빅데이터 분석 시각화

Homework#9

학번 : 2016039029
이름 : 이종영
마감일자 : 23/10/31

Contents

1. Data Analysis Processing

1.1 데이터 분석에 사용될 라이브러리 및 데이터를 호출

```python
# 1. Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn import metrics
```

✓ 0.0s

```python
# 2. Import the dataset (same directory - py)
df = pd.read_csv('cancer.csv')
```

✓ 0.0s

1.2 데이터 정보 확인 헤더정보, 크기, 컬럼의 타입

```python
# 3. exploratory data analysis: Data Information
print(df.head())
print(df.shape)
print(df.info())
```

✓ 0.0s

```
2                1                          2           2
3                1                          3           4
4                3                          2           1

   Bland_Chromatin  Normal_Nucleoli1  Mitoses  Class
0                3                 1        1      2
1                3                 2        1      2
2                3                 1        1      2
3                3                 7        1      2
4                3                 1        1      2
(699, 11)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Id                           699 non-null    int64
 1   Clump_thickness              699 non-null    int64
 2   Uniformity Cell Size         699 non-null    int64
 3   Uniformity Cell Shape        699 non-null    int64
 4   Marginal Adhesion            699 non-null    int64
 5   Single_Epithelial_Cell_Size  699 non-null    int64
 6   Bare_Nuclei                  699 non-null    object
 7   Bland_Chromatin              699 non-null    int64
 8   Normal_Nucleoli              699 non-null    int64
 9   Mitoses                      699 non-null    int64
 10  Class                        699 non-null    int64
dtypes: int64(10), object(1)
memory usage: 60.2+ KB
None
```

1.3 오브젝트 타입 제거, EDA 수행

```
# 2-1. object delete
df = df.drop(columns=['Bare_Nuclei'])
# 3. Exploratory data analysis: Class Inbalance
print(df['Class'].value_counts())
print()
print(df['Mitoses'].value_counts())
print()
print(df['Bland_Chromatin'].value_counts())
print()
```

```
Class
2    458
4    241
Name: count, dtype: int64

Mitoses
1     579
2      35
3      33
10     14
4      12
7       9
8       8
5       6
6       3
Name: count, dtype: int64

Bland_Chromatin
2     166
3     165
1     152
7      73
4      40
5      34
8      28
```

표본의 형태를 고려한 Class 속성 선정

```
print(df['Class'].value_counts()/np.float64(len(df)))
```

```
Class
2    0.655222
4    0.344778
Name: count, dtype: float64
```

1.4 누락된 필드 체크

```
# 3. Exploratory data analysis: Check Missing Data
print(df.isnull().sum())
```

✓ 0.0s

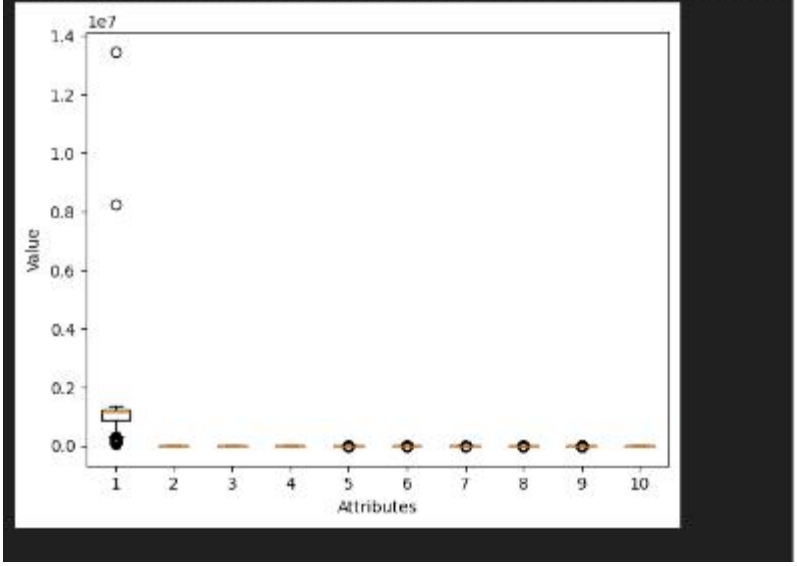
```
Id                             0
Clump_thickness                0
Uniformity_Cell_Size           0
Uniformity_Cell_Shape          0
Marginal_Adhesion              0
Single_Epithelial_Cell_Size    0
Bland_Chromatin                0
Normal_Nucleoli                0
Mitoses                        0
Class                          0
dtype: int64
```

따로 없으니 전처리는 하지 않음.

