다이아몬드 가격 예측

컴퓨터공학전공 2022208025 이순우

목차

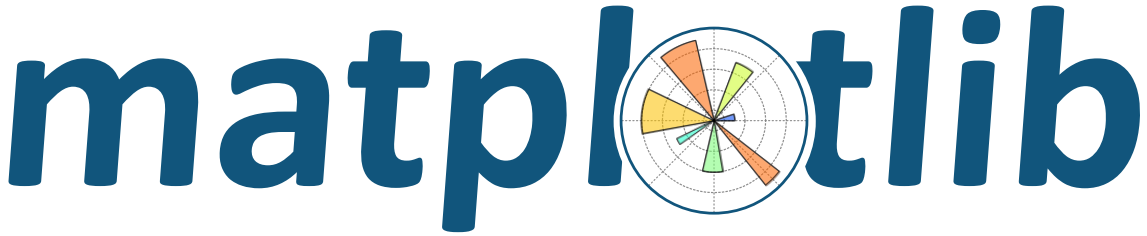
1. 개요

- 다이아몬드 데이터를 기초로 다이아몬드의 특성에 따른 가격에 대한 예측 모형을 분석하고 가장 적절한 모형에 따른 가격을 예측하기 위해서
- 예측하기 어려운 다이아몬드의 구매예측 가격 제시
- 다이아몬드가 평가 요소인 중량, 색, 투명도에 따른 가격의 변화를 분석

2. 사용 Library

NumPy

행렬이나 일반적으로 대규모 다차원 배열을 쉽게 처리할 수 있도록 지원하는 파이썬 라이브러리로 데이터 구조 외에도 수치 계산을 위해 효율적으로 구현된 기능을 제공



Matplotlib

다양한 데이터를 많은 방법으로 도식화 할 수 있도록 하는 파이썬 라이브러리

2. 사용 Library

Seaborn

Matplotlib을 기반으로 다양한 색상 테마와 통계용 차트 등의 기능을 추가한 시각화 패키지

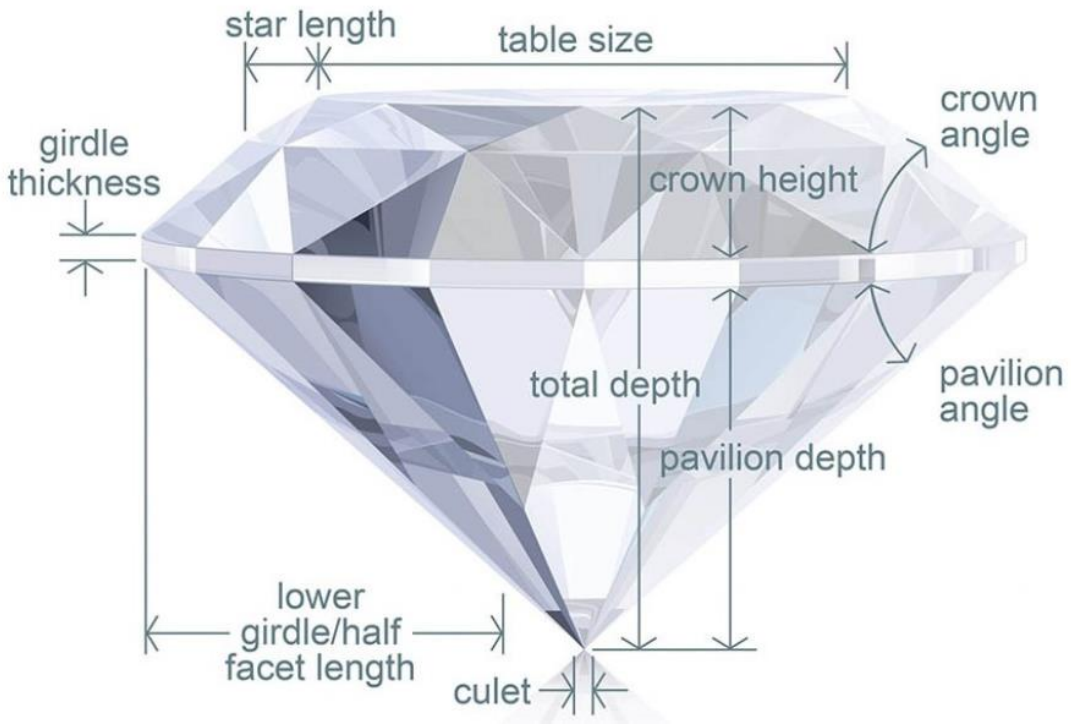
Sklearn

파이썬에서 머신러닝을 분석할 때 사용되는 라이브러리

Pandas

데이터 조작 및 분석, 처리를 위한 라이브러리

3. 데이터 소개



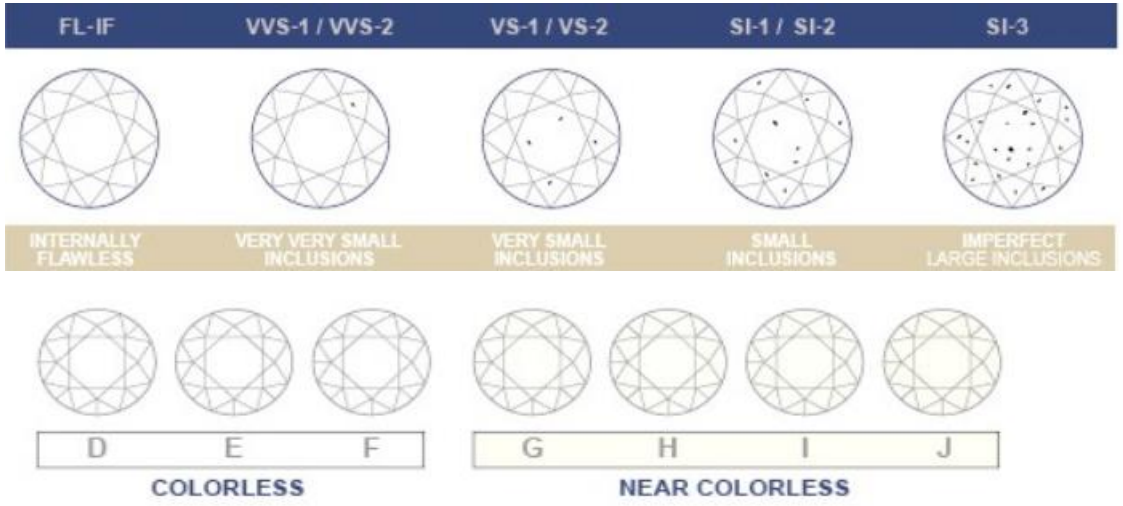

```
data_df = pd.read_csv("../input/diamonds/diamonds.csv")
data_df.sample(10)
```

