KOREATECH 한국기술교육대학교

13 주차 과제

학번	2020136087
이름	윤아현
수강 과목	머신러닝및실습
담당 교수	이해윤 교수님
제출일	2024.06.02

머신러닝 및 실습 과제 4

1 번. 데이터 설명

데이터셋은, 레드 와인과 화이트 와인의 화학적 성분을 기반으로 한 와인의 품질이다.

독립 변수: 와인의 화학적 성분

종속 변수: 와인의 품질 점수 (0~10)

데이터는 11 개의 독립 변수와 1 개의 종속 변수로 구성되어 있다.

이 때, 종속 변수인 와인의 품질 점수는 3 ~ 9 점 사이에 있다.

전체 데이터 개수는 6497 개이며, 품질 점수의 개수는 아래의 표와 같다.

quality	values
3	30
4	216
5	2,138
6	2,836
7	1,079
8	193
9	5

2 번. 데이터 전처리 과정

1) 기본 통계량 확인하기

종속 변수(와인의 품질)에 대한 기본 통계량을 확인하였을 시에, 평균이 5.8 이고 표준편차가 0.8 이라는 것을 확인할 수 있다.

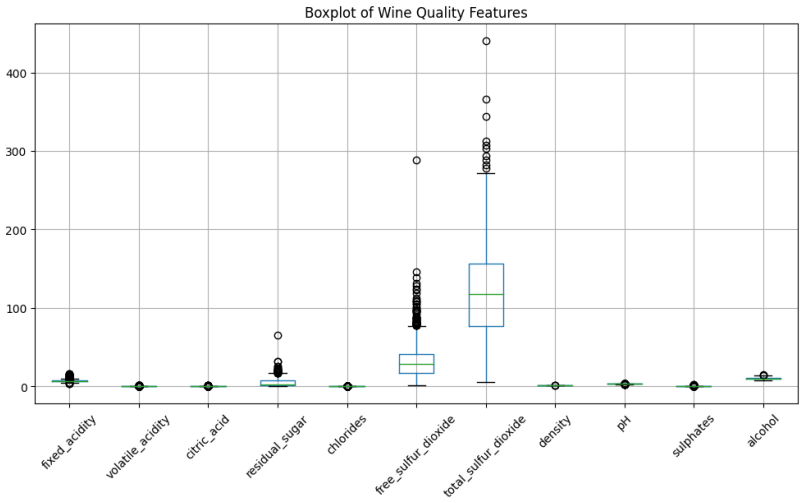
이는 데이터 값들이 평균 주변에 근접하게 분포하고 있음을 보여준다.

2) 결측치 확인하기

독립 변수 및 종속 변수 모두 결측치가 존재하지 않았다.

3) 이상치 확인하기

이상치를 쉽게 확인하고, 비교하기 위해 독립 변수들에 대한 Box-Plot 을 그려 확인하였다.



대부분의 독립 변수에서 이상치로 판단될 값들이 보이지 않았지만,

독립변수인 residual_sugar, free_sulfur_dioxide, 및 total_sulfur_dioxide 에서는 튀는 값이 조금씩 존재한다는 것을 확인하였다.

위 3 가지 변수에 대해 이상치 제거를 수행하였다.

1. Residual_sugar

   값이 50 이상인 데이터를 삭제하였다.

2. free_sulfur_dioxide

   값이 200 이상인 데이터를 삭제하였다.

3. total_sulfur_dioxide

   값이 300 이상인 데이터를 삭제하였다.

전체 6,497 개의 데이터에서 총 7 개의 데이터가 삭제되었다.

4) 상관관계 확인하기

독립 변수와 종속 변수 사이에 강한 상관관계를 나타내는 변수는 없었으며, 가장 큰 상관관계를 나타낸 독립 변수는 alcohol 이었다. (0.446)

5) 데이터 불균형 해소

위의 데이터 설명 표에 나타나 있듯이, 각 데이터 간의 불균형이 심한 것을 확인 할 수 있다.

