데이터분석기초 기말 대체 과제

기계정보공학과 2019430003 고영훈

Track 1. Python 환경에서 차량 연비에 관한 'mpg' 데이터셋을 활용하여 데이터분석을 수행

1.1 자료에 포함된 각 변수의 의미를 파악하고 데이터 구조를 설명

최초에 데이터 분석을 위해 필요한 외부 라이브러리를 호출하고 mpg 데이터 세트를 불러온 뒤 데이터 분석을 진행했다.

```
# 데이터 분석을 위해 필요한 외부 라이브러리 호출
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
from matplotlib import pyplot as plt
from patsy import dmatrices
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
# mpg 데이터 세트 불러오기(https://vincentarelbundock.github.io/Rdatasets/datasets.html 에서 다운로드) 및 데이터 확인
mpg = pd.read_csv('/content/mpg.csv')
mpg.head()
```

그림 1, 2. 외부 라이브러리 호출 및 mpg 데이터 세트 로드

Pandas의 read_csv를 이용해 자료를 불러와 head() 함수를 이용해 데이터를 제대로 가져왔는지 확인하고 이때 데이터 column중 rownames의 경우 데이터가 아닌 번호이기 때문에 drop()을 통해 제거했다.

	rownames	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
0	1	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact
1	2	audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact
2	3	audi	a4	2.0	2008	4	manual(m6)	f	20	31	p	compact
3	4	audi	a4	2.0	2008	4	auto(av)	f	21	30	p	compact
4	5	audi	a4	2.8	1999	6	auto(l5)	f	16	26	p	compact

그림 3. head() 결과

```
# 데이터 전처리 및 확인
mpg.drop(columns=['rownames'], inplace=True)
mpg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 234 entries, 0 to 233
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   manufacturer  234 non-null    object
 1   model         234 non-null    object
 2   displ         234 non-null    float64
 3   year          234 non-null    int64
 4   cyl           234 non-null    int64
 5   trans         234 non-null    object
 6   drv           234 non-null    object
 7   cty           234 non-null    int64
 8   hwy           234 non-null    int64
 9   fl            234 non-null    object
 10  class         234 non-null    object
dtypes: float64(1), int64(4), object(6)
memory usage: 20.2+ KB
```

그림 4. Data type 구분

이후 info() 함수를 통해 mpg 데이터 세트에 포함된 각 변수의 의미와 데이터 구조를 파악했다.

mpg 데이터 세트는 자동차의 연비(Miles Per Gallon)와 관련된 다양한 정보를 포함한다.

1. manufacturer: 자동차 제조사
   - 데이터 타입: object
   - 예: 'audi', 'ford', 'toyota' 등
2. model: 자동차 모델
   - 데이터 타입: object
   - 예: 'a4', 'mustang', 'corolla' 등
3. displ: 엔진 배기량 (리터)
   - 데이터 타입: float64
   - 예: 1.8, 2.0, 3.1 등
4. year: 연식
   - 데이터 타입: int64
   - 예: 1999, 2008
5. cyl: 실린더 수
   - 데이터 타입: int64
   - 예: 4, 6, 8 등
6. trans: 변속기 유형
   - 데이터 타입: object
   - 예: 'auto(l5)', 'manual(m6)' 등
7. drv: 구동 방식
   - 데이터 타입: object
   - 'f': 전륜구동 (front-wheel drive)
   - 'r': 후륜구동 (rear-wheel drive)
   - '4': 사륜구동 (4-wheel drive)
8. cty: 도시 연비 (MPG)
   - 데이터 타입: int64
   - 예: 18, 21, 24 등
9. hwy: 고속도로 연비 (MPG)
   - 데이터 타입: int64
   - 예: 25, 28, 31 등
10. fl: 연료 종류
    - 데이터 타입: object
    - 'c': CNG (Compressed Natural Gas)
    - 'd': Diesel
    - 'e': Ethanol
    - 'p': Premium

- 'r': Regular

11. class: 자동차 유형
    - 데이터 타입: object
    - 예: 'compact', 'suv', 'midsize', '2seater' 등

1.2 다음의 과정을 포함하여 자료의 특징을 분석

(A) 범주형 자료에 대해서는 빈도수, 양적 자료에 대해서는 자료의 분포를 파악

범주형 자료에 대해서 빈도수를 파악하기 위해서 아래와 같은 코드를 적용했다.

```
# 제조사 빈도수 및 바 그래프
manufacturer_counts = mpg['manufacturer'].value_counts()
print(manufacturer_counts)
plt.figure(figsize=(10, 6))
sns.barplot(x=manufacturer_counts.values, y=manufacturer_counts.index)
plt.title('manufacturer')
plt.xlabel('frequency')
plt.ylabel('manufacturer')
plt.show()
```

그림 5. 빈도수 및 바 그래프 출력 함수

제조사를 제외하고도 모델, 변속기, 구동 방식, 연료 타입, 자동차 유형에도 동일한 코드를 적용했고 결과는 다음과 같다.

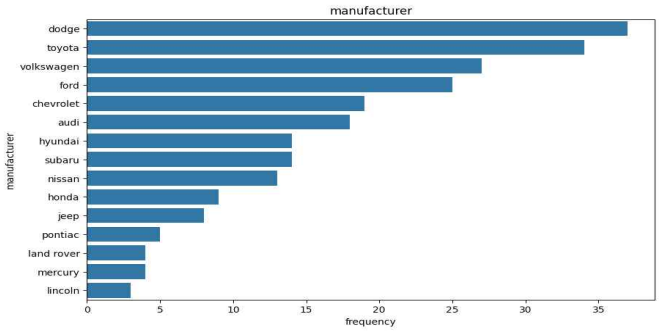


그림 6. 제조사 빈도 그래프

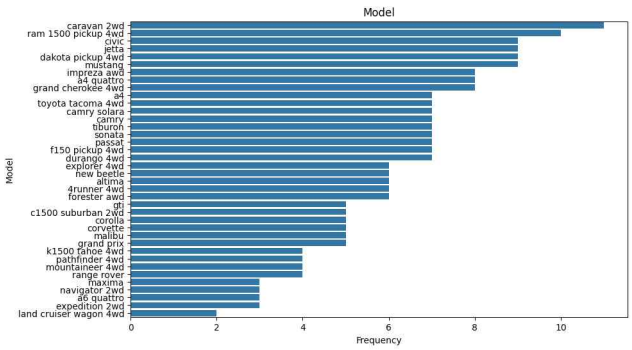


그림 7. 차량 모델 빈도 그래프

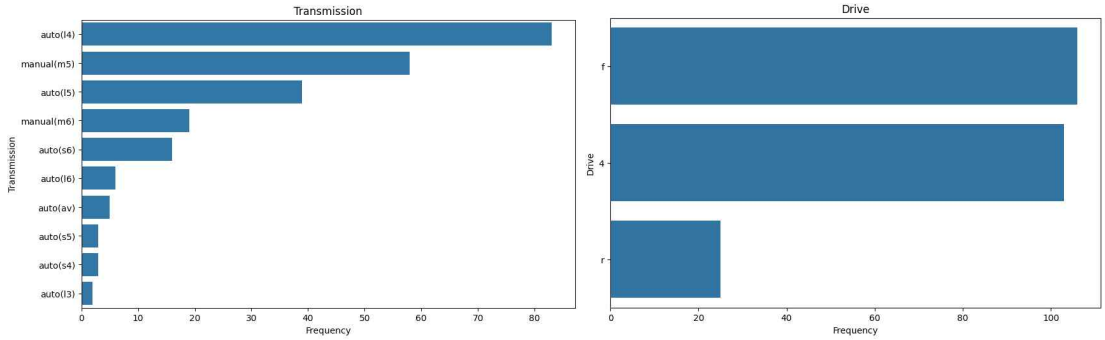


그림 8, 9. 변속기, 구동 방식 빈도 그래프

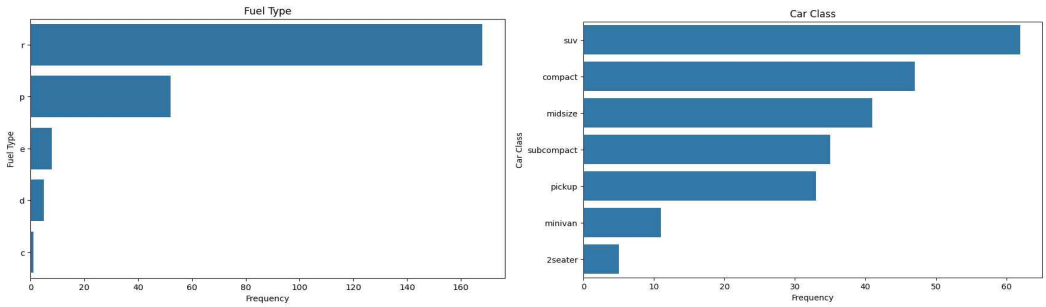


그림 10, 11. 연료 타입, 자동차 유형 빈도 그래프

양적 자료의 경우에는 최초에 describe() 함수를 통해서 요약 통계량을 확인했다.

```
1 # 양적 자료 요약 통계량 확인
2 mpg.describe()
```

	displ	year	cyl	cty	hwy
count	234.000000	234.000000	234.000000	234.000000	234.000000
mean	3.471795	2003.500000	5.888889	16.858974	23.440171
std	1.291959	4.509646	1.611534	4.255946	5.954643
min	1.600000	1999.000000	4.000000	9.000000	12.000000
25%	2.400000	1999.000000	4.000000	14.000000	18.000000
50%	3.300000	2003.500000	6.000000	17.000000	24.000000
75%	4.600000	2008.000000	8.000000	19.000000	27.000000
max	7.000000	2008.000000	8.000000	35.000000	44.000000

그림 12. mpg 데이터 세트 양적 자료 요약 통계량

요약 통계량을 살펴보는 것을 통해서는 자료의 분포를 정확하게 파악하기 힘들었기에 seaborn 라이브러리의 pairplot() 함수를 통해서 산점도 및 히스토그램을 출력하고 그에 대한 관계를 확인했다.

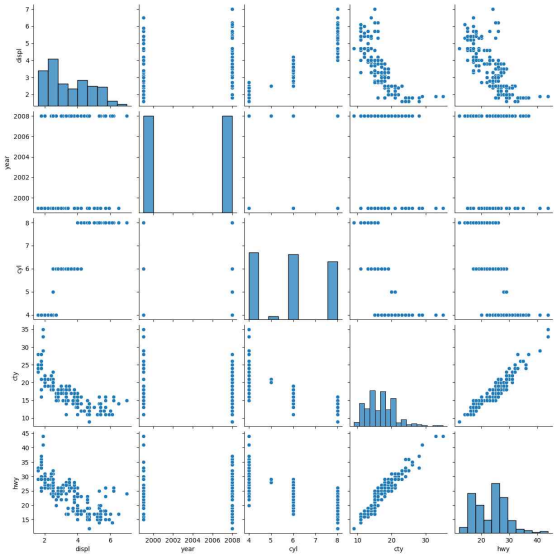


그림 13. mpg 데이터 세트 양적 자료 산점도 및 히스토그램

산점도를 통해 hwy,cty 사이의 비례관계와 displ과 hwy,cty 사이의 음의 비례 관계가 있음을 확인할 수 있다. 그러나 이때 보이는 히스토그램으로는 확실하게 빈도를 파악하기 힘들기에 각각에 대하여 분포를 파악할 수 있는 히스토그램을 출력하도록 했다.

```
# 배기량 분포 확인
displ_counts = mpg['displ'].value_counts().sort_index()
print(displ_counts)
plt.figure(figsize=(10, 6))
sns.histplot(mpg['displ'], kde=True)
plt.title('Distribution of Displacement')
plt.xlabel('Displacement')
plt.ylabel('Frequency')
plt.show()
```

그림 14. 배기량 빈도 및 히스토그램 출력 함수

위와 같은 빈도 및 히스토그램 출력 함수를 배기량, 연식, 실린더 수, 도시 연비, 고속도로 연비에 적용해 다음과 같은 결과를 얻었다.

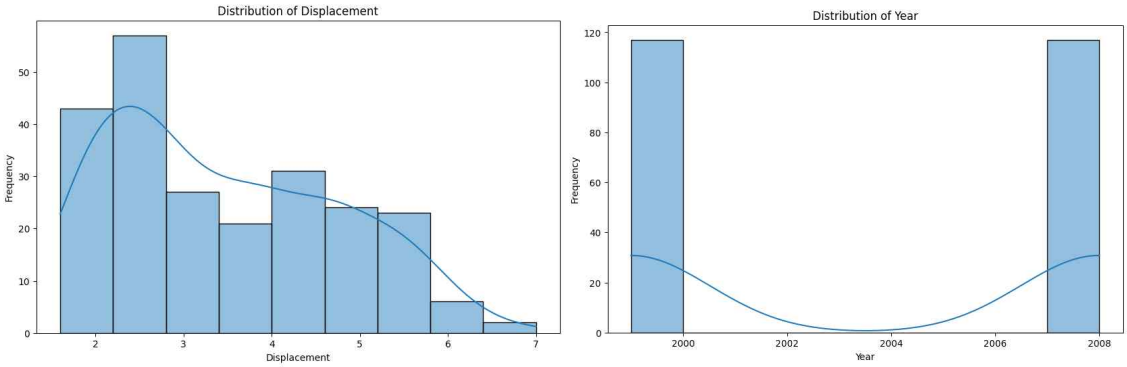


그림 15, 16. 배기량 및 연식 히스토그램

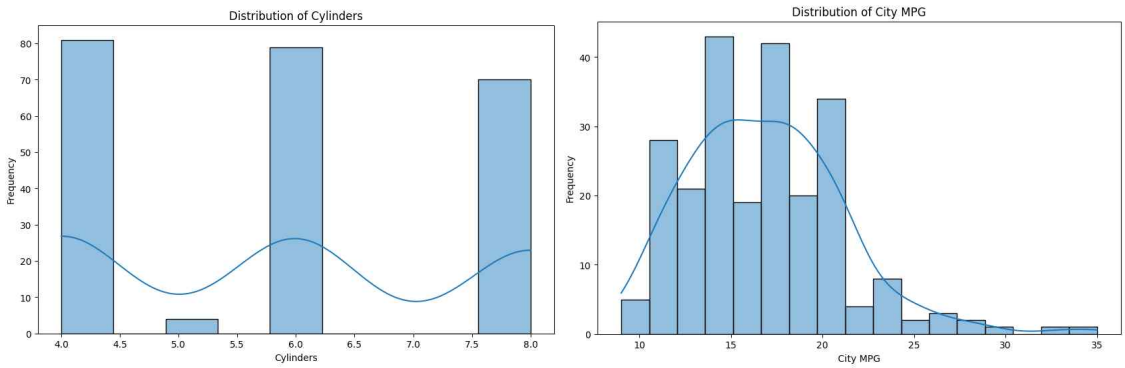


그림 17, 18. 실린더 수, 도시 연비 히스토그램

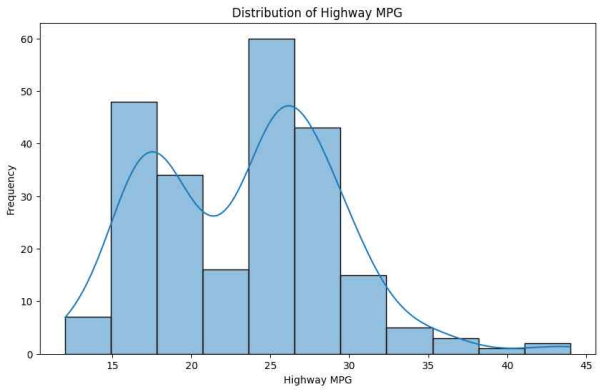


그림 19. 고속도로 연비 히스토그램

이 히스토그램을 통해서 제조년도는 1999년과 2008년 두 개밖에 없으며 실린더 같은 경우에도 연속적인 값이 아닌 4개, 5개, 6개, 8개로 이산적이며 실린더가 5개인 차는 드물다는 것을 확인할 수 있다.
그리고 배기량과 두 연비의 히스토그램의 개형이 유사한데 이는 후에 있을 상관 분석을 통해 알아보기로 하였다.