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
53443	53444	0.75	Good	E	SI1	63.4	57.0	2674	5.75	5.77	3.65
23611	23612	1.72	Very Good	E	SI2	61.0	57.0	11638	7.67	7.76	4.71
38491	38492	0.42	Ideal	D	VS2	61.2	56.0	1031	4.80	4.84	2.95
121	122	0.74	Ideal	E	SI1	62.3	54.0	2762	5.80	5.83	3.62
41056	41057	0.42	Very Good	H	VS2	60.9	55.0	1190	4.87	4.89	2.97
48533	48534	0.71	Very Good	H	SI1	62.5	63.0	1986	5.66	5.63	3.53
40309	40310	0.40	Ideal	E	VS1	62.6	56.0	1125	4.72	4.67	2.94
28127	28128	0.29	Very Good	F	VVS2	63.1	57.0	664	4.22	4.30	2.69
7692	7693	1.08	Very Good	I	SI1	60.5	60.0	4276	6.61	6.67	4.02
26283	26284	2.01	Very Good	H	SI2	63.0	59.0	15773	7.90	7.93	4.99

3. 데이터 소개

```
data_df.head()
```

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75



- Price: 가격(기준 : 미국 달러)
- Carat: 다이아몬드의 무게(1캐럿 = 0.2)
- Cut: 커팅 품질(다이아몬드 광채의 품질에 영향)
- Color: 다이아몬드 색깔
- Clarity: 다이아몬드의 선명도
- X : 다이아몬드의 길이
- Y : 다이아몬드의 넓이
- Z : 다이아몬드의 깊이
- Depth : 전체 깊이 %를 의미(z/mean(x, y)) mean:평균
- Table : 가장 넓은 부분 기준으로 다이아몬드의 상단부분의 넓이

3. 데이터 소개

```
data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  53940 non-null  int64
 1   carat       53940 non-null  float64
 2   cut         53940 non-null  object
 3   color       53940 non-null  object
 4   clarity     53940 non-null  object
 5   depth       53940 non-null  float64
 6   table       53940 non-null  float64
 7   price       53940 non-null  int64
 8   x           53940 non-null  float64
 9   y           53940 non-null  float64
 10  z           53940 non-null  float64
dtypes: float64(6), int64(2), object(3)
memory usage: 4.5+ MB
```

53940개가 모두 non-null이므로 누락된 값은 없다.

cut, color, clarity의 데이터 유형은 Object이므로 데이터 전처리에서 숫자 변수로 변경해야 한다.

4. 데이터 분석 및 시각화

```
plt.figure(figsize=(10,8))
cols = ["#A0522D","#A52A2A","#CD853F","#F4A460","#DEB887"]
ax = sns.violinplot(x="cut",y="price", data=data_df, palette=cols,scale= "count")
ax.set_title("Diamond Cut for Price", color="#774571", fontsize = 20)
ax.set_ylabel("Price", color="#4e4c39", fontsize = 15)
ax.set_xlabel("Cut", color="#4e4c39", fontsize = 15)
plt.show()
```

Cut 품질은 fair, good, very good, premium, ideal순으로 좋다.

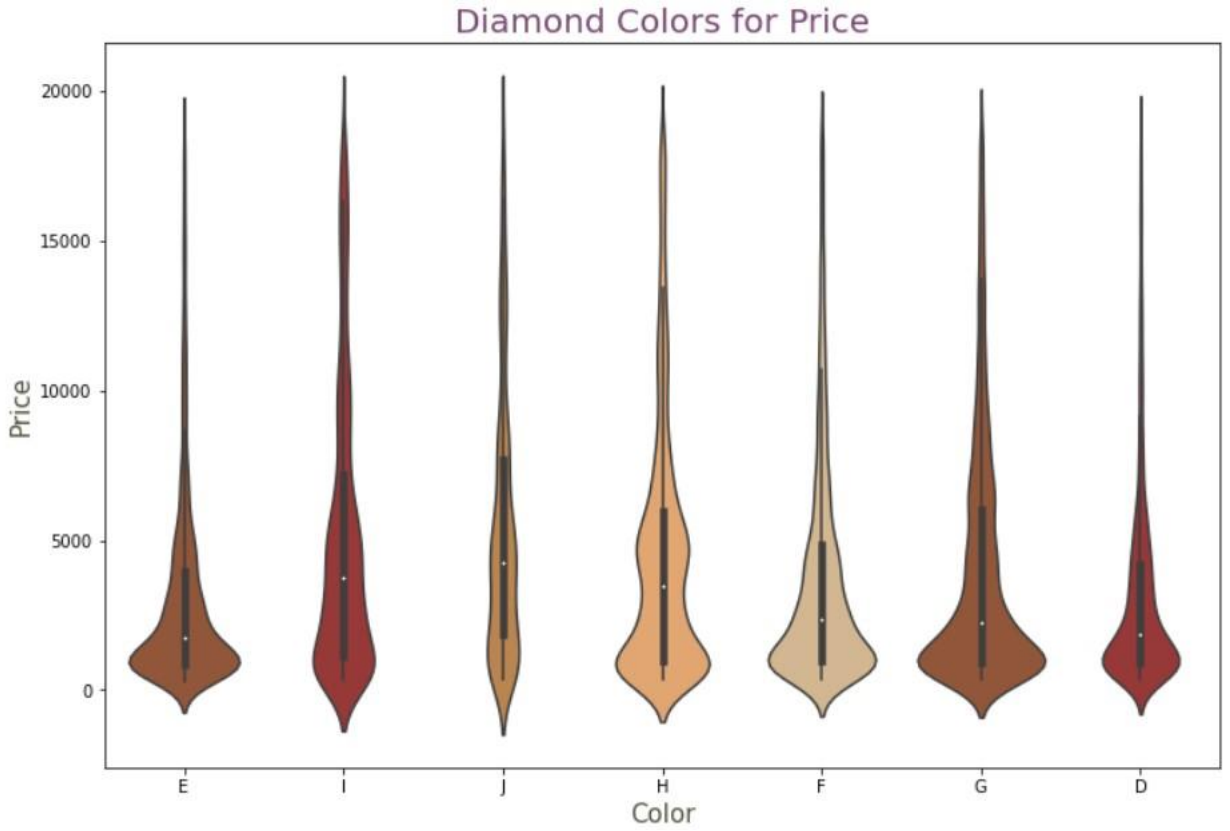
Ideal cut이 가장 많고 Fair는 가장 적다.

4. 데이터 분석 및 시각화

```
plt.figure(figsize=(12,8))
ax = sns.violinplot(x="color",y="price", data=data_df, palette=cols,scale= "count")
ax.set_title("Diamond Colors for Price", color="#774571", fontsize = 20)
ax.set_ylabel("Price", color="#4e4c39", fontsize = 15)
ax.set_xlabel("Color", color="#4e4c39", fontsize = 15)
plt.show()
```

