna)
   1095     if copy or is_object_dtype(arr) or is_object_dtype(dtype):
   1096         # Explicit copy, or required since NumPy can't view from / to object.
-> 1097         return arr.astype(dtype, copy=True)
   1098
   1099     return arr.view(dtype)

ValueError: could not convert string to float: 'null'
```

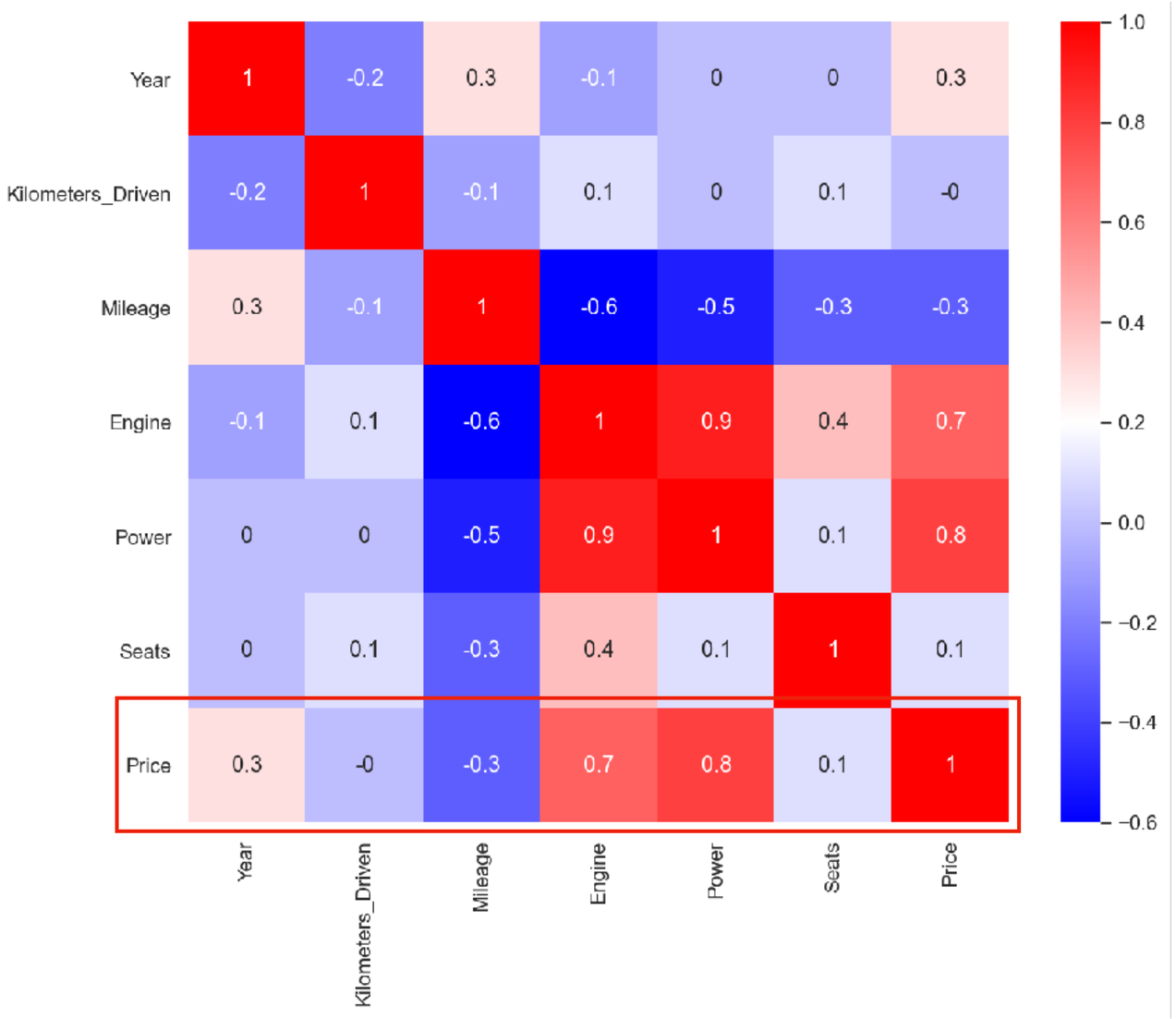
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error


tmp1 = usedcar_origin[~(usedcar_origin["Power"] == "null")]
tmp2 = usedcar_origin[(usedcar_origin["Power"] == "null")]
tmp1["Power"] = tmp1["Power"].astype(float)


lr = LinearRegression()
lr.fit(X=tmp1[["Engine"]], y=tmp1[["Power"]])
rmse =np.sqrt(mean_squared_error(tmp1[["Power"]], lr.predict(tmp1[["Engine"]])))
print(rmse)


# power가 null었던 부분에 예측값으로 대체
usedcar_origin.loc[(usedcar_origin["Power"].str.split(" ", expand=True)[0].str.strip() == "null"), "Power"] = \
lr.predict(tmp2[["Engine"]])


# Power도 새로 들어온 데이터 위해 다시 형변환
usedcar_origin["Power"] = usedcar_origin["Power"].astype(float)
```

```
26.912935568392896
```

- 회귀분석으로 power==null인 데이터 예측값으로 대치

- train data의 power~engine rmse=26.9 (ref. tmp1['power'] std=53.8)