(B) 그림과 상관계수의 값을 통해 양적 자료에 해당하는 변수들 사이의 연관성을 파악
상관계수의 값을 계산하고 이에 대한 시각화를 위해 히트맵을 출력하는 함수를 적용했다.

```
1 # 양적자료의 히트맵과 상관계수를 통한 상관 분석
2 quantitative_vars = ['displ', 'cty', 'hwy', 'year', 'cyl']
3 correlation_matrix = mpg[quantitative_vars].corr()
4 correlation_matrix
5 sns.heatmap(correlation_matrix, annot=True)
```

그림 20. 양적 자료의 상관계수 계산 및 히트맵 코드

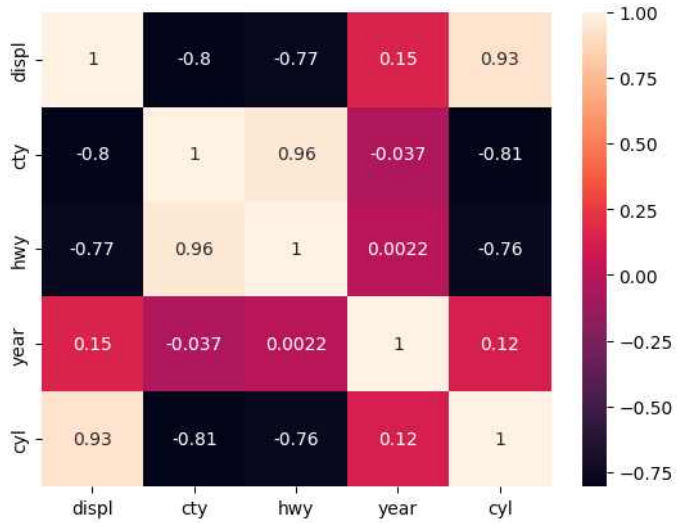


그림 21. 상관관계 히트맵

상관관계 히트맵을 통해서 mpg 데이터 세트의 양적 자료 간의 상관계수를 확인하고 관계를 분석할 수 있었다.

히트맵은 변수 간의 상관관계를 시각적으로 표현한 것으로, 상관계수 값은 -1에서 1 사이의 값을 가진다. 1에 가까울수록 강한 양의 상관 관계를, -1에 가까울수록 강한 음의 상관 관계를 나타냄. 0에 가까울수록 상관관계가 거의 없음을 의미한다.

히트맵 분석 결과

1. displ (배기량)와 cty (도시 연비)
   - 상관계수: -0.8
   - 해석: 배기량이 증가할수록 도시 연비가 감소하는 경향이 강하게 나타난다. 이는 일반적으로 배기량이 큰 차는 연비가 낮다는 것을 의미한다.
2. displ (배기량)와 hwy (고속도로 연비)
   - 상관계수: -0.77
   - 해석: 배기량이 증가할수록 고속도로 연비도 감소하는 경향이 강하게 나타난다.
3. displ (배기량)와 cyl (실린더 수)
   - 상관계수: 0.93
   - 해석: 배기량이 증가할수록 실린더 수도 증가하는 경향이 매우 강하게 나타난다. 이는 배기량이 큰 엔진일수록 실린더 수가 많기 때문으로 보인다.
4. cty (도시 연비)와 hwy (고속도로 연비)
   - 상관계수: 0.96
   - 해석: 도시 연비와 고속도로 연비 사이에 매우 강한 양의 상관 관계가 있다. 도시에서 연비가 좋은 차는 고속도로에서도 연비가 좋은 경향이 있다. 일반적으로 자동차 시스템의 연비가 좋으면 도시에서의 연비와 고속도로에서의 연비가 좋기 때문으로

생각된다.
5. cty (도시 연비)와 cyl (실린더 수)
   - 상관계수: -0.81
   - 해석: 도시 연비가 감소할수록 실린더 수가 증가하는 경향이 강하게 나타난다. 이는 실린더 수가 많은 차는 연비가 낮다는 것을 의미한다.
6. hwy (고속도로 연비)와 cyl (실린더 수)
   - 상관계수: -0.76
   - 해석: 고속도로 연비가 감소할수록 실린더 수가 증가하는 경향이 강하게 나타난다.
7. year (연식)과 다른 변수들
   - year와 displ: 0.15 (약한 양의 상관관계)
   - year와 cty: -0.037 (거의 상관관계 없음)
   - year와 hwy: 0.0022 (거의 상관관계 없음)
   - year와 cyl: 0.12 (약한 양의 상관 관계)
   - 해석: 연식은 다른 변수들과 약하거나 거의 상관관계가 없다. 이는 연식이 차량의 배기량, 연비, 실린더 수와 직접적인 관련이 적음을 보여준다.

요약하면, 배기량, 실린더 수와 연비 사이에는 강한 상관관계가 있으며, 실린더가 많으면 배기량이 많고 배기량이 많은 차량은 연비가 낮은 경향이 있다. 연비가 좋은 차는 도시와 고속도로 모두에서 연비가 좋은 경향이 있다. 연식은 다른 변수들과 큰 상관관계가 없는 것으로 보인다.

(C) 양적 자료와 범주형 자료를 교차하여 범주가 구분된 자료의 분포를 시각화하고 그 결과를 해석

고속도로 연비와 도시 연비는 상관계수가 1에 가까운 상관관계가 아주 큰 변수들이기에 이를 평균 낸 데이터에 대해서 데이터 분석을 실시 분석 결과가 흔들리지 않는다고 판단 된다.
그래서 이에 대해서 평균을 낸 평균 연비 값에 대한 데이터 추가하고 이에 대한 분석을 실시하려고 한다.
또한 연식은 예상과 달리 연비와 큰 상관이 없었기에 연식에 대해서는 제외하고 나머지 데이터 분석을 실시하였다.

```
1 # 평균 연비(고속도로 연비와 도시 연비의 평균) 데이터 추가 및 연식 데이터 제거
2 mpg['mpg'] = ((mpg['cty']+mpg['hwy'])/2)
3 mpg.drop(columns=['year'], inplace=True)
4 mpg.head()
```

그림 22. 평균 연비 데이터 추가 및 연식 데이터 제거 코드

그리고 양적 자료와 범주형 자료의 선택에서 이 프로젝트는 연비에 큰 영향을 미치는 요인에 대해서 알아보려고 하기 때문에 양적 자료의 경우 연비를 선택하고 이에 대한 범주형 자료의 분포를 시각화하려고 한다.
범주형 자료 중 차량 모델의 경우 너무 다양하여 분석에 오히려 방해가 될 것으로 예상해 분석 대상에서 제외했다.

```
# 평균 연비와 제조사의 관계 시각화
plt.figure(figsize=(15, 8))
sns.boxplot(x='manufacturer', y='mpg', data=mpg)
plt.title('MPG by Manufacturer')
plt.xlabel('Manufacturer')
plt.ylabel('MPG')
plt.show()
```

그림 23. 평균 연비와 제조사의 관계 시각화 코드

위와 같은 평균 연비와 범주형 자료 사이의 관계를 시각화하는 코드를 차량 모델을 제외한 나머지 범주형 자료에도 적용하여 다음과 같은 결과를 얻었다.

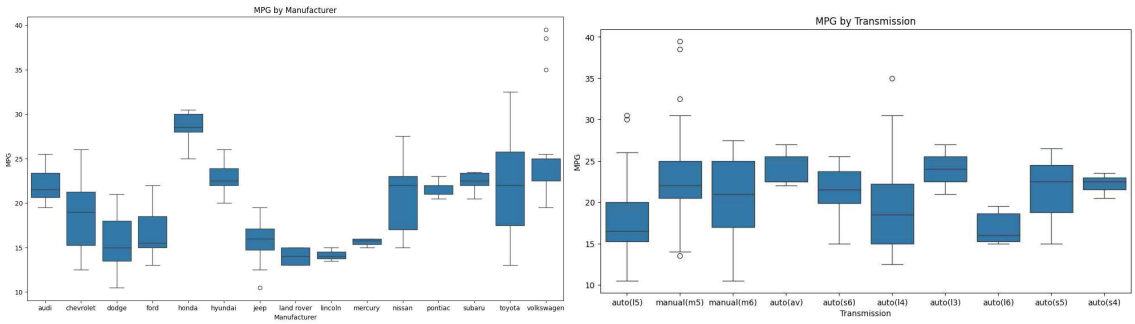


그림 24, 25. 평균 연비와 제조사, 변속기 간의 관계 그래프

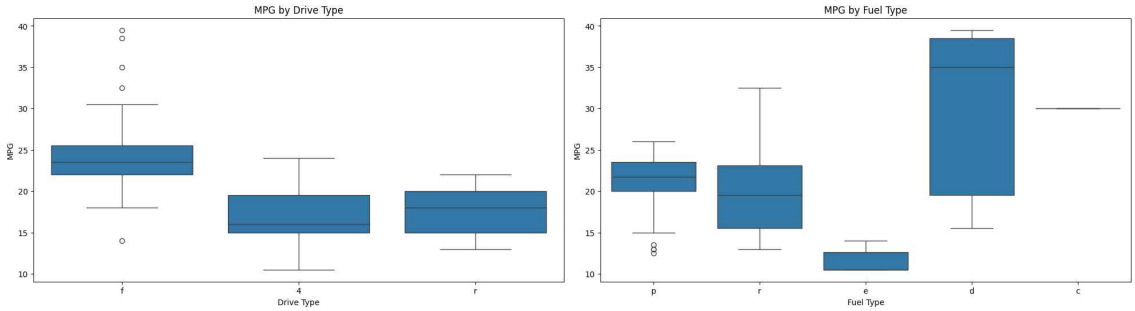


그림 26, 27. 평균 연비와 구동 방식, 연료 종류 간의 관계 그래프

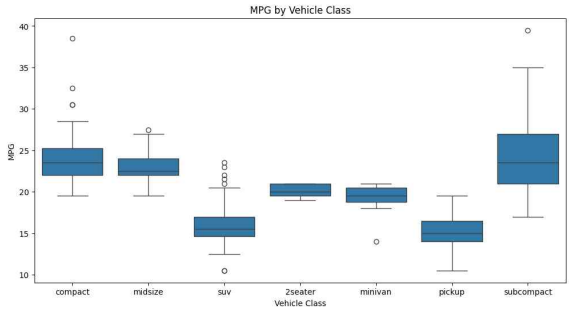


그림 28. 평균 연비와 차량 종류에 따른 관계 그래프

제조사의 경우 제조사와 평균 연비가 관계있어 보이긴 하지만 이는 제조사에서 어떤 차량을 주로 만드느냐에 의한 차이로 비롯된 것 같다.
변속기의 경우 평균적으로 어떤 변속기가 좋다고 말하기에는 애매한 감이 있다. auto(l5)의 경우에는 평균적으로 다른 변속기에 비해서 확실히 뒤처지는 연비를 보여주지만 나머지 변속기들의 경우에는 평균적으로 봤을 때는 큰 차이를 보이지 않는다.
구동 방식의 경우에는 확실하게 연비에 영향을 미치는 모습을 보이는데 구동 방식이 f 즉 전륜구동일 때가 나머지 두 방식일 때에 비해 연비가 좋은 모습을 보이며 구동 방식이 4 즉 사륜구동일 때는 나머지 두 방식일 때에 비해 연비가 나쁜 모습을 보인다. 즉 f > r > 4 순으로 연비가 차이가 나는 모습을 확인할 수 있다.
그리고 연료 종류에 따른 연비에서도 확실하게 영향을 확인할 수 있는데, 연료가 디젤(d)인 경우 다른 연료 종류에 비해서 연비가 좋고, 그다음 c > p > r > e 순으로 연비가 좋은 것을 확인할 수 있다.
마지막으로 자동차 유형과 연비의 경우에는 확실히 자동차 유형에 따라서 연비가 구분되는 모습을 보이지만 이는 자동차 유형에 따라서 구동 방식, 연료 종류, 배기량이 다르기 때문에 나타나는 모습으로 생각된다.
즉 요약하면 범주형 자료 중 연비에 큰 영향을 미치는 요인은 구동 방식, 연료 종류이고 작은 영향을 미치는 요인은 변속기의 종류로 예측된다.