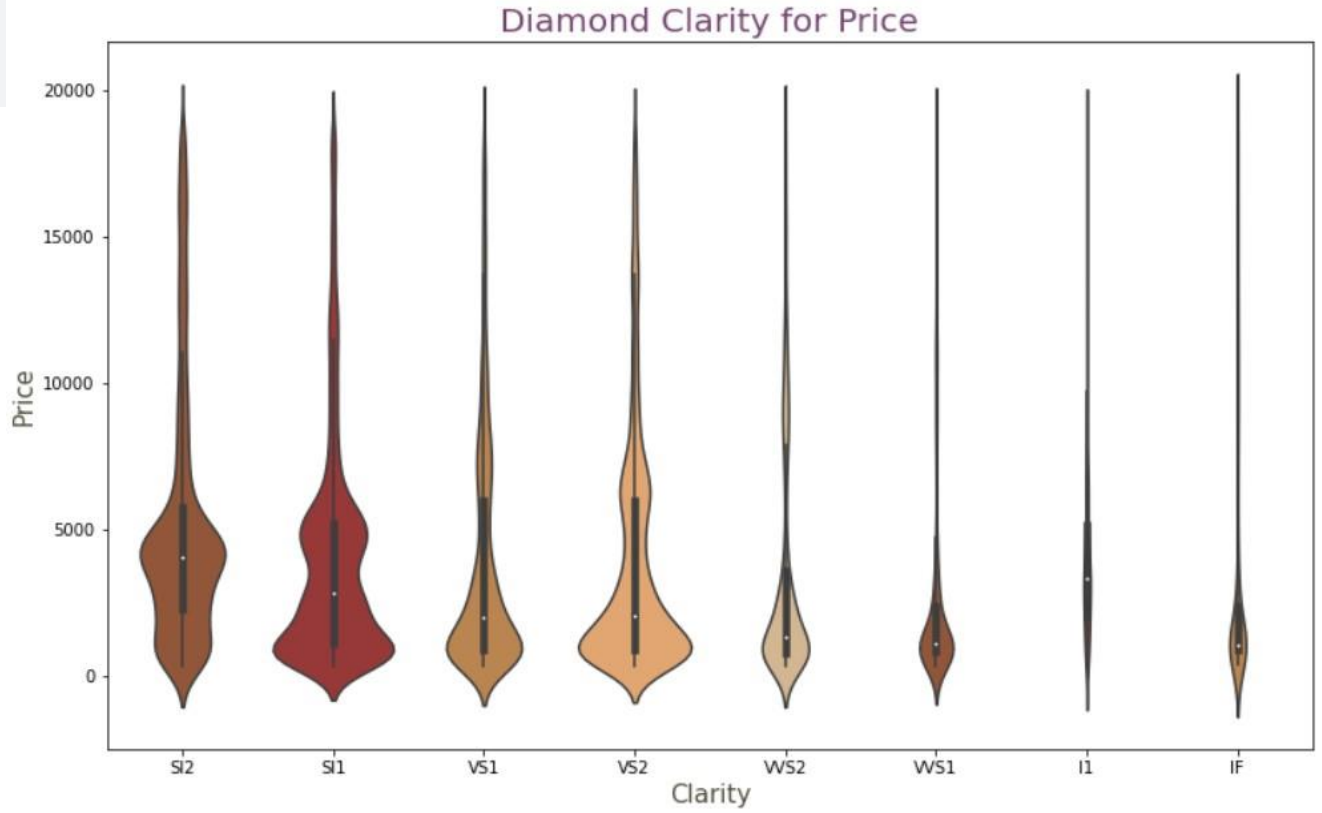
제일 품질이 낮은 J 색깔이 가장 적었고

H와 G는 품질이 낮음에도 많은 양을 차지하고 있다.



4. 데이터 분석 및 시각화

```
plt.figure(figsize=(13,8))
ax = sns.violinplot(x="clarity",y="price", data=data_df, palette=cols,scale= "count")
ax.set_title("Diamond Clarity for Price", color="#774571", fontsize = 20)
ax.set_ylabel("Price", color="#4e4c39", fontsize = 15)
ax.set_xlabel("Clarity", color="#4e4c39", fontsize = 15)
plt.show()
```



가장 좋은 "IF" 선명도와 최악의 "I1" 선명도의

다이아몬드는 매우 희귀하고

중간 선명도가 나머지 대부분을 차지하고 있음

4. 데이터 분석 및 시각화

```
data_df.describe()
```

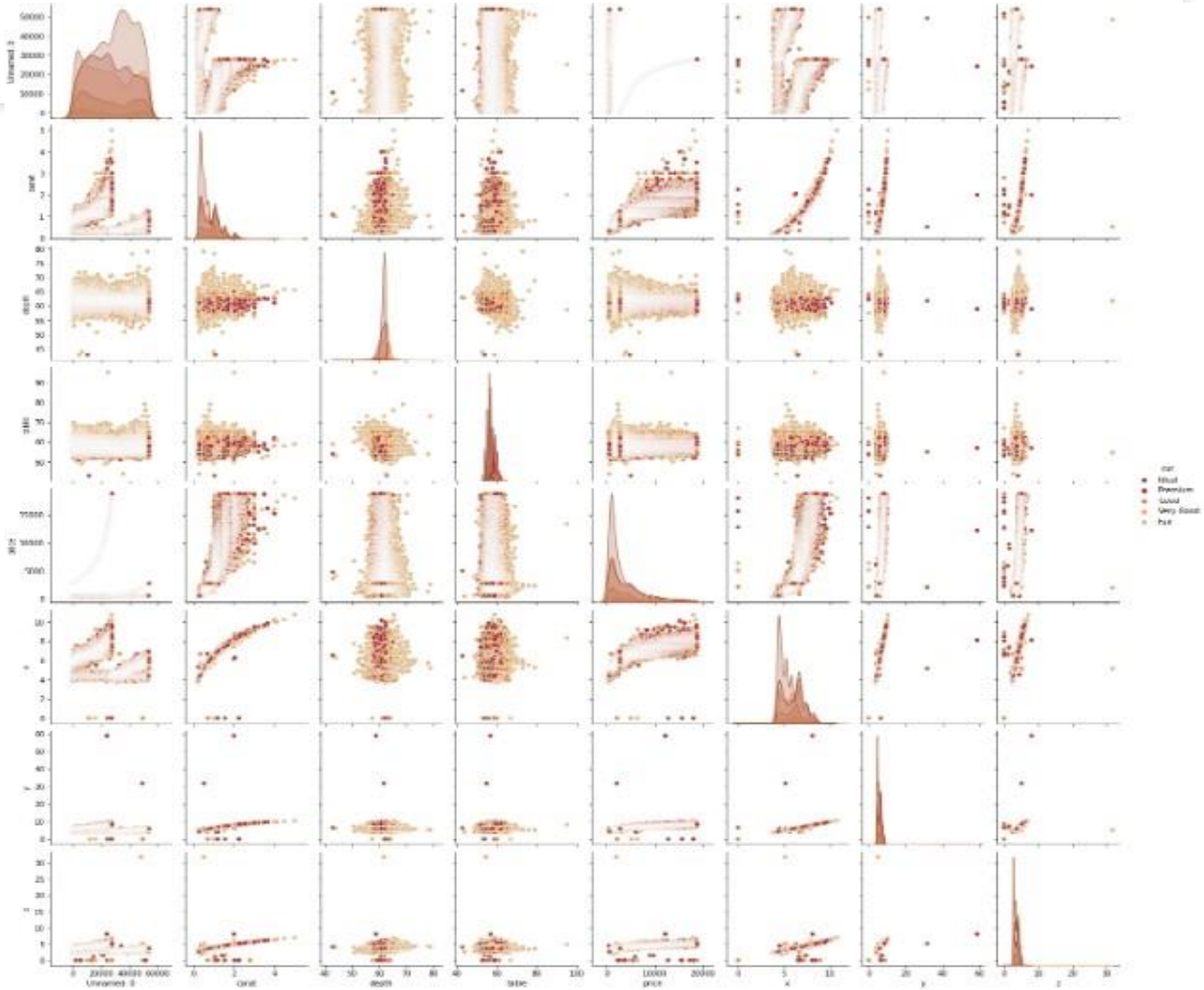
수치형 데이터의 count, 표준편차, min, max 등을 보여준다.

