빅데이터 분석 시각화

Homework#10

학번 : 2016039029
이름 : 이종영
마감일자 : 23/11/20

Contents

1. Data Analysis Processing

1.1 데이터 분석에 사용될 라이브러리 및 데이터를 호출

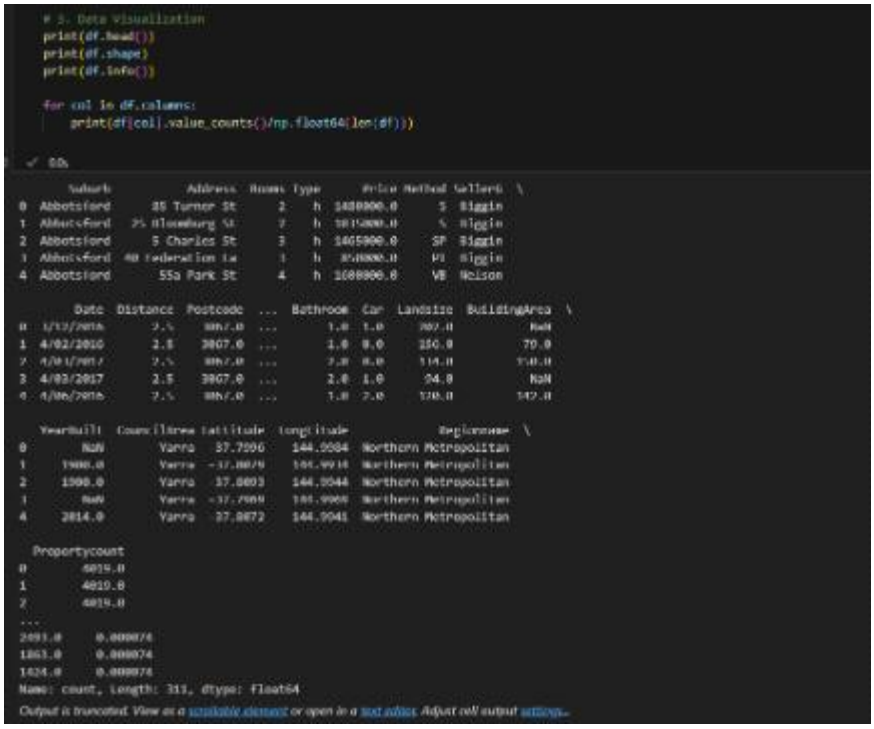
```
# 1. Import necessary libraries

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
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

```
# 2. Import the dataset (same directory)
df = pd.read_csv('melb_data.csv')
```

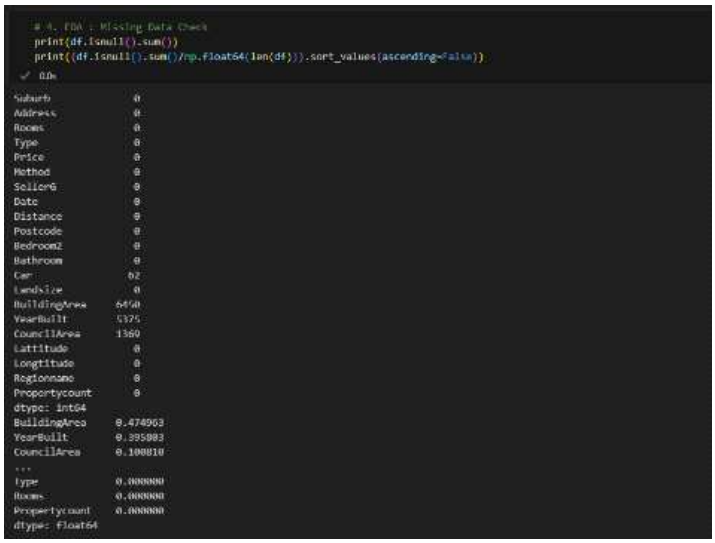
1.2 데이터 정보 확인 헤더정보, 크기, 데이터 형태



1.3 EDA 수행 종속변수 관계 확인 속성 선정