처음에는 오버샘플링 기법을 모든 class 에 대해 수행하려고 했지만, class 9 같은 경우에는 데이터가 10 개 미만이기에, 사용이 불가능하였고 과적합이 발생할 우려가 있었다.

먼저, 클래스를 3 개로 재조정을 수행하였다.

quality	values
0 (3, 4)	243
1 (5, 6, 7)	6,049
2 (8, 9)	198

범주를 조정하여도 각 클래스 간의 불균형은 해소되지 않았다.

이를 해결하기 위해, 데이터 정규화를 수행한 뒤 훈련 데이터에 대해서만 오버샘플링 및 언더샘플링을 진행한다.

6) 데이터 정규화

훈련 데이터와 테스트 데이터로 분할한 뒤, 정규화를 수행하였다. (test_size=0.2)

모든 독립 변수의 단위를 동일하게 맞추기 위해 "StandardScaler(표준정규화)"를 적용하였다.

추가적으로, 클래스 간의 불균형을 해결하기 위해 class 1 은 UnderSampling 을, class 0 과 2 는 OverSampling 을 수행한다.

이 경우의 문제점은,

class 0 과 2 에 대한 class 를 1000 개 이상으로 늘리게 되면, 테스트 데이터에서 전체적인 정확도는 높을지라도 class 0 과 2 로 분류하는 성능이 현저히 낮음을 알 수 있다.

또한, 데이터의 왜곡 문제로 합성 데이터에 초점이 맞춰진 모델이 생성될 수 있다.

이는 합성된 데이터가 데이터의 패턴을 제대로 인식하지 못해 생긴 문제로, **일반화 능력이 아주 낮다고** 해석할 수 있다.

그러므로 전체적인 정확도가 높지 않더라도, class 1 에 대한 데이터 수를 줄여 일반화 능력을 키우도록 한다.

class 1 에 대한 샘플 수는 700 개로 언더샘플링하였고, class 0 과 2 는 500 개로 오버샘플링하였다.

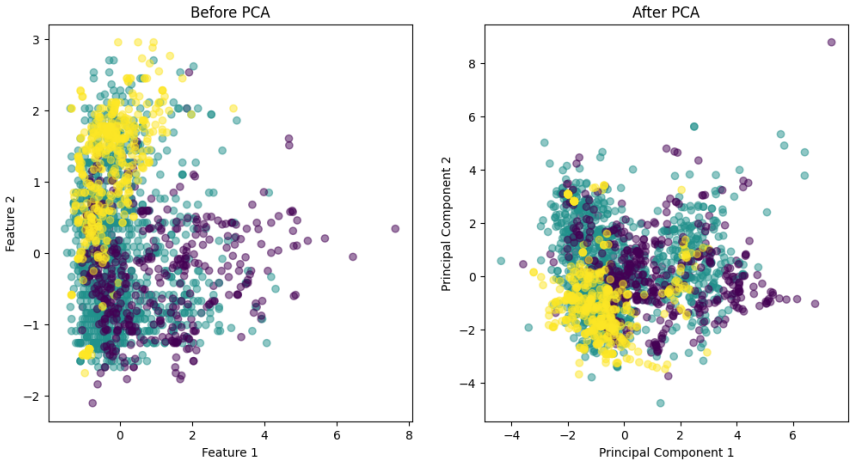
7) Label Encoder

종속 변수에 대해 Encoding 을 수행하여 범주형 데이터를 수치형 데이터로 변환한다.

3 번. 차원 축소 과정 및 결과 분석

차원 축소의 경우, 4 차원 이상은 시각화를 수행할 수 없기 때문에 2 차원과 3 차원으로 대폭 축소를 해본 뒤 시각화를 수행한다.