	Unnamed: 0	carat	depth	table	price	x	y	z
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	26970.500000	0.797940	61.749405	57.457184	3932.799722	5.731157	5.734526	3.538734
std	15571.281097	0.474011	1.432621	2.234491	3989.439738	1.121761	1.142135	0.705699
min	1.000000	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	13485.750000	0.400000	61.000000	56.000000	950.000000	4.710000	4.720000	2.910000
50%	26970.500000	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	40455.250000	1.040000	62.500000	59.000000	5324.250000	6.540000	6.540000	4.040000
max	53940.000000	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

'x', 'y' 및 'z'의 특징에서 최소값이 0인 값이 있다. 이런 데이터 값은 적절한 값으로 바꾸거나 완전히 폐기해 줘야 한다.

4. 데이터 분석 및 시각화

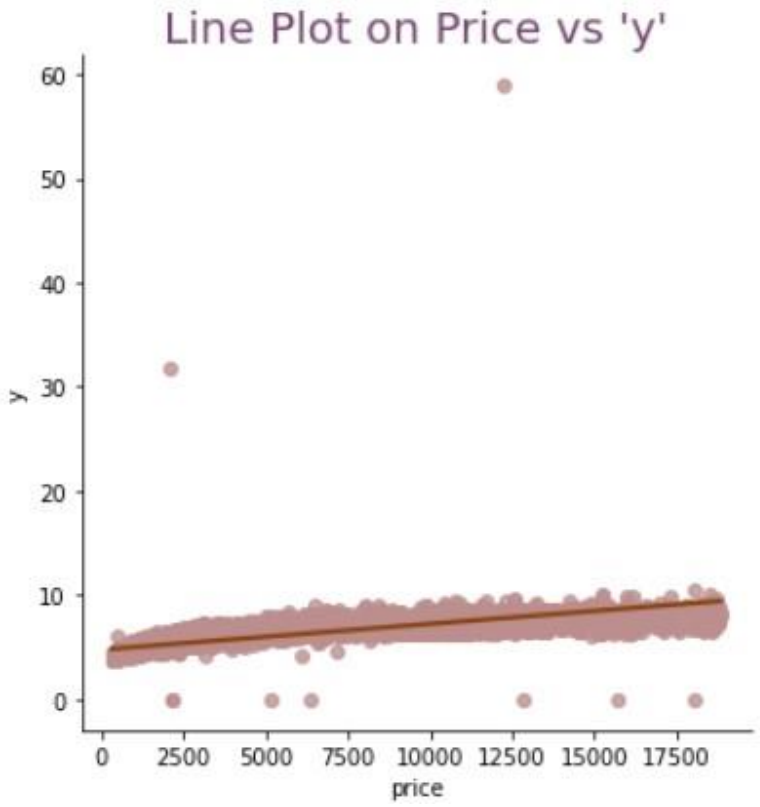
```
ax = sns.pairplot(data_df, hue= "cut", palette = cols)
```



쓸모없는 기능 "unnamed"가 있어 이를 제거해야 하며 값들 중에 멀리 떨어져 있는 이상값으로 보이는 데이터가 있음을 알 수 있다.

"y" 및 "z“, “depth”, “table”에는 데이터에서 제거해야 하는 이상값이 있으며 line plot를 통해 검사하여 확인한 후 값을 제한해야 한다.

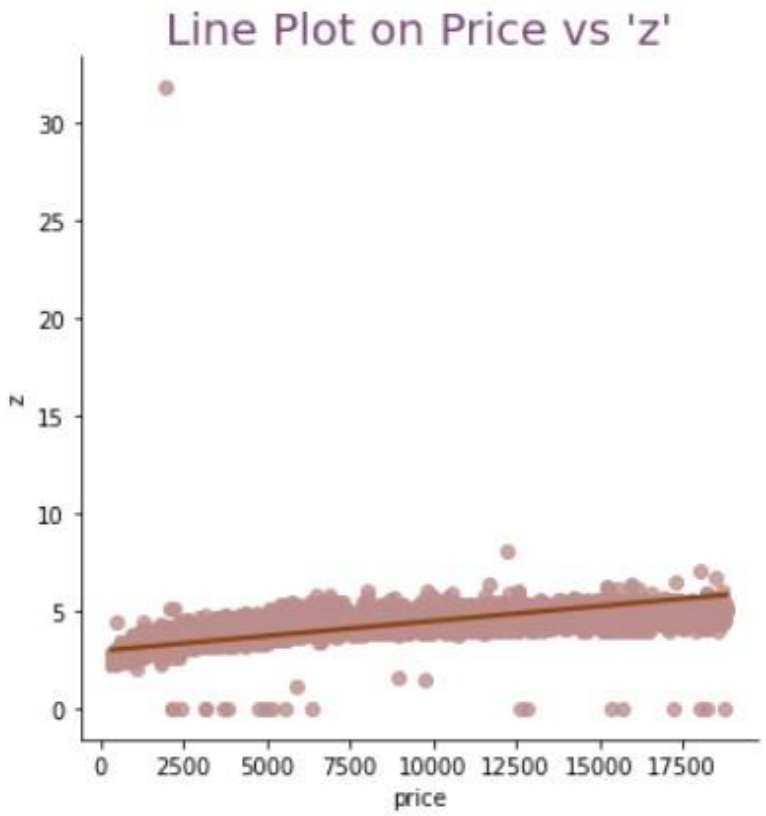
4. 데이터 분석 및 시각화




```
lm = sns.lmplot(x="price", y="y", data=data_df,
                scatter_kws={"color": "#BC8F8F"}, line_kws={"color": "#8B4513"})
plt.title("Line Plot on Price vs 'y'", color="#774571", fontsize = 20)
plt.show()
```

