머신러닝과 산업응용 분석보고서 #1

2021190002 장서현

Part 1. Regression – 자전거 대여 수 예측 모델

[EDA]

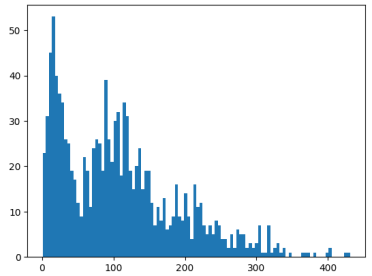
주어진 데이터에 대한 EDA를 진행하였다.

(1) 히스토그램을 통해 count 변수의 분포를 알아보았다.

(2) (3) 주어진 train data의 결측치를 시각화하였다.

(4) (5) 스케일링 전 데이터들의 박스플롯을 확인해본 결과, 스케일링의 필요성을 느끼고, Standard Scaler을 사용하여 스케일링한 뒤, 다시 데이터들의 박스플롯을 그렸다.

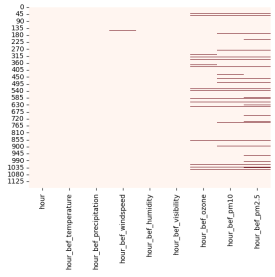
(6) (7) 변수들의 상관계수를 히트맵과 페어플랏으로 표현함으로써, 변수들끼리의 관계성을 시각화하였다.



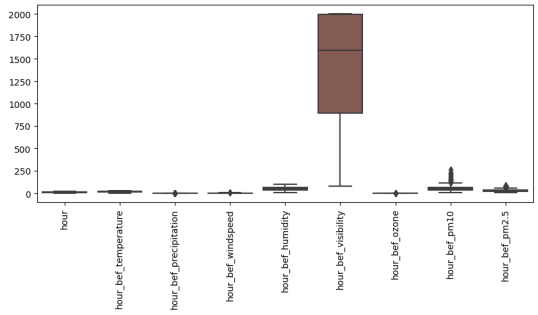
- (1)

```
hour                      0
hour_bef_temperature      2
hour_bef_precipitation    2
hour_bef_windspeed        8
hour_bef_humidity         2
hour_bef_visibility       2
hour_bef_ozone           57
hour_bef_pm10            69
hour_bef_pm2.5           88
dtype: int64
```

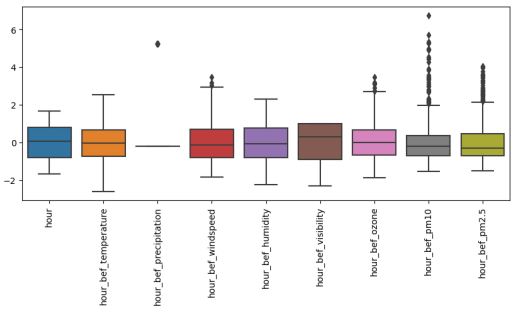
- (2)



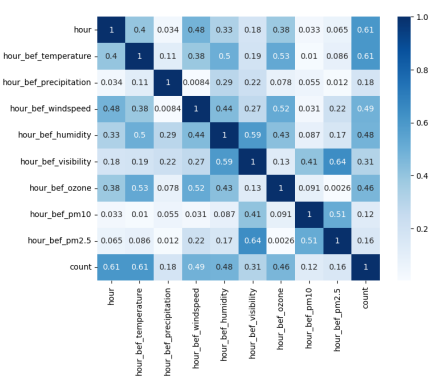
- (3)



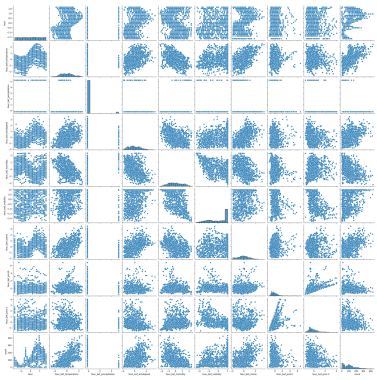
- (4)



- (5)



- (6)



- (7)

[전처리]

1) (이상치 대체 및 제거)

데이터 수 보존을 위해 이상치 제거는 하지 않기로 결정하였다. 또, 예측 모델의 과적합 방지를 위해 이상치 대체도 하지 않기로 결정하였다.