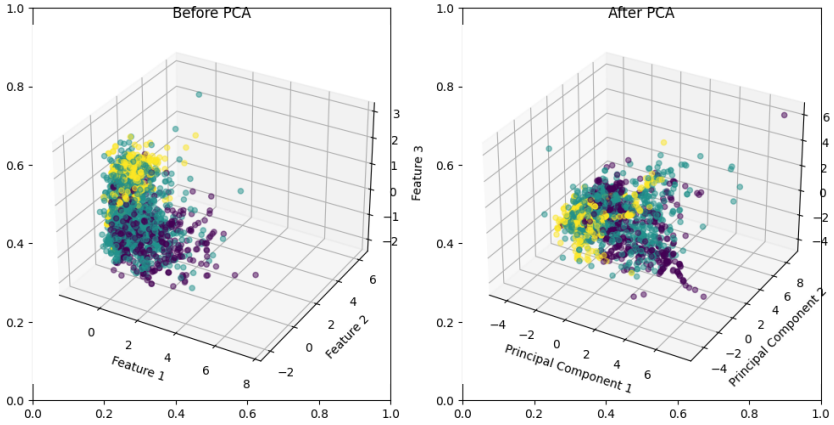
1) 2 차원 PCA 수행



PCA 전 그래프는 종속 변수와 가장 상관관계가 높은 alcohol 과 volatile_acidity 를 기준으로 시각화하였다.

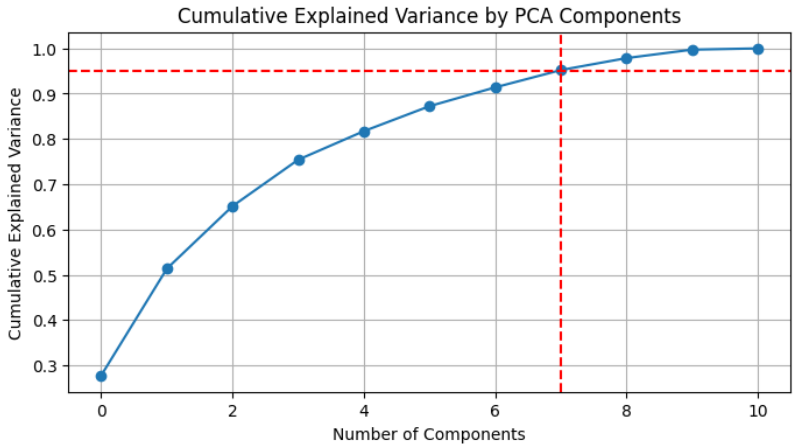
PCA 전과 후의 차이를 보면, PCA 후에 데이터 분포가 비교적 명확하게 보이는 것을 확인할 수 있다.

2) 3 차원 PCA 수행



2 차원으로 PCA 를 수행했을 때와 동일하게 PCA 후에 데이터 분포가 비교적 명확하게 보이는 것을 확인할 수 있다.

3) 축소할 차원 지정하기



PCA 결과에 따르면, 7 차원으로 줄였을 때 데이터의 95%까지 설명이 가능하다는 것을 확인할 수 있다.

이를 통해, 전체 독립 변수를 사용했을 때와 7 개의 주성분을 사용했을 때 모델 성능을 비교해본다.

4 번. 분류 모델 구축 및 성능 평가

PCA 를 수행하기 전과, 수행한 후의 성능을 비교해본다.

대표적인 성능 지표로 "accuracy_score"를 사용한다.

모델	PCA 수행 전	PCA 수행 후
1. 로지스틱 회귀	61.24%	-
2. 의사결정 나무	66.95%	68.57%
3. K-NN	59.63%	-
4. SVM	63.56%	-

*가장 성능이 좋은 모델에만 PCA 후 데이터셋에 대한 모델 생성을 추가적으로 수행하였음

의사 결정 나무 결과 (PCA 후):

```
accuracy score 0.6856702619414484


confusion matrix:
  [[ 29  18   2]
 [197 839 174]
 [  1  16  22]]


classification report:
              precision    recall  f1-score   support

           0       0.13      0.59      0.21        49
           1       0.96      0.69      0.81      1210
           2       0.11      0.56      0.19        39

    accuracy                           0.69      1298
   macro avg       0.40      0.62      0.40      1298
weighted avg       0.90      0.69      0.76      1298
```

결과를 보면, class 1 에 대한 예측 성능이 좋지만 class 0 과 class 2 에 대한 정밀도(precision) 및 f1-score 가 현저히 낮은 것을 볼 수 있다.