```
# 4. EDA : target attribute - Type
# h - house,cottage,villa, semi,terrace; u - unit, duplex; t - townhouse;
print(df['Type'].value_counts())
print()
print(df['Type'].value_counts()/np.float64(len(df)))
```

1.4 누락된 필드 체크



```
# 4. EDA : Missing Data Check
print(df.isnull().sum())
print((df.isnull().sum()/np.float64(len(df))).sort_values(ascending=False))
```

```
Suburb           0
Address          0
Rooms            0
Type             0
Price            0
Method           0
SellerG          0
Date             0
Distance         0
Postcode         0
Bedroom2         0
Bathroom         0
Car             62
Landsize         0
BuildingArea    6450
YearBuilt       5375
CouncilArea     1369
Lattitude        0
Longtitude       0
Regionname       0
Propertycount    0
dtype: int64
BuildingArea    0.474963
YearBuilt       0.395883
CouncilArea     0.100810
...
Type            0.000000
Rooms           0.000000
Propertycount   0.000000
dtype: float64
```

'BuildingArea','YearBuilt','CouncilArea''Car'

4가지 속성에 대해 전처리 필요

이때, 10% 초과인 3개 속성에 대해서는 CDA 진행 그 외

속성에 대해서는 결측값에 대해 대체값을 삽입한다.

각 튜플의 값이 다수건이 확인 되므로 단순대체를 이용한다.

```
# 4. EDA : CDA 10% OVER
df = df.drop(columns=['BuildingArea','YearBuilt','CouncilArea'])

# 4. EDA : SimpleImputer 10% UNDER
imputer = SimpleImputer(strategy='mean')
Car = df['Car'].values.reshape(-1, 1)
imputed_Car = imputer.fit_transform(Car)
df['Car'] = imputed_Car

print(df.isnull().sum())
```

```
Suburb          0
Address         0
Rooms           0
Type            0
Price           0
Method          0
SellerG         0
Date            0
Distance        0
Postcode        0
Bedroom2        0
Bathroom        0
Car             0
Landsize        0
Lattitude       0
Longtitude      0
Regionname      0
Propertycount   0
dtype: int64
```

1.5 이상값 확인

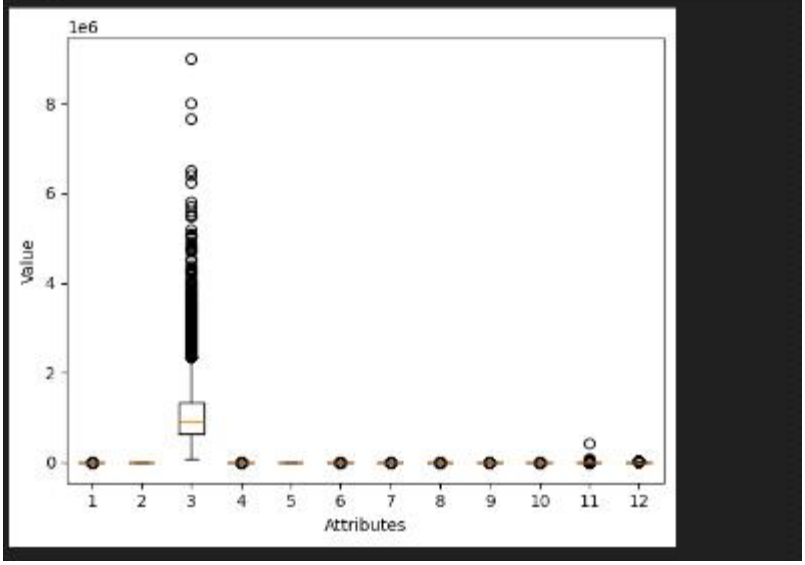
이전에 확인한 데이터 중 범주형 데이터에 대한 작업을 진행하는데 방해되는 속성을 제거하며 그 외에는 라벨인코딩으로 동일한 데이터형으로 바꾸어 진행한다