2) (결측치 대체 및 제거)

주어진 train data의 각 샘플이 몇 개의 결측치를 가지고 있는지 확인한 결과, 1068개의 샘플은 결측치가 0개, 31개의 샘플은 결측치가 1개, 16개의 샘플은 결측치가 2개, 49개의 샘플은 결측치가 3개, 1개의 샘플이 결측치가 4개, 2개의 샘플은 결측치가 8개임을 확인할 수 있었다. 결측치가 8개인 2개의 샘플은 소실된 정보가 너무 많아서 샘플로서의 가치가 없다고 판단했기에, 제거하였다. 나머지 결측치들에 대해서는, 데이터의 중앙값을 사용하기로 결정하였다. 최빈값이나 평균을 사용하였을 때 보다 최종 성능이 미세하게 좋았기 때문이다.

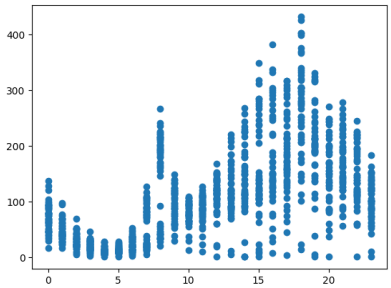
```
0    1068
3      49
1      31
2      16
8       2
4       1
```

```
# 결측치가 8개인 행 삭제
train.drop(train.loc[(train.isna().sum(axis = 1) == 8)].index, inplace = True)
train.reset_index(drop=True, inplace=True)

# 결측치를 데이터의 중앙값으로 대체
train.fillna(train.median(), inplace = True)
X_test.fillna(X_test.median(), inplace = True)
```

3) (스케일링)

A) hour 변수 처리

hour 변수의 분포를 확인해보니, 0 이상 23 이하의 정수값을 가지는 불연속한 discrete한 변수임을 알 수 있었다. 그에 따라, one-hot encoding을 활용하여 해당하는 시간이 1, 다른 모든 시간에는 0의 값을 갖도록 설정하여서, 데이터를 이진 벡터 값으로 변환 받았고, 24개의 새로운 이진벡터 값 데이터들이 각각 24개의 category가 되도록 데이터 타입을 변형해주었다. 따라서, hour 변수를 24개의 카테고리를 가지는 범주형 데이터라고 취급하였다.

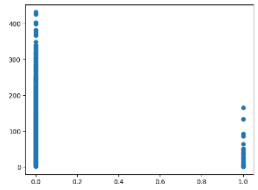



```
# 정수형 hour 데이터를 one-hot encoding을 이용하여 이진 벡터의 값으로 변환
train = pd.concat([pd.get_dummies(train['hour']), train], axis = 1)
train = train.drop(['hour'], axis = 1)
X_test = pd.concat([pd.get_dummies(X_test['hour']), X_test], axis = 1)
X_test = X_test.drop(['hour'], axis = 1)
```

```
# 24개의 새롭게 생긴 이진 hour 데이터를 category형 데이터로 변환
for i in range(24):
  train[str(i)] = train[str(i)].astype('category').values
  X_test[str(i)] = X_test[str(i)].astype('category').values
```