이는 class 0 과 2 에 대한 합성 데이터 개수가 많아 데이터가 왜곡되었을 가능성이 있다고 추측된다.

PCA 후의 모델 성능이 더 높은 것을 보아, 주요 특성을 잘 찾아내고 불필요한 변동성을 제거된 것으로 추측된다.

5 번. 앙상블 기법 적용 결과

모델	PCA 수행 전	PCA 수행 후
1. Bagging(DT)	77.5%	-
2. AdaBoosting	59.1%	-
3. RandomForest	66.1%	-
4. XGBoost	79.43%	76.89%
5. LGBM	74.5%	

*가장 성능이 좋은 모델에만 PCA 후 데이터셋에 대한 모델 생성을 추가적으로 수행하였음

XGBoost 결과 (PCA 전):

```
accuracy score 0.7942989214175655


confusion matrix:
  [[ 28  20   1]
 [120 980 110]
 [  0  16  23]]


classification report:
              precision    recall  f1-score   support

           0       0.19      0.57      0.28        49
           1       0.96      0.81      0.88      1210
           2       0.17      0.59      0.27        39

    accuracy                           0.79      1298
   macro avg       0.44      0.66      0.48      1298
weighted avg       0.91      0.79      0.84      1298
```

결과를 보면, 의사결정나무 모델과 동일하게 class 1 에 대한 예측 성능이 좋지만 class 0 과 class 2 에 대한 정밀도(precision) 및 f1-score 가 현저히 낮은 것을 볼 수 있다.

또한, 단일 모델인 의사결정나무와 비교해보았을 때에도 class 1 에 대한 예측 성능은 더욱이 높아졌지만 다른 클래스에 대한 예측 성능은 동일하다.

이는 합성 데이터로 인한 데이터 왜곡이 발생했을 가능성이 높다는 것을 말해준다.

XGBoost 모델 같은 경우, PCA 전이 PCA 후보다 성능이 더 좋다.

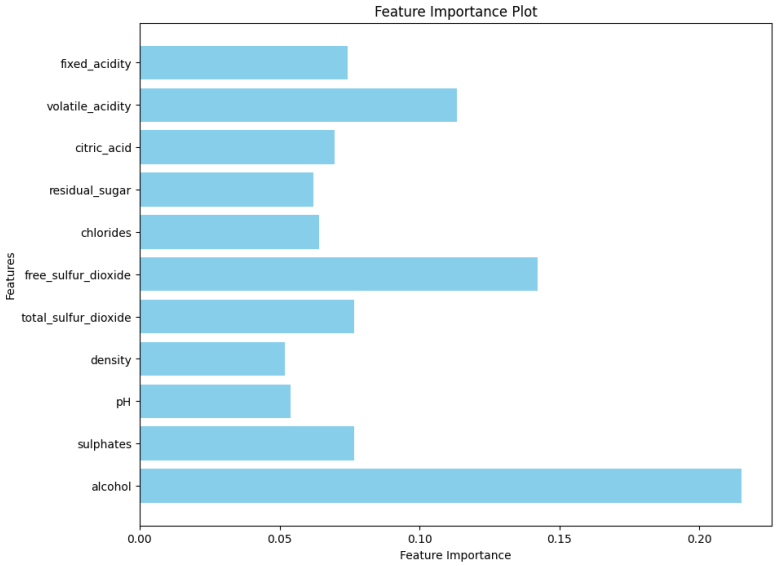
이유를 추측해보았을 때, 앙상블 모델인 XGBoost 는 복잡한 알고리즘을 가지고 있어 주요 특징들만을 추출했을 때보다 전체 독립 변수를 모두 가지고 있을 때 정보 손실이 덜 일어나 성능이 비교적 좋은 것이라고 판단된다.