1.5 이상값 확인

```
# 3. Exploratory data analysis: Check Outliers
plt.boxplot(df)
plt.xlabel('Attributes')
plt.ylabel('Value')
plt.show()
```

✓ 0.0s

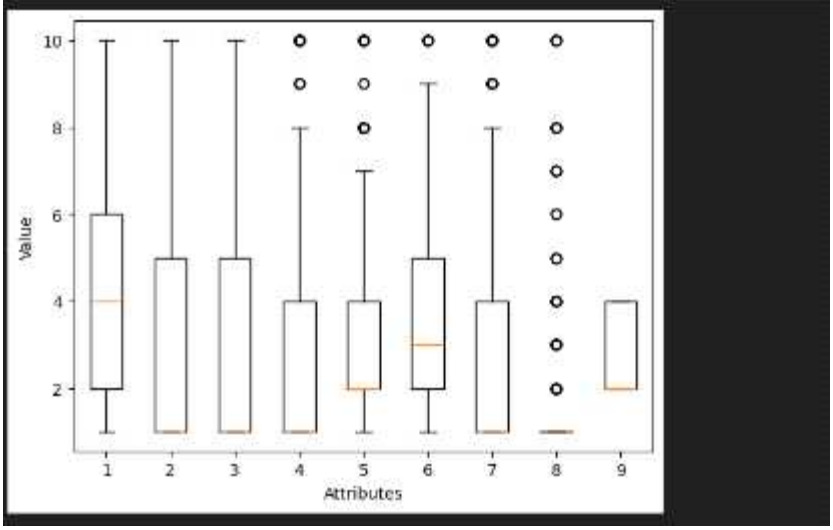


이때, 첫번째 컬럼으로 인한 가시성이 떨어짐을 확인 ID는 사실상 기본키의 역할로 해당값을 배제함.

1.5.1 제거 및 다시 확인

```
# 3-1. Exploratory data analysis: Outliers Detail
data_to_boxplot = df.drop(columns=['Id'])
plt.boxplot(data_to_boxplot)
plt.xlabel('Attributes')
plt.ylabel('Value')
plt.show()
df = df.drop(columns=['Id'])
```

✓ 0.0s



4~7까지는 전처리 필요, 8번째 컬럼은 고른 분포가 확인되며 정확도에 영향을 줄 것으로 추측

1.6 트레이닝셋을 7의 비율로 세트 및 확인

```
# 4. Split data into separate training and test set 7:3
training_points = df.drop(columns=['Class'])
training_labels = df['Class']

X_train, X_test, y_train, y_test = train_test_split(
    training_points,
    training_labels,
    test_size=0.3,
    random_state=4)

print(X_train)
print(y_train)
print(X_test)
print(y_test)

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

✓ 0.0s

```
     Clump_Thickness  Uniformity_Cell_Size  Uniformity_Cell_Shape  \
35                 7                     4                      6
605                1                     1                      1
340                3                     4                      4
420                2                     1                      1
202                1                     4                      3
..               ...                   ...                    ...
599                3                     2                      4
560                6                    10                     10
439                5                     1                      1
174                8                     4                      5
522               10                    10                     10

     Marginal_Adhesion  Single_Epithelial_Cell_Size  Bland_Chromatin  \
35                   4                            6                4
605                  1                            2                1
560                 10                            6                6
429                  1                            2                2
202                 10                            4                5
..                 ...                          ...              ...
520                  1                            3                1
560                 10                           10                6
426                  1                            3                1
174                  4                            3                6
522                  3                           10                5