```
# 4. EDA : Not Categorical data
df = df.drop(columns=['Suburb','Address','Date','Lattitude','Longtitude','Regionname'])


# 4. EDA : Categorical data Labelencoding
encoder = LabelEncoder()
df['Method'] = encoder.fit_transform(np.array(df['Method'].values.reshape(-1,1)))
df['SellerG'] = encoder.fit_transform(np.array(df['SellerG'].values.reshape(-1,1)))
df['Type'] = encoder.fit_transform(np.array(df['Type'].values.reshape(-1,1)))
```

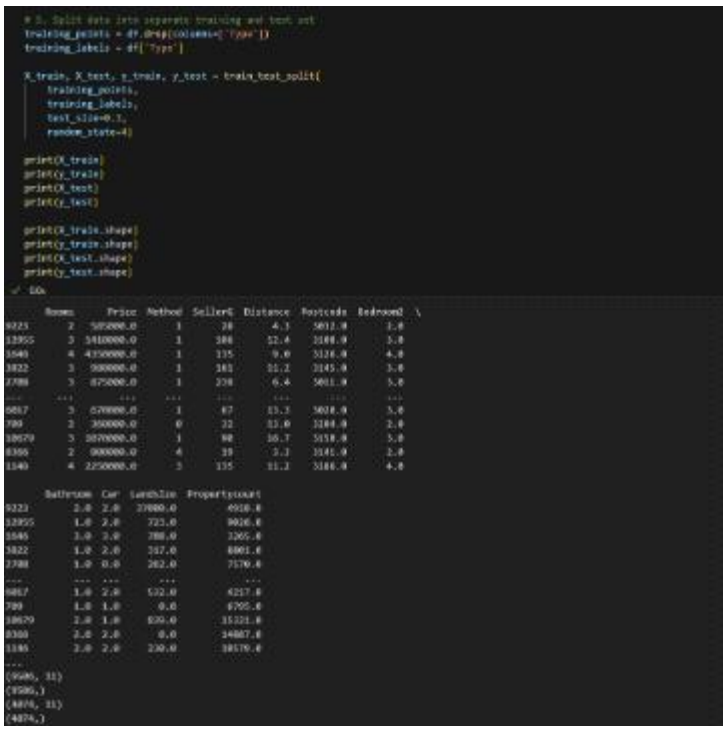
이후 이상값을 다시 확인한다.

```
# 4. EDA: Outliers
plt.boxplot(df)
plt.xlabel('Attributes')
plt.ylabel('Value')
plt.show()
✓ 0.0s
```



다음의 작업에서 3번 컬럼에 대해 조치 되는지 확인하기로 한 다음 다음 절차를 진행한다.

1.6 트레이닝셋을 7의 비율로 세트 및 확인



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

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)
```

✓ 0s

```
      Rooms      Price  Method  SellerG  Distance  Postcode  Bedroom2  \
8223      2   585000.0       1       28       4.3    3012.0       2.0
12955     3  1410000.0       1      388      12.4    3188.0       3.0
1646      4  4350000.0       1      135       9.0    3126.0       4.0
3822      3   980000.0       1      103      11.2    3145.0       3.0
2788      3   875000.0       1      230       6.4    3011.0       3.0
...     ...        ...     ...      ...       ...       ...       ...
6057      3   670000.0       1       67      15.3    3028.0       3.0
788       2   360000.0       0       22      13.0    3188.0       2.0
10579     3  1070000.0       1       98      16.7    3158.0       3.0
8366      2   900000.0       4       39       3.3    3141.0       2.0
1140      4  2250000.0       5      135      11.2    3186.0       4.0

      Bathroom  Car  Landsize  Propertycount
8223       2.0  2.0   37000.0         4910.0
12955      1.0  2.0     723.0         9026.0
1646       3.0  3.0     788.0         3265.0
3822       1.0  2.0     167.0         8801.0
2788       1.0  0.0     202.0         7570.0
...        ...  ...       ...            ...
6057       1.0  2.0     532.0         4217.0
788        1.0  1.0       0.0         6795.0
10579      2.0  1.0     639.0        15321.0
8366       2.0  2.0       0.0        14807.0
1140       2.0  2.0     230.0        10579.0
...
(9486, 11)
(9486,)
(4070, 11)
(4070,)
```

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

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

print(guesses)
```

94] ✓ 0.0s

```
[0 0 2 ... 2 0 0]
```

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

95] ✓ 0.0s

```
[[2605   32  169]
 [ 271   15   58]
 [ 452   25  447]]
0.7528227785959745
```

K=5, 0.752...