6 번. 결과 분석

가장 좋은 성능을 가진 모델은 정확도가 79.43%인 PCA 를 수행하지 않은 XGBoost Model 이다.

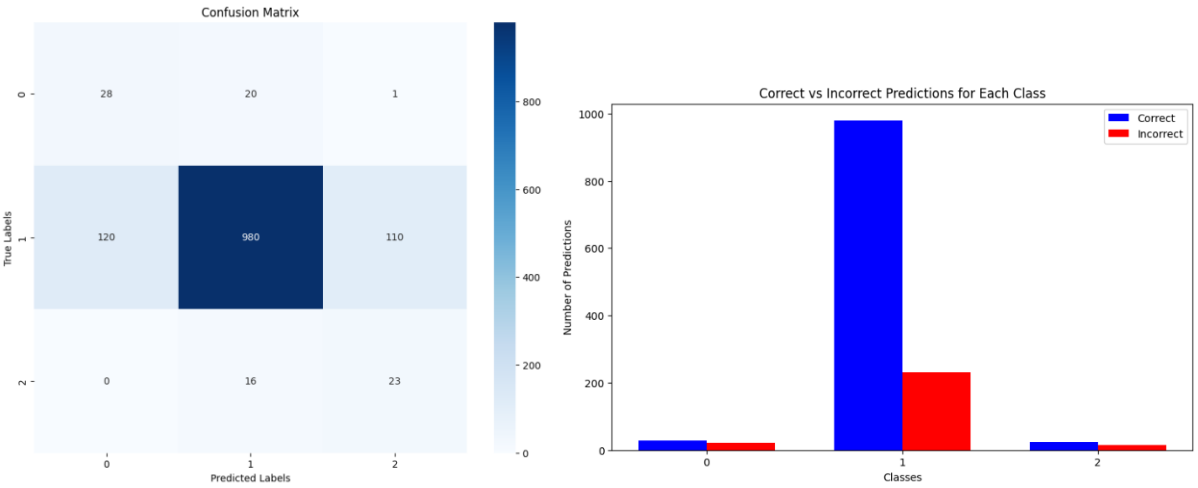
1) Feature Importance 추출

Feature Importance 를 추출하였을 때, 결과는 아래의 표와 같다.



종속 변수와 상관관계가 가장 높았던 alcohol 이 가장 변수 중요도가 높다는 것을 확인할 수 있다.

2) 예측 결과 시각화



위의 그림을 보았을 때에도 class 1 에 대한 예측 정확도는 높지만, 다른 class 는 정확도가 현저히 낮고 제대로 예측을 수행하지 못한다는 것을 볼 수 있다.

Quality 점수가 5, 6, 7 에 몰려 있지 않고 분산되어 있거나 더 다양한 데이터가 수집되었다면, 더 좋은 성능이 나왔을 것이라고 생각된다.

첨부. 수행 코드

데이터 불러오기

1 번. "ucimlrepo library"를 설치하여 features 와 targets 를 불러옴

```
from ucimlrepo import fetch_ucirepo

wine_quality = fetch_ucirepo(id=186)

X = wine_quality.data.features
y = wine_quality.data.targets
```

데이터 전처리 과정

1 번. 기본 통계량 확인하기

```
X.describe(include = 'all')
```

```
y.describe()
```

2 번. 결측치 확인하기

```
# 결측치 확인하기
X.isnull().sum()
```

```
y.isnull().sum()
```

3 번. 이상치 제거하기

```
# 이상치 확인하기
# box-plot 그리기

plt.figure(figsize=(12, 6))
X.boxplot()
plt.xticks(rotation=45)
plt.title('Boxplot of Wine Quality Features')
plt.show()
```

```
## residual_sugar

mask = X['residual_sugar'] < 50
X = X[mask].reset_index(drop = True)
y = y[mask].reset_index(drop = True)
X
```