B) hour_bef_precipitation 변수 처리

hour_bef_precipitation은 0과 1을 갖는 binary 변수임을 확인할 수 있었다. 0과 1이 '값'으로서 의미를 가지면 안되기에, 이를 범주형 데이터가 되도록 데이터 타입을 변경하였다.



```
# 이진 hour_bef_precipitation 데이터를 category형 데이터로 변환
train['hour_bef_precipitation'] = train['hour_bef_precipitation'].astype('category').values
X_test['hour_bef_precipitation'] = X_test['hour_bef_precipitation'].astype('category').values
train_n['hour_bef_precipitation'] = train_n['hour_bef_precipitation'].astype('category').values
X_test_n['hour_bef_precipitation'] = X_test_n['hour_bef_precipitation'].astype('category').values
```

C) 나머지 변수들 스케일링

Category형 데이터(hour, hour_bef_precipitation)를 제외한 나머지 변수들의 경우, 연속적인 분포를 지니는 변수들이다. 이들은 MinMaxScaler를 활용하여 스케일링하였고, StandardScaler를 활용할때보다 MinMaxScaler를 활용할 때 성능이 더 좋아서 최종적으로 결정하였다.

```
# category형 데이터를 제외한 상태로 MinMaxScaler 활용하여 Scaling

scaler = MinMaxScaler()

X_numbers = X_train.select_dtypes(exclude = 'category')
scaler.fit(X_numbers)
X_numbers = pd.DataFrame(scaler.transform(X_numbers), columns = X_numbers.columns)
X_train = pd.concat([X_train.select_dtypes(exclude = 'float64'), X_numbers], axis = 1)

X_test_numbers = X_test.select_dtypes(exclude = 'category')
scaler.fit(X_test_numbers)
X_test_numbers = pd.DataFrame(scaler.transform(X_test_numbers), columns = X_test_numbers.columns)
X_test = pd.concat([X_test.select_dtypes(exclude = 'float64'), X_test_numbers], axis = 1)

X_numbers_n = X_train_n.select_dtypes(exclude = 'category')
scaler.fit(X_numbers_n)
X_numbers_n = pd.DataFrame(scaler.transform(X_numbers_n), columns = X_numbers_n.columns)
X_train_n = pd.concat([X_train_n.select_dtypes(exclude = 'float64'), X_numbers_n], axis = 1)

X_test_numbers_n = X_test_n.select_dtypes(exclude = 'category')
scaler.fit(X_test_numbers_n)
X_test_numbers_n = pd.DataFrame(scaler.transform(X_test_numbers_n), columns = X_test_numbers_n.columns)
X_test_n = pd.concat([X_test_n.select_dtypes(exclude = 'float64'), X_test_numbers_n], axis = 1)
```

[변수선택]

A) 상관계수

```
hour_bef_precipitation    0.180252
...
hour_bef_visibility       0.313437
hour_bef_ozone            0.454629
hour_bef_pm10             0.117806
hour_bef_pm2.5            0.144289
Name: count, dtype: float64
```

우선, 상관계수를 출력해 영향력 없는 변수들이 있는지 확인하고자 했다. 상관계수가 0.1보다 작은 변수들은 없어서, 변수들은 모두 유지되었다.

B) T-test

20	79.9195	16.401	4.873	0.000	47.740	112.099
21	96.6731	16.455	5.875	0.000	64.386	128.960
22	86.7254	16.051	5.403	0.000	55.233	118.218
23	47.0658	15.834	2.972	0.003	15.999	78.133
hour_bef_precipitation	-59.5555	7.413	-8.034	0.000	-74.100	-45.011
hour_bef_temperature	136.0458	9.077	14.988	0.000	118.237	153.855
hour_bef_windspeed	19.4980	9.473	2.058	0.040	0.912	38.084
hour_bef_humidity	-29.0337	10.166	-2.856	0.004	-48.981	-9.087
hour_bef_visibility	11.3594	7.611	1.492	0.136	-3.574	26.293
hour_bef_ozone	24.0372	9.700	2.478	0.013	5.005	43.070
hour_bef_pm10	-74.8251	13.311	-5.621	0.000	-100.943	-48.707
hour_bef_pm2.5	6.5127	10.072	0.647	0.518	-13.250	26.276