...
(480, 8)
(480,)
(210, 8)
(210,)
```

1.7 K-최근접 이웃법(KNN) 실시 주변 데이터셋 변경하며 정확도 확인

```
# 5. Fit K Neighbours Classifier to the training set 5
classifier = KNeighborsClassifier(n_neighbors = 5)
classifier.fit(X_train, y_train)
guesses = classifier.predict(X_test)

print(guesses)
```

✓ 0.0s

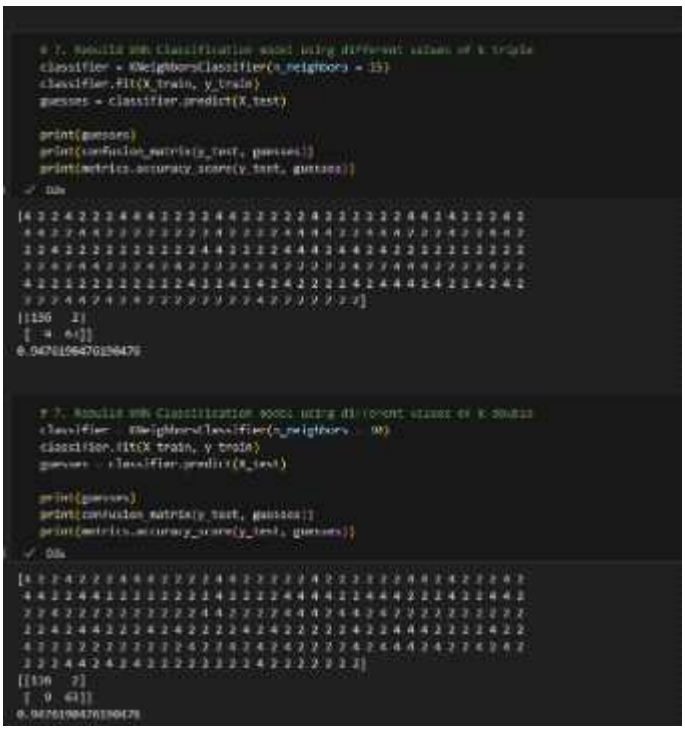
```
[4 2 2 4 2 2 2 4 4 4 2 2 2 2 4 4 2 2 2 2 2 4 2 2 2 2 2 2 4 4 2 4 2 2 2 4 2
 4 4 2 2 4 4 2 2 2 2 2 2 2 2 4 2 2 2 2 4 4 4 4 2 2 4 4 4 2 2 2 4 2 2 4 4 2
 2 2 4 2 2 2 2 2 2 2 2 2 2 2 4 4 2 2 2 2 4 4 4 2 2 4 2 4 2 2 2 2 2 2 2 2 2
 2 2 4 2 4 4 2 2 2 4 2 4 2 2 2 2 4 2 4 2 2 2 2 4 2 2 2 4 4 4 2 2 2 2 2 4 2 2
 4 2 2 2 2 2 2 2 2 2 2 2 4 2 2 4 2 4 2 4 2 2 2 2 4 2 4 4 4 2 4 2 2 4 2 4 2 4 2
 2 2 2 4 4 2 4 2 4 2 2 2 2 2 2 2 2 4 2 2 2 2 2 2 2]
```

```
# 6. Check Accuracy Score
print(confusion_matrix(y_test, guesses))
print(metrics.accuracy_score(y_test, guesses))
```

✓ 0.0s