컬럼 별 상관관계



- Power & Engine Price 와 강한 양의 상관관계

- Year는 약한 양의 상관관계

- Mileage는 약한 음의 상관관계

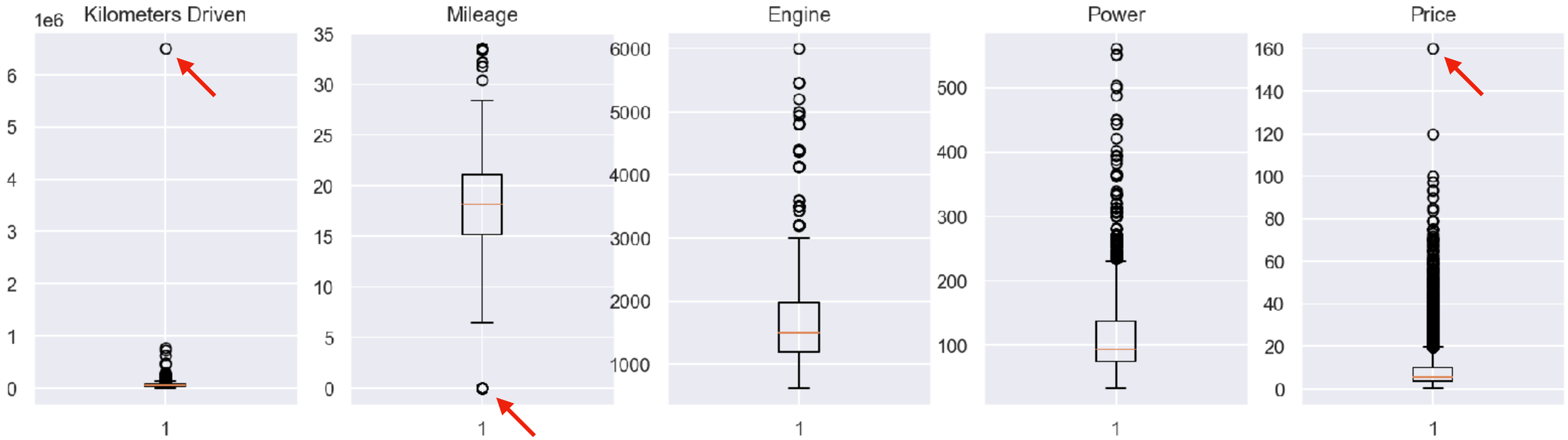
- Power 와 Engine은 강한 양의 상관관계

- Mileage는 Power & Engine은 음의 상관관계

이상치 확인

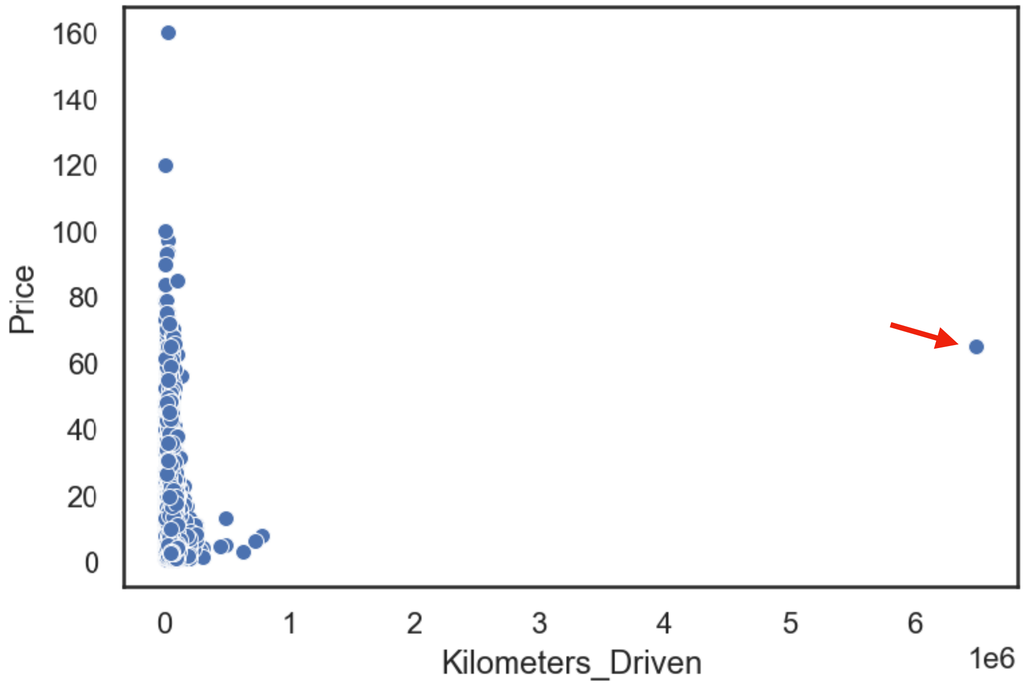
```
usedcar_origin.describe()
```

	Year	Kilometers_Driven	Mileage	Engine	Power	Seats	Price
count	5975.000000	5.975000e+03	5975.000000	5975.000000	5975.000000	5975.000000	5975.000000
mean	2013.386778	5.867431e+04	18.179408	1621.606695	112.955782	5.278828	9.501647
std	3.247238	9.155851e+04	4.521801	601.036987	53.725132	0.808959	11.205736
min	1998.000000	1.710000e+02	0.000000	624.000000	34.200000	0.000000	0.440000
25%	2012.000000	3.390800e+04	15.200000	1198.000000	75.000000	5.000000	3.500000
50%	2014.000000	5.300000e+04	18.160000	1493.000000	93.700000	5.000000	5.650000
75%	2016.000000	7.300000e+04	21.100000	1984.000000	138.100000	5.000000	9.950000
max	2019.000000	6.500000e+06	33.540000	5998.000000	560.000000	10.000000	160.000000

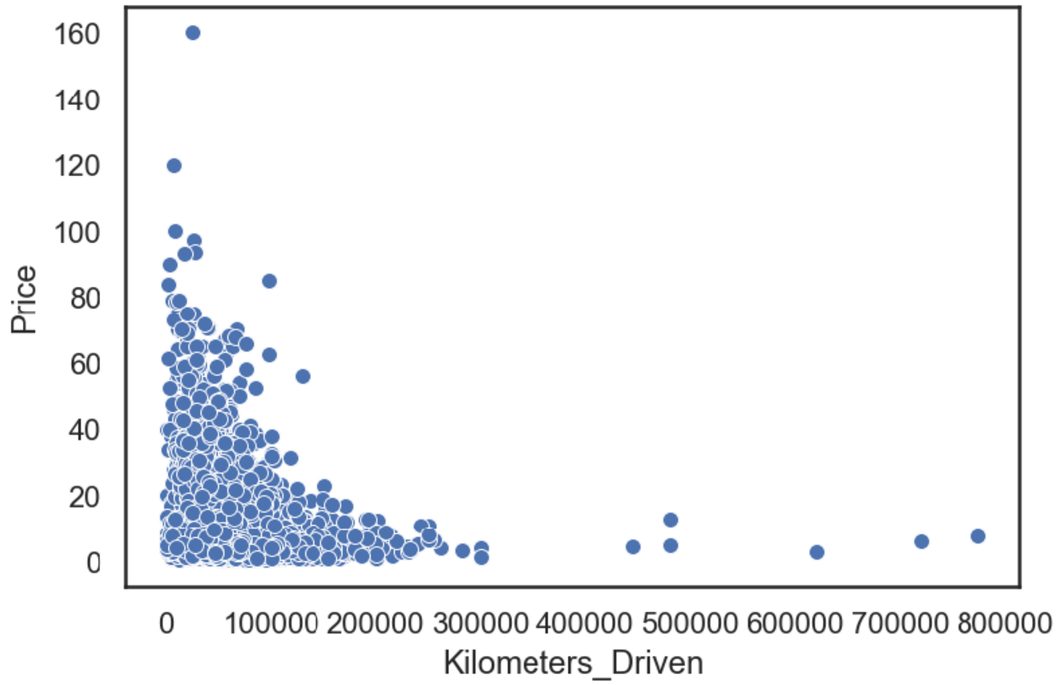


Kilometers_Driven

이상치 제거 전



이상치 제거 후




```
# 최신 2019년식일 때 2017년식인데 650만 km..? 이정도면 시속 240이상으로 24시간 3년 달려야하는거리.. 말이 안됨. 삭제~
raw_tr[raw_tr["Kilometers_Driven"]>=5000000]
```

	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	Price	Brand
2328	Chennai	2017	6500000	Diesel	Automatic	First	15.97	2993.0	258.0	5.0	65.0	bmw

Mileage, Seats

```
usedcar_origin[usedcar_origin["Mileage"]==0.0].describe()
```

	Year	Kilometers_Driven	Mileage	Engine	Power	Seats	Price
count	56.000000	56.000000	56.0	56.000000	56.000000	56.000000	56.000000
mean	2009.410714	73994.017857	0.0	1763.303571	114.706444	5.125000	11.925714
std	4.670125	47859.011314	0.0	825.287717	57.852234	0.895697	15.398214
min	2001.000000	4000.000000	0.0	799.000000	49.134459	2.000000	0.550000
25%	2006.000000	45875.000000	0.0	1086.000000	64.651309	5.000000	1.435000
50%	2009.000000	66500.000000	0.0	1164.000000	77.452710	5.000000	2.615000
75%	2012.000000	90000.000000	0.0	2245.750000	165.000000	5.000000	18.875000
max	2019.000000	227000.000000	0.0	3597.000000	262.600000	10.000000	49.240000

```
usedcar_origin["Seats"].value_counts()
```

```
5.0     5012
7.0      674
8.0      134
4.0       99
6.0       31
2.0       16
10.0       5
9.0        3
0.0        1
Name: Seats, dtype: int64
```

- Mileage → 56개 데이터. 우선 삭제. 추후 대체값 채워서 재학습 예정

- Seats가 0인 행이 하나 있음 → 해당 차종 구글링해서 5인승 확인

Name

```
usedcar_origin = pd.read_csv("datas/train-data.csv", index_col=0)
usedcar_origin.head()
```

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	Price
0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg	998 CC	58.16 bhp	5.0	NaN	1.75
1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	5.0	NaN	12.50
2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2 kmpl	1199 CC	88.7 bhp	5.0	8.61 Lakh	4.50
3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77 kmpl	1248 CC	88.76 bhp	7.0	NaN	6.00
4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	NaN	17.74

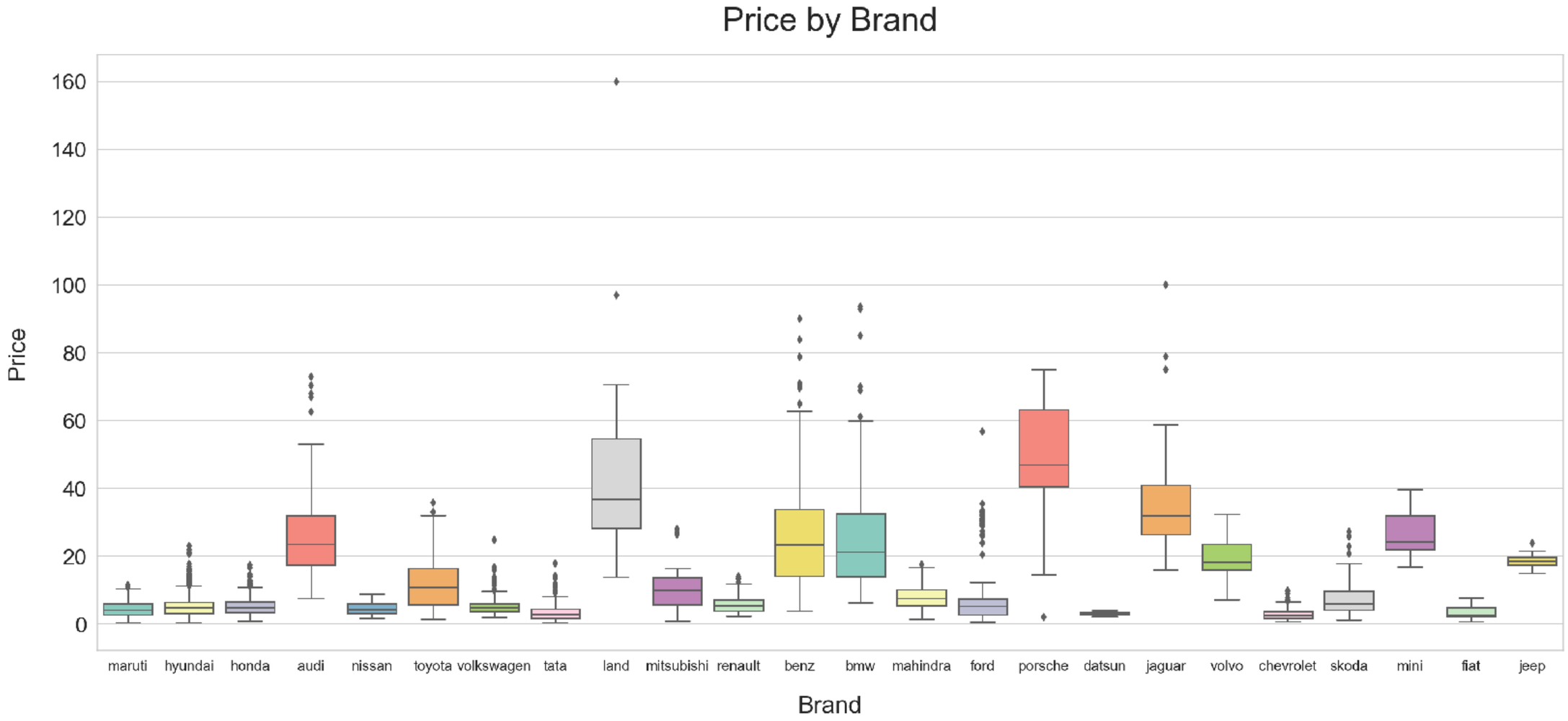
```
usedcar_origin["Brand"] = usedcar_origin["Name"].str.split(" ", expand=True)[0].\
str.lower()
```