```
# 7. Rebuild KNN Classification model using different values of k
classifier = KNeighborsClassifier(n_neighbors = 15)
classifier.fit(X_train, y_train)
guesses = classifier.predict(X_test)

print(guesses)
print(confusion_matrix(y_test, guesses))
print(metrics.accuracy_score(y_test, guesses))
```

✓ 0.0s

```
[0 0 2 ... 0 0 0]
[[2627    2  177]
 [ 283    0   61]
 [ 471    1  402]]
0.7055806470299459
```

```
# 7. Rebuild KNN Classification model using different values of k
classifier = KNeighborsClassifier(n_neighbors = 30)
classifier.fit(X_train, y_train)
guesses = classifier.predict(X_test)

print(guesses)
print(confusion_matrix(y_test, guesses))
print(metrics.accuracy_score(y_test, guesses))
```

✓ 0.0s

```
[0 0 2 ... 0 0 0]
[[2591    0  215]
 [ 285    0   59]
 [ 408    0  516]]
0.7626411389297087
```

K=15, 0.765… K=30, 0.762…

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

1.7.1 KNN 범위에 대한 시각 분석화

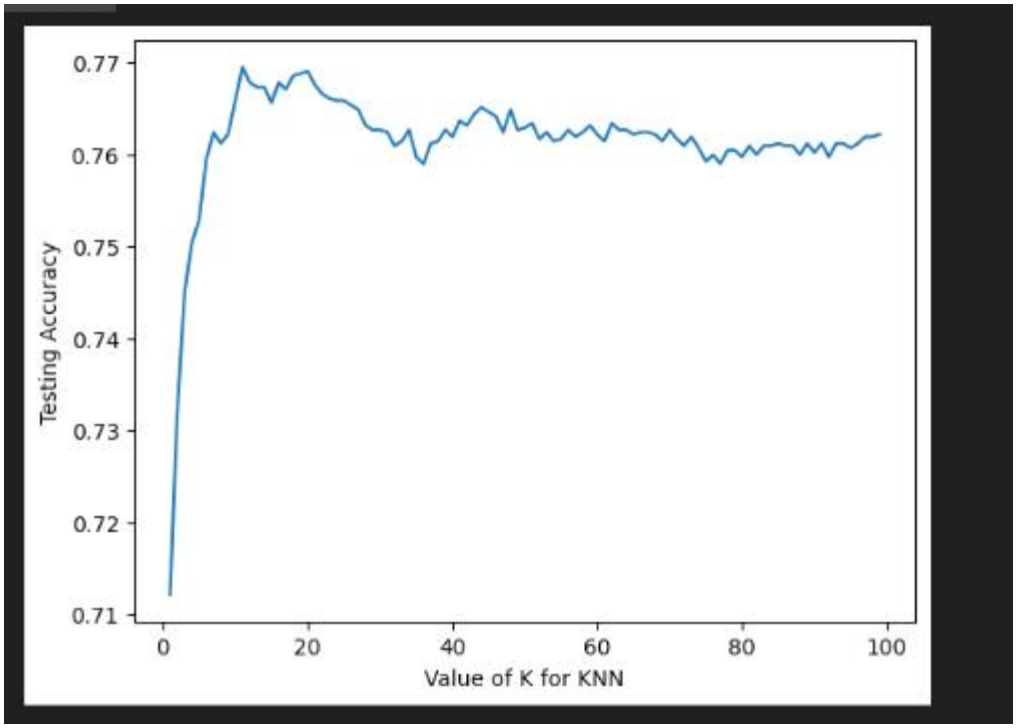
```
# 7. Improving Accuracy: Tuning k parameter
k_range = range(1, 100)

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.



