گزارش مروری بر روی دیتاست diamonds

1403/09/01

مریم محمدی

استاد: حوری رضوی

آشنایی با دیتاست :

- **مقدمه :**

ساختار الماس : **ساختار یک الماس در به نمایش گذاشتن کیفیت آن بسیار مهم است و برش (cut)یکی از مهم ترین عوامل آن است. تراش الماس به تناسبات، تقارن و جلا دادن آن اشاره دارد که در نهایت نحوه شکست نور در سنگ را تعیین می کند و در نتیجه درخشش آن ایجاد می شود.**

وزن قیراط (carat) یکی دیگر از عناصر ضروری ساختار الماس است .قیراط واحد وزنی است که برای اندازه گیری اندازه الماس استفاده می شود و یک قیراط معادل 200 میلی گرم است. هر چه وزن قیراط الماس بزرگتر باشد، کمیاب تر و ارزشمندتر است.

با این حال، وزن قیراط تنها عامل تعیین کننده ارزش الماس نیست. عواملی مانند برش، شفافیت و رنگ نیز نقش مهمی در ارزیابی کیفیت و ارزش کلی الماس دارند. شفافیت (clearity) و رنگ (color) دو عنصر اضافی هستند که ساختار الماس را تشکیل می دهند.

شفافیت به وجود ایرادات داخلی (آخال ها) و لکه های خارجی (لکه ها) در یک الماس اشاره دارد که می تواند بر شفافیت و ظاهر کلی آن تأثیر بگذارد. موسسه گوهرشناسی آمریکا (GIA) برای ارزیابی شفافیت الماس، مقیاس درجه بندی شفافیتی را از بی عیب (بدون درج یا لکه) تا شامل (شاخص های قابل مشاهده با چشم غیر مسلح) ایجاد کرده است. در همین حال، درجه بندی رنگ از D (بی رنگ) تا Z (زرد روشن یا قهوه ای) متغیر است که الماس های بی رنگ با ارزش ترین و مطلوب ترین آنها هستند. تأثیر ساختار الماس، از جمله برش، قیراط، شفافیت و رنگ، فراتر از جذابیت زیبایی شناختی آنها به ارزش مالی و پتانسیل سرمایه گذاری آنها است.

4Cs (برش، قیراط، شفافیت و رنگ) به عنوان عوامل کلیدی تعیین کننده کیفیت و ارزش یک الماس به طور جهانی شناخته می شوند. الماس هایی با درجه تراش بالا، وزن قیراط بزرگتر، شفافیت عالی و درجه رنگ معمولاً ارزشمندتر هستند و در بازار به دنبال آن هستند. ساختار یک الماس نه تنها بر قیمت آن، بلکه بر ارزش فروش مجدد و پتانسیل سرمایه گذاری آن نیز تأثیر می گذارد، و آن را برای خریداران و فروشندگان صنعت الماس مورد توجه قرار می دهد.

د). 4Cs of Diamond Quality: 4Cs of Diamond Qualityچیست؟

https://4cs.gia.edu/en-us/4cs-of-diamond-quality/

DIAMOND **EDUCATION**

DIAMOND **SHAPE**

Round | Princess | Emerald | Cushion | Oval | Heart | Pear | Marquise

DIAMOND **CARAT**

0.50 ct	0.75 ct	1.00 ct	1.25 ct	1.50 ct	2.00 ct	2.50 ct	3.00 ct	Dime
5.1mm	5.8mm	6.4mm	7.00mm	7.4mm	8.1mm	8.6mm	9.3mm	17.91mm

DIAMOND **CLARITY**

IF	VVS1-VVS2	VS1-VS2	SI1	SI2	I1	I2	I3
Internally Flawless	Very Very Slight Inclusions	Very Slight Inclusions	Slight Inclusions		Inclusions		