위와 같은 방식으로 3 개의 변수에 대한 이상치를 제거해주었음

4 번. 상관관계 확인하기

```
corr_df = pd.concat([X, y], axis=1)
corr = corr_df.corr()
print(corr["quality"].sort_values(ascending=False))
```

5 번. Target 데이터 범위 재조정하기

```
def change_target(data):
    if data <= 4:
        return 0  # Low Quality
    elif data <= 7:
        return 1  # Medium Quality
    else:
        return 2  # High Quality
```

```
y = y['quality'].apply(change_target)
```

6 번. 데이터 정규화하기

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.2,
                                                    stratify=y,
                                                    random_state = 42)
```

```
sc = StandardScaler()
X_train_sc = sc.fit_transform(X_train)
X_test_sc = sc.transform(X_test)
X_train_sc[:5]
```

7 번. 데이터 불균형 해소하기

```
pipeline = Pipeline([
    ('under', RandomUnderSampler(sampling_strategy={1: 700})),
    ('over', SMOTE(sampling_strategy={0: 500, 2: 500}))
])

X_train_res, y_train_res = pipeline.fit_resample(X_train_sc, y_train)
```

8 번. Label Encoding 수행하기

```
encoder = LabelEncoder()
```

```
# y_train_enc = encoder.fit_transform(y_train_smote)
y_train_enc = encoder.fit_transform(y_train_res)
y_test_enc = encoder.transform(y_test)
```

차원 축소 과정 및 결과 분석

1 번. 2 차원 PCA 수행하기

```
# 2차원 pca 결과
pca_2d = PCA(n_components=2)
X_train_pca_2d = pca_2d.fit_transform(X_train_res)
X_test_pca_2d = pca_2d.transform(X_test_sc)
```

```
fig, ax = plt.subplots(1, 2, figsize=(12, 6))

ax[0].scatter(X_train_res[:, 1], X_train_res[:, 10], c=y_train_enc,
              cmap='viridis', alpha=0.5)
ax[0].set_title('Before PCA')
ax[0].set_xlabel('Feature 1')
ax[0].set_ylabel('Feature 2')

ax[1].scatter(X_train_pca_2d[:, 0], X_train_pca_2d[:, 1], c=y_train_enc,
              cmap='viridis', alpha=0.5)
ax[1].set_title('After PCA')
ax[1].set_xlabel('Principal Component 1')
ax[1].set_ylabel('Principal Component 2')

plt.show()
```

2 번. 3 차원 PCA 수행하기

```
# 3차원 pca 결과
pca_3d = PCA(n_components=3)
X_train_pca_3d = pca_3d.fit_transform(X_train_res)
X_test_pca_3d = pca_3d.transform(X_test_sc)
```

```
fig, ax = plt.subplots(1, 2, figsize=(12, 6))

ax1 = fig.add_subplot(121, projection='3d')
ax1.scatter(X_train_res[:, 1], X_train_res[:, 2], X_train_res[:, 10],
            c=y_train_enc, cmap='viridis', alpha=0.5)
ax1.set_title('Before PCA')
ax1.set_xlabel('Feature 1')
ax1.set_ylabel('Feature 2')
ax1.set_zlabel('Feature 3')

ax2 = fig.add_subplot(122, projection='3d')
ax2.scatter(X_train_pca_3d[:, 0], X_train_pca_3d[:, 1], X_train_pca_3d[:, 2],
            c=y_train_enc, cmap='viridis', alpha=0.5)
ax2.set_title('After PCA')
ax2.set_xlabel('Principal Component 1')
ax2.set_ylabel('Principal Component 2')
ax2.set_zlabel('Principal Component 3')

plt.show()
```

3 번. 축소할 차원 선정

```
pca = PCA().fit(X_train_res)
cumulative_variance = np.cumsum(pca.explained_variance_ratio_)
```