정확도는 위와 같이 확인됨.

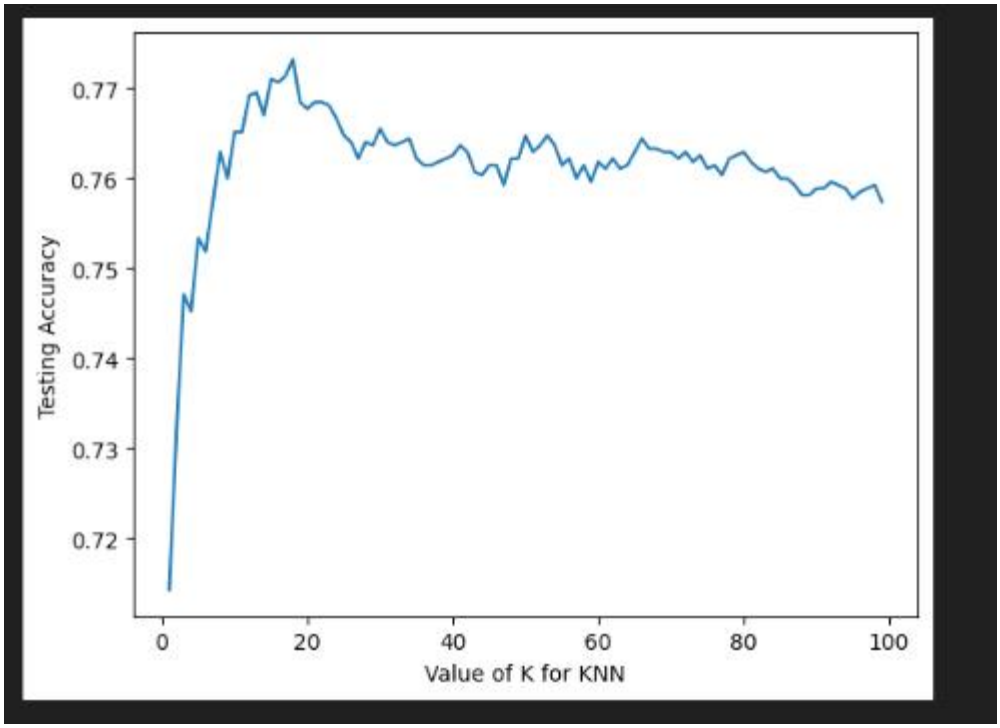
1.8 트레이닝 셋, 테스트 셋 비율 변경 (8:2)

```
# 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 늘어난 것으로 식별됨 따라서 이후 셋은 8:2 비율로 진행됨.

1.9 정확도 높이기를 위한 속성 관계 시각화

이때, 상관계수에 대해 관계 없는 (0에 수렴) 하는 속성값을 제거 함.