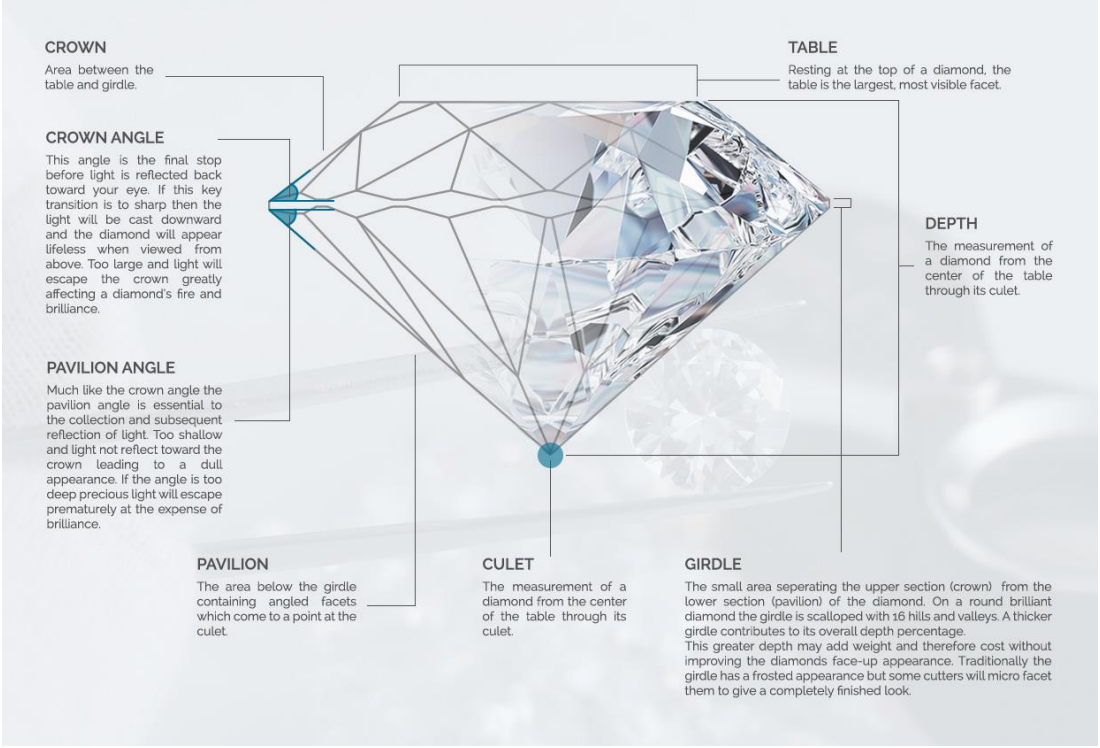
DIAMOND **COLOR**

D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Colorless			Near Colorless				Faint Yellow			Very Light Yellow					Light Yellow							

DIAMOND **CUT**

*No two diamonds are identical-inclusions will vary by stone.
*All MM sizes are approximate.

LULURIA



Informations about dataset :

- This dataset contains information about 53,940 round-cut diamonds.
- There are 10 variables measuring various pieces of information about the diamonds. There is 1 variable that has an integer structure: price
- There are 3 variables with an ordered factor structure: cut, color, & clarity. An ordered factor arranges the categorical values in a low-to-high rank order. For example, there are 5 categories of diamond cuts with “Fair” being the lowest grade of cut to ideal being the highest grade.
- There are 6 variables that are of numeric structure: carat, depth, table, x, y, z

Feature description:

price price in US dollars (326--18,823).This is the target column containing tags for the features.

The 4 Cs of Diamonds:

carat (0.2--5.01) The carat is the diamond’s physical weight measured in metric carats. One carat equals 1/5 gram and is subdivided into 100 points. Carat weight is the most objective grade of the 4Cs.

cut (Fair, Good, Very Good, Premium, Ideal) In determining the quality of the cut, the diamond grader evaluates the cutter's skill in the fashioning of the diamond. The more precise the diamond is cut, the more captivating the diamond is to the eye.

color, from J (worst) to D (best) The colour of gem-quality diamonds occurs in many hues. In the range from colourless to light yellow or light brown. Colourless diamonds are the rarest. Other natural colours (blue, red, pink for example) are known as "fancy," and their colour grading is different than from white colorless diamonds.

clarity (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best)) Diamonds can have internal characteristics known as inclusions or external characteristics known as blemishes. Diamonds without inclusions or blemishes are rare; however, most characteristics can only be seen with magnification.

Dimensions

x length in mm (0--10.74)

y width in mm (0--58.9)

z depth in mm (0--31.8)

depth total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43--79) The depth of the diamond is its height (in millimetres) measured from the culet (bottom tip) to the table (flat, top surface).

table width of the top of the diamond relative to widest point (43--95)

A diamond's table refers to the flat facet of the diamond seen when the stone is face up. The main purpose of a diamond table is to refract entering light rays and allow reflected light rays from within the diamond to meet the observer's eye. The ideal table cut diamond will give the diamond stunning fire and brilliance.

حال وارد مرحله اصلی شده و دیتاست را در load کرده و هر یک از کتابخانه های زیر را import میکنیم :

```
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

با دستور pd.read_csv() یک پیش نمایش از دیتاست مان را میبینیم .