그리고 이에 대해서 추가적으로 확인하기 위해서 평균 연비와 배기량, 실린더 간의 회귀분석을 시각화하고 이에 변속기, 구동 방식, 연료에 대한 분석을 추가해 그래프를 확인하였다.

```
# 평균 연비와 배기량 간의 회귀 분석 시각화
sns.lmplot(data=mpg, x='displ', y='mpg')
plt.title('MPG - Displacement')
plt.xlabel('Displacement')
plt.ylabel('MPG')
plt.show()

sns.lmplot(data=mpg, x='displ', y='mpg', hue='trans') # 변속기에 대한 추가 분석
plt.title('MPG - Displacement by Transmission')
plt.xlabel('Displacement')
plt.ylabel('MPG')
plt.show()
```

그림 29. 평균 연비와 배기량 간의 회귀분석 시각화 및 변속기 추가 분석 코드

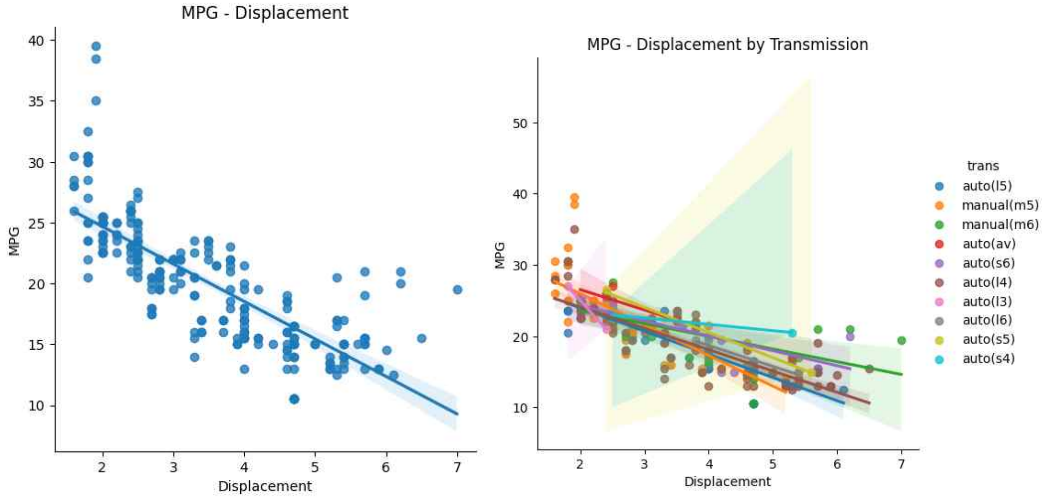


그림 30, 31. 연비-배기량 회귀분석 + 변속기 종류 분석

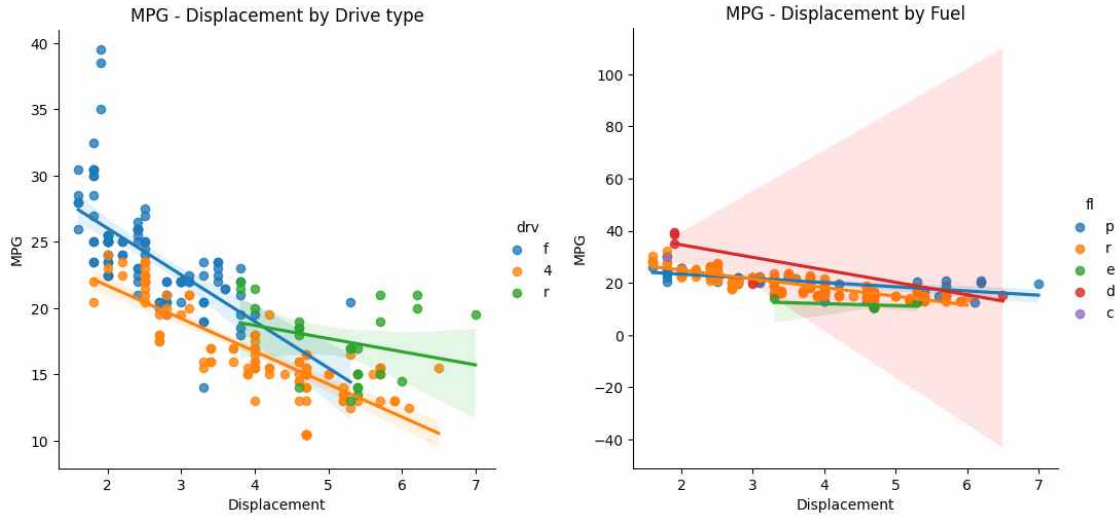


그림 32, 33. 연비-배기량 회귀분석 + 구동 방식, 연료 종류 분석

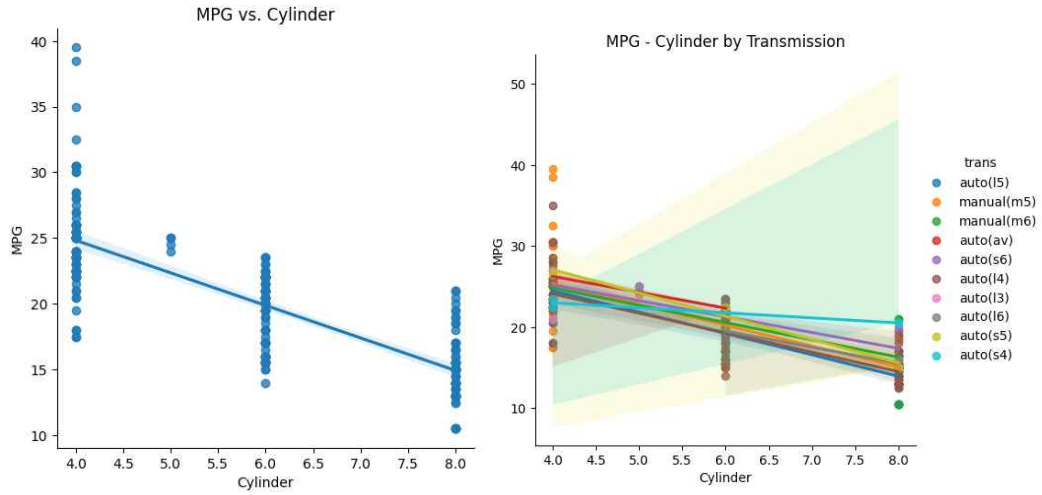


그림 34, 35. 연비-실린더 수 회귀분석 + 변속기 종류

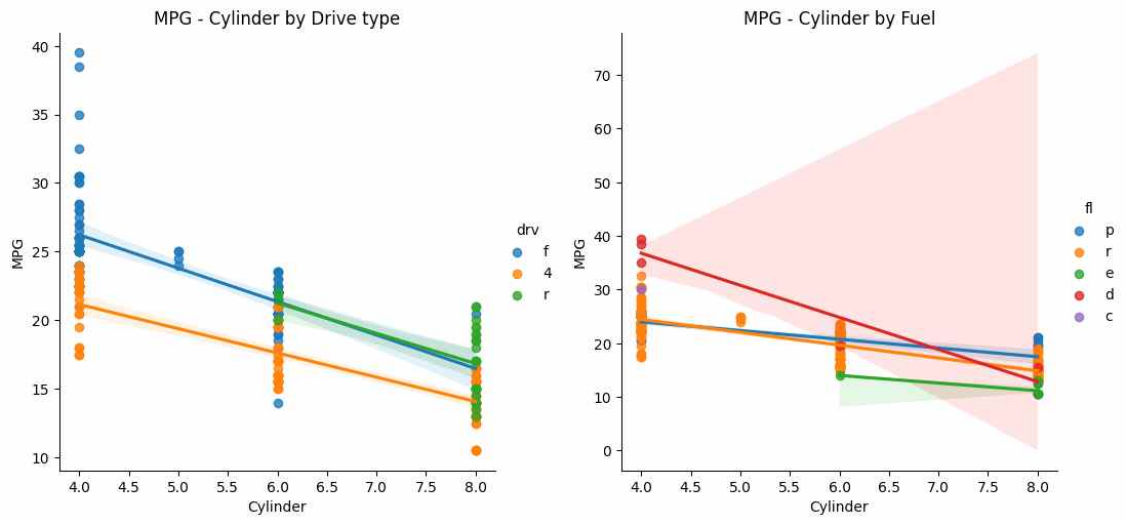


그림 36, 37. 연비-실린더 수 회귀분석 + 구동 방식, 연료 종류

위의 그래프들에 대한 분석을 통해서 확실히 연비와 배기량, 실린더 수, 변속기 종류, 구동 방식, 연료 종류가 상관관계가 있음을 확인할 수 있었다.

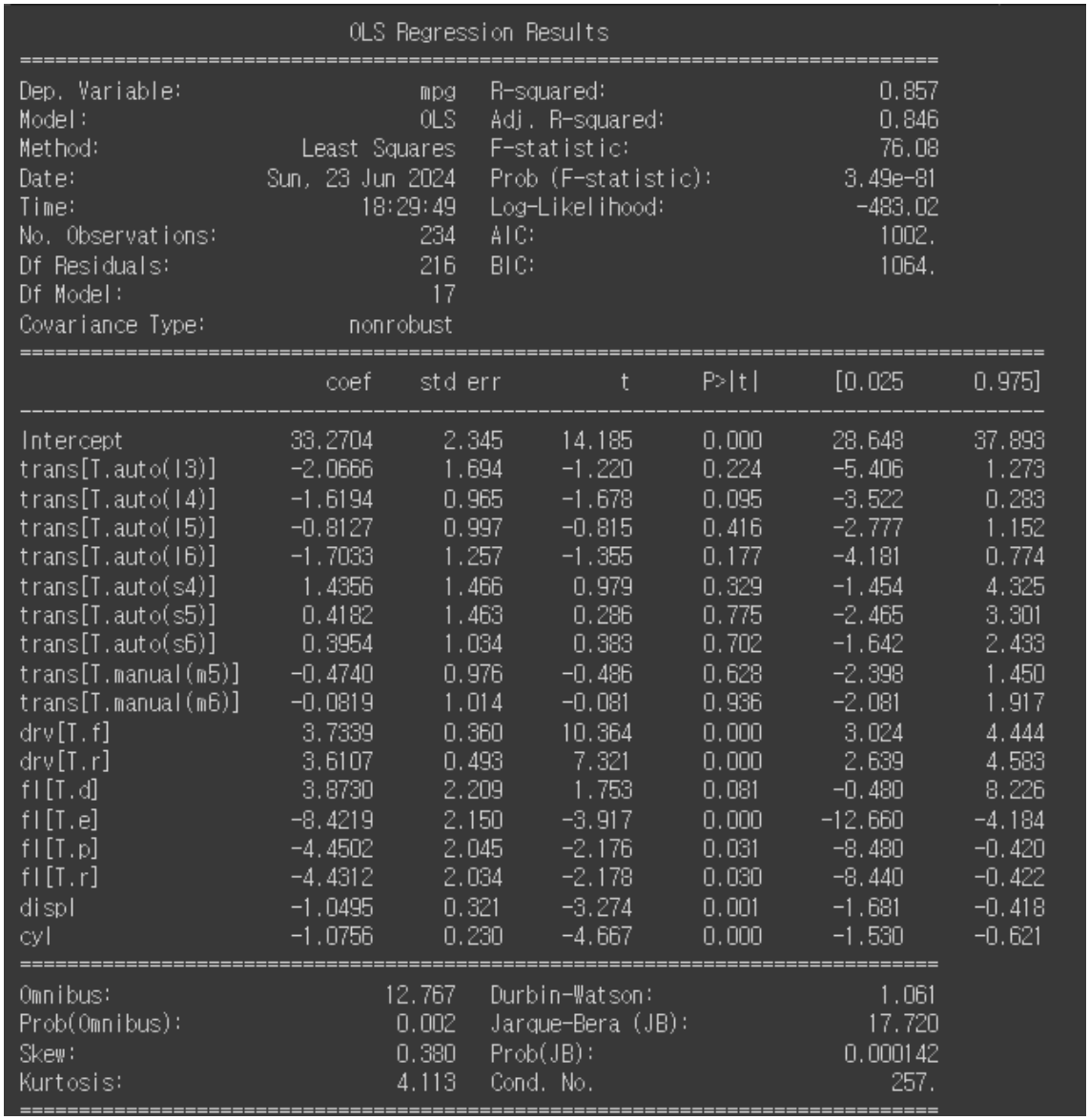
그렇기에 연비에 가장 큰 영향을 미치는 요인을 식별하기 위해 연비를 종속 변수로 두고 배기량, 실린더 수, 변속기 종류, 구동 방식, 연료 종류를 독립 변수로 두고 다중 선형 회귀를 진행하였다.

(D) 회귀분석을 통해 연비에 큰 영향을 미치는 요인을 식별하고 그 결과를 종합적으로 분석

```
# 선형 회귀 실시 및 요약 출력
y, X = dmatrices('mpg ~ displ + cyl + trans + drv + fl', data=mpg, return_type='dataframe')
model = sm.OLS(y, X)
result = model.fit()
print(result.summary())
```

그림 38. 배기량, 실린더 수, 변속기, 구동 방식, 연료 종류를 포함한 다중 선형 회귀 분석 코드

연비를 종속 변수로 배기량, 실린더 수, 변속기, 구동 방식, 연료 종류를 독립 변수로 하여 다중 선형 회귀 분석을 진행하기 위해 다음과 같은 코드를 실행하였다. 결과는 다음과 같다.



OLS Regression Results

Dep. Variable:	mpg	R-squared:	0.857
Model:	OLS	Adj. R-squared:	0.846
Method:	Least Squares	F-statistic:	76.08
Date:	Sun, 23 Jun 2024	Prob (F-statistic):	3.49e-81
Time:	18:29:49	Log-Likelihood:	-483.02
No. Observations:	234	AIC:	1002.
Df Residuals:	216	BIC:	1064.
Df Model:	17		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	33.2704	2.345	14.185	0.000	28.648	37.893
trans[T.auto(l3)]	-2.0666	1.694	-1.220	0.224	-5.406	1.273
trans[T.auto(l4)]	-1.6194	0.965	-1.678	0.095	-3.522	0.283
trans[T.auto(l5)]	-0.8127	0.997	-0.815	0.416	-2.777	1.152
trans[T.auto(l6)]	-1.7033	1.257	-1.355	0.177	-4.181	0.774
trans[T.auto(s4)]	1.4356	1.466	0.979	0.329	-1.454	4.325
trans[T.auto(s5)]	0.4182	1.463	0.286	0.775	-2.465	3.301
trans[T.auto(s6)]	0.3954	1.034	0.383	0.702	-1.642	2.433
trans[T.manual(m5)]	-0.4740	0.976	-0.486	0.628	-2.398	1.450
trans[T.manual(m6)]	-0.0819	1.014	-0.081	0.936	-2.081	1.917
drv[T.f]	3.7339	0.360	10.364	0.000	3.024	4.444
drv[T.r]	3.6107	0.493	7.321	0.000	2.639	4.583
fl[T.d]	3.8730	2.209	1.753	0.081	-0.480	8.226
fl[T.e]	-8.4219	2.150	-3.917	0.000	-12.660	-4.184
fl[T.p]	-4.4502	2.045	-2.176	0.031	-8.480	-0.420
fl[T.r]	-4.4312	2.034	-2.178	0.030	-8.440	-0.422
displ	-1.0495	0.321	-3.274	0.001	-1.681	-0.418
cyl	-1.0756	0.230	-4.667	0.000	-1.530	-0.621

Omnibus:	12.767	Durbin-Watson:	1.061
Prob(Omnibus):	0.002	Jarque-Bera (JB):	17.720
Skew:	0.380	Prob(JB):	0.000142
Kurtosis:	4.113	Cond. No.	257.

그림 39. 다중 선형 회귀 분석 결과

다중 선형 회귀 분석 결과는 위와 같고 이때 변속기의 종류는 auto(av)를 기준, 연료는 c를 기준, 구동 방식은 사륜구동 방식을 기준으로 하여 선형 회귀가 진행되었다. 위를 분석하면 다음과 같다.