```
# 10. Improving Accuracy: Feature Engineering
df = df.drop(columns=['Method','SellerG','Postcode','Landsize','Propertycount'])
```

✓ 0.0s

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

X_train, X_test, y_train, y_test = train_test_split(
    training_points,
    training_labels,
    test_size=0.2,
    random_state=4)

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

다시 모델을 생섬함.

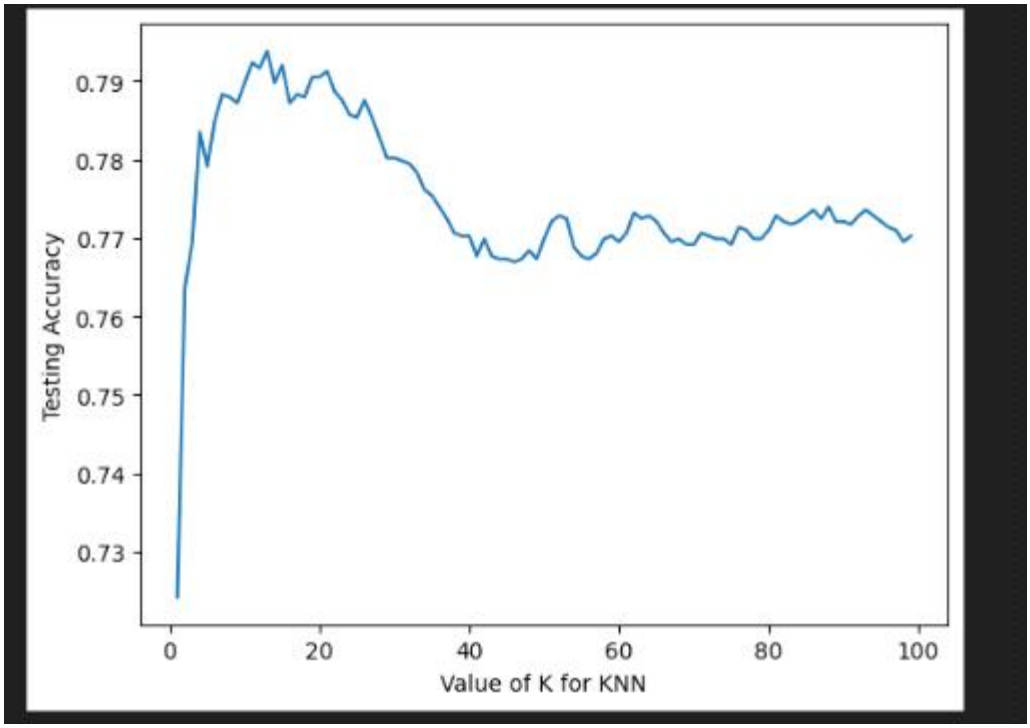
1.10 생성된 모델을 이용 다시한번 정확도를 시각화 함.

```
# 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 0.77 -> 0.79 으로 상승함.

1.12 스케일링 및 속성 분포 확인

```
# 12. Improving Accuracy: Feature Scaling
df.describe()
```

✓ 0.0s

	Rooms	Type	Price	Distance	Bedroom2	Bathroom	Car
count	13580.000000	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000
mean	2.937997	0.526362	1.075684e+06	10.137776	2.914728	1.534242	1.610075
std	0.955748	0.832878	6.393107e+05	5.868725	0.965921	0.691712	0.960433
min	1.000000	0.000000	8.500000e+04	0.000000	0.000000	0.000000	0.000000
25%	2.000000	0.000000	6.500000e+05	6.100000	2.000000	1.000000	1.000000
50%	3.000000	0.000000	9.030000e+05	9.200000	3.000000	1.000000	2.000000
75%	3.000000	1.000000	1.330000e+06	13.000000	3.000000	2.000000	2.000000
max	10.000000	2.000000	9.000000e+06	48.100000	20.000000	8.000000	10.000000

값의 범위 차이가 큰 것을 확인 MINMAX 스케일링을 사용하여 다시한번 모델을 생성함.

1.13 정규화 및 트레이닝

```
# 12. Improving Accuracy: Feature Scaling

df = df.copy()

scaler = MinMaxScaler()

features = [['Price', 'Distance']]
for feature in features:
    df[feature] = scaler.fit_transform(df[feature])

plt.boxplot(df)
plt.xlabel('Attributes')
plt.ylabel('Value')
plt.show()
```

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

X_train, X_test, y_train, y_test = train_test_split(
    training_points,
    training_labels,
    test_size=0.2,
    random_state=4)