- Name 값의 첫 부분이 브랜드명임을 파악, 해당 규칙으로 Brand 열 추가

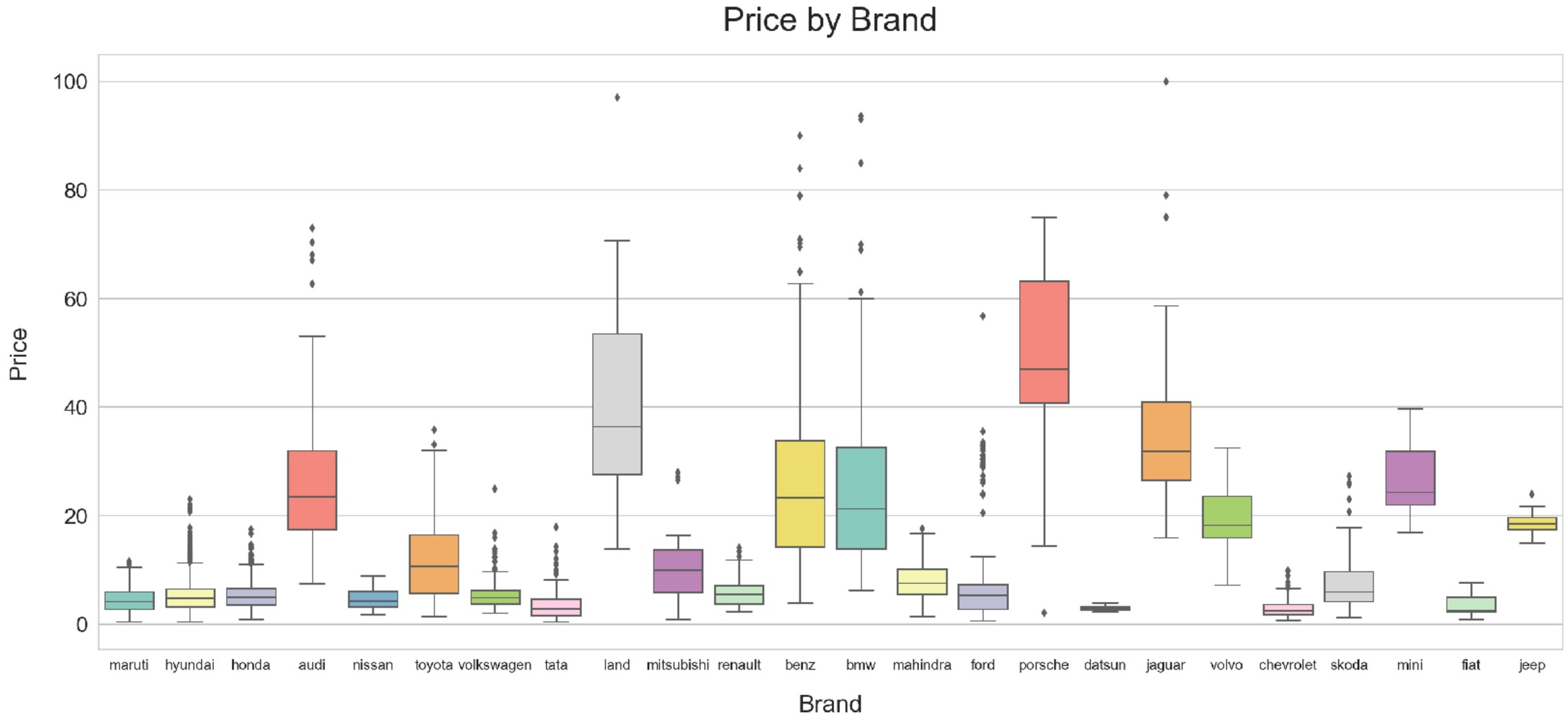
```
print(usedcar_origin["Brand"].value_counts())
ls = usedcar_origin["Brand"].value_counts()[usedcar_origin["Brand"].value_counts()<10].keys()
usedcar_origin = usedcar_origin[~(usedcar_origin["Brand"].isin(ls))]
usedcar_origin.drop(columns=["Name"], inplace=True)
```