주요 요인 분석

통계적으로 유의미한 변수:

구동 방식 (drv):
drv[T.f] (전륜구동): 계수 = 3.7339, p-값 = 0.000

전륜구동 차량은 기준(drv=4)에 비해 연비가 3.7339 단위만큼 높다.
drv[T.r] (후륜구동): 계수 = 3.6107, p-값 = 0.000
후륜구동 차량도 기준(drv=4)에 비해 연비가 3.6107 단위만큼 높다.

연료 유형 (fl):
fl[T.e] (연료 유형 e): 계수 = -8.4219, p-값 = 0.000
연료 유형이 e인 차량은 기준(fl=c)에 비해 연비가 8.4219 단위만큼 낮다.
fl[T.p] (연료 유형 p): 계수 = -4.4502, p-값 = 0.031
연료 유형이 p인 차량도 기준(fl=c)에 비해 연비가 4.4502 단위만큼 낮다.
fl[T.r] (연료 유형 r): 계수 = -4.4312, p-값 = 0.030
연료 유형이 r인 차량 역시 기준(fl=c)에 비해 연비가 4.4312 단위만큼 낮다.

배기량 (displ): 계수 = -1.0495, p-값 = 0.001
배기량이 증가할수록 연비가 기준(trans=auto(av))에 비해 1.0495 단위만큼 낮아진다.

실린더 수 (cyl): 계수 = -1.0756, p-값 = 0.000
실린더 수가 증가할수록 연비가 기준(trans=auto(av))에 비해 1.0756 단위만큼 낮아진다.

통계적으로 유의미하지 않은 변수:
변속기 유형 (trans): 대부분의 변속기 유형은 통계적으로 유의미하지 않다. 이는 변속기 유형이 연비에 큰 영향을 미치지 않음을 의미한다.

종합 분석
구동 방식 (drv):
전륜구동(drv[T.f])과 후륜구동(drv[T.r]) 차량은 기준(drv=4)에 비해 연비가 높다. 이는 구동 방식이 연비에 중요한 영향을 미친다는 것을 나타낸다.

연료 유형 (fl):
연료 유형이 e, p, r인 차량은 기준(fl=c)에 비해 연비가 낮다. 특히 연료 유형 e의 경우 연비가 크게 낮아지는 것을 볼 수 있다. 이는 연료 유형이 연비에 중요한 영향을 미친다는 것을 나타낸다.

배기량 (displ)과 실린더 수 (cyl):
배기량과 실린더 수가 증가할수록 연비가 감소한다. 이는 큰 엔진을 가진 차량이 작은 엔진을 가진 차량보다 연비가 낮다는 것을 의미한다.

변속기 유형 (trans):
대부분의 변속기 유형은 통계적으로 유의미하지 않다. 이는 변속기 유형이 연비에 큰 영향을 미치지 않는다는 것을 나타낸다.

결론
연비에 큰 영향을 미치는 요인은 구동 방식, 연료 유형, 배기량, 그리고 실린더 수이다. 특히 전륜구동과 후륜구동 차량은 연비가 좋고, 연료 유형이 e, p, r인 차량은 연비가 나쁘다. 또한, 배기량과 실린더 수가 많을수록 연비가 낮아진다. 이러한 결과를 바탕으로 차량의 연비 개선을 위해서는 적절한 구동 방식과 연료 유형을 선택하고, 배기량과 실린더 수를 최적화하는 것이 중요할 것이다.

이러한 결론을 내었지만 굉장히 많은 독립 변수에 대해서 선형회귀를 진행하였기 때문에 다중공선성에 의한 문제가 발생할 수 있다고 생각하여 위의 분석을 제외하고도 다중공선성에 의한 영향을 고려한 분석 또한 진행하였다.
VIF를 확인하여 다중공선성이 존재하는지 여부를 확인하고 전진 선택법 및 후진 제거법을 진행하여 선택된 독립 변수 들에 대하여 분석을 진행하려고 한다.

```python
# VIF를 이용한 다중공선성 분석
vif = pd.DataFrame()
vif['Feature'] = X.columns
vif['VIF Factor'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif = vif.sort_values(by='VIF Factor', ascending=False)
vif = vif.reset_index(drop=True)
print(vif)
```

그림 40. VIF를 이용한 다중공선성 분석 코드

위의 코드를 실행하여 얻은 결과는 다음과 같다.

	Feature	VIF Factor
0	Intercept	326.930113
1	fl[T.r]	49.794962
2	fl[T.p]	42.946478
3	trans[T.auto(l4)]	12.676463
4	trans[T.manual(m5)]	10.556522
5	displ	10.151383
6	fl[T.e]	9.073317
7	trans[T.auto(l5)]	8.197361
8	cyl	8.162501
9	fl[T.d]	6.062804
10	trans[T.manual(m6)]	4.561535
11	trans[T.auto(s6)]	4.045862
12	trans[T.auto(l6)]	2.345432
13	drv[T.f]	1.911404
14	trans[T.auto(s4)]	1.616937
15	trans[T.auto(s5)]	1.609091
16	trans[T.auto(l3)]	1.445629
17	drv[T.r]	1.379466

그림 41, 42. 독립 변수들의 VIF 값

VIF 값을 통해서 4개의 독립변수의 VIF 값이 10 이상이며 많은 독립변수의 VIF 값이 3 이상임을 확인했기에 위의 독립변수들에 다중공선성이 존재한다고 보았고, 그에 따라서 전진 선택법 및 후진 제거법을 통해서 독립 변수들을 선택하고 이에 대한 선형 회귀 분석을 진행하려고 한다.

```python
# 전진 선택법 적용
# 종속 변수와 독립 변수 설정
y, X = dmatrices('mpg ~ displ + cyl + trans + drv + fl', data=mpg, return_type='dataframe')

# 전진 선택법 함수 정의
def forward_selection(data, target, significance_level=0.05):
    initial_features = data.columns.tolist()
    best_features = []
    while initial_features:
        remaining_features = list(set(initial_features) - set(best_features))
        new_pval = pd.Series(index=remaining_features)
        for new_column in remaining_features:
            model = sm.OLS(target, sm.add_constant(data[best_features + [new_column]])).fit()
            new_pval[new_column] = model.pvalues[new_column]
        min_p_value = new_pval.min()
        if min_p_value < significance_level:
            best_features.append(new_pval.idxmin())
        else:
            break
    return best_features

selected_features = forward_selection(X, y)
print("Selected features:", selected_features)
```

```
Selected features: ['Intercept', 'cyl', 'drv[T.f]', 'fl[T.d]', 'drv[T.r]', 'fl[T.e]', 'trans[T.auto(l4)]', 'displ']
```

그림 43. 전진 선택법을 통한 독립 변수 선택

```python
# 후진 제거법 적용
y, X = dmatrices('mpg ~ displ + cyl + trans + drv + fl', data=mpg, return_type='dataframe')

# 후진 제거법 함수 정의
def backward_elimination(data, target, significance_level=0.05):
    features = data.columns.tolist()
    while len(features) > 0:
        model = sm.OLS(target, sm.add_constant(data[features])).fit()
        max_p_value = model.pvalues.max()
        if max_p_value > significance_level:
            excluded_feature = model.pvalues.idxmax()
            features.remove(excluded_feature)
        else:
            break
    return features

selected_features = backward_elimination(X, y)
print("Selected features:", selected_features)
```

```
Selected features: ['Intercept', 'trans[T.auto(l4)]', 'drv[T.f]', 'drv[T.r]', 'fl[T.d]', 'fl[T.e]', 'displ', 'cyl']
```

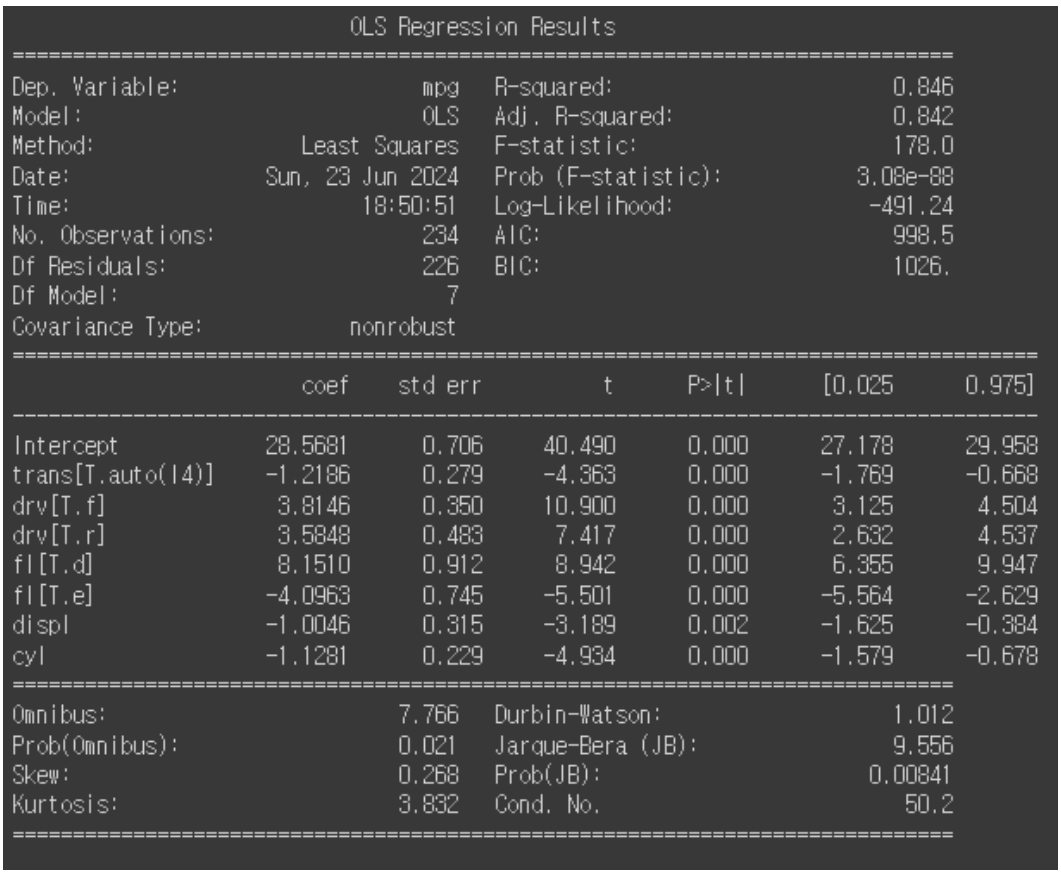
그림 44. 후진 제거법을 통한 독립 변수 선택

전진 선택법 및 후진 제거법을 통하여 선택된 독립변수는 동일하며 ['Intercept', 'cyl', 'drv[T.f]', 'fl[T.d]', 'drv[T.r]', 'fl[T.e]', 'trans[T.auto(l4)]', 'displ'] 이었다. 그러므로 이 독립변수 들에 대해서 다중 선형 회귀 분석을 진행하였다.

```python
# 선택된 독립 변수들에 대한 다중 선형 회귀 분석
X_selected = X[selected_features]
model = sm.OLS(y, X_selected).fit()
print(model.summary())
```

그림 45. 선택된 변수들에 대한 다중 선형 회귀 분석 실행 코드

위의 코드를 실행한 결과는 다음과 같다.



OLS Regression Results

Dep. Variable:	mpg	R-squared:	0.846
Model:	OLS	Adj. R-squared:	0.842
Method:	Least Squares	F-statistic:	178.0
Date:	Sun, 23 Jun 2024	Prob (F-statistic):	3.08e-88
Time:	18:50:51	Log-Likelihood:	-491.24
No. Observations:	234	AIC:	998.5
Df Residuals:	226	BIC:	1026.
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	28.5681	0.706	40.490	0.000	27.178	29.958
trans[T.auto(l4)]	-1.2186	0.279	-4.363	0.000	-1.769	-0.668
drv[T.f]	3.8146	0.350	10.900	0.000	3.125	4.504
drv[T.r]	3.5848	0.483	7.417	0.000	2.632	4.537
fl[T.d]	8.1510	0.912	8.942	0.000	6.355	9.947
fl[T.e]	-4.0963	0.745	-5.501	0.000	-5.564	-2.629
displ	-1.0046	0.315	-3.189	0.002	-1.625	-0.384
cyl	-1.1281	0.229	-4.934	0.000	-1.579	-0.678

Omnibus:	7.766	Durbin-Watson:	1.012
Prob(Omnibus):	0.021	Jarque-Bera (JB):	9.556
Skew:	0.268	Prob(JB):	0.00841
Kurtosis:	3.832	Cond. No.	50.2

그림 46. 선택한 변수들에 대한 다중 선형 회귀 분석 결과

전진 선택법 및 후진 제거법을 통해서 나온 결과로 선택된 데이터들에 대해서만 다중 선형 회귀분석을 진행한 결과

변속기 유형 (trans):

변속기 유형의 경우 하나만 선택되었기에 하나로만 전체 모델을 표현할 수 있다는 것은 변속기의 유형에 따라서 연비에 큰 차이가 없다는 것이다.

구동 방식 (drv):

전륜구동(drv[T.f])과 후륜구동(drv[T.r]) 차량은 기준(drv=4)에 비해 연비가 높다.

이는 구동 방식이 연비에 중요한 영향을 미친다는 것을 나타낸다.

연료 유형 (fl):
연료 유형이 e인 차량은 기준(fl=c)에 비해 연비가 많이 낮다. 그리고 연료 유형이 d인 차량은 기준(fl=c)에 비해 연비가 매우 높습니다. 이는 연료 유형이 연비에 중요한 영향을 미친다는 것을 나타낸다.

배기량 (displ)과 실린더 수 (cyl):
배기량과 실린더 수가 증가할수록 연비가 감소한다. 이는 실린더 수가 증가하면 배기량도 증가하고 이렇게 실린더 수가 많은 엔진을 가진 자동차가 연비가 안 좋다는 것을 나타낸다.

그런데 이때의 결과는 위의 다중공선성을 고려하지 않은 다중 선형 회귀 분석의 결과와 크게 다르지 않으므로 다음과 같은 결론을 내렸다.

즉 연비에 큰 영향을 미치는 요인은 구동 방식, 연료 유형, 배기량 및 실린더 수로 구동 방식이 사륜구동이 아닐 때, 연료가 디젤일 때, 엔진의 배기량이 크지 않을 때 연비가 좋아질 것으로 보인다.