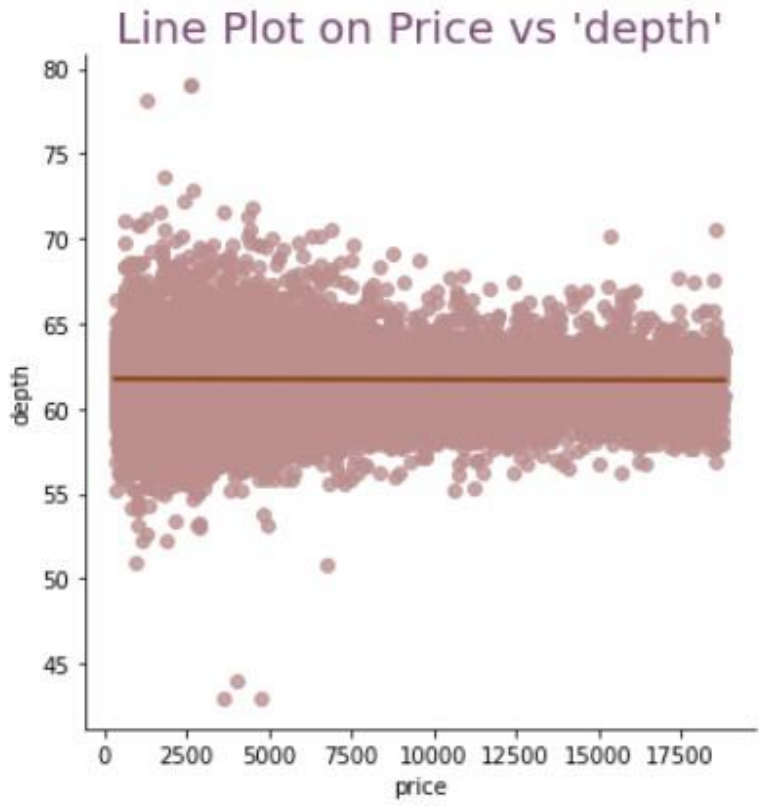
Line plot을 이용하면 알고리즘에 데이터를 공급하기 전에 제거할 이상치를 쉽게 발견할 수 있다.

4. 데이터 분석 및 시각화




```
lm = sns.lmplot(x="price", y="z", data=data_df,
                scatter_kws={"color": "#BC8F8F"}, line_kws={"color": "#8B4513"})
plt.title("Line Plot on Price vs 'z'", color="#774571", fontsize = 20)
plt.show()
```

4. 데이터 분석 및 시각화




```
lm = sns.lmplot(x="price", y="depth", data=data_df,
                scatter_kws={"color": "#BC8F8F"}, line_kws={"color": "#8B4513"})
plt.title("Line Plot on Price vs 'depth'", color="#774571", fontsize = 20)
plt.show()
```

4. 데이터 분석 및 시각화


```
lm = sns.lmplot(x="price", y="table", data=data_df,
                scatter_kws={"color": "#BC8F8F"}, line_kws={"color": "#8B4513"})
plt.title("Line Plot on Price vs 'Table'", color="#774571", fontsize = 20)
plt.show()
```

5. 데이터 전처리

```
data_df = data_df.drop(["Unnamed: 0"], axis=1)
data_df.shape
```

```
(53940, 10)
```

학습에 필요 없는 데이터인 Unnamed를 제거한다.

```
data_df = data_df.drop(data_df[data_df["x"]==0].index)
data_df = data_df.drop(data_df[data_df["y"]==0].index)
data_df = data_df.drop(data_df[data_df["z"]==0].index)
data_df.shape
```

```
(53920, 10)
```

‘x’,’ y’, ‘z’의 최소값이 0인 데이터 제거한다.

5. 데이터 전처리

```
# 적절한 측정값을 정의하여 이상값 제거
data_df = data_df[(data_df["depth"]<75)&(data_df["depth"]>45)]
data_df = data_df[(data_df["table"]<80)&(data_df["table"]>40)]
data_df = data_df[(data_df["x"]<30)]
data_df = data_df[(data_df["y"]<30)]
data_df = data_df[(data_df["z"]<30)&(data_df["z"]>2)]
data_df.shape
```

```
(53907, 10)
```

이상값으로 모델을 학습 시 영향을 줄 수 있으므로 제거해야 한다.

각각의 데이터마다 범위를 정해서 이상치를 제거한다.

5. 데이터 전처리

```
# 원본 데이터를 유지하기 위해 복사본 만들기
data1 = data_df.copy()

# 범주형 데이터가 있는 열에 encoder
columns = ['cut','color','clarity']
label_encoder = LabelEncoder()
for col in columns:
    data1[col] = label_encoder.fit_transform(data1[col])
data1.describe()
```

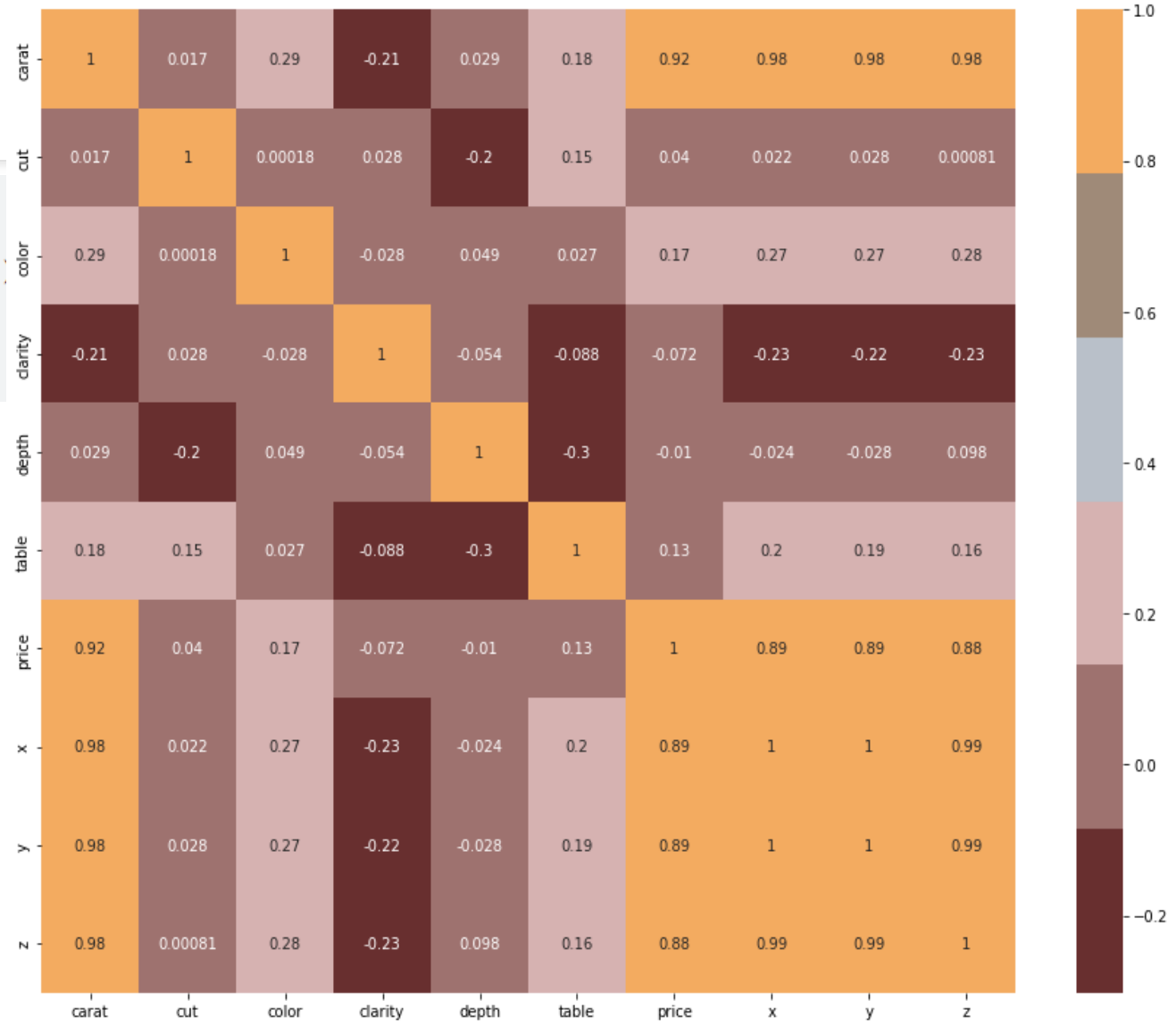
머신러닝 알고리즘을 사용하려면 결측 값과 문자열이 허용되지 않으므로 범주형 데이터인 ‘cut’, ‘color’, ‘clarity’를 인코딩해서 숫자로 변환시켜준다.