```
[[136   2]
 [  9  63]]
0.9476190476190476
```

K=5, 0.947...



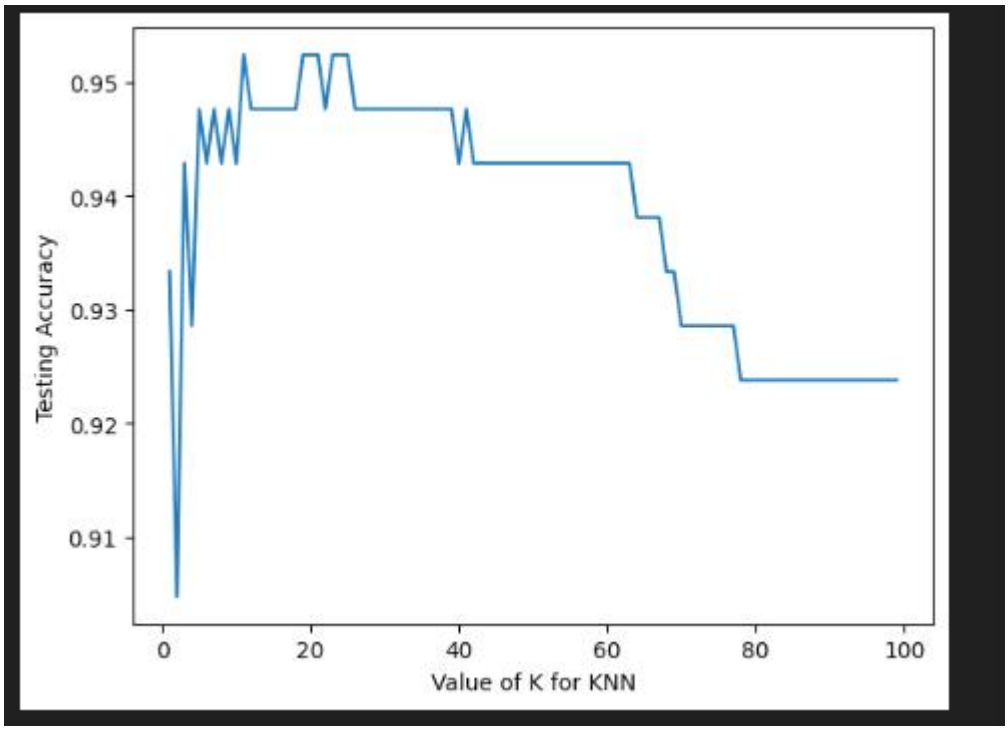
K=15, 0.947 K=30, 0.947

유의미한 차이가 없음 즉 K값을 범위로 두어 시각분석화 할 필요가 있음.

1.7.1 KNN 범위에 대한 시각 분석화

```
# 7. Improving Accuracy: Tuning k parameter 1 to 100
k_range = range(1, 100)

accuracy_scores = []

for k in k_range:
    classifier = KNeighborsClassifier(n_neighbors = k)
    classifier.fit(X_train, y_train)
    guesses = classifier.predict(X_test)
    accuracy_scores.append(metrics.accuracy_score(y_test, guesses))
print(accuracy_scores)