다음으로, OLS의 t-test를 활용하였다. 변수의 영향력이 0일 것이라는 귀무가설 하에, 유의수준 0.01에서 판단하였을 때, hour_bef_windspeed, hour_bef_visibility, hour_bef_ozone, hour_bef_pm2.5가 제거 후보로 선정되었다. hour_bef_ozone은 p-value가 0.013으로, 다른 제거 대상들에 비해 p-value가 작아서 바로 제거하지 않고, 후속 과정에서 여러 번의 실험을 통해 다양한 조합으로 성능을 평가하였다. 최종적으로, hour_bef_windspeed, hour_bef_visibility, hour_bef_pm2.5, 3개의 변수만 제거하고, hour_bef_ozone은 제거를 안하기로 결정하기로 하였다.

20	108.2360	9.623	11.248	0.000	89.355	127.116
21	124.7130	9.388	13.285	0.000	106.294	143.132
22	113.4529	9.139	12.414	0.000	95.521	131.385
23	73.0657	9.000	8.118	0.000	55.407	90.724
hour_bef_precipitation	-60.6079	7.368	-8.226	0.000	-75.065	-46.151
hour_bef_temperature	131.4541	8.907	14.758	0.000	113.977	148.931
hour_bef_humidity	-41.8126	7.354	-5.685	0.000	-56.242	-27.383
hour_bef_ozone	26.4285	9.358	2.824	0.005	8.068	44.789
hour_bef_pm10	-84.4758	10.682	-7.909	0.000	-105.434	-63.518

세 개의 변수 제거 후, t-test를 다시 진행해보았을 때, 남은 변수들은 모두 유의미함을 알 수 있었다.

C) VIF

변수들 사이 다중공선성이 존재하는지 검사하기 위해 VIF를 활용하였고, 모두 VIF가 10보다 작다는 것으로 미루어보아, 다중공선성의

	column	VIF
0	hour_bef_temperature	7.467092
1	hour_bef_humidity	2.908726
2	hour_bef_ozone	6.336139
3	hour_bef_pm10	3.079956

위험은 적다고 판단하였다.

→ 위와 같은 전반적인 과정과 다양한 변수 조합의 후속 실험 과정을 통해, 최종 모델의 변수들은 hour, hour_bef_precipitation, hour_bef_temperature, hour_bef_humidity, hour_bef_ozone, hour_bef_pm10으로 확정하였다. 이때 MSE는 1727.3945가 나왔다.

[성능분석]

Ridge와 Lasso 두 경우 모두, 하이퍼파라미터 λ의 설정에 따라 다른 MSE 값이 나온다는 것을 확인할 수 있다. Ridge의 경우, λ가 1일 때, MSE가 1814.1079로 나오며, 다른 λ값들에 비해 좋은 성능을 보였다. Lasso의 경우, λ가 0.001일 때, MSE가 1814.4361로 나오며, 다른 λ값들에 비해 좋은 성능을 보였다. Lasso가 Ridge에 비해 더 높은 민감도를 보이기 때문에 최적의 λ값이 더 작다는 것 또한 확인 가능하다.

	0.001	0.01	0.1	1	10
Ridge	1814.490515	1814.467045	1814.253762	1814.107919	1923.073596
Lasso	1814.436066	1814.479850	1821.872971	2365.822807	6871.795657

Decision Tree Regressor의 경우, one-hot encoding 값이 모델의 성능을 저하시킨다. 그 이유는 one-hot encoding으로 많은 0이 포함된 희소행렬들이 생성되는데, 이로 인해 트리 구조가 필요이상으로 복잡해지기 때문이다. 따라서, hour 변수를 초기 데이터 형태로 사용하였다. 결과는 다음과 같이 MSE는 2878.6097로 나온다.

	max_depth	max_leaf_nodes	max_features	min_samples_split	min_samples_leaf	MSE
Decision Tree Regressor	10	None	0.8	0.2	0.1	2878.60974

Part 2. Classification – 와인 품종 분류 모델

[EDA]

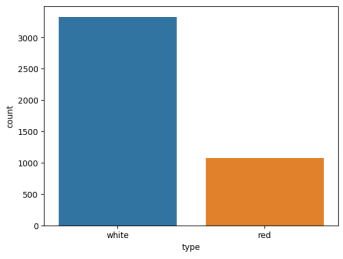
주어진 데이터에 대한 EDA를 진행하였다.