Track 2. 관심있는 주제로부터 군집분석(Cluster Analysis)을 수행하기에 적합한 다변량 데이터를 확보하고 다음 과정을 포함하여 분석을 진행

2.1 데이터 선정 배경과 출처, 포함된 변수의 의미 등을 설명

평소에 NBA에 대한 관심이 많았기에 NBA 선수들의 데이터에 데이터 분석 기초 강의를 통해 배운 내용을 적용하면 흥미로운 내용을 도출할 수 있을 것으로 예상하여 2023/2024 시즌 NBA 선수들의 게임 당 선수의 데이터를 선택함
출처는 Basketball-reference.com으로 Basketball-reference.com에서는 NBA 23/24 시즌의 Player Per Game의 데이터를 콤마(,)로 구분된 텍스트 파일로 제공한다.
이를 엑셀의 기능을 통해 CSV 파일로 변환하여 2324.csv 파일을 만들어 사용했다.

```
# 데이터 분석을 위해 필요한 외부 라이브러리 호출
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

```
# 데이터 불러오기(Stats라는 변수에 저장) 및 확인
Stats = pd.read_csv('/content/2324.csv')
Stats.head()
```

그림 47, 48. 외부 라이브러리 호출 및 2324.csv 로드

Pandas의 read_csv를 이용해 자료를 불러와 head() 함수를 이용해 데이터를 제대로 가져왔는지 확인했다.

	Rk	Player	Pos	Age	Tm	G	GS	MP	FG	FGA	...	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS	Player-additional
0	1	Precious Achiuwa	PF-C	24	TOT	74	18	21.9	3.2	6.3	...	2.6	4.0	6.6	1.3	0.6	0.9	1.1	1.9	7.6	achiupr01
1	2	Bam Adebayo	C	26	MIA	71	71	34.0	7.5	14.3	...	2.2	8.1	10.4	3.9	1.1	0.9	2.3	2.2	19.3	adebaba01
2	3	Ochai Agbaji	SG	23	TOT	78	28	21.0	2.3	5.6	...	0.9	1.8	2.8	1.1	0.6	0.6	0.8	1.5	5.8	agbajoc01
3	4	Santi Aldama	PF	23	MEM	61	35	26.5	4.0	9.3	...	1.2	4.6	5.8	2.3	0.7	0.9	1.1	1.5	10.7	aldamsa01
4	5	Nickeil Alexander-Walker	SG	25	MIN	82	20	23.4	2.9	6.6	...	0.4	1.6	2.0	2.5	0.8	0.5	0.9	1.7	8.0	alexani01

그림 49. head() 결과

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 572 entries, 0 to 571
Data columns (total 31 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Rk                572 non-null    int64
 1   Player            572 non-null    object
 2   Pos               572 non-null    object
 3   Age               572 non-null    int64
 4   Tm                572 non-null    object
 5   G                 572 non-null    int64
 6   GS                572 non-null    int64
 7   MP                572 non-null    float64
 8   FG                572 non-null    float64
 9   FGA               572 non-null    float64
 10  FG%               568 non-null    float64
 11  3P                572 non-null    float64
 12  3PA               572 non-null    float64
 13  3P%               540 non-null    float64
 14  2P                572 non-null    float64
 15  2PA               572 non-null    float64
 16  2P%               567 non-null    float64
 17  eFG%              568 non-null    float64
 18  FT                572 non-null    float64
 19  FTA               572 non-null    float64
 20  FT%               537 non-null    float64
 21  ORB               572 non-null    float64
 22  DRB               572 non-null    float64
 23  TRB               572 non-null    float64
 24  AST               572 non-null    float64
 25  STL               572 non-null    float64
 26  BLK               572 non-null    float64
 27  TOV               572 non-null    float64
 28  PF                572 non-null    float64
 29  PTS               572 non-null    float64
 30  Player-additional 572 non-null    object
dtypes: float64(23), int64(4), object(4)
memory usage: 138.7+ KB
```

그림 50, 51. Data type 구분

이후 info() 함수를 통해 mpg 데이터 세트에 포함된 각 변수의 의미와 데이터 구조를 파악했다.

데이터 정보 및 타입

0. RK: 이름을 알파벳 순으로 정렬해 오름차순으로 부여한 숫자(int64)
1. Player: 이름(object)
2. Pos: 포지션(object)
3. Age: 나이(int64)
4. Tm: 소속 팀(object)
5. G: 출전 경기 수(int64)
6. GS: 스타팅 출천 경기 수(int64)
7. MP: 경기 당 출천 시간(분)(float64)
8. FG: 경기 당 필드골(float64)
9. FGA: 경기 당 필드골 시도(float64)
10. FG%: 필드골 성공률(float64)
11. 3P: 경기 당 3점슛(float64)
12. 3PA: 경기 당 3점슛 시도(float64)
13. 3P%: 3점슛 성공률(float64)
14. 2P: 경기 당 2점슛(float64)

15. 2PA: 경기 당 2점슛 시도(float64)
16. 2P%: 2점슛 성공률(float64)
17. eFG%: 2점슛과 3점슛 사이의 효율 차이를 보정한 필드골 성공률(float64)
18. FT: 경기 당 자유투(float64)
19. FTA: 경기 당 자유투 시도(float64)
20. FT%: 자유투 성공률(float64)
21. ORB: 경기 당 공격 리바운드(float64)
22. DRB: 경기 당 수비 리바운드(float64)
23. TRB: 경기 당 총 리바운드(float64)
24. AST: 경기 당 어시스트(float64)
25. STL: 경기 당 스틸(float64)
26. BLK: 경기 당 블록(float64)
27. TOV: 경기 당 턴오버(float64)
28. PF: 경기 당 개인파울(float64)
29. PTS: 경기 당 총 점(float64)
30. Player-additional: 동일 이름을 구분하기 위한 선수 ID(object)

2.2 다음의 각 과정을 포함하여 분석을 진행

(A) 군집분석을 포함하여 예상되는 분석 절차를 구체화

군집분석을 진행할 때의 예상되는 분석 절차는 다음과 같다.

1. 결측치, 이상치 확인 및 제거, 분석에 필요한 데이터들에 대한 선택을 포함한 데이터 전처리
이때 분석에 필요한 데이터 같은 경우 농구에서 1차 지표라고 부르는 데이터들에 대한 선택을 포함한다.

2. k-means 군집분석을 통한 포지션 별 데이터와 군집 별 데이터 간의 비교를 통해 군집 분류로 포지션을 찾아낼 수 있는지 확인
이때 포지션별 데이터는 포지션의 object 값을 PG ~ C에 대하여 1 ~ 5로 매핑을 하여 이에 대한 데이터를 출력하고
군집 별 데이터 같은 경우에는 군집을 5개로 나누어 군집분석을 했을 때 군집이 5개의 포지션을 나타낼 수 있도록 하여 출력하려고 한다.

3. 결과 확인 이후 성공 혹은 실패 이유 분석
성공했다면 어떤 이유로 성공했는지, 실패했다면 왜 실패했는지에 대한 분석을 통해 군집분석의 정확도를 향상시킬 수 있을 것으로 생각한다.

(B) 결측치, 이상치 등이 있는지 확인하고 필요하다면 적절한 전처리 과정을 수행

isnull().sum()을 통해 각 변수에 대해 결측치의 개수를 확인하여 결측치를 확인했다.

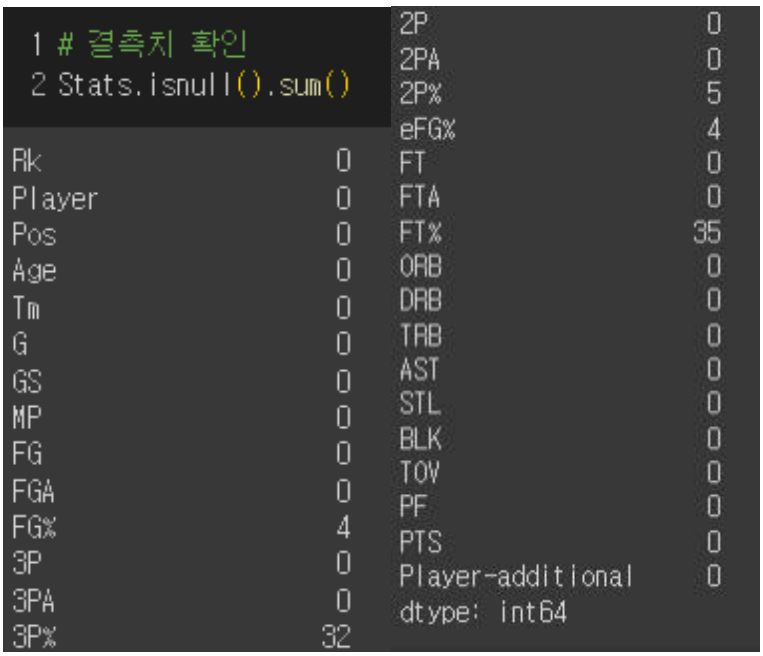


그림 52, 53. 결측치 확인

결측치 같은 경우 3P%, 2P%, eFG%, FT%에 대해서 나타났는데 이 값들은 모두 확률에 대한 값으로 슛 성공 횟수/슛 시도 횟수로 나타나는 값들이다.
이는 전체 시즌에서 시도를 아예 하지 않은 경우 0으로 나누기 때문에 결측치가 된 것으로 판단하여 다음 코드를 통해 결측치 부분을 모두 0으로 변환하였다.

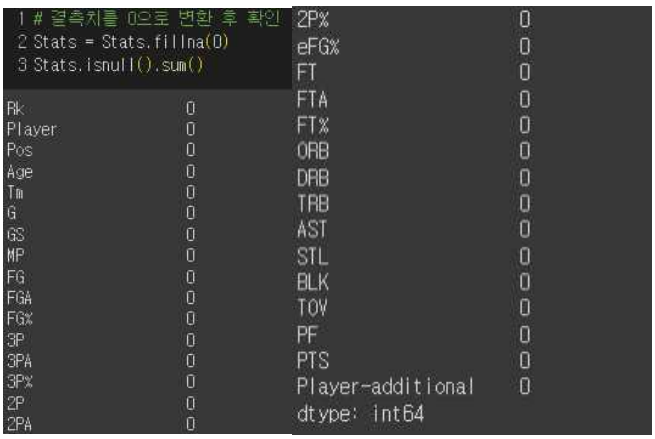


그림 52, 53. 결측치 처리

위의 실행 결과를 통해서 결측치가 처리된 것을 확인할 수 있다.
현재 군집 분석에서는 전체 데이터에 대한 군집분석이 아닌 중요하다고 생각되는 데이터들에 대한 군집분석을 하려고 하기에 중요하다고 생각되는 데이터를 선택하려고 했다.
선택된 데이터는 1차 스탯인 eFG%, 출장 시간, 리바운드, 스틸, 어시스트, 블록, 포인트과 군집분석의 대상인 포지션이다.

그리고 이때 이런 선택된 데이터에 대해서 전처리를 통해서 데이터를 가공하고자 한다.
우선 포지션 데이터에 대한 전처리 및 매핑을 진행하였다.

```
# 포지션 데이터에 대한 전처리 및 매핑
Stats['Pos'] = Stats['Pos'].apply(lambda x: x.split('-')[0]) # 멀티 포지션으로 된 선수들의 포지션을 더 작은 사이즈의 포지션으로 한정
pos_mapping = {'PG': 1, 'SG': 2, 'SF': 3, 'PF': 4, 'C': 5}
Stats['Pos_Num'] = Stats['Pos'].map(pos_mapping)
```

그림 54. 포지션 데이터에 대한 전처리(포지션 매핑)

정성적 데이터인 포지션을 다른 데이터와의 회귀 분석에 활용하기 위해 포지션이 멀티포지션인 선수의 경우 Precious Achiuwa 선수와 같이 PF-C 같은
작은 포지션-큰 포지션의 표현형식을 가지고 있기 때문에 '-'을 기준으로 앞의 값으로 반환함으로 더 작은 사이즈의 포지션으로 한정했음
포지션을 회귀분석에 사용하기 위해 포지션이 PG일때 1, SG일 때 2, SF일 때 3, PF일 때 4, C일때 5로 하여 Pos_Num이라는 새로운 데이터를 생성해 이때의 5개의 포지션에 따른 다른 변수들의 분포와 억지로 군집을 5개로 나눴을 때의 분포가 유사하게 나오는지 판단하여 군집분석을 실행하려고 한다.

```
# 데이터 전처리
selected_columns = ['Pos_Num', 'MP', 'eFG%', 'TRB', 'STL', 'AST', 'BLK', 'PTS'] # 분석을 위한 데이터 선택
Stats = Stats[selected_columns]
Stats.head()
```

그림 55. 데이터 선택 및 전처리

	Pos_Num	MP	eFG%	TRB	STL	AST	BLK	PTS
0	4	21.9	0.529	6.6	0.6	1.3	0.9	7.6
1	5	34.0	0.529	10.4	1.1	3.9	0.9	19.3
2	2	21.0	0.483	2.8	0.6	1.1	0.6	5.8
3	4	26.5	0.528	5.8	0.7	2.3	0.9	10.7
4	2	23.4	0.560	2.0	0.8	2.5	0.5	8.0

그림 56. head() 결과

head() 함수의 결과를 통해서 데이터 전처리가 정상적으로 처리된 것을 확인할 수 있다.
그리고 데이터 분석에 들어가기에 앞서 이 데이터에 대해서 이상치를 확인하였다.

```
# 이상치 확인
sns.boxplot(data=Stats)
```

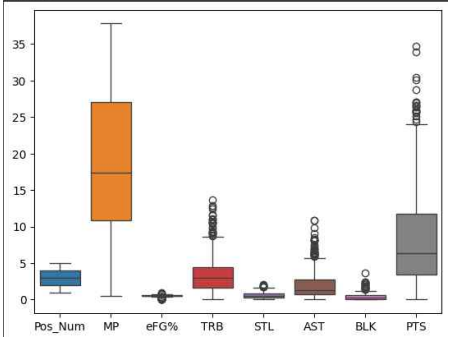


그림 57. 이상치 확인 결과

sns.boxplot() 함수 실행 결과 튀는 값들은 있었으나 데이터를 보았을 때 동떨어진 데이터들도 충분히 가능한 수치라고 생각해 이상치에 대한 처리를 하지 않았다.

데이터의 전처리를 마무리하고 군집분석을 수행하였다.