plt.plot(k_range, accuracy_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')
plt.show()
```

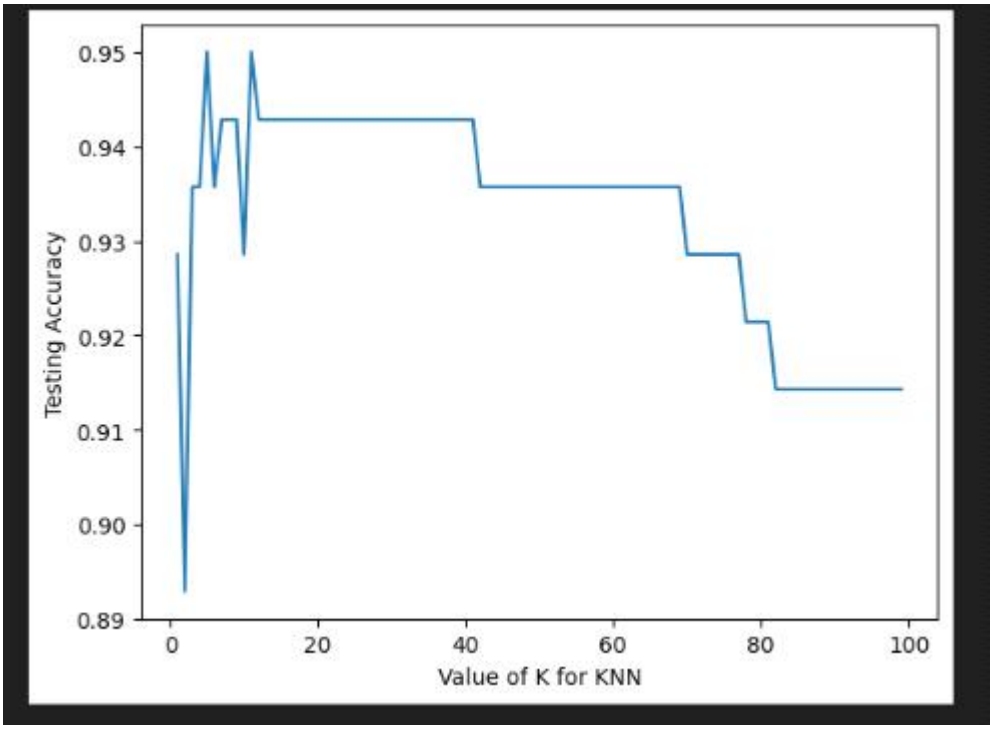
K = 30까지도 무의미한 차이가 있었으니 값은 100까지 크게 잡아서 식별.



전체적으로 정확도가 높으나 70 이후 값이 쳐짐이 확인됨.

1.8 트레이닝 셋, 테스트 셋 비율 변경 (8:2)

```
# 8. Improving Accuracy: Changing split ratio (8:2)
# Tuning k parameter
k_range = range(1, 100)

accuracy_scores = []

for k in k_range:
    classifier = KNeighborsClassifier(n_neighbors = k)
    classifier.fit(X_train, y_train)
    guesses = classifier.predict(X_test)
    accuracy_scores.append(metrics.accuracy_score(y_test, guesses))
print(accuracy_scores)

plt.plot(k_range, accuracy_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')
plt.show()
```



정확도가 줄어든것으로 식별됨 따라서 이후 셋은 7:3 비율로 진행됨.

1.9 정확도 높이기를 위한 속성 관계 시각화

```
# 9. Improving Accuracy: Feature Engineering
dfcorr = df.corr()
plt.figure(figsize=(12,8))
sns.heatmap(data = dfcorr, annot=True)
```

이때, Class에 대해 Mitoses(8번째 컬럼, 0.42) Single_Epithelial_Cell_Size(5번째 컬럼, 0.68) 약한 속성을 식별하여 제거 함.
처음 데이터 분석 과정에서 확인한 4~7 속성 및 8번째 속성임을 알 수 있음.

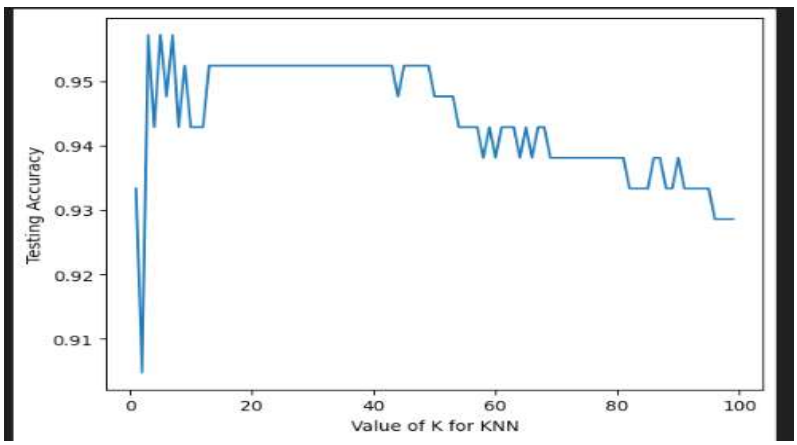
1.10 해당컬럼 제거