	carat	cut	color	clarity	depth	table	price	x	y	z
count	53907.000000	53907.000000	53907.000000	53907.000000	53907.000000	53907.000000	53907.000000	53907.000000	53907.000000	53907.000000
mean	0.797628	2.553379	2.594023	3.835569	61.749741	57.455948	3930.584470	5.731463	5.733292	3.539441
std	0.473765	1.027442	1.701286	1.724572	1.420119	2.226153	3987.202815	1.119384	1.111252	0.691434
min	0.200000	0.000000	0.000000	0.000000	50.800000	43.000000	326.000000	3.730000	3.680000	2.060000
25%	0.400000	2.000000	1.000000	2.000000	61.000000	56.000000	949.000000	4.710000	4.720000	2.910000
50%	0.700000	2.000000	3.000000	4.000000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	1.040000	3.000000	4.000000	5.000000	62.500000	59.000000	5322.000000	6.540000	6.540000	4.040000
max	5.010000	4.000000	6.000000	7.000000	73.600000	79.000000	18823.000000	10.740000	10.540000	6.980000

범주형 데이터가 수치형 데이터로 변해서 describe함수를 사용했을 때 나타나는 것을 알 수 있다.

5. 데이터 전처리

```
# heatmap을 이용하여 상관관계 살펴보기
cmap = sns.diverging_palette(205, 133, 63, as_cmap=True)
cols = (["#682F2F", "#9E726F", "#D6B2B1", "#B9C0C9", "#9F8A78", "#F3AB60"])
corrmat= data1.corr()
f, ax = plt.subplots(figsize=(15,12))
sns.heatmap(corrmat,cmap=cols,annot=True)
```



"carat", "x", "y", "z"는 목표 변수인 가격과 높은 상관관계가 있다.

"cut", "clarity", "depth"는 매우 낮은 상관 관계(<|0.1|)를 가진다.

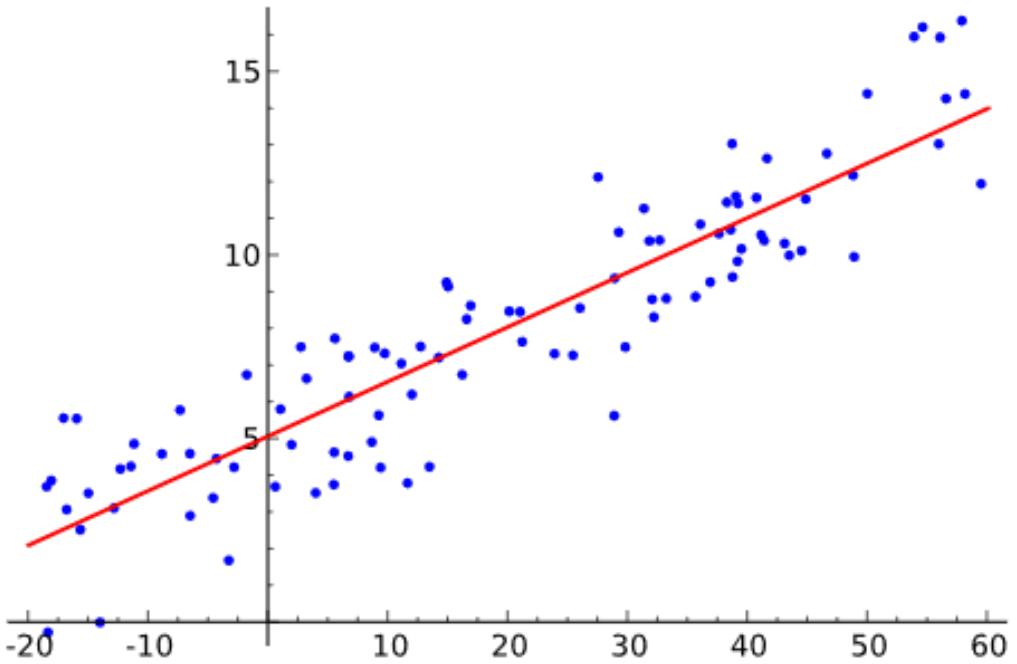
6. 머신러닝 알고리즘

- Linear Regression
- Lasso
- Decision Tree Regressor
- Random Forest Regressor
- KNN Regressor
- XGBoost Regressor

6. 머신러닝 알고리즘

Linear Regression

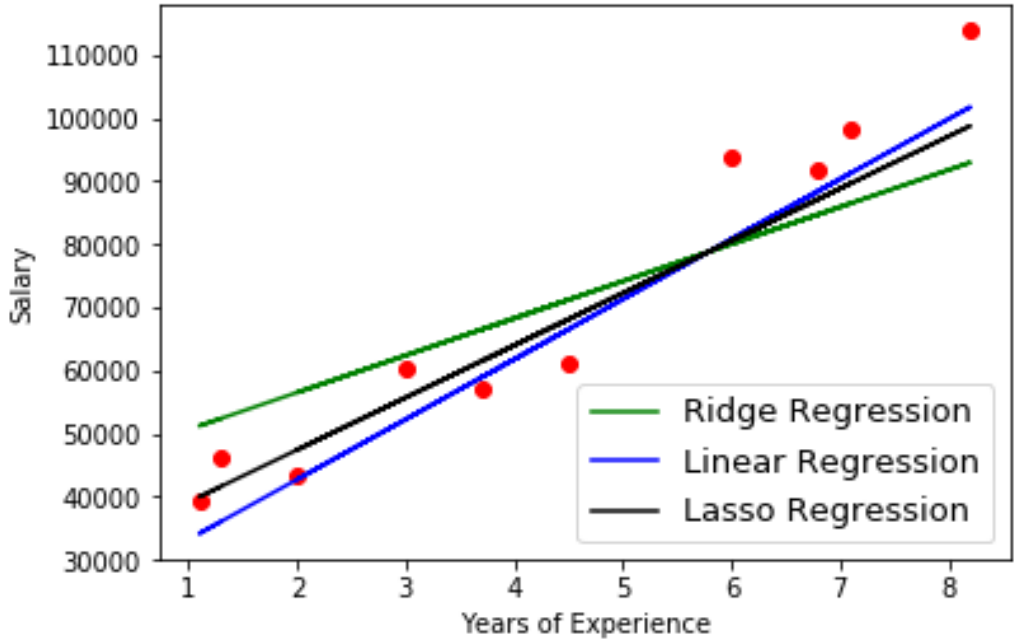
x와 y사이의 관계를 설명할 수 있는 선형함수(선)을 찾아내고 새로운 데이터의 x값이 주어졌을 때 그에 해당하는 y값을 예측하는 것이다.