- 데이터 수가 10 이하인 브랜드 5종 -> 원활한 학습 위해 5종 제거 후 Name 열도 drop

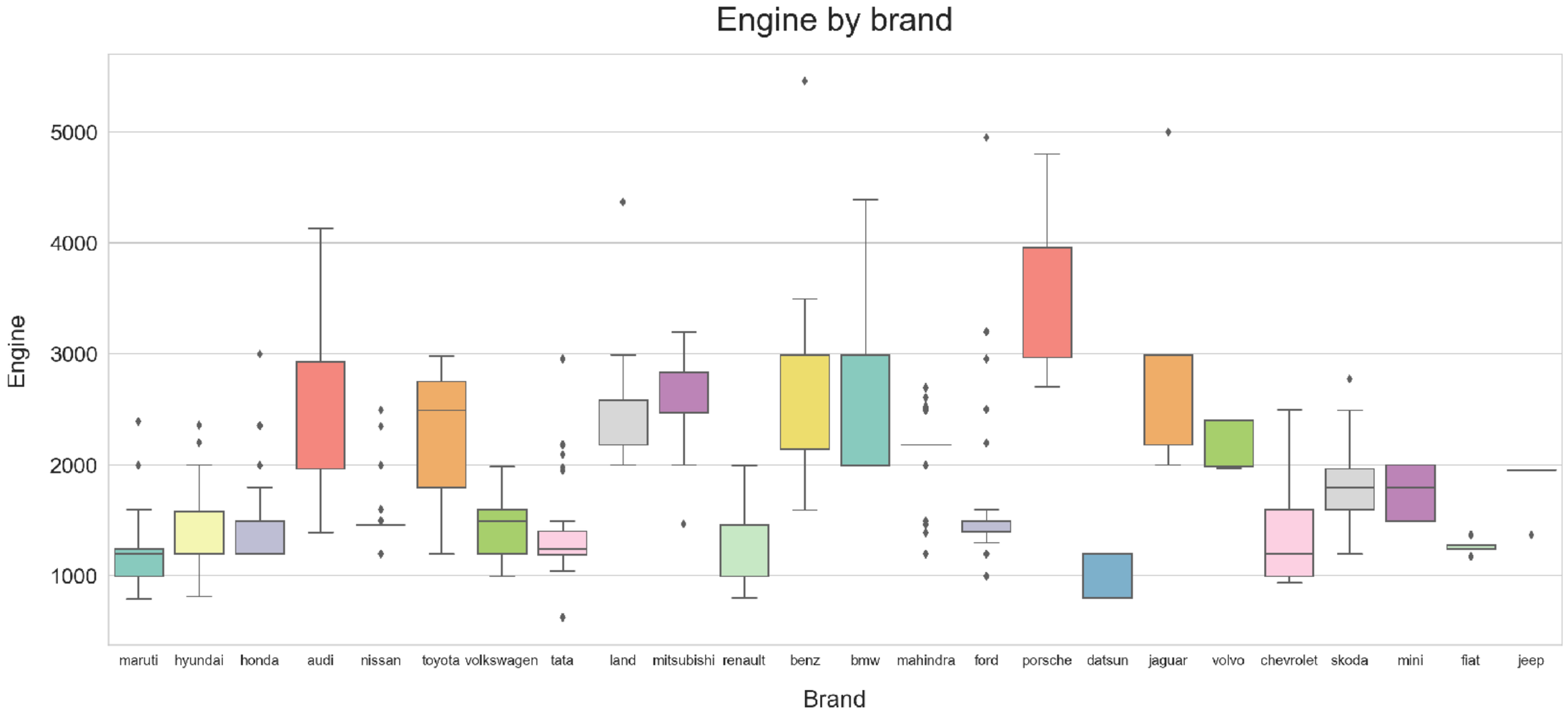
Price max값 제거 전



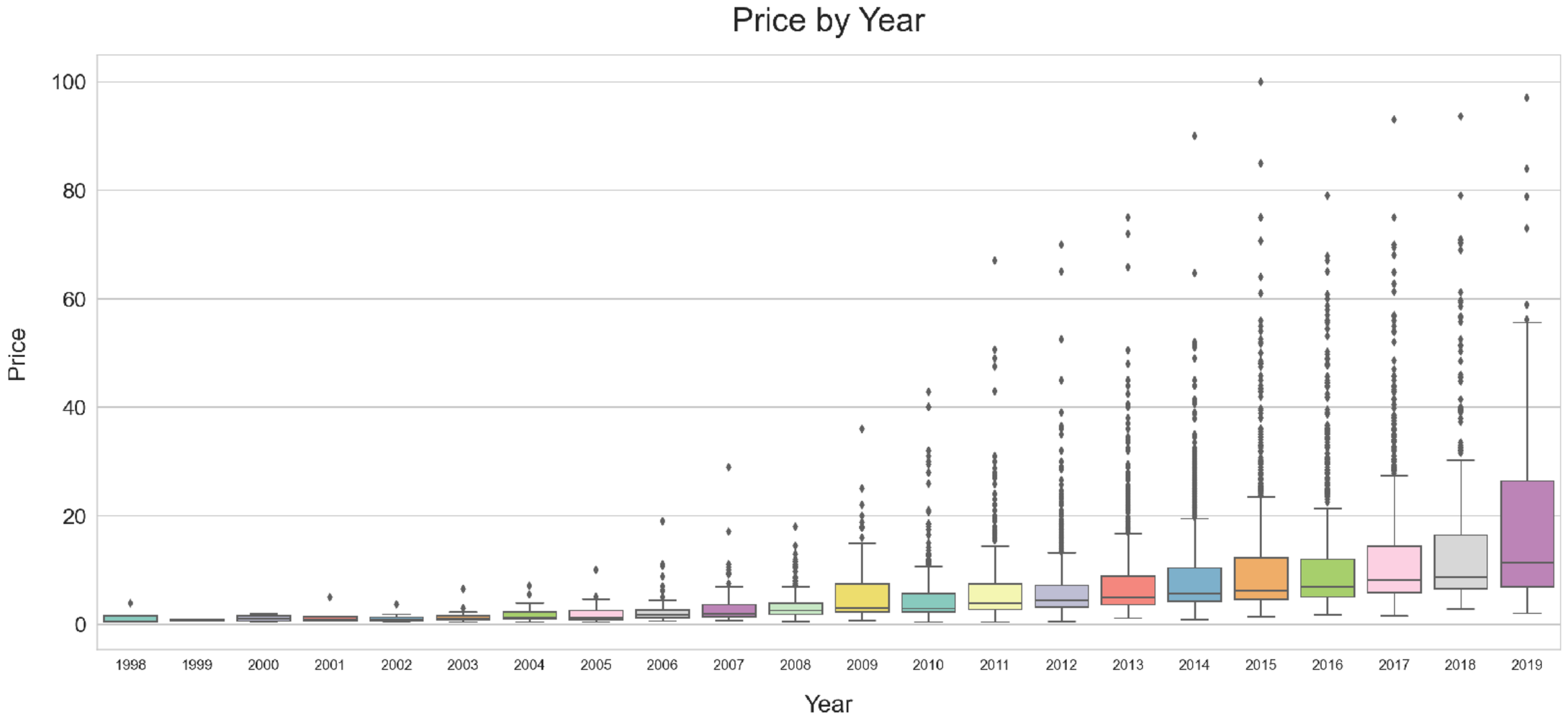
Price max값 제거 후



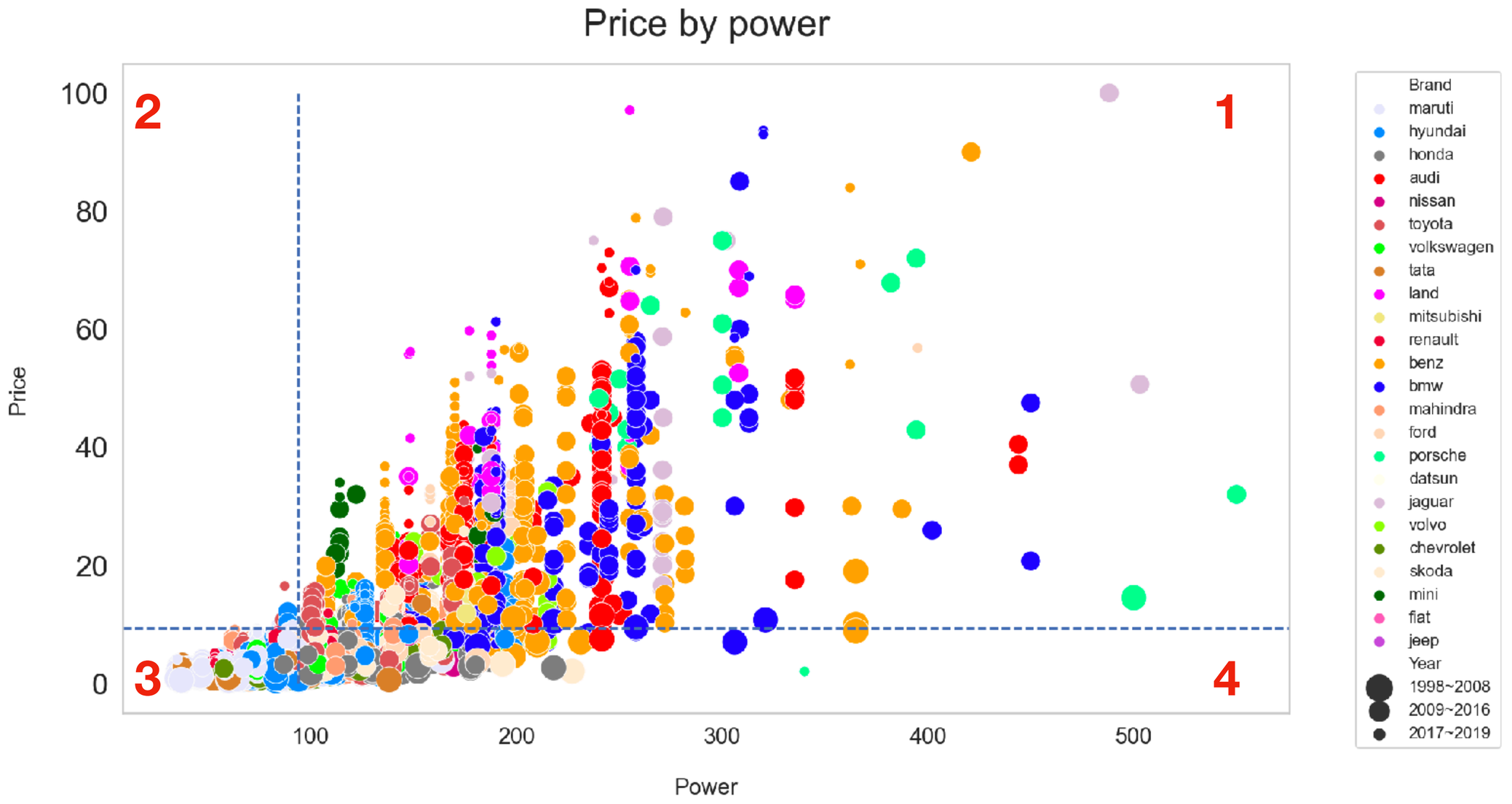
브랜드 별 엔진 성능 boxplot



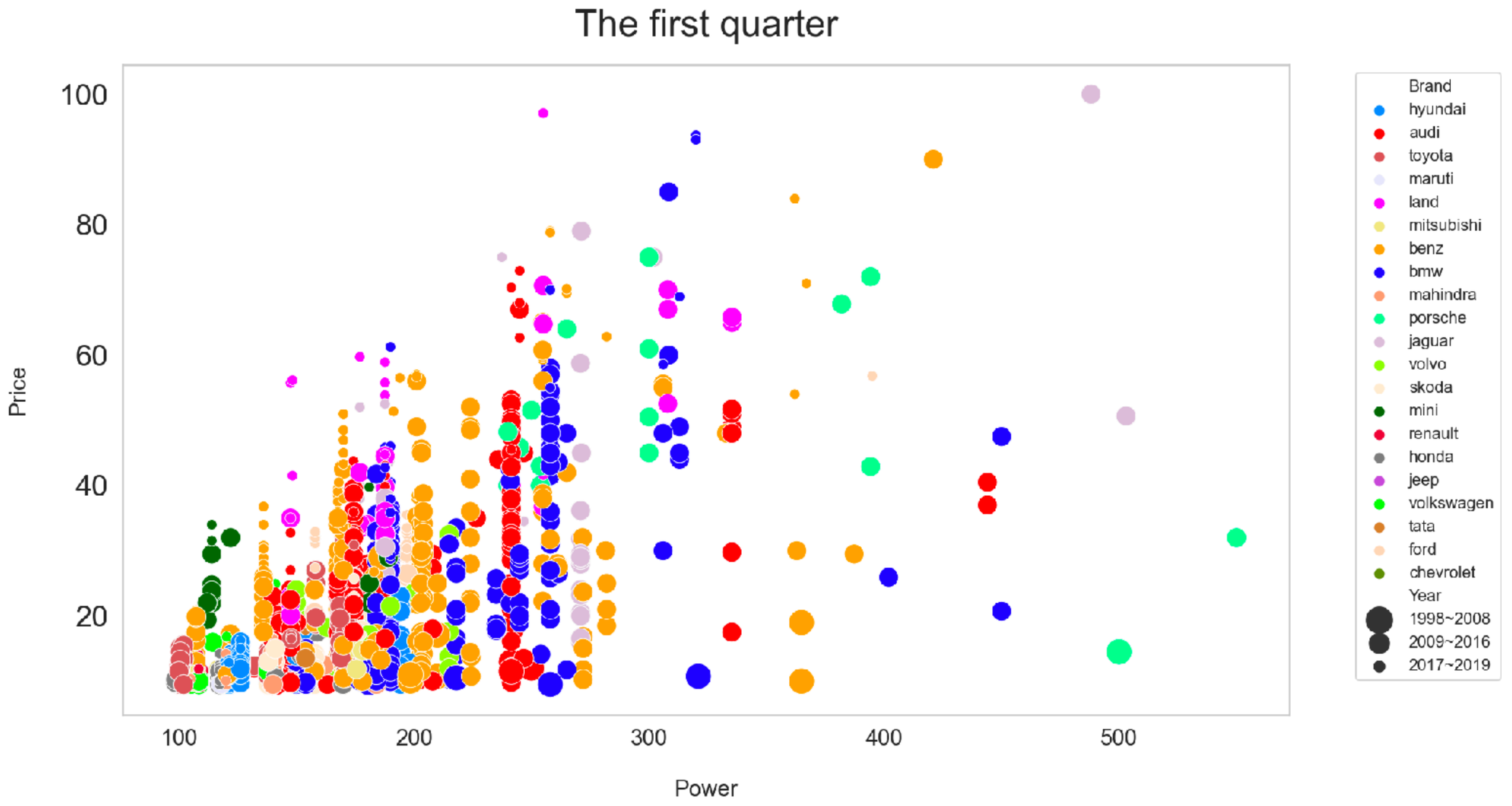
년도 별 가격 boxplot



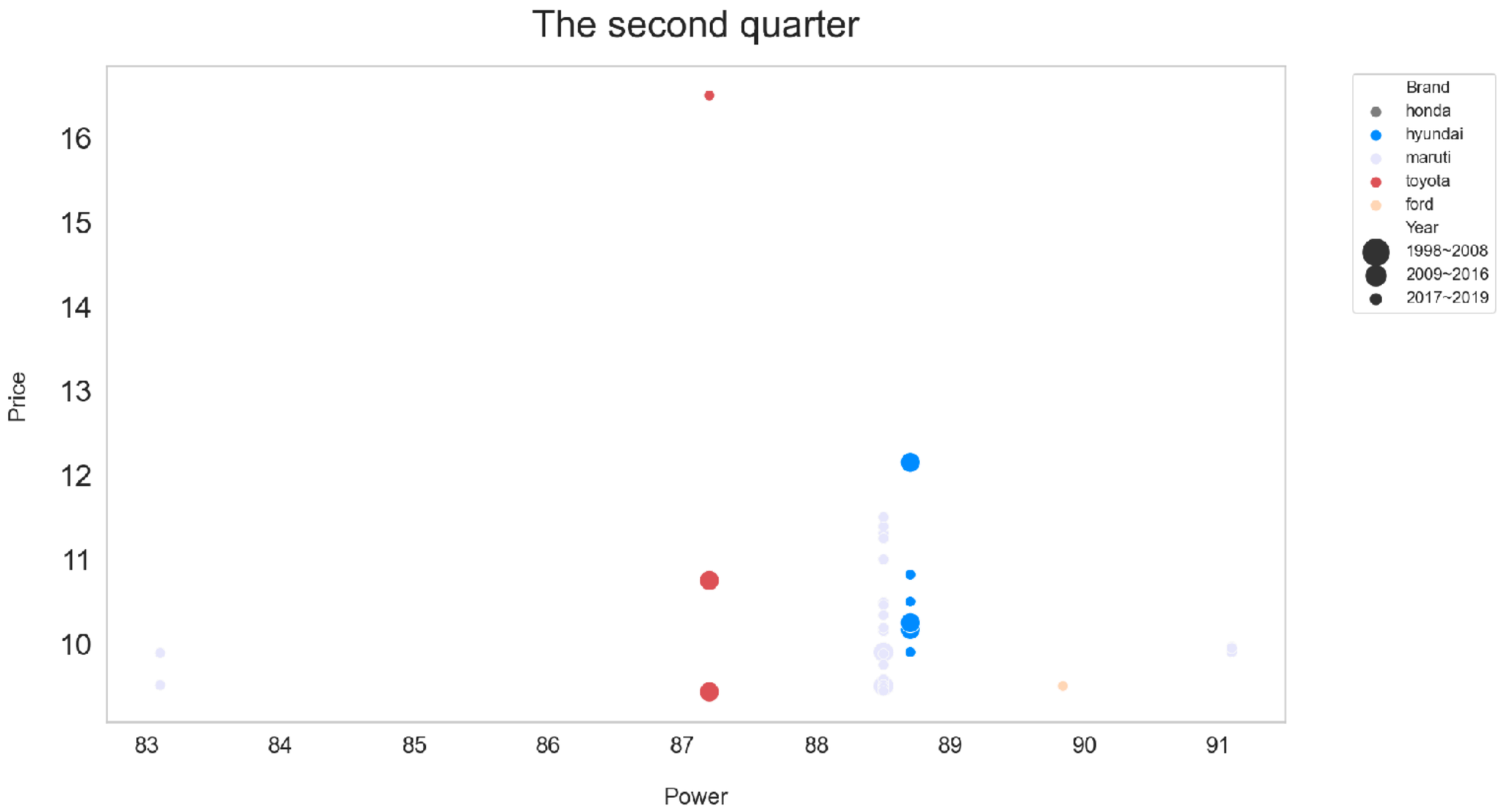
년도에 따른 Price와 Power



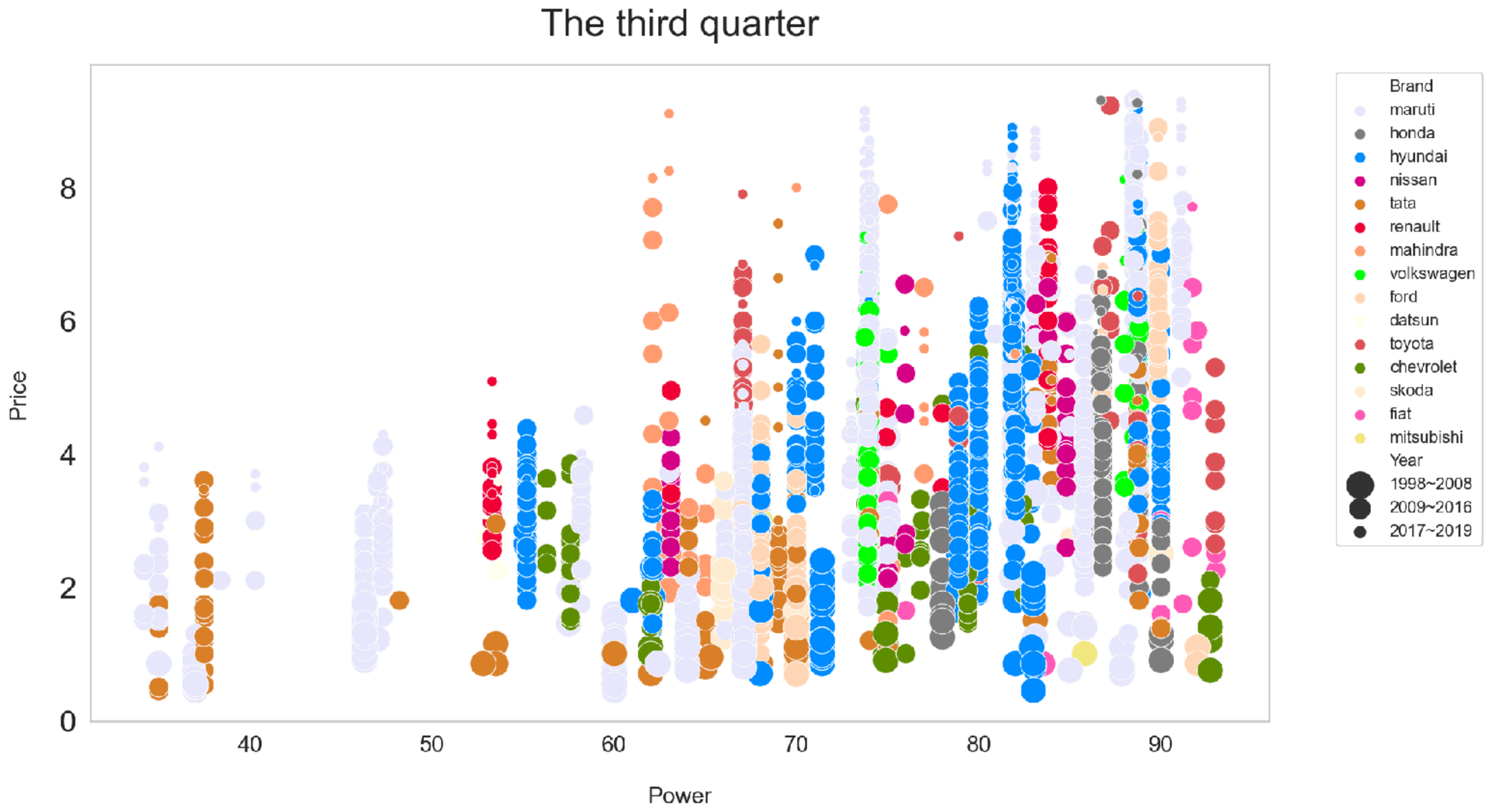
제1사분면


The first quarter
Price
100
80
60
40
20
100
200
300
400
500
Power
Brand
hyundai
audi
toyota
maruti
land
mitsubishi
benz
bmw
mahindra
porsche
jaguar
volvo
skoda
mini
renault
honda
jeep
volkswagen
tata
ford
chevrolet
Year
1998~2008
2009~2016
2017~2019

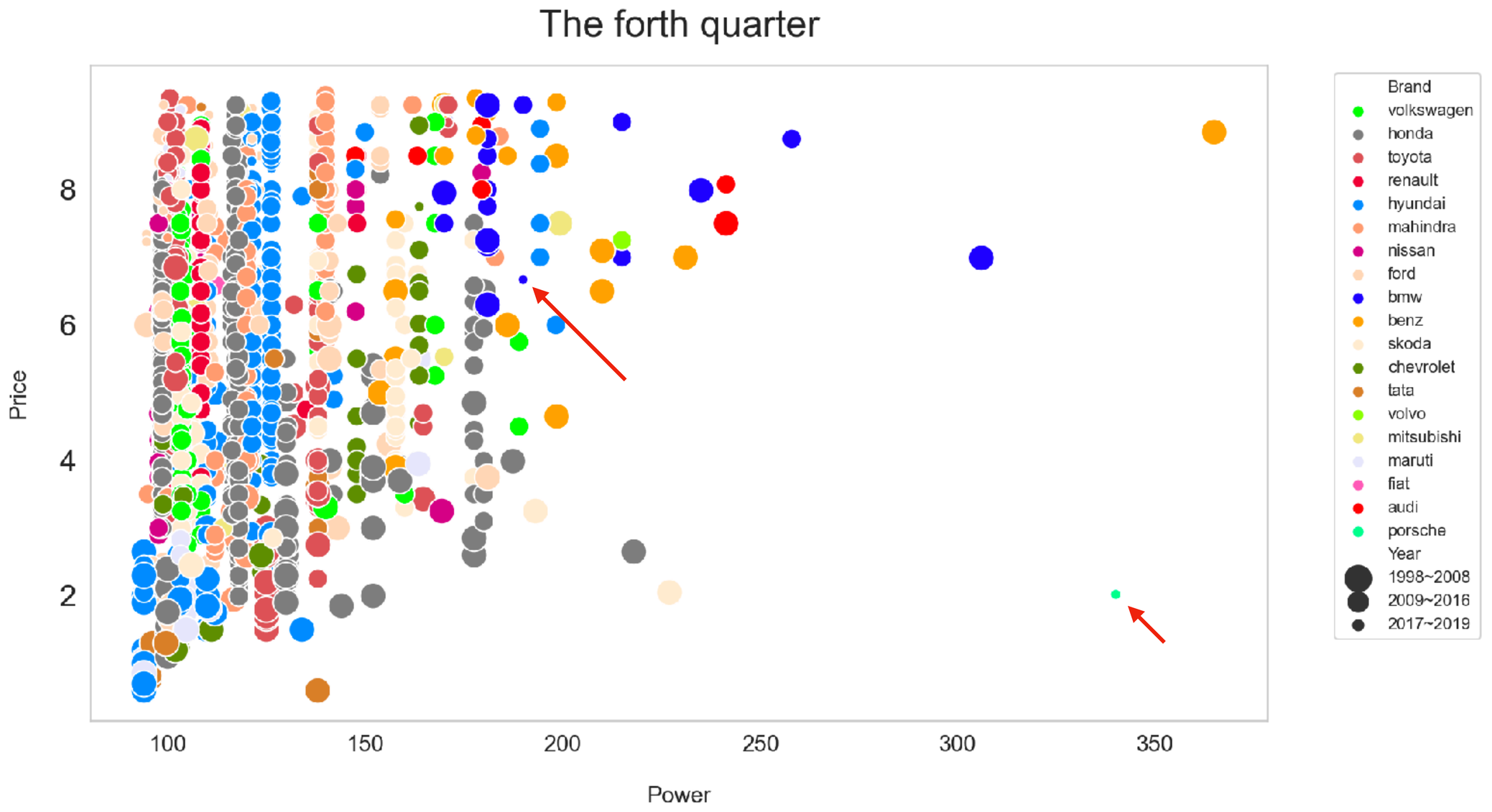
제2사분면


The second quarter
Price
16
15
14
13
12
11
10
83
84
85
86
87