(1) 카운트플랏을 통해 클래스 균형을 시각화하였다.

(2) (3) 주어진 train data의 결측치를 시각화하였다.

(4) (5) 스케일링 전 데이터들의 박스플롯을 확인해본 결과, 스케일링의 필요성을 느끼고, Standard Scaler을 사용하여 스케일링한 뒤, 다시 데이터들의 박스플롯을 그렸다.

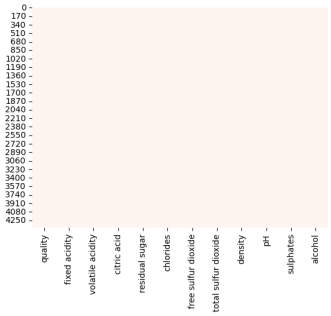
(6) 전체 상관계수들을 히트맵으로 표현함으로써, 변수들끼리의 관계성을 시각화하였다.



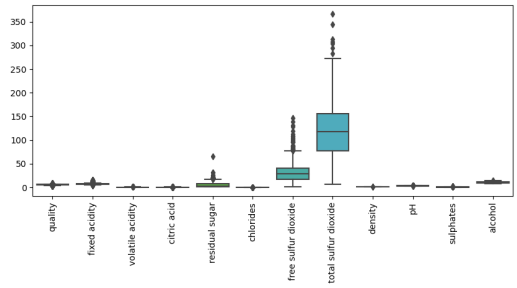
- (1)

```
quality                0
fixed acidity          0
volatile acidity       0
citric acid            0
residual sugar         0
chlorides              0
free sulfur dioxide    0
total sulfur dioxide   0
density                0
pH                     0
sulphates              0
alcohol                0
dtype: int64
```

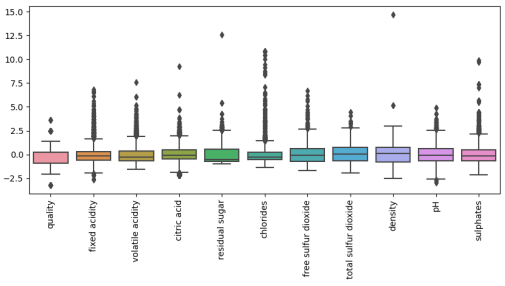
- (2)



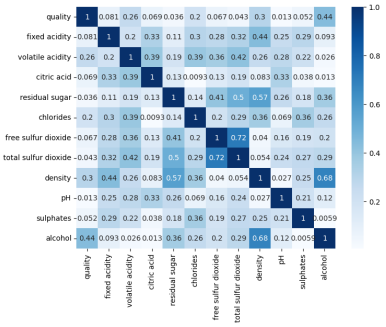
- (3)



- (4)



- (5)



- (6)

[전처리]

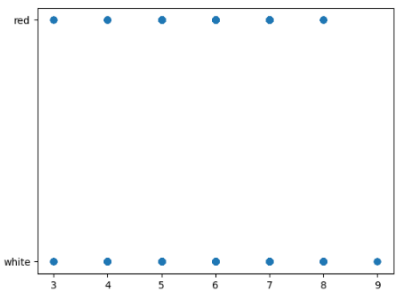
A) 문제 조건 충족

문제에서 주어진 조건대로 red는 1, white는 0으로 설정하였다.

```
Y_train2 = train2['type'].apply(lambda x: 1 if x == 'red' else 0)  # 'red'를 1로, 나머지를 0으로 변환
```