6. 머신러닝 알고리즘

Lasso

선형회귀 모델에 규제를 주어 복잡도를 감소시키는 방법으로 계수의 절대값을 기준으로 규제 적용한다.

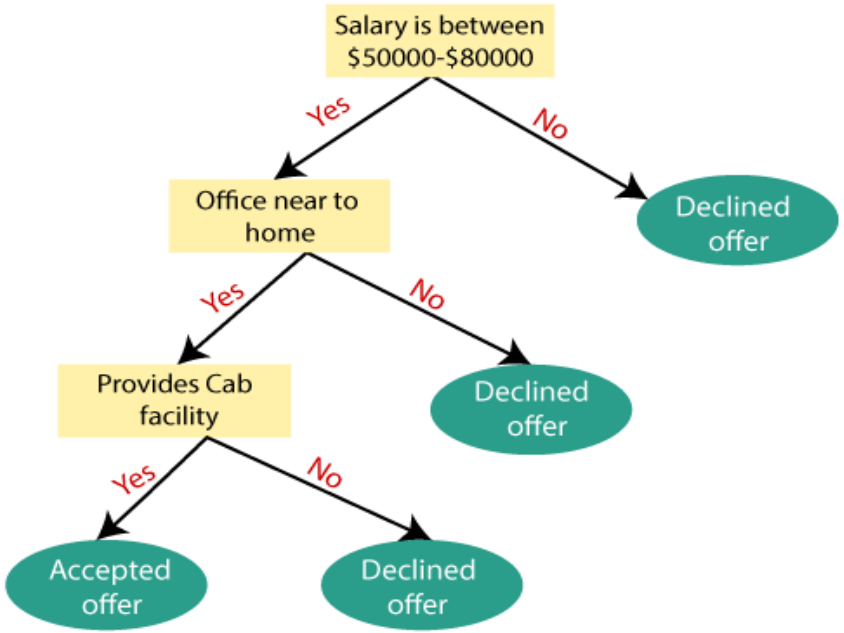


6. 머신러닝 알고리즘

Decision Tree Regression

데이터를 분석하여 이들 사이에 존재하는 패턴을 예측 가능한 규칙들의 조합으로 나타내며,

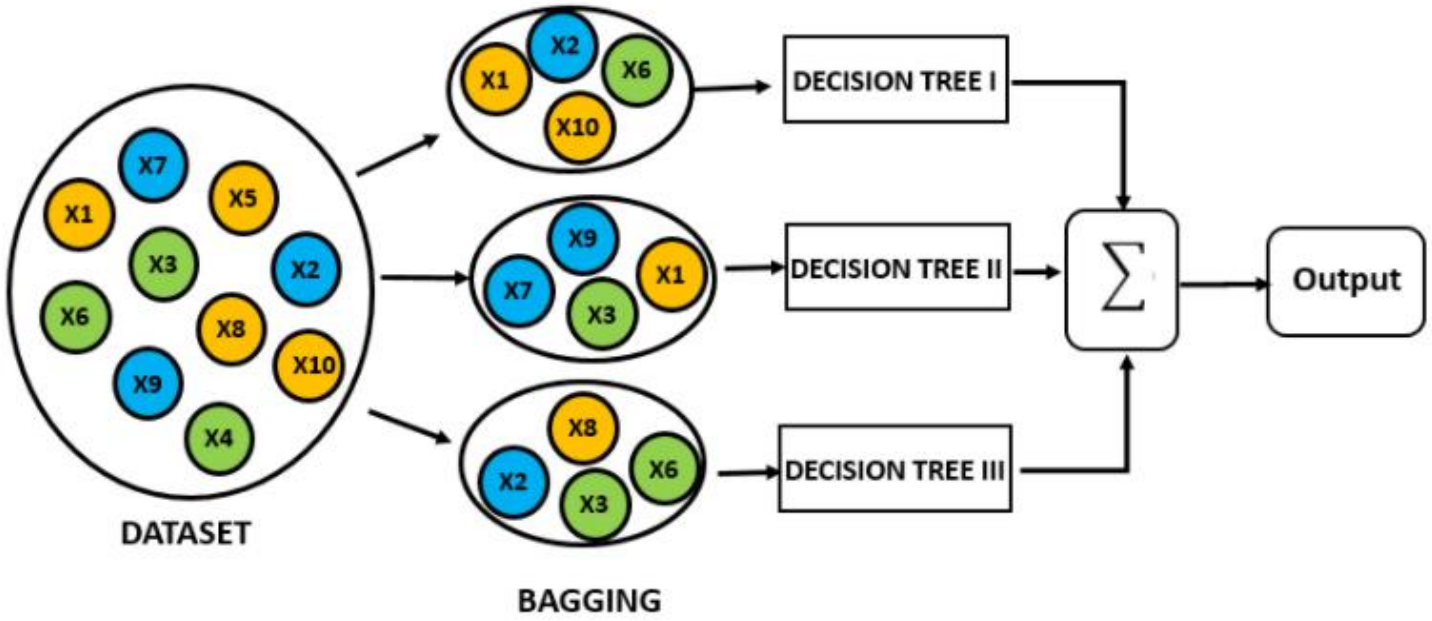
그 모양이 나무와 같다고 해서 의사결정 트리라 불린다.



6. 머신러닝 알고리즘

Random Forest Regression

Bagging이라는 앙상블 기법을 사용한 모델로 기존 bagging의 이점을 살리고 변수를 랜덤으로 선택하는 과정을 통해 개별 나무들의 상관성을 줄여 예측력을 향상시킨 앙상블 모형이다.

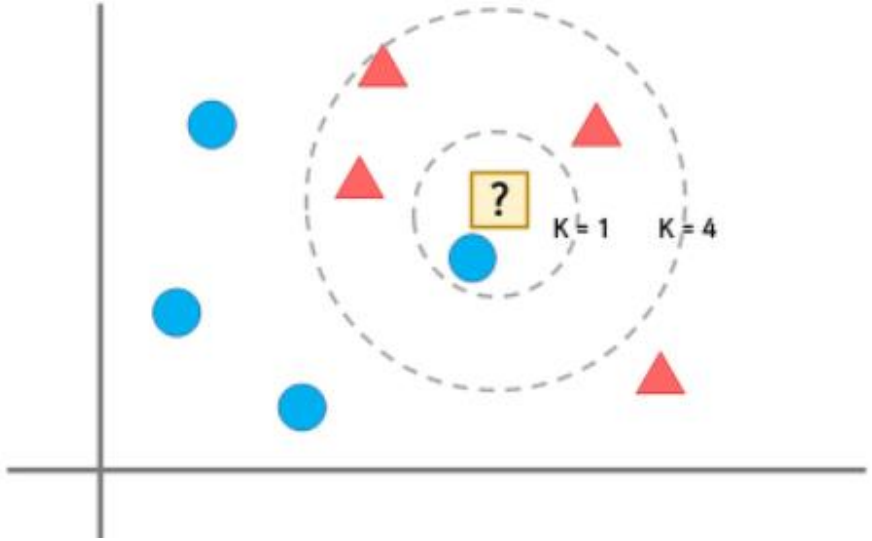


6. 머신러닝 알고리즘

KNN Regression

특정 입력 데이터가 주어졌을 때 입력 데이터와 가까운 k개의 데이터를 이용하여 예측하는 모형이다.