```
data = pd.read_csv("diamonds.csv")
data
```

مشاهده میکنیم که دیتاست ما دارای 53940 سطر و 11 ستون است . البته ستون اول همان ردیف ها میباشد که در ادامه حذف خواهد شد و ستون های اصلی ما 10 تا خواهد بود . مشاهده میکنیم که برخی از این ستون ها numeric هستند و برخی خیر.

	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
...	...	...	...	...	...	...	...	...	...	...	...
53935	53936	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53936	53937	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53937	53938	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53938	53939	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	53940	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

53940 rows × 11 columns

به مرحله ی DATA PREPROCESSING **میرسیم . در این مرحله باید :**

steps :

- Data cleaning
- Identifying and removing outliers
- Encoding categorical variables

ستون اول همان ردیف ها هستند (" Unnamed: 0") بنابراین ما آن را حذف می کنیم.

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

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

حداقل مقدار "x"، "y"، "z" صفر است. این نشان می دهد که مقادیر معیوب در داده هایی وجود دارد که الماس های بی بعد یا دو بعدی را نشان می دهد. بنابراین باید آن‌ها را فیلتر کنیم، زیرا نقاط داده‌ای به وضوح معیوب هستند.

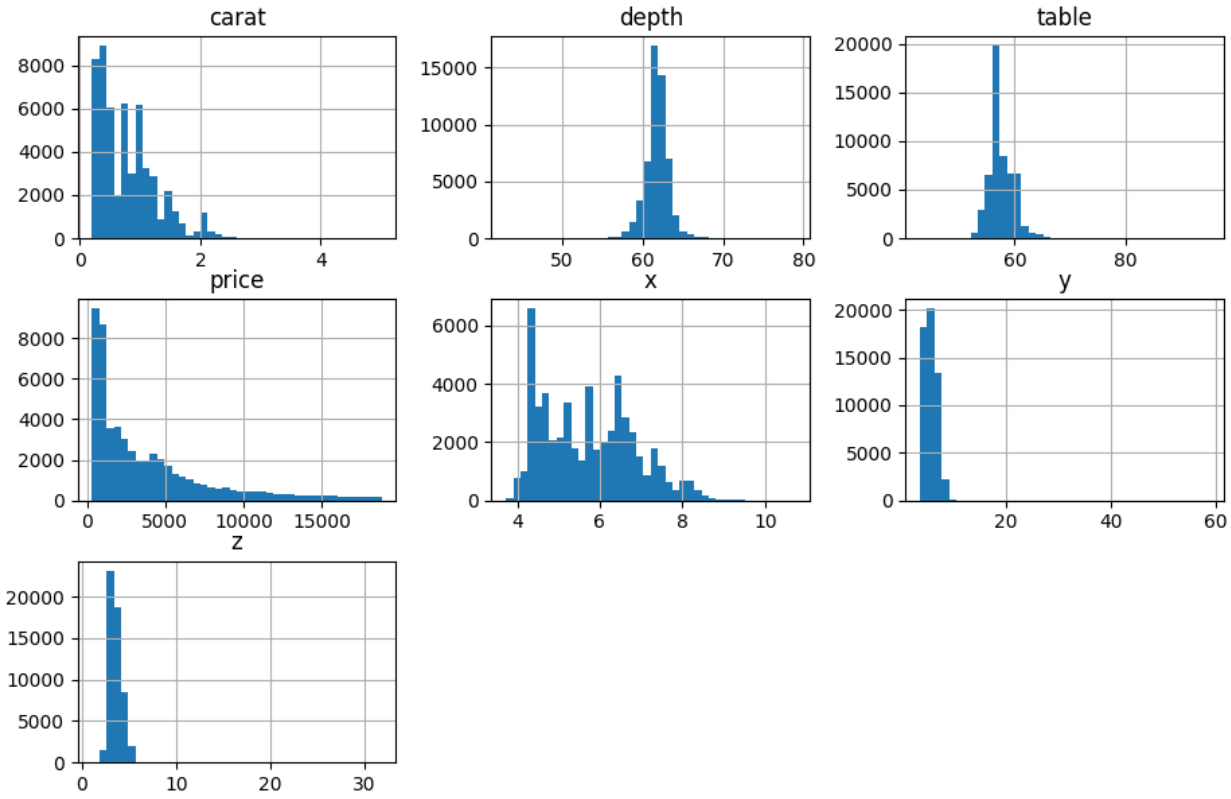
```
data = data.drop(data[data["x"]==0].index)
data = data.drop(data[data["y"]==0].index)
data = data.drop(data[data["z"]==0].index)
data.shape
```

```
(53920, 10)
```

We lost **20** data points by deleting the dimensionless(2-D or 1-D) diamonds.

Histogram:

```
data.hist(bins=40, figsize=(11,7))
```



بر اساس هیستوگرام‌هایی که برای داده هایمان کشیده ایم، می‌توان چند نکته را در مورد هر کدام از feature**ها فهمید:**

1. **قیراط:**
   - **توزیع وزن قیراط به سمت راست متمایل است (چولگی به راست)، یعنی بیشتر الماس‌ها وزن قیراطی کمتر از** 1 **دارند. این نشان می‌دهد که الماس‌های کوچک‌تر در مجموعه داده هایمان بیشتر هستند.**

2. **depth:**
   - **توزیع عمق به صورت نرمال به نظر می‌رسد، و بیشتر الماس‌ها دارای عمقی بین** 60 **تا** 65 **هستند. این بازه معمولاً برای عمق الماس‌ها مناسب است و نشان می‌دهد که اکثر الماس‌ها در محدوده استاندارد عمق قرار دارند.**

3. **اندازه سطح بالای الماس:(Table)**

- **اندازه سطح بالای الماس نیز به صورت نرمال توزیع شده و اوج آن بین 55 تا 60 است. این مقدار نشان‌دهنده درصد اندازه سطح بالای الماس‌هاست که اغلب به عنوان محدوده ایده‌آل برای برش در نظر گرفته می‌شود.**

4. **Xطول:**

- **طول الماس‌ها در محدوده 4 تا 6 دارای اوج است که به این معناست که بیشتر الماس‌ها دارای ابعادی در این محدوده هستند. یک دنباله طولانی به سمت مقادیر بزرگ‌تر وجود دارد که نشان‌دهنده تعداد کمی از الماس‌های بزرگ‌تر است.**