B) quality 변수 scaling

```
X_train2['quality'].value_counts(ascending = True)
✓ 0.0s
quality
9       4
3      20
8     122
4     157
7     749
5    1428
6    1917
Name: count, dtype: int64
```



quality 데이터는 continuous 분포를 가지는 다른 변수들과 다르게 3~9 사이의 discrete한 분포를 가진다. 그렇기에 다른 변수들과는 다른 스케일링 기법을 사용하여 전처리를 진행하였다.

quality 데이터의 경우, 나머지 데이터의 전처리 결과와 같이 (이후 설명) 평균이 0이 되도록 설정하였고, 최댓값과 최솟값의 차이가 1이 되어서, 결과적으로 -0.5 ~ 0.5 사이에 값을 가지도록 처리하였다. 그러기 위해서 MinMaxScaling과 비슷한 메커니즘을 활용하여, $f(x) = \frac{x-6}{6}$의 함수를 통해, 3~9 사이에 분포하던 quality 값들을 -0.5~0.5이 되도록 변경하였다.

```
   # (3+9)/2 = 6

   X_train2['quality'] = X_train2['quality'].apply(lambda x: (x - 6) / 6)
   X_test2['quality'] = X_test2['quality'].apply(lambda x: (x - 6) / 6)
   X_train2['quality'].value_counts(ascending = True)
```

✓ 0.0s

```
quality
 0.500000       4
-0.500000      20
 0.333333     122
-0.333333     157
 0.166667     749
-0.166667    1428
 0.000000    1917
Name: count, dtype: int64
```

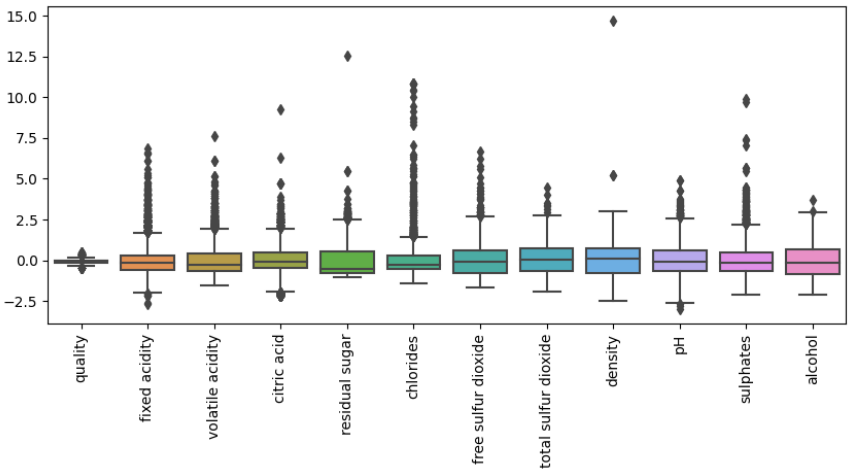
-0.5과 0.5 사이로 quality 데이터가 바뀌었음을 확인할 수 있다.