```
# 10. Improving Accuracy: Feature Engineering ( low 2 columns 0.68, 0.42 )
df = df.drop(columns=['Single_Epithelial_Cell_Size','Mitoses'])
```

1.11 다시한번 트레이닝 및 시각화

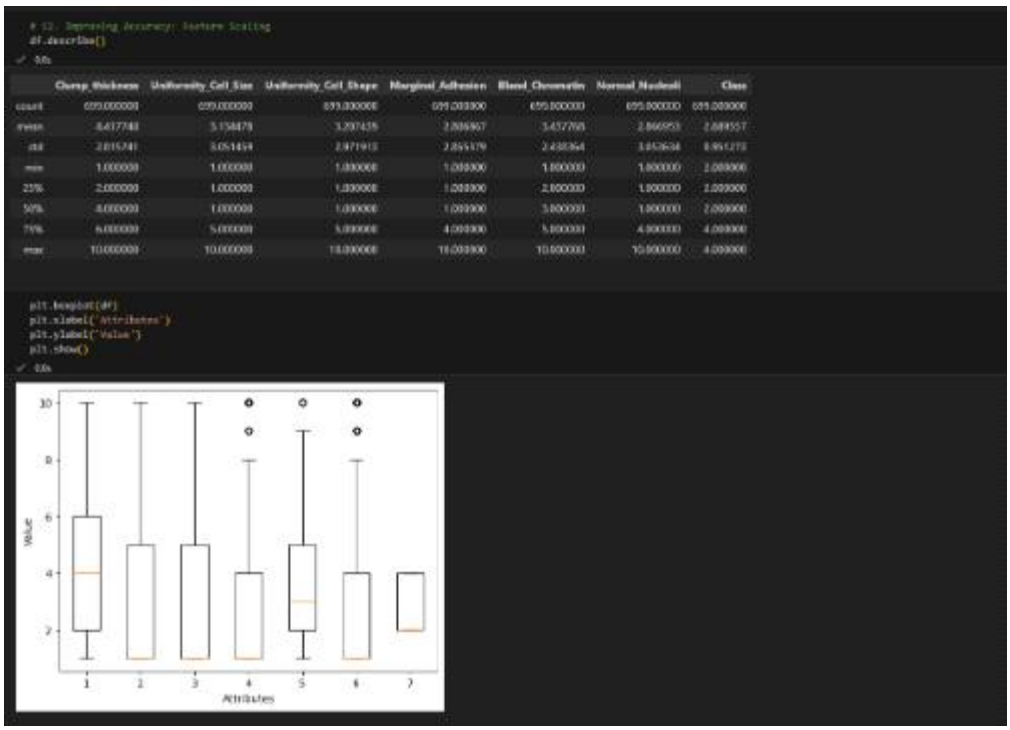
```
# 11. Split data into separate training and test set
training_points = df.drop(columns=['Class'])
training_labels = df['Class']

X_train, X_test, y_train, y_test = train_test_split(
    training_points,
    training_labels,
    test_size=0.3,
    random_state=4)

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
# 11. Tuning k parameter
k_range = range(1, 100)

accuracy_scores = []

for k in k_range:
    classifier = KNeighborsClassifier(n_neighbors = k)
    classifier.fit(X_train, y_train)
    guesses = classifier.predict(X_test)
    accuracy_scores.append(metrics.accuracy_score(y_test, guesses))
print(accuracy_scores)