5. **Yعرض:**

- **عرض الماس‌ها نیز مشابه طول در محدوده 4 تا 6 اوج دارد و یک دنباله به سمت مقادیر بزرگ‌تر دارد، اما بیشتر الماس‌ها در همان محدوده قرار دارند.**

6. **Zارتفاع:**

- **توزیع ارتفاع مشابه X و Y است، به طوری که اوج آن بین 2 تا 5 است و تعداد کمی از الماس‌ها ارتفاع بزرگ‌تری دارند. تمرکز الماس‌ها در این محدوده نشان‌دهنده تناسب ابعاد بیشتر الماس‌ها است.**

7. **قیمت:**

- **توزیع قیمت هم به شدت چولگی به راست دارد، یعنی بیشتر الماس‌ها قیمت کمتر از 2000 دارند و تعداد کمی از الماس‌ها با قیمت‌های بسیار بالا وجود دارند.**

به طور کلی، داده‌ها نشان می‌دهند که اکثر الماس‌ها در محدوده ابعاد و قیمتی خاص و متوسط قرار دارند و تعداد کمتری از الماس‌های بزرگ‌تر و گران‌تر در مجموعه دیده می‌شود.

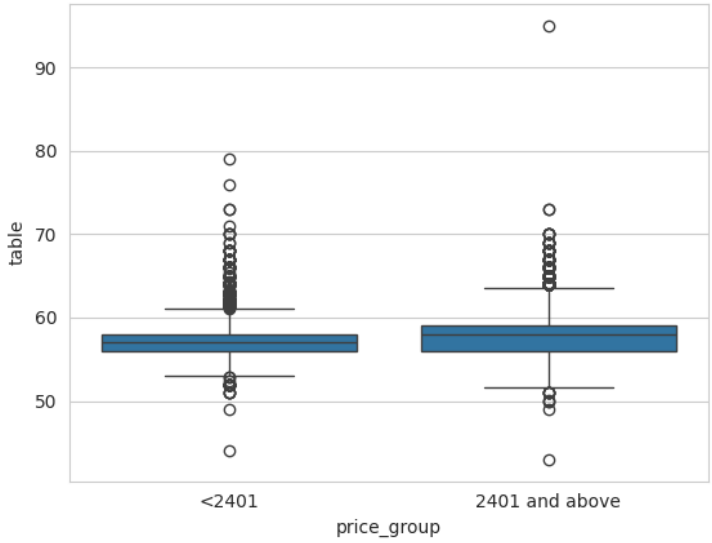
❖ ایده اول برای دسته بندی مقادیر price

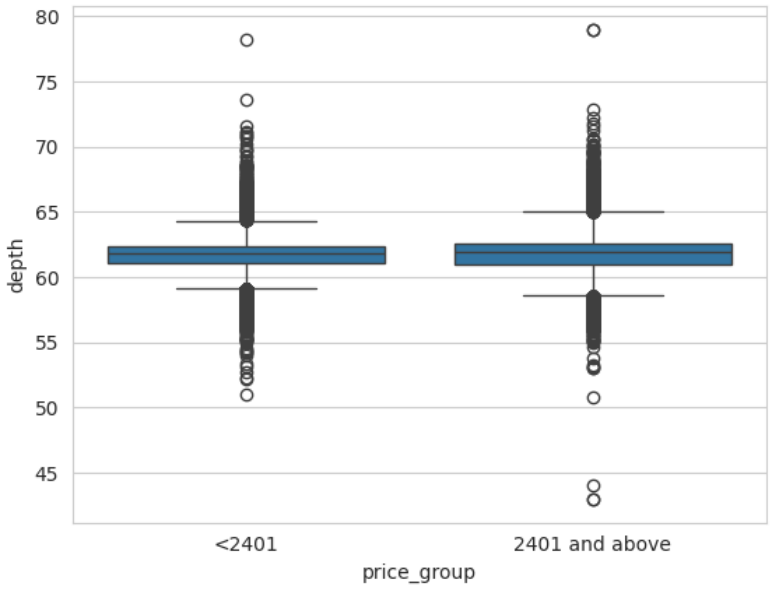
ایده اول این بود که با استفاده از مقدار وسط **price** آن را به دو دسته ی بزرگتر مساوی **2401** و کوچکتر از **2401** تقسیم کرده و یک **price group** برای ان ها تشکیل داده و بعد مابقی نتایج با این **price group** مقایسه شوند. این روند در ابتدا درست به نظر میرسید ...