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

✓ 0.0s

```
(10864, 6)
(10864,)
(2716, 6)
(2716,)
```

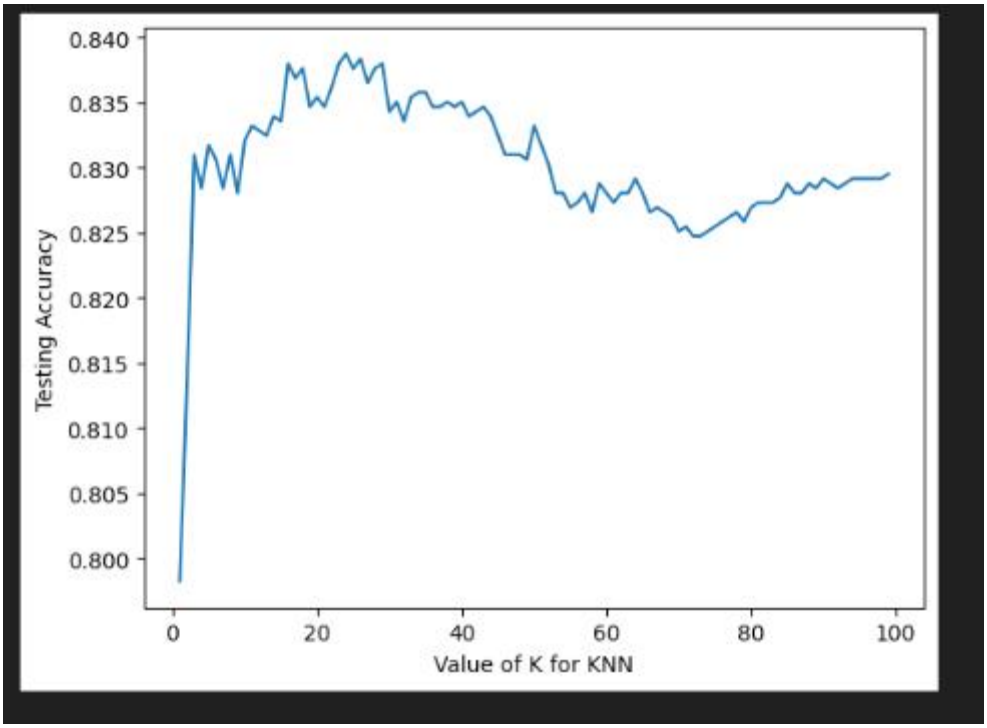
1.14 마지막 정확도 확인

```
# 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 0.79 -> 0.83 이상으로 정확도를 확보하였고 k값이 적을때의 정확도 또한 증가한것을 식별함.

1.15 NB 적용시 정확도 확인

```
    # 15. Check Accuracy Score : NB
    classifier = GaussianNB()
    classifier.fit(X_train, y_train)
    guesses = classifier.predict(X_test)
    print(guesses)

    print(confusion_matrix(y_test, guesses))
    print(metrics.accuracy_score(y_test, guesses))
```

✓ 0.0s

```
[0 0 0 ... 0 2 0]
[[1584   32  243]
 [ 147   17   71]
 [  91   13  518]]
0.7801914580265096
```

1.16 SVM 적용시 정확도 확인

```
    # 15. Check Accuracy Score : SVM
    classifier = SVC(kernel = 'linear')
    classifier.fit(X_train, y_train)
    guesses = classifier.predict(X_test)
    print(guesses)

    print(confusion_matrix(y_test, guesses))
    print(metrics.accuracy_score(y_test, guesses))
```

✓ 1.2s

```
[0 0 0 ... 0 2 0]
[[1693    0  166]
 [ 169    0   66]
 [ 119    0  503]]
0.8085419734904271
```

1.17 각 정확도 비교

KNN - 0.8306332842415317

NB - 0.7801914580265096

SVM - 0.8085419734904271

NB < SVM < KNN

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 8:2으로 선정하였습니다.
정확도를 높이기 위한 속성간 관계를 시각화 하고 제거한 뒤 정규화를 위한 값의 분포를 확인하고 MINMAXScaler를 진행하여 최종적으로 정확도를 높이는데 성공하였습니다.
3개의 알고리즘의 정확도를 비교한 결과 해당 데이터셋에 대한
타입 속성에 대한 정확도를 높게 얻어내는 알고리즘은 KNN 이었습니다.