88
89
90
91
Power
Brand
honda
hyundai
maruti
toyota
ford
Year
1998~2008
2009~2016
2017~2019

제3사분면


The third quarter
Price
Power
0
2
4
6
8
40
50
60
70
80
90
Brand
maruti
honda
hyundai
nissan
tata
renault
mahindra
volkswagen
ford
datsun
toyota
chevrolet
skoda
fiat
mitsubishi
Year
1998~2008
2009~2016
2017~2019

제4사분면


The forth quarter
Price
8
6
4
2
100
150
200
250
300
350
Power
Brand
volkswagen
honda
toyota
renault
hyundai
mahindra
nissan
ford
bmw
benz
skoda
chevrolet
tata
volvo
mitsubishi
maruti
fiat
audi
porsche
Year
1998~2008
2009~2016
2017~2019

제4사분면 이상치

```
usedcar_bmw = usedcar_origin.loc[(usedcar_origin["Brand"] == "bmw") & (usedcar_origin["Price"] < 8) & (usedcar_origin["Year"] > 2016)]
usedcar_bmw
```

```
usedcar_porshe = usedcar_origin.loc[(usedcar_origin["Brand"] == "bmw") & (usedcar_origin["Price"] < 4)]
usedcar_porshe
```

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	Price	Brand
3132	Porsche Cayenne Base	Kochi	2019	14298	Petrol	Automatic	First	13.33 kmpl	2995 CC	340 bhp	5.0	1.36 Cr	2.02	porsche

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	Price	Brand
3059	BMW 3 Series 320d Luxury Line	Delhi	2019	87000	Diesel	Automatic	First	22.69 kmpl	1995 CC	190 bhp	5.0	52.46 Lakh	6.67	bmw

- BMW 3 Series 320d Luxury Line → 6.67 Lakh(한화 약 1,000만원)

- Porsche Cayenne Base → 2.02 Lakh (한화 약 310만원)

Train & Test set 구분

Train & Test set 구분

test_size = 0.3

* 데이터셋의 행수가 5,909로 적은 편
  → 테스트 데이터의 수를 확보하기 위해 테스트 데이터셋의 크기를 0.3으로 결정

random_state = 13

* 반복실행 시 일정한 결과 반환을 위해 설정했으며, 13은 임의의 숫자

stratify = df[“Brand”]