(C) 데이터의 특징을 파악하고 자유로운 분석을 통해 의미를 도출
군집분석에 앞서 포지션에 따라서 다른 데이터들이 어떤 의미를 보여주는지 확인하였다. 그를 위해 pairplot을 통해서 포지션들에 대한 다른 데이터의 분포를 확인했다.

```
1 # 포지션과 다른 데이터 간의 Pairplot
2 sns.pairplot(Stats, hue='Pos_Num', diag_kind='kde')
3 plt.show()
```

그림 58. 포지션과 다른 데이터 간의 관계 분석을 위한 코드

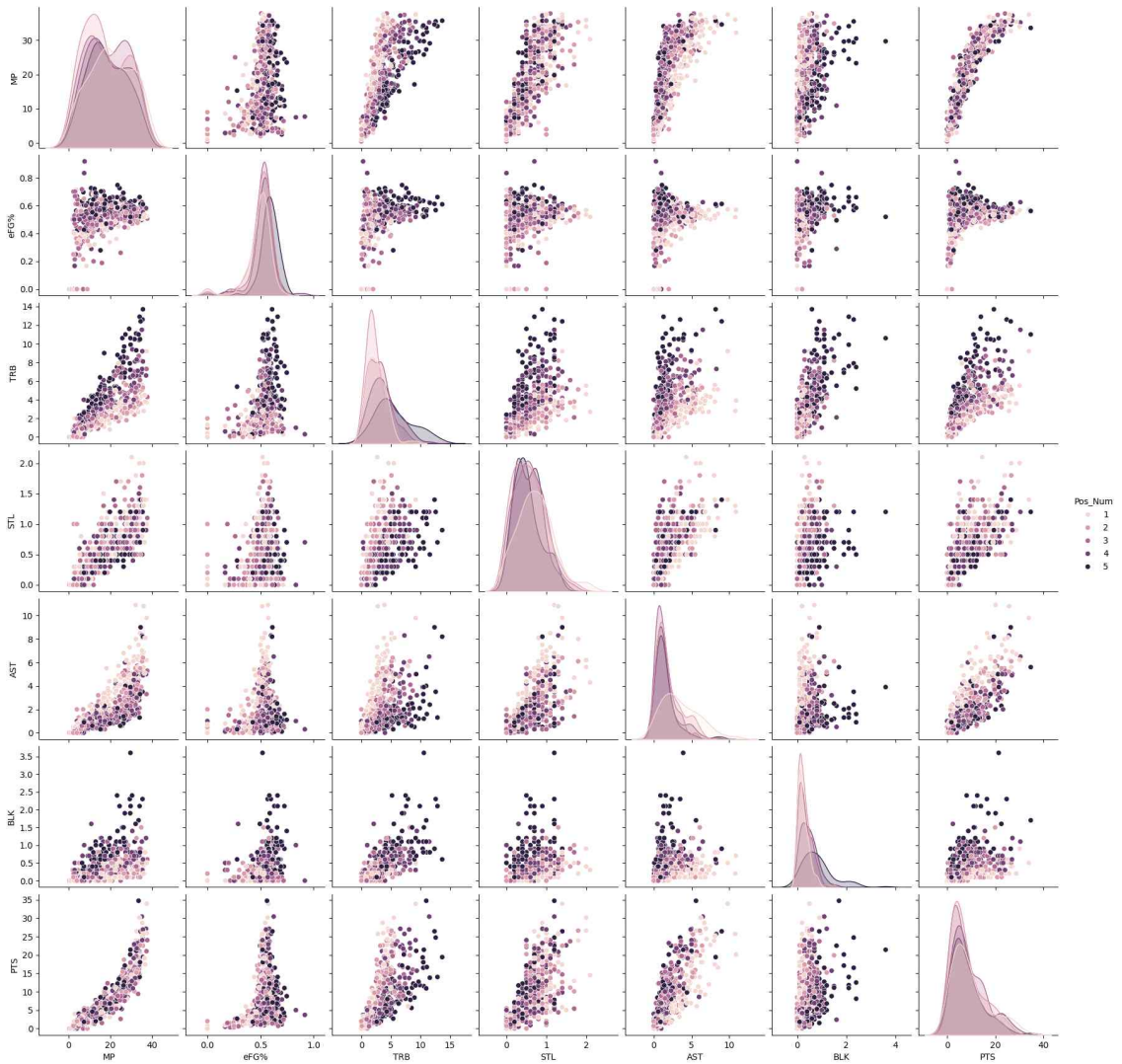


그림 59. 포지션에 대한 다른 데이터들의 pairplot

pairplot의 분포도와 포지션간의 데이터를 분석하면 다음과 같다.
1. PG(포인트 가드): 평균적으로 리바운드, 블락 숫자가 낮고, 스틸, 어시스트가 높음, 다른 포지션과 비슷하지만 득점에 있어서 평균적으로 약간의 우위가 있는 것으로 보임, 확연한 차이점은 어시스트에 있음
2. SG(슈팅 가드): 포인트 가드 포지션과 유사한 분포를 보이지만 슈팅 성공률이 높음
3. SF(스몰 포워드): 포인트 가드 포지션과 센터의 중간 정도의 분포를 보임
4. PF(파워 포워드): 센터 포지션과 유사한 분포를 보이지만 블록 수치가 확연히 낮음
5. C(센터): 평균적으로 스틸, 어시스트가 낮지만, 리바운드, 블락의 숫자가 높음, eFG%에 있어서 다른 포지션에 비해 평균적으로 높은 값을 보임

그리고 이에 대해서 군집분석을 수행해 비슷한 분포도를 보이는지 확인하려고 한다.
이때 군집분석을 할 때 포지션의 영향을 제외하기 위해 포지션을 제외한 데이터에 대해서 군집분석을 진행하였고, 위에서 언급했듯이 군집의 수는 5개로 정했다.

```python
# 군집 분석(포지션의 영향을 제외하기 위해 포지션 제외)
Stats_cluster = Stats.copy()
Stats_cluster = Stats_cluster.drop(columns='Pos_Num')
kmeans = KMeans(n_clusters=5) # 포지션 별 구분이 가능할 수 있도록 5개로 구분
kmeans.fit(Stats_cluster)
Stats_cluster['Cluster'] = kmeans.labels_
sns.pairplot(Stats_cluster, hue='Cluster', diag_kind='kde')
plt.show()
```

그림 60. 포지션을 제외한 데이터의 군집분석 코드

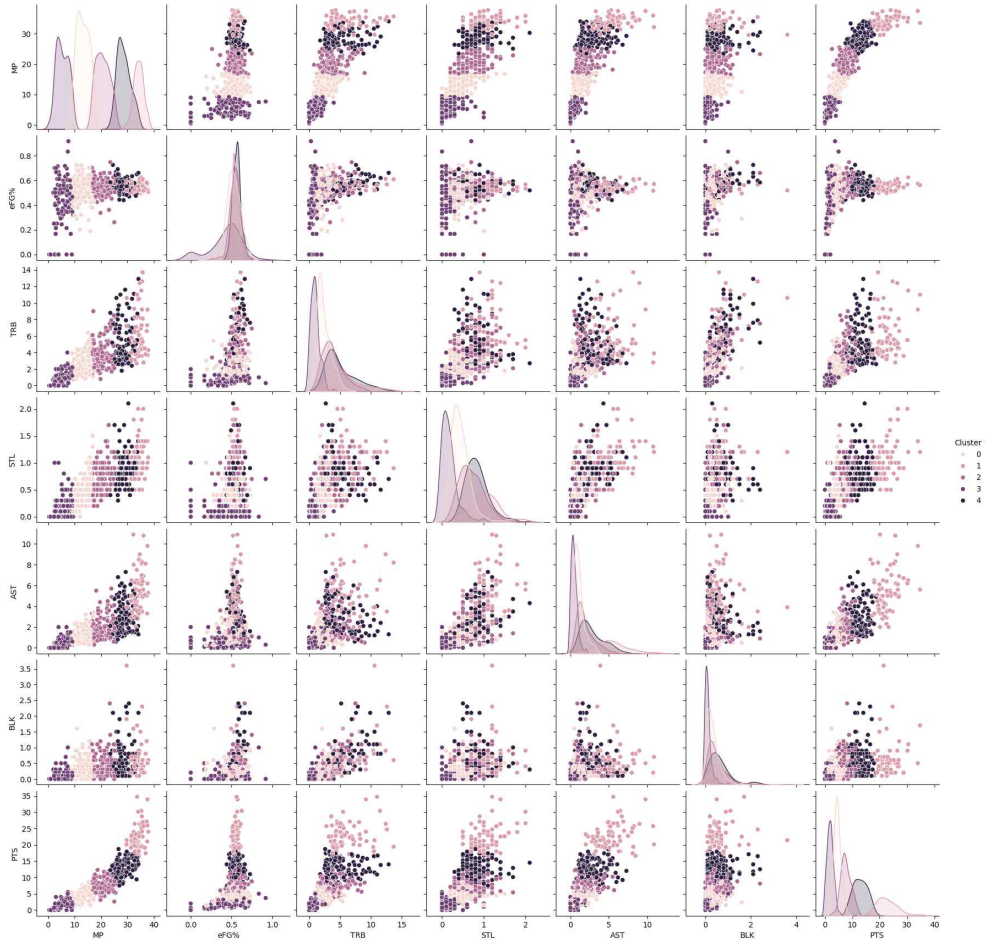


그림 61. 포지션을 제외한 데이터의 군집분석 결과 pairplot

군집분석 결과 이전의 포지션별 데이터와 다르게 출전 시간이 더 군집 분류에 더 큰 영향을 미친것으로 확인되었다.
군집을 5개로 나누어 포지션별의 각 데이터에 대한 특성이 비슷하게 나타난다면 군집분석을 통해서 포지션을 정할 수 없는 선수에 대해서도 포지션 분류를 할 수 있을 것으로 예상했지만 실패하였다. 그래서 출전 시간을 제외하고 군집 분류를 시행하였다.

```
# 군집 분석(출전 시간의 영향을 제외하기 위해 출전시간도 제외)
Stats_cluster = Stats.copy()
Stats_cluster = Stats_cluster.drop(columns='Pos_Num') # 포지션 제외
Stats_cluster = Stats_cluster.drop(columns='MP') # 출전 시간 제외
kmeans = KMeans(n_clusters=5) # 포지션에 대한 군집화가 가능하도록 5개로 설정
kmeans.fit(Stats_cluster)
Stats_cluster['Cluster'] = kmeans.labels_
sns.pairplot(Stats_cluster, hue='Cluster', diag_kind='kde')
plt.show()
```

그림 62. 포지션과 출전 시간을 제외한 데이터의 군집분석 코드

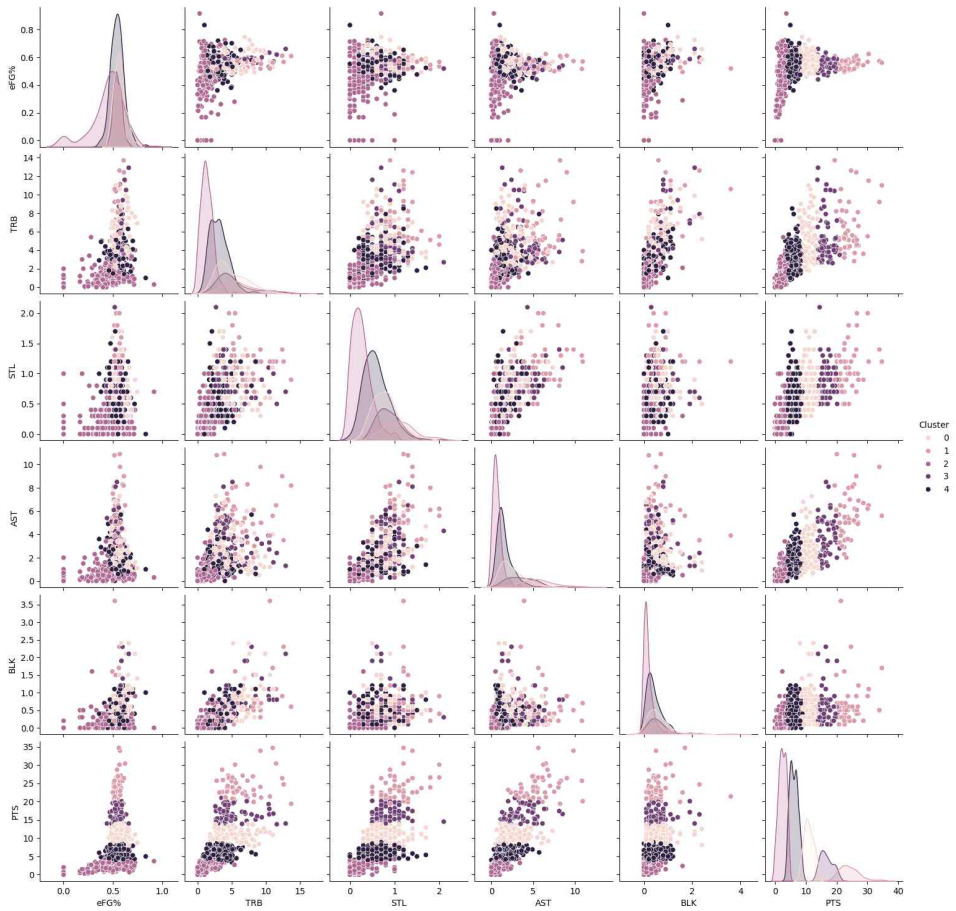


그림 63. 포지션과 출전 시간을 제외한 데이터의 군집분석 결과 pairplot

그런데 이때 결과가 출전 시간을 제외하기 전 군집분석 결과 pairplot과 거의 똑같았다. 그래서 다른 데이터에 대해서 출전 시간이 영향을 미치는 것으로 보이기 때문에 군집 분류가 실패한 것으로 분석했다. 그래서 포지션에 대한 군집화 효과를 보정해주기 위해서 포지션을 포함하면 어떻게 될지 한번 분석해봤다.

```python
# 포지션 데이터가 포함된 군집 분석(출전 시간은 제외하여 출전 시간에 대한 영향을 줄임)
Stats_cluster = Stats.copy()
Stats_cluster = Stats_cluster.drop(columns='MP') # 출전 시간만 제외
kmeans = KMeans(n_clusters=5)
kmeans.fit(Stats_cluster)
Stats_cluster['Cluster'] = kmeans.labels_
sns.pairplot(Stats_cluster, hue='Cluster', diag_kind='kde')
plt.show()
```