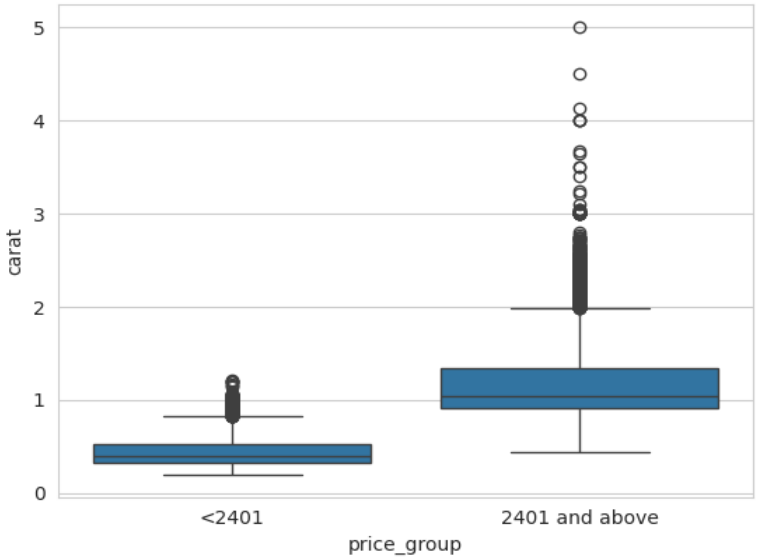
```
data["price_group"] = pd.cut(data['price'] , bins = [0,2401,float('inf')],
labels=['<2401' , '2401 and above'])
lower_prices = data[data['price_group']== '<2401']
higher_prices = data[data['price_group']== '2401 and above']
data.columns
```

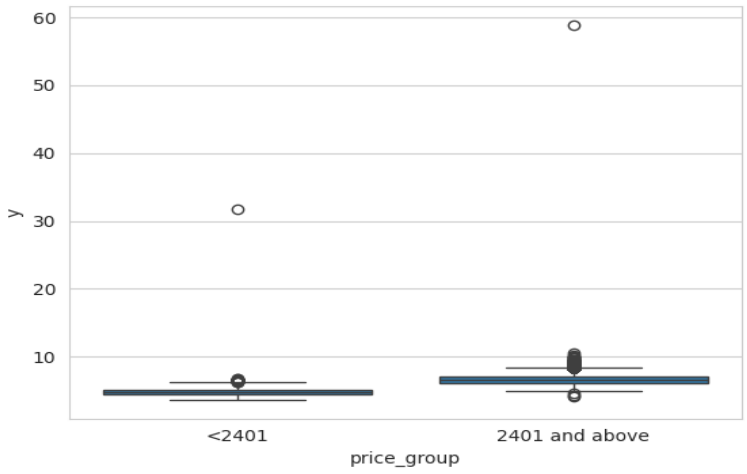
```
Index(['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'price', 'x', 'y',
       'z', 'price_group'],
      dtype='object')
```

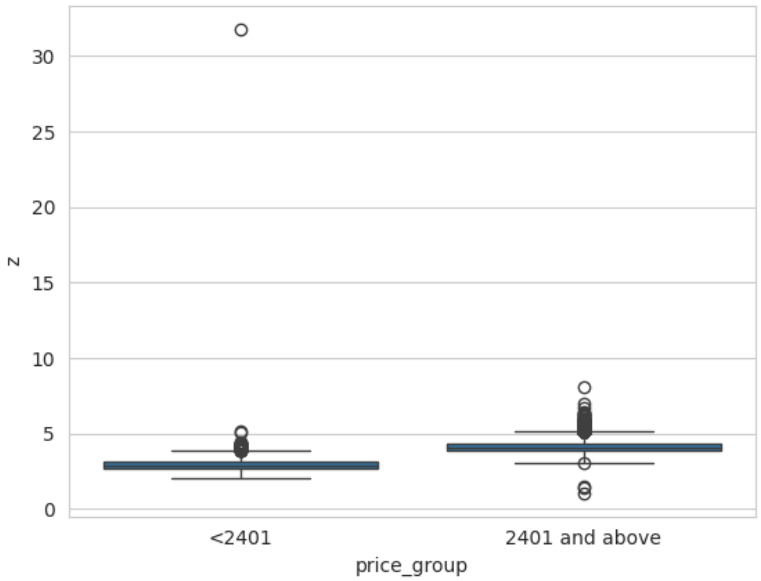
Boxplot




80
75
70
65
60
55
50
45
depth
<2401
2401 and above
price_group

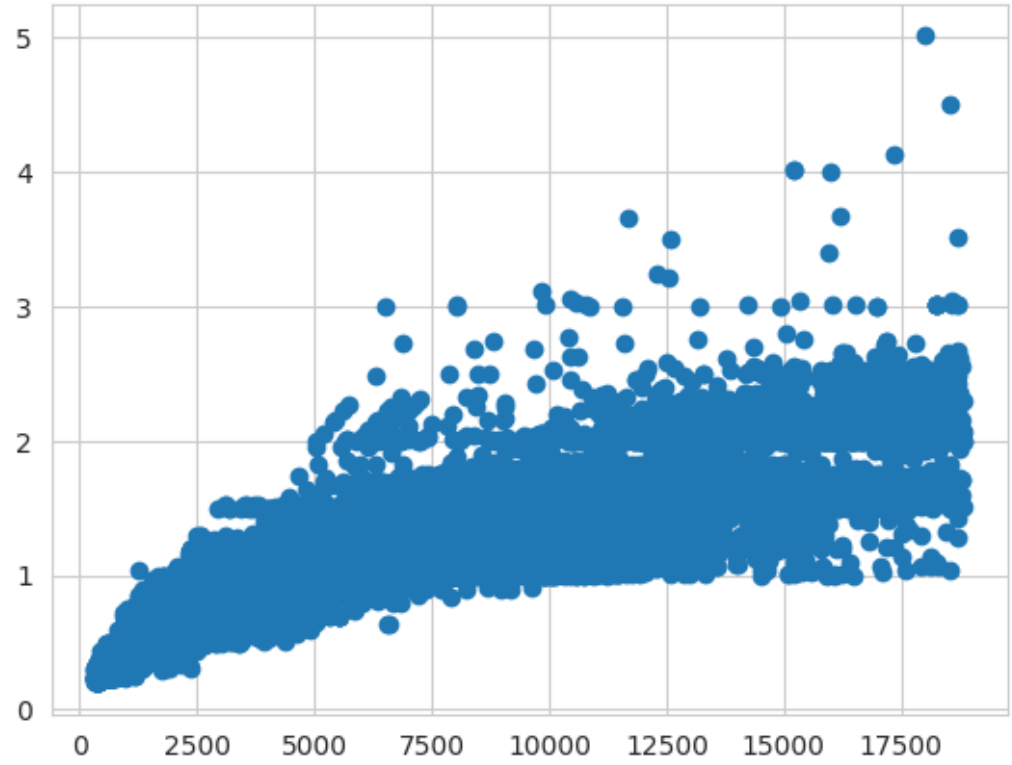

5
4
3
2
1
0
carat
<2401
2401 and above
price_group


60
50
40
30
20
10
y
<2401
2401 and above
price_group

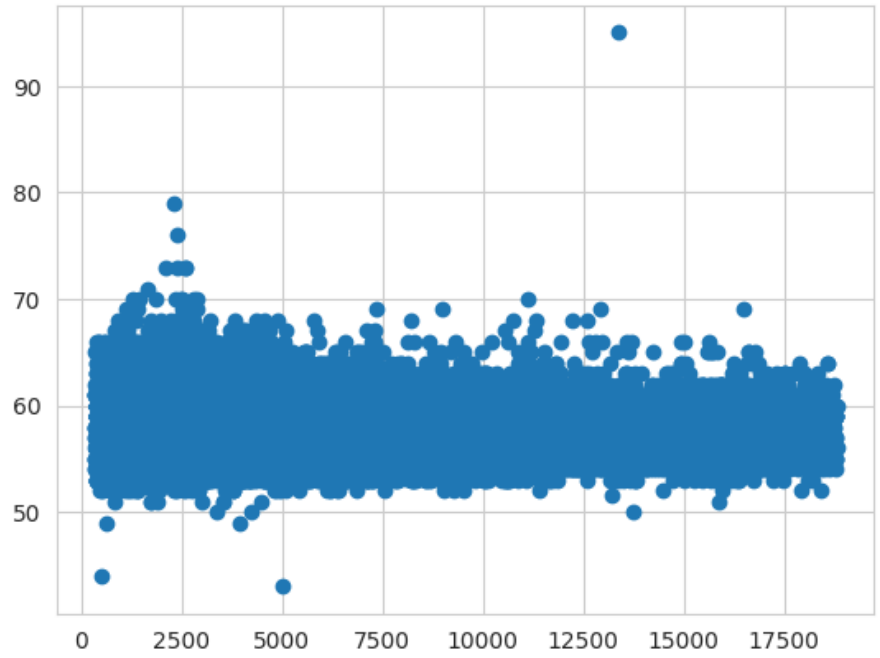


Scatter plot:

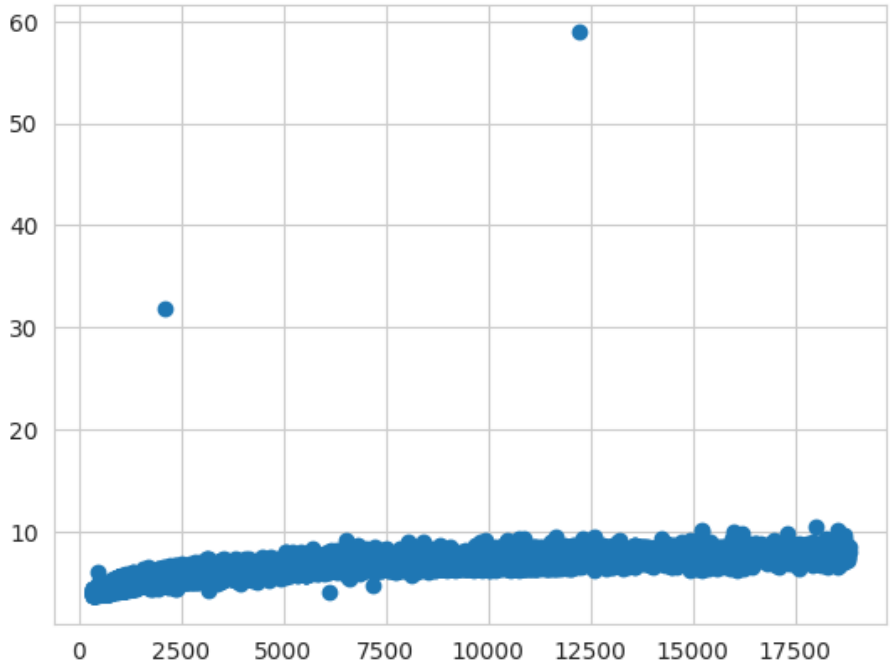
```
plt.scatter(data.price, data.carat)
plt.show()
```



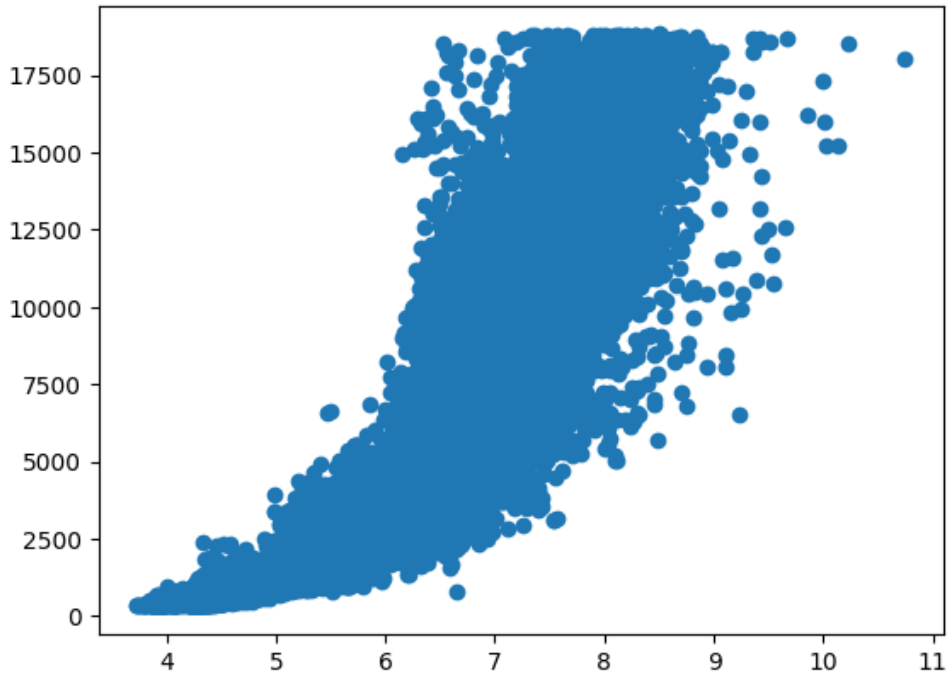