C) 나머지 변수들 scaling

quality 데이터를 제외하고 제외한 다른 데이터들은 모두 연속적인 분포를 가졌다. 나머지 데이터들은 StandardScaler를 활용하여 Scaling하였다. 그 이유는 이후 실험과정에서 MinMaxScaler를 활용했을 때 보다 StandardScaler가 결과적으로 미세하게 더 우수한 성능을 보여주었기 때문이다.

```
# quality 데이터를 제외하고 StandardScaler 활용하여 Scaling
X_numbers2 = X_train2.select_dtypes(exclude='int64')

# StandardScaler 적용
scaler = StandardScaler()
scaler.fit(X_numbers2)
X_numbers2 = pd.DataFrame(scaler.transform(X_numbers2), columns=X_numbers2.columns)

# 인덱스를 초기화한 후 병합하여 인덱스가 어긋나지 않도록 함
X_train2 = X_train2.reset_index(drop=True)
X_numbers2 = X_numbers2.reset_index(drop=True)

# 스케일링된 데이터와 다른 변수 병합 (인덱스 초기화 후)
X_train2 = pd.concat([X_train2.select_dtypes(exclude='float64').reset_index(drop=True),
                      X_numbers2.reset_index(drop=True)], axis=1)

# X_test에 대해 동일한 전처리 수행
X_test_numbers2 = X_test2.select_dtypes(exclude='int64')
scaler.fit(X_test_numbers2)
X_test_numbers2 = pd.DataFrame(scaler.transform(X_test_numbers2), columns=X_test_numbers2.columns)

# 인덱스를 초기화한 후 병합하여 인덱스가 어긋나지 않도록 함
X_test2 = X_test2.reset_index(drop=True)
X_test_numbers2 = X_test_numbers2.reset_index(drop=True)

# 스케일링된 데이터와 다른 변수 병합 (인덱스 초기화 후)
X_test2 = pd.concat([X_test2.select_dtypes(exclude='float64').reset_index(drop=True),
                     X_test_numbers2.reset_index(drop=True)], axis=1)
```

→ 이는 모든 전처리 과정이 끝난 데이터들의 boxplot이다.



[변수선택]

A) 상관계수

우선, 상관계수를 출력해 과적합 여부를 판단하며, 영향력 없는 변수들이 있는지 확인하고자 했

다. Alcohol 데이터의 상관계수가 0.0196으로 나타나, alcohol 데이터는 제거하기로 판단하였다. 나머지 변수들은 모두 유의미한 상관계수 값을 가지고 있었다.

```
quality                0.111226
fixed acidity          0.472788
volatile acidity       0.644626
citric acid            0.196671
residual sugar         0.348647
chlorides              0.525568
free sulfur dioxide    0.474750
total sulfur dioxide   0.702426
density                0.373476
pH                     0.345740
sulphates              0.485436
alcohol                0.019554
Name: type, dtype: float64
```

-> 상관계수 출력 결과

	column	VIF
0	quality	1.340438
1	fixed acidity	3.126897
2	volatile acidity	1.817672
3	citric acid	1.578258
4	residual sugar	4.451789
5	chlorides	1.635668
6	free sulfur dioxide	2.148102
7	total sulfur dioxide	2.753615
8	density	5.550736
9	pH	1.844076
10	sulphates	1.406313

-> VIF 출력 결과

B) VIF

VIF값을 통해 다중공선성이 존재하는지 파악하였다. VIF값들이 모두 10 이하임을 확인하였고, 다중공선성의 위험이 낮다는 것을 알 수 있었다. 따라서, 더 이상 제거할 변수가 없기에, 최종 모델로 확정하였다.

→ 위와 같은 전반적인 과정과 다양한 변수 조합의 후속 실험 과정을 통해, 최종 모델의 변수들은 alcohol을 제외한 나머지로 선택되었다.

[성능분석]

Logistic Regression의 경우, 하이퍼파라미터 C의 값이 10일 때 비교적으로 가장 좋은 성능을 보이며, F1 score는 0.982174을 기록한다.

	0.01	0.1	1	10
Logistic Regression	0.974001	0.978278	0.980197	0.982174

KNN의 경우, 하이퍼파라미터 n_neighbors의 값이 1일 때 비교적으로 가장 좋은 성능을 보이며, F1 score 약 0.987323를 기록한다.

	1	3	5	7
KNN	0.987323	0.986901	0.984801	0.984580

Decision Tree Classifier의 경우, 최적의 하이퍼파라미터 값들은 다음과 같고, 이때의 F1 score는 0.8457을 기록한다.

	max_depth	max_leaf_nodes	max_features	min_samples_split	min_samples_leaf	F1
Decision Tree Classifier	None	50	0.6	0.6	0.1	0.8457