plt.plot(k_range, accuracy_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')
plt.show()
```



정확도가 0.91 -> 0.93 으로 상승함.

1.12 스케일링 및 속성 분포 확인



Min,max가 차이가 크지 않고 고르게 분포됨이 보임 이때 4,5,6 번째 속성에 대해 일부 오류값이 식별됨. 이에 대해 RobustScaler 정규화를 진행하기로 함.

1.13 정규화 및 트레이닝

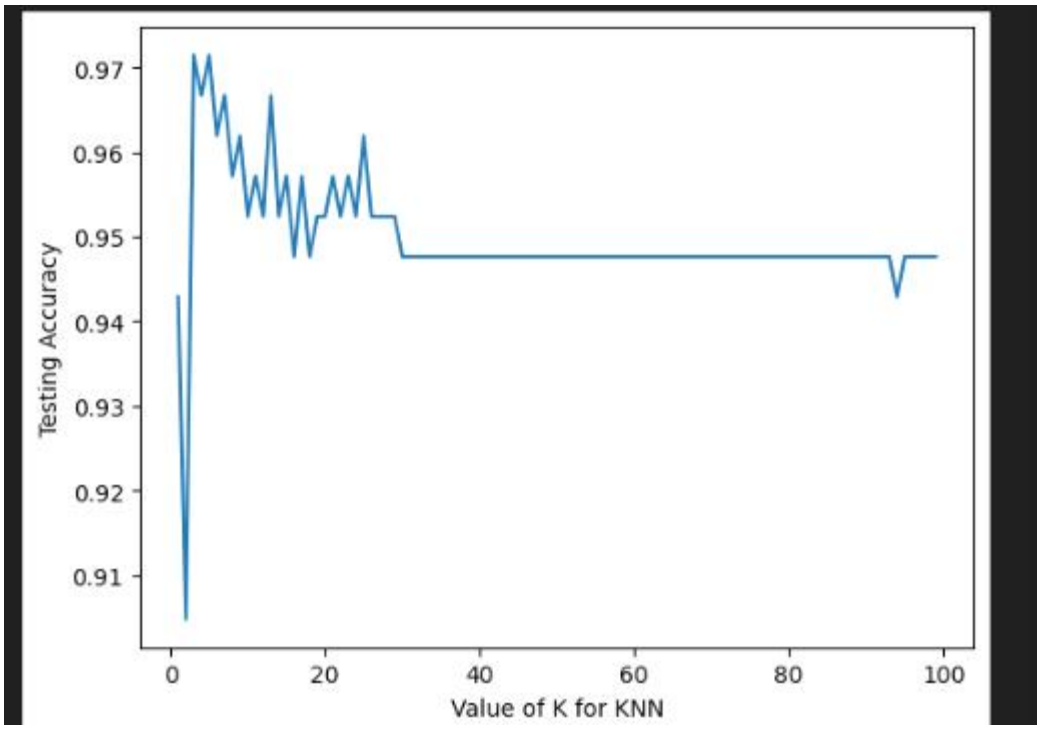
```
# 12. Improving Accuracy: Feature Scaling
from sklearn.preprocessing import RobustScaler

#Create copy of dataset.
df = df.copy()

scaler = RobustScaler()


features = [['Marginal_Adhesion', 'Normal_Nucleoli']]
for feature in features:
    df[feature] = scaler.fit_transform(df[feature])
```

0.0s

```
# 13. Split data into separate training and test set
training_points = df.drop(columns=['Class'])
training_labels = df['Class']

X_train, X_test, y_train, y_test = train_test_split(
    training_points,
    training_labels,
    test_size=0.3,
    random_state=4)

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

1.14 마지막 정확도 확인

```
# 14. Tuning k parameter
k_range = range(1, 100)

accuracy_scores = []

for k in k_range:
    classifier = KNeighborsClassifier(n_neighbors = k)
    classifier.fit(X_train, y_train)
    guesses = classifier.predict(X_test)
    accuracy_scores.append(metrics.accuracy_score(y_test, guesses))
print(accuracy_scores)

plt.plot(k_range, accuracy_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')
plt.show()
```



기존 0.93 -> 0.94 이상으로 정확도를 확보하였고 k값이 적을때의 정확도 또한 증가한것을 식별함.

2. Result

데이터 분석 후 정확도를 상승 시키는 전처리 과정을 진행하였습니다.
오브젝트 데이터를 지우고 기본키를 제거하여 전처리를 진행한 뒤 기준이 될 속성값을 선택하였습니다.
선택된 속성값에 대해 트레이닝셋과 테스트 셋의 비율은 7:3으로 선정하였습니다.
정확도를 높이기 위한 속성간 관계를 시각화 하고 제거한 뒤 정규화를 위한 값의 분포를 확인하고 RobustScaler를 진행하여 최종적으로 정확도를 높이는데 성공하였습니다.