```
plt.scatter(data.price, data.table)
plt.show()
```




```
plt.scatter(data.price, data.y)
plt.show()
```

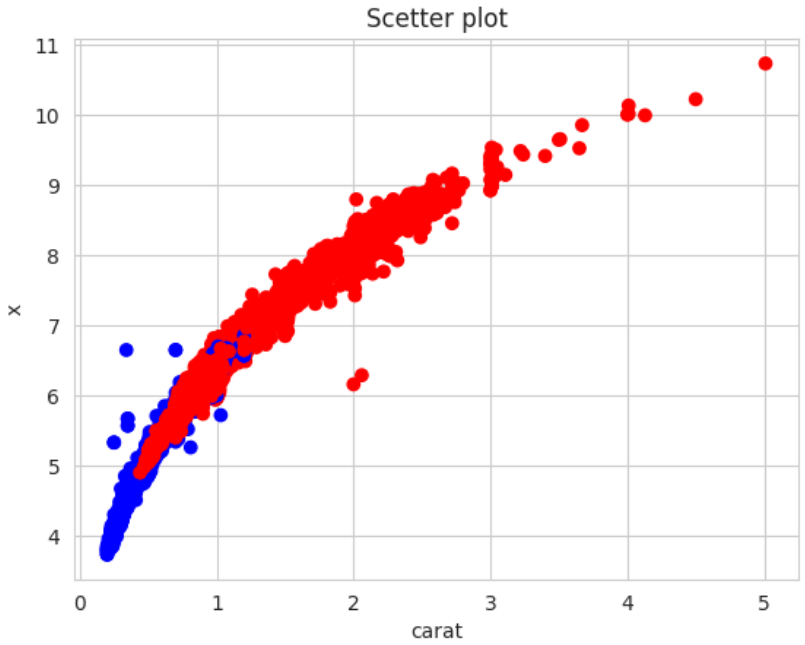



```
plt.scatter(data.x, data.price)
plt.show()
```

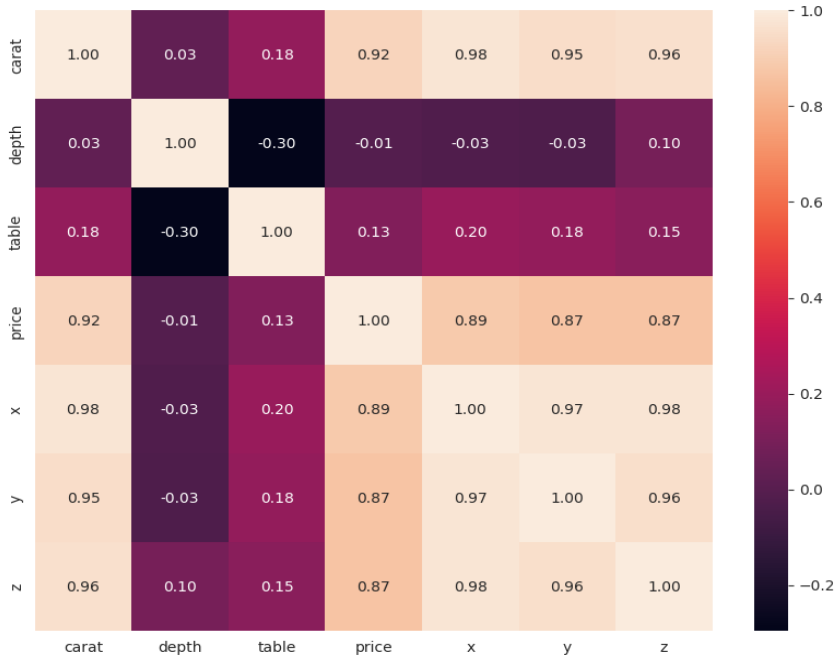