그림 64. 포지션은 포함, 출전 시간을 제외한 데이터의 군집분석 코드

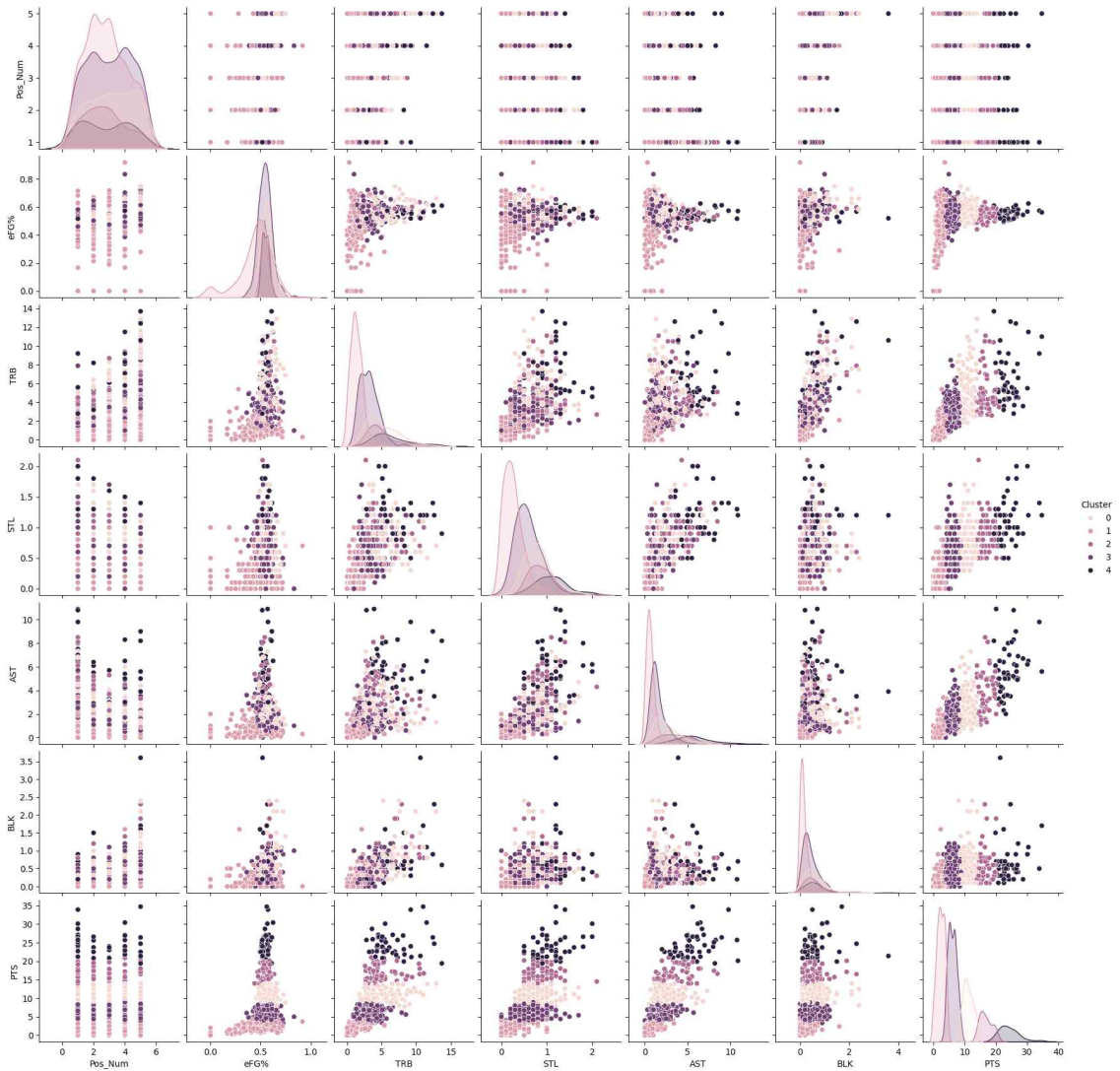


그림 65. 포지션은 포함, 출전 시간을 제외한 데이터의 군집분석 결과 pairplot

Pos_Num을 변수에 포함했음에도 포지션에 따른 pairplot의 결과와 상이한 결과가 나왔다. 이는 이미 다른 변수들에 MP에 대한 영향이 클 것으로 판단하여 MP와 다른 변수들에 대해서 각각의 선형 회귀 분석 실시하였다.

```
# 데이터 각각에 대한 선형 회귀 분석
sns.lmplot(data=Stats, x='MP', y='Pos_Num')
sns.lmplot(data=Stats, x='MP', y='eFG%')
sns.lmplot(data=Stats, x='MP', y='TRB')
sns.lmplot(data=Stats, x='MP', y='STL')
sns.lmplot(data=Stats, x='MP', y='AST')
sns.lmplot(data=Stats, x='MP', y='BLK')
sns.lmplot(data=Stats, x='MP', y='PTS')
```

그림 66. 출전 시간에 대한 데이터 각각의 선형 회귀 분석 코드

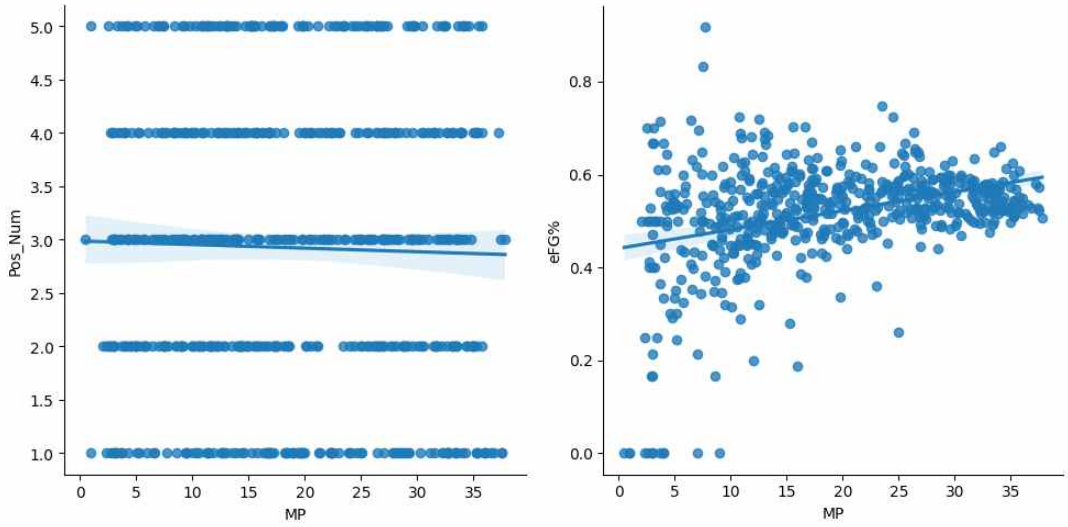


그림 67, 68. 출전 시간에 대한 포지션, eFG% 선형 회귀 분석

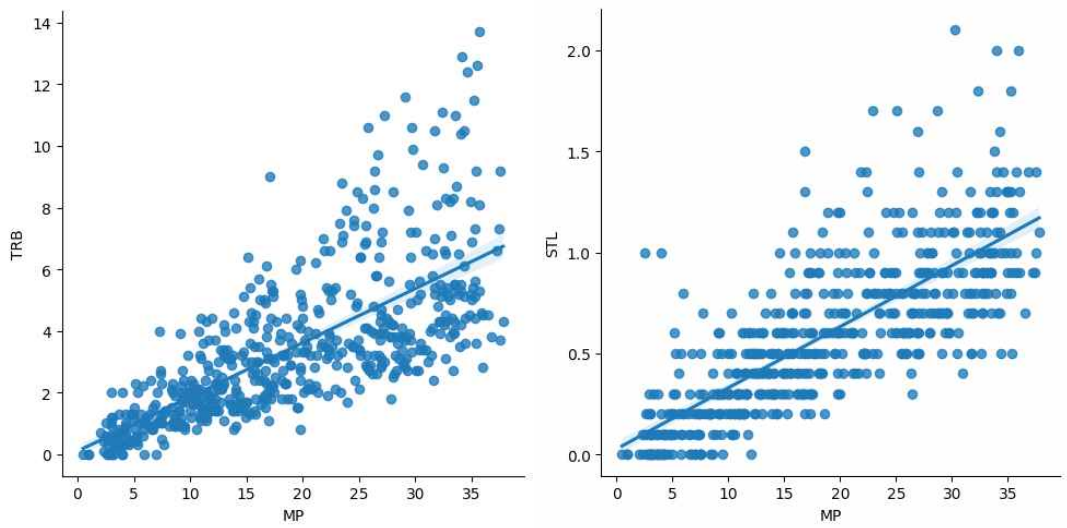


그림 69, 70. 출전 시간에 대한 리바운드, 스틸 선형 회귀 분석

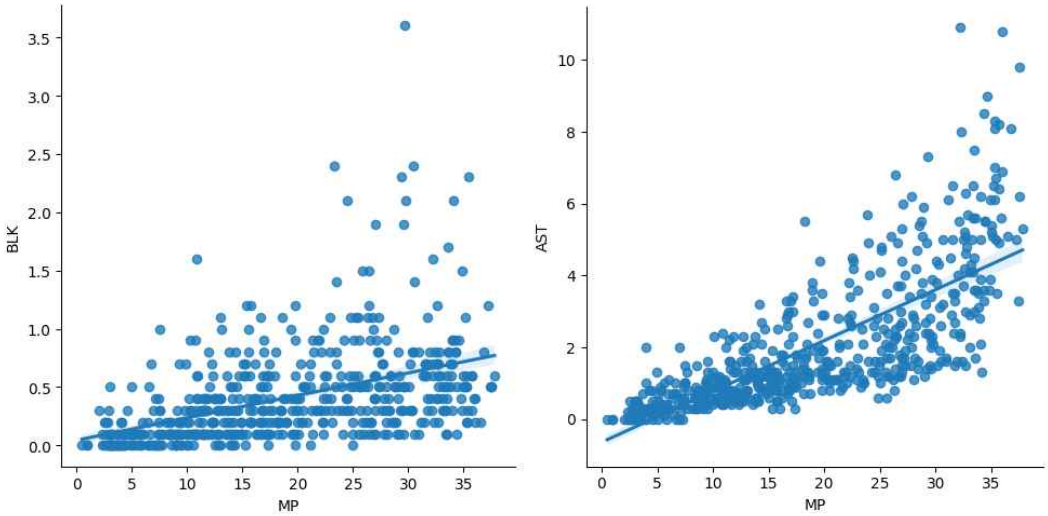


그림 71, 72. 출전 시간에 대한 블락, 어시스트 선형 회귀 분석

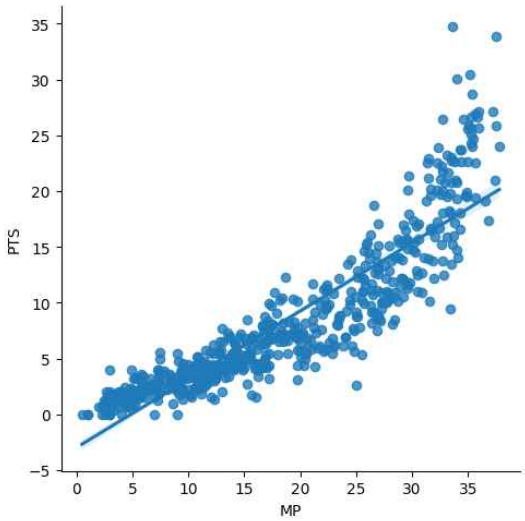


그림 73. 출전 시간에 대한 득점 선형 회귀 분석

회귀분석 결과로서 이미 출전 시간에 대해서 다른 데이터들이 선형관계를 가지고 있음이 판명되었고 이에 따라서 출전 시간을 제외하더라도 출전 시간이 포함된 효과를 내는 것이 확인되었다. 선수들에 대한 군집화는 지금과 같은 이른바 1차 스탯으로 표현되는 데이터들에 대해서는 출전 시간에 대해서 밖에 군집화가 진행되지 않음을 증명하였다.
이는 생각해보면 당연한 것이 다른 1차 스탯의 경우 출전시간이 길어질 경우 스탯을 쌓을 확률이 높아지기 때문이다.

1차 스탯을 출전 시간과 성공률 등에 가중치를 두어 가공한 2차 스탯으로 군집 분석을 진행하면 포지션별 군집화가 이루어질 수 있을지에 대해서 궁금하게 되었고 나중에 기회가 된다면 2차 스탯에 대한 군집분석을 진행하고 싶다.

2.3 다음을 고려하여 군집분석을 수행

위의 실험에서 원하는 결과 대로 군집분석을 하는 데 실패하였기에 처음 데이터를 재가공하여 추가 군집분석을 진행하였다.

```
# Data scaling 적용 분석
Stats = pd.read_csv('/content/2324.csv')

# 결측치 제거 및 확인
Stats = Stats.fillna(0)
Stats.isnull().sum()
```

그림 74. 추가 회귀분석 진행을 위한 데이터 reload 및 결측치 제거 및 확인 코드

```
Rk                0
Player            0
Pos               0
Age               0
Tm                0
G                 0
GS                0
MP                0
FG                0
FGA               0
FG%               0
3P                0
3PA               0
3P%               0
2P                0
2PA               0
2P%               0
eFG%              0
FT                0
FTA               0
FT%               0
ORB               0
DRB               0
TRB               0
AST               0
STL               0
BLK               0
TOV               0
PF                0
PTS               0
Player-additional 0
dtype: int64
```

그림 75, 76. 결측치 제거 확인

새로 진행할 군집분석의 목적은 슛을 많이 던지는 선수가 잘 던지는 선수인가? 에 대한 궁금증을 해결하기 위해 진행하였다. 또한 2점 슛을 잘 던지는 선수가 3점 슛 또한 잘 던지는 지에 대해서도 궁금하였다.
이를 위해서 데이터를 추가로 가공하였다.

```
# 던지는 확률과 던지는 횟수에 대한 군집 분석
selected_columns = ['2PA', '2P%', '3PA', '3P%']
Stats = Stats[selected_columns]
Stats.head()
```

그림 77. 데이터 가공 코드

	2PA	2P%	3PA	3P%
0	5.0	0.562	1.3	0.268
1	13.7	0.528	0.6	0.357
2	2.8	0.523	2.7	0.294
3	4.3	0.534	5.0	0.349
4	2.5	0.517	4.1	0.391

그림 78. head() 결과

(A) 군집분석을 수행할 때 Data Scaling이 필요한 이유 및 Data Scaling 적용

Data Scaling이란 데이터의 값의 범위를 조정하는 것으로 전처리 과정에 속한다. 데이터 값의 범위가 다 제각각이기에 범위 차이가 클 경우, 모델을 학습할 때 0으로 수렴하거나, 무한으로 발산할 수 있다. 그렇기에 피처들의 데이터 분포나 범위를 조정해 줌으로써 데이터 분석 시 더 유용한 결과가 나올 수 있도록 도와준다.
그리고 군집분석은 데이터의 유사성을 측정하여 높은 대상 집단을 분류하고, 군집간 상이성을 규명하는 방법이다. 이는 비지도 학습이기에 predict와 raw data를 대조하지 않는다. 군집분석을 할 때는 그러한 과정이 필요한데 이는 왜곡 정도가 높은 데이터에 K-Means를 적용할 경우, 중심의 개수를 증가시키더라도 변별력이 떨어지는 군집화가 수행되기 때문이다. 그래서 로그 변환 스케일링 등 여러 가지 방법을 통해 데이터 스케일링을 적용하고 군집분석을 실시한다. 혹은 넓은 범위를 가질 때 평균이 0 분산이 1인 가우시안 정규분포로 조절해 수치들의 영향을 조금 변화시켜 더 유의미한 결과를 이끌어 낸다.