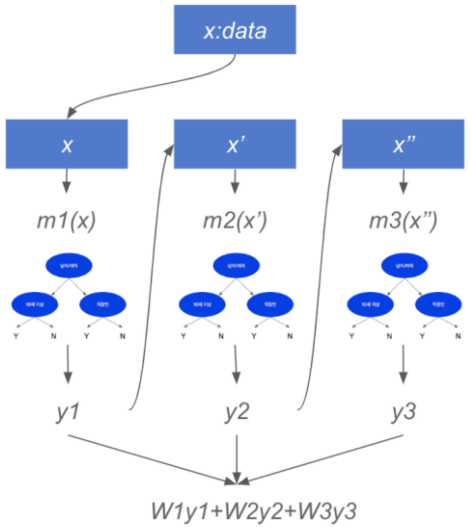
데이터 범위 밖의 새로운 데이터는 예측이 불가능하다.



6. 머신러닝 알고리즘

XGBoost Regression

Boosting 이란? 약한 예측 모형들의 학습 에러에 가중치를 두고, 순차적으로 다음 학습 모델에 반영하여 강한 예측모형을 만드는 것이다. Boosting 기법을 이용하여 구현한 알고리즘은 Gradient Boost 가 대표적인데, 이 알고리즘을 병렬 학습이 지원되도록 구현한 라이브러리가 XGBoost 이다.



7. 데이터 학습

```
# 종속변수와 독립변수 정의
X= data1.drop(["price"],axis =1)
y= data1["price"]
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.20, random_state=25)
```

price를 제거한 값을 X(입력 데이터)로 할당하고 price를 y(타깃 데이터)로 할당

전체 데이터의 20%를 테스트 데이터로 사용

random_state에 난수값을 넣어 호출할 때마다 동일한 학습/ 테스트용 데이터 세트를 생성

7. 데이터 학습

```
pipeline_lr=Pipeline([("scalar1",StandardScaler()),
                      ("lr",LinearRegression())])

pipeline_lasso=Pipeline([("scalar2", StandardScaler()),
                       ("lasso",Lasso())])

pipeline_dt=Pipeline([("scalar3",StandardScaler()),
                      ("dt",DecisionTreeRegressor())])

pipeline_rf=Pipeline([("scalar4",StandardScaler()),
                      ("rf",RandomForestRegressor())])


pipeline_kn=Pipeline([("scalar5",StandardScaler()),
                      ("kn",KNeighborsRegressor())])


pipeline_xgb=Pipeline([("scalar6",StandardScaler()),
                      ("xgb",XGBRegressor())])

# 모든 pipeline 목록
pipelines = [pipeline_lr, pipeline_lasso, pipeline_dt, pipeline_rf, pipeline_kn, pipeline_xgb]

# pipline 사전
pipeline_dict = {0: "LinearRegression", 1: "Lasso", 2: "DecisionTree", 3: "RandomForest",4: "KNeighbors", 5: "XGBRegressor"}

# pipelines 대입
for pipe in pipelines:
    pipe.fit(X_train, y_train)
```

스케일링

데이터 값의 범위가 다 제각각 이기 때문에 범위 차이가 클 경우 데이터를 갖고 모델을 학습할 때 0으로 수렴하거나 무한으로 발산할 수 있다.

따라서 이를 방지하기 위해 모든 특성들의 데이터 분포나 범위를 동일하게 조정

StandardScaler( )를 통해 각 특성의 평균을 0, 분산을 1로 스케일링

Pipeline을 이용해 5가지의 데이터 전처리를 하는 모델을 한데 묶어서 fit 시킨다.

7. 데이터 학습

회귀 모델 성능 평가 지표 - RMSE, MSE, MAE

MSE : 예측 값과 정답의 차이를 제곱하기 때문에, 이상치에 대해 민감하다.

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

즉, 정답에 대해 예측 값이 매우 다른 경우, 그 차이는 오차 값에 상대적으로 크게 반영된다.

MAE : 모델의 예측 값과 실제 값의 차이의 절대값의 평균으로 절대값을 취하기 때문에 가장 직관적으로 알 수 있는 지표이다.

$$\mathrm{MAE} = \frac{1}{n}\sum|\hat{y} - y|$$

RMSE : MSE의 루트를 취하는 형태로 MSE의 단점이 어느 정도 해소된다.

$$\mathrm{RMSE} = \sqrt{MSE} = \sqrt{\frac{\sum(\hat{y}-y)^2}{n}}$$

7. 데이터 학습

```
cv_results_rms = []
for i, model in enumerate(pipelines):
    cv_score = cross_val_score(model, X_train,y_train,scoring="neg_root_mean_squared_error", cv=12)
    cv_results_rms.append(cv_score)
    print("%s: %f " % (pipeline_dict[i], -1 * cv_score.mean()))
```

```
LinearRegression: 1343.967078
Lasso: 1345.083818
DecisionTree: 747.227556
RandomForest: 549.908967
KNeighbors: 810.257754
XGBRegressor: 544.678232
```

교차검증을 간단히 해주는 코드

cross_val_score(model, X, y, scoring=None, cv=None)

model : 회귀 분석 모형

X : 독립 변수 데이터

y : 종속 변수 데이터

scoring : 성능 검증에 사용할 함수 이름

cv : 교차검증 생성기 객체 또는 숫자.

None이면 KFold(3)

숫자 k이면 KFold(k)

7. 데이터 학습

```
# RMSE가 가장 적은 XGBClassifier로 테스트 데이터 예측
pred = pipeline_xgb.predict(X_test)
print("R^2:",metrics.r2_score(y_test, pred))
print("Adjusted R^2:",1 - (1-metrics.r2_score(y_test, pred))*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
```

```
R^2: 0.9816468215020165
Adjusted R^2: 0.9816314874316042
```

제일 좋은 성능 을 보여준 XGBRegressor 모델로
테스트 데이터를 예측

회귀 모델에 대한 적합도 측정값인 R2 score로
모델이 데이터에 얼마나 적합한지를 확인
(R2 score는 0과 1사이의 값을 가지며 1에 가까울 수록 회귀
모델이 데이터에 대해 높은 연관성을 지님)

R2 score는 독립변수가 유의하든 하지 않든 수가 많아
지면 결정계수가 높아지는 단점을 가지므로
수정된 결정계수(adjusted R square)를 추가로 계산

8. 결론 및 소감

- XGBRegressor를 사용해 다이아몬드의 가격을 예측했을 때 98%의 정확도를 갖는 것을 확인 할 수 있습니다.

- 이번 주제에 대해 공부하면서 머신러닝 알고리즘, 스케일링, 교차 검증 등의 개념에 대해 자세히 알게 되었고 가격을 예측하는데 왜 이런 개념들이 필요한지에 대해 배우는 계기가 되었습니다.

  배운 내용을 바탕으로 다음에는 미래의 다이아몬드 가격을 예측하는 프로그램을 작성하고 싶습니다.

감사합니다.