* 소수 브랜드가 특정 데이터셋에 집중되는 현상을 방지하기 위함

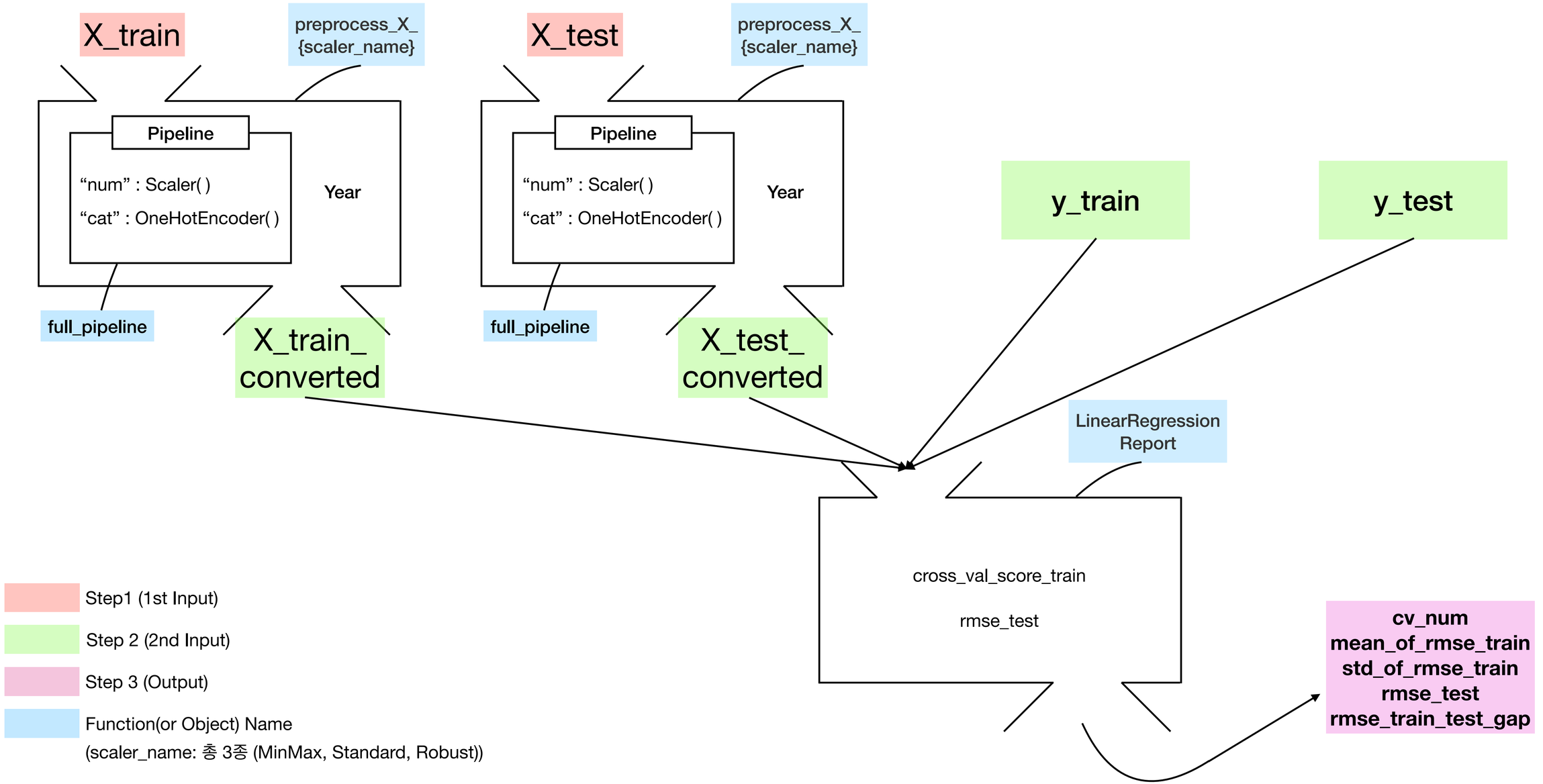
→ **Train data 4,136행, Test data 1,773행으로 분할**

```
1  df["Brand"].value_counts()
```

```
maruti           1196
hyundai          1073
honda             601
toyota            407
volkswagen        314
mercedes-benz     306
ford              298
mahindra          269
bmw               261
audi              235
tata              186
skoda             171
renault           145
chevrolet         121
nissan             91
land               51
jaguar             40
mitsubishi         27
mini               26
fiat               24
volvo              21
porsche            18
jeep               15
datsun             13
Name: Brand, dtype: int64
```

Pipeline 구축 및 모델링

파이프라인 및 구조도


X_train
preprocess_X_{scaler_name}
Pipeline
“num” : Scaler( )
“cat” : OneHotEncoder( )
Year
full_pipeline
X_train_converted
X_test
preprocess_X_{scaler_name}
Pipeline
“num” : Scaler( )
“cat” : OneHotEncoder( )
Year
full_pipeline
X_test_converted
y_train
y_test
LinearRegression Report
cross_val_score_train
rmse_test
cv_num
mean_of_rmse_train
std_of_rmse_train
rmse_test
rmse_train_test_gap
Step1 (1st Input)
Step 2 (2nd Input)
Step 3 (Output)
Function(or Object) Name
(scaler_name: 총 3종 (MinMax, Standard, Robust))

프로세스 코드 예시

Step 1 (1st Input) & Step 2 (2nd Input)

```
---------------- module test ----------------
```

```python
from modules import Preprocessing as pp
from modules import MakeReport as mrp
```

```python
# MinMaxScaler
X_train_minmax_ex = pp.preprocess_X_minmax(X_train, X_train)
X_test_minmax_ex = pp.preprocess_X_minmax(X_train, X_test)


# StandardScaler
X_train_standard_ex = pp.preprocess_X_standard(X_train, X_train)
X_test_standard_ex = pp.preprocess_X_standard(X_train, X_test)


# RobustScaler
X_train_robust_ex = pp.preprocess_X_robust(X_train, X_train)
X_test_robust_ex = pp.preprocess_X_robust(X_train, X_test)
```

```python
# MinMaxScaler
minmax_result_ex = mrp.LinearRegressionReport(X_train_minmax_ex, X_test_minmax_ex,
                                              y_train, y_test)
# StandardScaler
standard_result_ex = mrp.LinearRegressionReport(X_train_standard_ex,
                                                X_test_standard_ex, y_train, y_test)
# RobustScaler
robust_result_ex = mrp.LinearRegressionReport(X_train_robust_ex, X_test_robust_ex,
                                              y_train, y_test)
```

Step 3 (Output)

```python
minmax_result_ex.iloc[[0,2,4,6,7,8]]
```

	cv_num	mean_of_rmse_train	std_of_rmse_train	rmse_test	rmse_train_test_gap
0	2	5.233778	0.817294	5.504507	0.270729
2	4	5.143994	0.265452	5.504507	0.360513
4	6	5.146507	0.517286	5.504507	0.358000
6	8	5.120695	0.648637	5.504507	0.383812
7	9	5.153678	0.553872	5.504507	0.350829
8	10	5.070924	0.726388	5.504507	0.433583

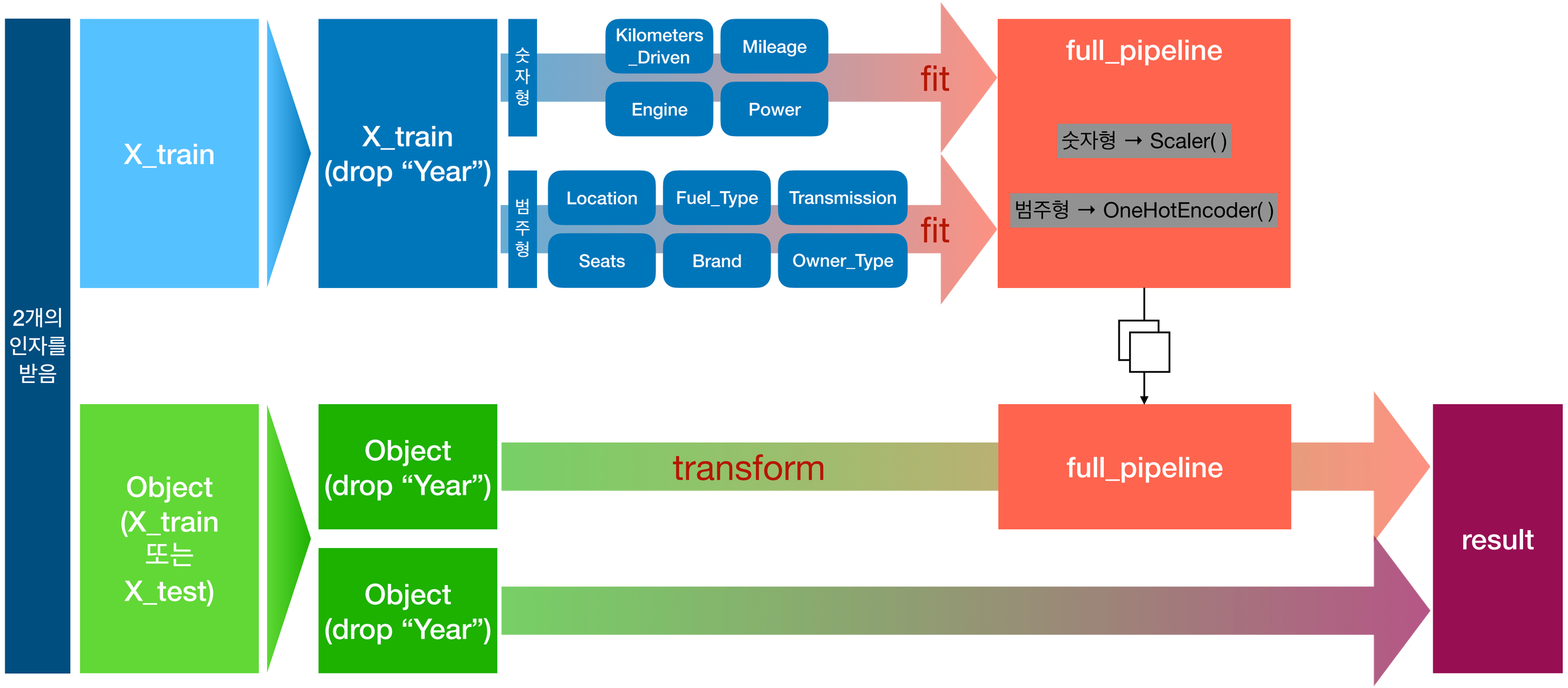
```python
standard_result_ex.iloc[[0,2,4,6,7,8]]
```

	cv_num	mean_of_rmse_train	std_of_rmse_train	rmse_test	rmse_train_test_gap
0	2	5.261797	0.810858	5.475912	0.214115
2	4	5.370868	0.112681	5.475912	0.105044
4	6	5.144803	0.525558	5.475912	0.331108
6	8	5.089911	0.585553	5.475912	0.386000
7	9	5.115479	0.536372	5.475912	0.360433
8	10	5.076227	0.730658	5.475912	0.399684