이때 2PA와 3PA와 같은 슛 시도 횟수는 2P%, 3P%의 0 ~ 1 사이에 분포하는 확률에 비해 넓은 범위를 가지고 있기에 정규화를 통한 Data Scaling을 진행하였다.

```
1 # 정규화
2 scaler = MinMaxScaler() # 최대, 최소값을 0과 1로 맞춰줌
3 Stats_normalized = scaler.fit_transform(Stats) # 각 데이터의 평균을 0, 표준편차를 1로 정규화
4 Stats_normalized = pd.DataFrame(Stats_normalized, columns=selected_columns)
```

그림 79. 정규화를 통한 Data Scaling 코드

정규화를 적용한 데이터와 그렇지 않은 데이터 모두 군집분석하고 이에 대한 비교 또한 추가적으로 진행하려고 한다.
그리고 이때 군집분석을 위한 최적 군집수를 정할 수 있는 두 가지 방법을 적용하려고 했다.
하나는 Elbow Method로 군집 내 평균 제곱 오차를 계산하여 그래프를 그린다. 이때 군집 내 평균 제곱 오차는 각 데이터 포인트와 해당 포인트가 속한 군집 중심 간의 거리의 제곱합을 의미한다. 군집수가 증가할수록 군집 내 평균 제곱 오차는 감소하지만, 특정 지점에서 그 감소율이 급격히 줄어들게 된다. 이 지점을 Elbow라고 하며 이 지점의 군집 수가 최적의 군집 수로 간주된다.
다른 하나는 Silhouette Score로 각 데이터 포인트의 군집 내 응집성과 군집 간 분리도를 측정하여 최적의 군집 수를 결정하는 방법이다. Silhouette Score는 -1에서 1 사이의 값을 가지며, 값이 1에 가까울수록 해당 데이터 포인트가 적절한 군집에 속해 있음을 나타낸다. 전체 데이터 세트에 대한 평균 Silhouette Score를 계산하여 가장 높은 값을 가지는 군집 수가 최적의 군집 수로 선택된다. Silhouette Score는 군집 수가 증가함에 따라 증가하다가 특정 지점에서 감소하기 시작하는데, 이 지점이 최적의 군집 수로 간주된다.

최적 군집 수 판단을 위한 함수 코드와 실행 결과는 다음과 같다.

```python
# 최적 군집수 판단을 위한 함수 정의(Elbow Method, Silhouette Score)
def plot_elbow_and_silhouette(data, max_clusters=10):
    """
    데이터에 대해 Elbow Method와 Silhouette Score를 계산하고 시각화하는 함수.

    Parameters:
    data (pd.DataFrame): 클러스터링을 할 데이터프레임.
    max_clusters (int): 최대 클러스터 수. 기본값은 10.

    Returns:
    None
    """
    # Elbow Method 사용
    ssd = []
    K = range(1, max_clusters + 1)
    for k in K:
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(data)
        ssd.append(kmeans.inertia_)

    # 결과 시각화(Elbow Method)
    plt.figure(figsize=(14, 5))

    plt.subplot(1, 2, 1)
    plt.plot(K, ssd, 'bx-')
    plt.xlabel('Number of clusters')
    plt.ylabel('Sum of squared distances')
    plt.title('Elbow Method For Optimal k')

    # Silhouette Score 계산
    silhouette_scores = []
    for k in range(2, max_clusters + 1):
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(data)
        score = silhouette_score(data, kmeans.labels_)
        silhouette_scores.append(score)

    # 결과 시각화(Silhouette Score)
    plt.subplot(1, 2, 2)
    plt.plot(range(2, max_clusters + 1), silhouette_scores, 'bx-')
    plt.xlabel('Number of clusters')
    plt.ylabel('Silhouette Score')
    plt.title('Silhouette Score For Optimal k')

    plt.tight_layout()
    plt.show()

plot_elbow_and_silhouette(Stats)
plot_elbow_and_silhouette(Stats_normalized)
```

그림 80. Elbow Method 및 Silhouette Score 코드

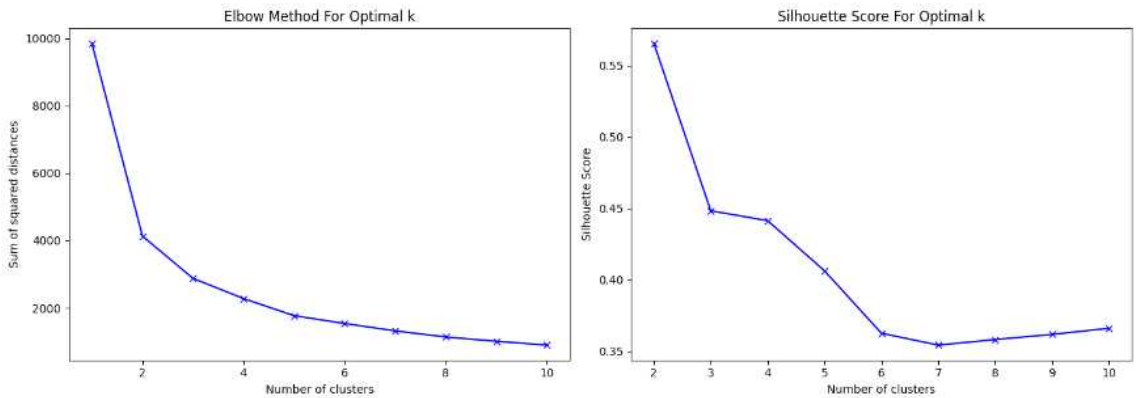


그림 81. Elbow Method 및 Silhouette Score 코드 실행 결과(Not Normalized)

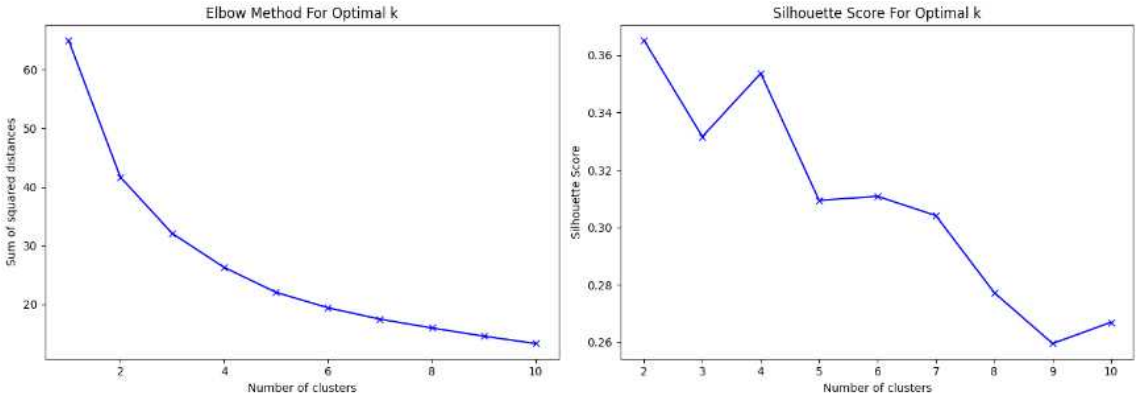


그림 82. Elbow Method 및 Silhouette Score 코드 실행 결과(Normalized)

정규화된 경우, 정규화되지 않은 경우 두 가지 모두에 대해서 최적 군집 수는 Elbow Method와 Silhouette Score에 대해서 모두 2로 나왔다.
군집 수가 2일 때가 최적이기에 군집 수를 2로 두고 군집분석을 시행했다.

(B) 군집분석을 수행하여 군집별 데이터를 추출한 후, 군집별 특징을 자유롭게 비교하여 분석

아래의 코드를 실행하여 정규화된 경우와 그렇지 않은 경우에서 모두 군집분석을 실행하였다.

```
# 군집 분석(정규화 적용 이전)
Stats_cluster = Stats.copy()
kmeans = KMeans(n_clusters=2)
kmeans.fit(Stats_cluster)
Stats_cluster['Cluster'] = kmeans.labels_
sns.pairplot(Stats_cluster, hue='Cluster', diag_kind='kde')
plt.show()
```

그림 83. 군집분석 코드(Not Normalized)

```
# 군집 분석(정규화)
Stats_cluster = Stats_normalized.copy()
kmeans = KMeans(n_clusters=2)
kmeans.fit(Stats_cluster)
Stats_cluster['Cluster'] = kmeans.labels_
sns.pairplot(Stats_cluster, hue='Cluster', diag_kind='kde')
plt.show()
```

그림 84. 군집분석 코드(Normalized)

그리고 그 실행 결과는 아래 그림과 같다.

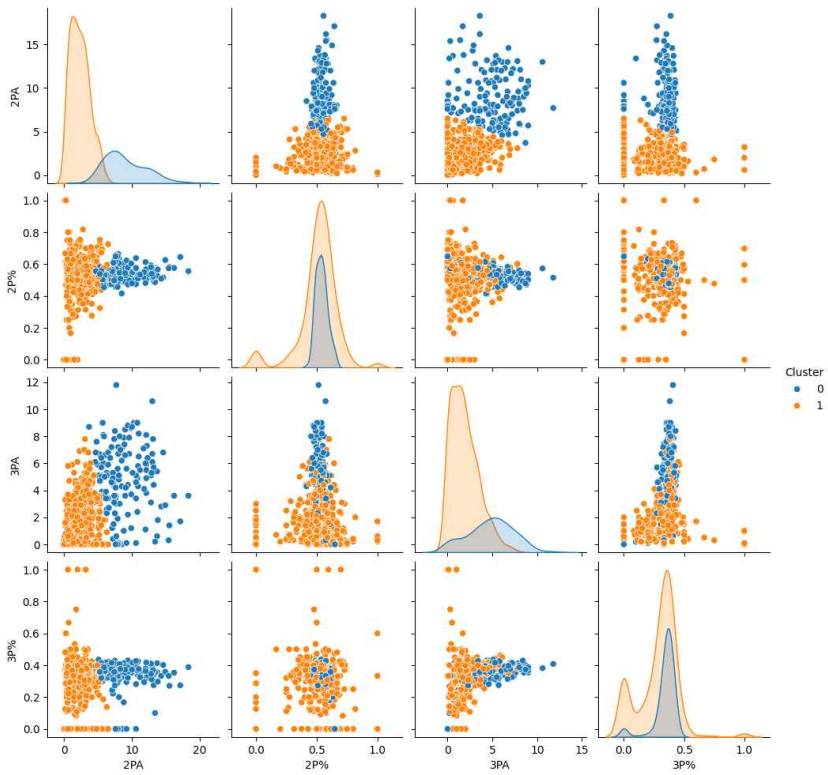


그림 85. 군집분석 실행 결과(Not Normalized)

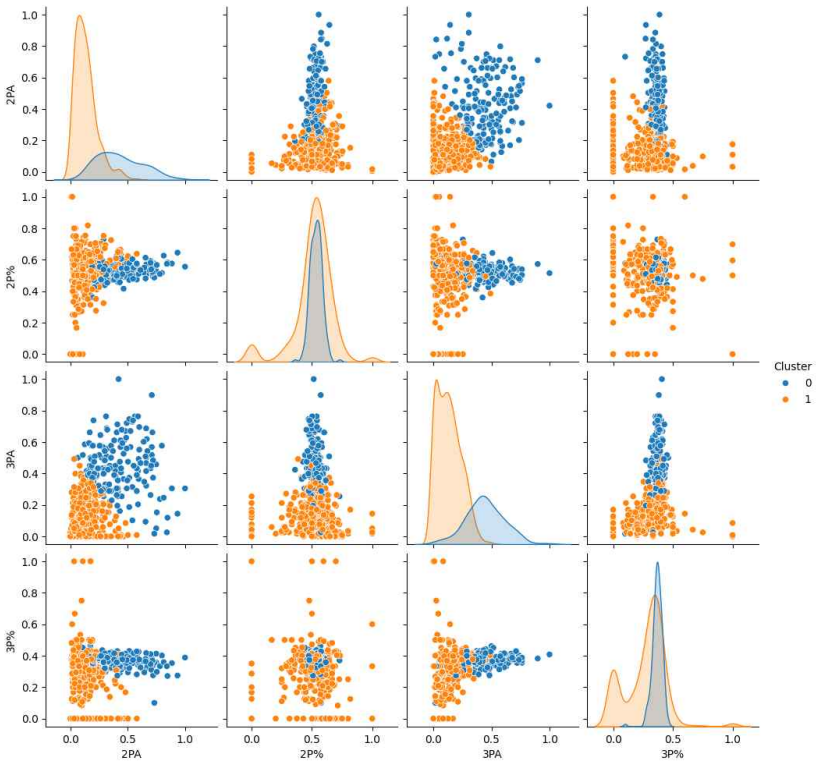


그림 86. 군집분석 실행 결과(Normalized)

정규화가 되지 않은 데이터에 대한 군집분석에 대해서 두 개의 군집에 대한 분석을 해보면

군집 1
2점 슛 시도: 평균 8회 정도
2점 슛 성공률: 거의 50%
3점 슛 시도: 평균 5회 이상이지만 분포가 다양함
3점 슛 성공률: 거의 40%
군집 2:
2점 슛 시도: 평균 5회 이하
2점 슛 성공률 : 거의 50%
3점 슛 시도: 평균 1 ~ 2회 사이
3점 슛 성공률 거의 40%, 군집 1의 그래프의 분포와 거의 비슷함

많이 던지는 그룹이 더 잘 던지는 모습이 보이지 않는다. 그래서 평균적으로 거의 비슷한 성공률을 가지는 두 그룹의 시도 횟수를 가르는 차이점에 대해서는 확인해 볼 수 없다. 정규화를 통한 군집 분석을 통해서 이 차이점을 확인해 보려고 한다.

정규화가 된 데이터에 대한 군집분석에 대해서 두 개의 군집에 대한 분석을 해보면 정규화가 되었기에 슛 시도에 대해서 정확한 수치를 알 수 없지만 빈도수는 알 수 있다.
군집 1:
2점 슛 시도: 적음
2점 슛 성공률 : 거의 50%, 편차가 있는 편
3점 슛 시도: 적음 거의 편차가 없이 모여있음
3점 슛 성공률 거의 30% 정도, 40% 이상 던지는 선수들은 절반이 안됨

군집 2:
2점 슛 시도: 많음
2점 슛 성공률 : 거의 50%, 편차가 거의 없음
3점 슛 시도: 평균적으로는 많은데, 편차가 큼
3점 슛 성공률 거의 40% 이상에 편차가 적음

정규화된 군집분석의 군집 2는 정규화 이전 군집분석의 군집 1과 정규화된 군집분석의 군집 1은 정규화 이전 군집분석의 군집 2와 대응되는데 정규화를 통한 군집분석에서는 많이 던지는 군집이 확실히 성공률에서 강점을 보이는 것이 확인되었다.

이를 통해서 확실히 많이 던지는 선수들은 그들이 많이 던지는 이유가 있음을 증명하였다. 그리고 정규화를 통해서 Data Scaling을 할 경우 군집 분석에서 데이터에 대한 보정을 통해 데이터 내부의 유의미한 데이터를 이끌어 낼 수 있다는 것을 확인할 수 있었다.

코드의 경우 github.com/yonhun/MIE_DA에 올려놓았으니 확인하고 싶으시면 들어가서 확인해주시면 감사하겠습니다.