```
plt.figure(figsize=(8, 4))
plt.plot(cumulative_variance, marker='o')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Cumulative Explained Variance by PCA Components')
plt.grid(True)
plt.axhline(y=0.95, color='r', linestyle='--')  # 95% 설명된 분산을 나타내는 선
plt.axvline(x=np.argmax(cumulative_variance >= 0.95), color='r', linestyle='--')
plt.show()
```

4 번. 차원 축소 (d = 7)

```
## 결과
pca = PCA(n_components=7)
X_train_pca = pca.fit_transform(X_train_res)
X_test_pca = pca.transform(X_test_sc)
```

분류 모델 구축 및 성능 평가

1 번. Decision Tree Model (PCA X)

```
### Decision Tree
dt = DecisionTreeClassifier(random_state=42)

dt.fit(X_train_res, y_train_enc)
y_pred = dt.predict(X_test_sc)

acc = accuracy_score(y_test, y_pred)
mat = confusion_matrix(y_test, y_pred)
cpt = classification_report(y_test, y_pred)

print("accuracy score", acc)
print("\n\nconfusion matrix:\n ", mat)
print("\n\nclassification report:\n", cpt)
```

2 번. Decision Tree Model (PCA O)

```
### PCA 수행 후
dt2 = DecisionTreeClassifier(random_state=42)

dt2.fit(X_train_pca, y_train_enc)
y_pred = dt2.predict(X_test_pca)

acc = accuracy_score(y_test, y_pred)
mat = confusion_matrix(y_test, y_pred)
cpt = classification_report(y_test, y_pred)

print("accuracy score", acc)
print("\n\nconfusion matrix:\n ", mat)
print("\n\nclassification report:\n", cpt)
```

다른 모델도 이와 비슷하게 수행하였음

앙상블 기법 적용 결과

1 번. XGBoost Model (PCA X)

```
## XGBoost
xgb = XGBClassifier(random_forest=42)

xgb.fit(X_train_res, y_train_enc)
y_pred = xgb.predict(X_test_sc)

acc = accuracy_score(y_test, y_pred)
mat = confusion_matrix(y_test, y_pred)
cpt = classification_report(y_test, y_pred)

print("accuracy score", acc)
print("\n\nconfusion matrix:\n ", mat)
print("\n\nclassification report:\n", cpt)
```

2 번. XGBoost Model (PCA O)

```
## XGBoost
xgb = XGBClassifier(random_forest=42)

xgb.fit(X_train_pca, y_train_enc)
y_pred = xgb.predict(X_test_pca)

acc = accuracy_score(y_test, y_pred)
mat = confusion_matrix(y_test, y_pred)
cpt = classification_report(y_test, y_pred)

print("accuracy score", acc)
print("\n\nconfusion matrix:\n ", mat)
print("\n\nclassification report:\n", cpt)
```

다른 모델도 이와 비슷하게 수행하였음

결과 분석

1 번. 변수 중요도

```
feature_importances = xgb.feature_importances_
feature_names = X_train.columns.tolist()
print(feature_importances)
print(feature_names)
```

```
# 특성 중요도 시각화
plt.figure(figsize=(10, 8))
plt.barh(feature_names, feature_importances, align='center', color='skyblue')
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Feature Importance Plot')
plt.gca().invert_yaxis()
plt.show()
```

2 번. 예측 결과 시각화

```
conf_matrix = confusion_matrix(y_test_enc, y_pred)

plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels="012", yticklabels='012')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

```
correct = np.diag(conf_matrix)
incorrect = np.sum(conf_matrix, axis=1) - correct

labels = [0, 1, 2]

fig, ax = plt.subplots(figsize=(10, 6))
bar_width = 0.35
index = np.arange(len(labels))

bar1 = ax.bar(index, correct, bar_width, label='Correct', color='b')
bar2 = ax.bar(index + bar_width, incorrect, bar_width,
              label='Incorrect', color='r')

ax.set_xlabel('Classes')
ax.set_ylabel('Number of Predictions')
ax.set_title('Correct vs Incorrect Predictions for Each Class')
ax.set_xticks(index + bar_width / 2)
ax.set_xticklabels(labels)
ax.legend()

plt.show()
```