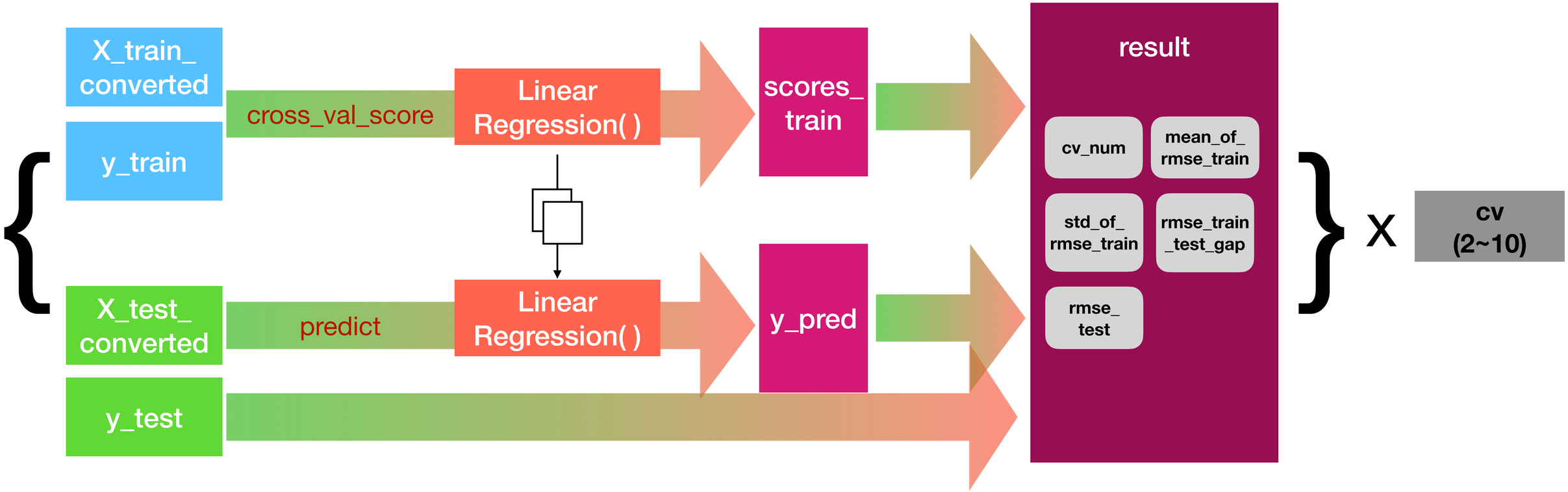
```python
robust_result_ex.iloc[[0,2,4,6,7,8]]
```

	cv_num	mean_of_rmse_train	std_of_rmse_train	rmse_test	rmse_train_test_gap
0	2	5.206931	0.850863	5.475019	0.268088
2	4	5.293141	0.414788	5.475019	0.181878
4	6	5.164714	0.534673	5.475019	0.310305
6	8	5.211075	0.763854	5.475019	0.263944
7	9	5.211679	0.518041	5.475019	0.263340
8	10	5.181605	0.753791	5.475019	0.293414

preprocess_X 모듈 상세



LinearRegressionReport 모듈 상세



결 론

결론

주어진 데이터셋에 대해 **MinMaxScaler**를 사용한 회귀분석 모델이 가장 적절하다고 판단됨

판단 기준:
1) 교차검증 결과 rmse의 평균 (mean_of_rmse_train)은 차이가 0.5 이하
   (한화 약 75만원)는 무차별,
2) 교차검증 결과 rmse의 표준편차 (std_of_rmse_train)는 작을수록 우수
   (과적합을 방지하기 위해 편차는 작을수록 좋다고 판단했기 때문)
3) 총 9가지의 fold (cv=2~10) 중 최다 득표 모델 선정

결과:
- MinMax : 4표
- Standard : 2표
- Robust : 3표

▶ 결론: 4표로 최다 득표한 MinMax Scaler를 사용한 모델이 가장 우수한 모델

cv fold별 결과

cv=2

cv_num	model	mean_of_rmse_train	std_of_rmse_train	rmse_test	rmse_train_test_gap
2	MinMax	5.316748	0.199835	5.207228	0.109520
2	Robust	5.433297	0.199398	5.283853	0.149445
2	Standard	5.508004	0.129860	5.171666	0.336338

cv=3

cv_num	model	mean_of_rmse_train	std_of_rmse_train	rmse_test	rmse_train_test_gap
3	MinMax	5.403592	0.486608	5.207228	0.196364
3	Robust	5.358126	0.486484	5.283853	0.074274
3	Standard	5.359962	0.478112	5.171666	0.188295

cv=4

cv_num	model	mean_of_rmse_train	std_of_rmse_train	rmse_test	rmse_train_test_gap
4	MinMax	5.316628	0.517379	5.207228	0.109399
4	Robust	5.435243	0.433618	5.283853	0.151391
4	Standard	5.274277	0.530608	5.171666	0.102611

cv=5

cv_num	model	mean_of_rmse_train	std_of_rmse_train	rmse_test	rmse_train_test_gap
5	MinMax	5.257987	0.534563	5.207228	0.050759
5	Robust	5.281580	0.524966	5.283853	0.002272
5	Standard	5.261054	0.556549	5.171666	0.089388

cv=6

cv_num	model	mean_of_rmse_train	std_of_rmse_train	rmse_test	rmse_train_test_gap
6	MinMax	5.352471	0.630293	5.207228	0.145243
6	Robust	5.370736	0.788202	5.283853	0.086883
6	Standard	5.286830	0.662869	5.171666	0.115164

cv=7

cv_num	model	mean_of_rmse_train	std_of_rmse_train	rmse_test	rmse_train_test_gap
7	MinMax	5.270576	0.559255	5.207228	0.063347
7	Robust	5.297726	0.651493	5.283853	0.013874
7	Standard	5.266905	0.642781	5.171666	0.095239

cv=8

cv_num	model	mean_of_rmse_train	std_of_rmse_train	rmse_test	rmse_train_test_gap
8	MinMax	5.291163	0.767897	5.207228	0.083935
8	Robust	5.237303	0.643013	5.283853	0.046549
8	Standard	5.351069	0.643873	5.171666	0.179403

cv=9

cv_num	model	mean_of_rmse_train	std_of_rmse_train	rmse_test	rmse_train_test_gap
9	MinMax	5.315184	0.698779	5.207228	0.107955
9	Robust	5.205423	0.748950	5.283853	0.078430
9	Standard	5.259610	0.743519	5.171666	0.087943

cv=10

cv_num	model	mean_of_rmse_train	std_of_rmse_train	rmse_test	rmse_train_test_gap
10	MinMax	5.298887	0.759492	5.207228	0.091659
10	Robust	5.295833	0.854979	5.283853	0.011981
10	Standard	5.291190	0.793649	5.171666	0.119524

※ mean_of_rmse_train : 교차검증 결과 RMSE 값들의 평균값 (train set ~ validation set)
std_of_rmse_train : 교차검증 결과 RMSE 값들의 표준편차값
rmse_test : 전체 train 데이터로 fit시킨 모델에 test 데이터를 통한 예측값과 실제 라벨값 간의 RMSE 값
(train set ~ test set)
rmse_train_test_gap : ‘mean_of_rmse_train’과 ‘rmse_test’ 간의 차이값

추후 개선 방향

추후 개선 방향

1) preprocess_X 함수 개선

: scaler 별로 반복문 돌도록 함수 하나로 합치기
또는 함수를 여러 개 만든 후 Preprocessing 모듈 내에 통합

2) LinearRegressionReport 함수 개선

: model 별로 반복문 돌도록 함수 하나로 합치기
또는 함수를 여러 개 만든 후 MakeReport 모듈 내에 통합

Q & A

E.O.D