```
col = np.where(data.price < 2401 ,'b','r')
plt.scatter(data.depth,data.carat,c = col)
plt.title("Scetter plot")
plt.xlabel("depth")
plt.ylabel("carat")

plt.show()
```

```
col = np.where(data.price < 2401 ,'b','r')
plt.scatter(data.carat,data.x,c = col)
plt.title("Scetter plot")
plt.xlabel("carat")
plt.ylabel("x")

plt.show()
```



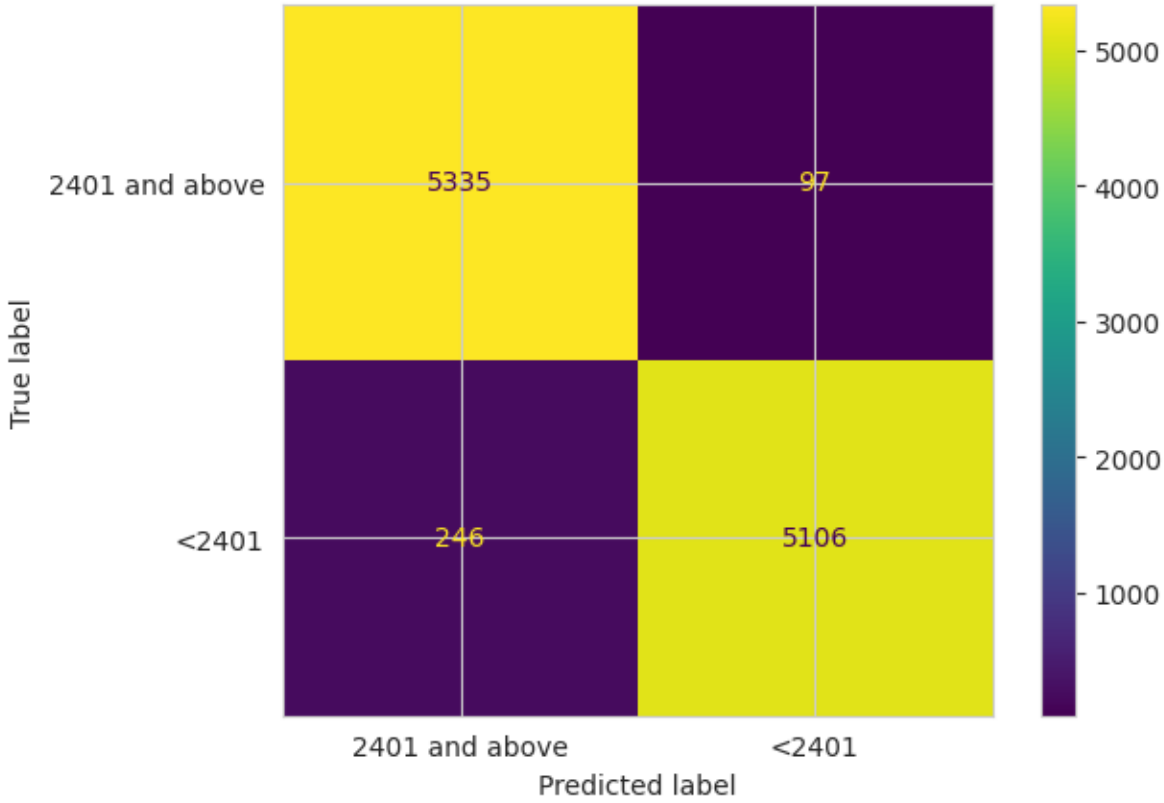
Heatmap:



KNN :

```
[[5335   97]
 [ 246 5106]]
accuracy =  0.9681936201780416
```

ConfusionMatrix



اما اشتباه این ایده در آنجا بود که : به هنگام گرفتن confusion matrix accuracy از خودش نشانه هایی از overfit شدن نشان میداد مخصوصا در روش Decision Tree

- **علاوه بر این ، بالاتر با رسم هیستوگرام برای داده هایمان متوجه شده بودیم که ستون قیمت شدت چولگی به راست دارد، یعنی بیشتر الماس‌ها قیمت کمتر از 2000 دارند و تعداد کمی از الماس‌ها با قیمت‌های بسیار بالا وجود دارند. پس ستون قیمت ما توزیع ناهمگون دارد.**

❖ ایده دوم برای دسته بندی مقادیر price

ایده دوم این بود که به جای دسته بندی عددی مقادیر price ، از چارک ها استفاده کنیم.

#fixing the problem with 'price' feature :

استفاده از صدک‌ها (Percentiles) برای دسته‌بندی

اگر در داده مان چولگی شدید وجود دارد، به جای تقسیم‌بندی داده‌ها بر اساس مقادیر مطلق (مانند تقسیم‌بندی به بازه‌های قیمتی ثابت)، می‌توانیم داده‌ها را بر اساس صدک‌ها دسته‌بندی کنیم تا گروه‌ها متعادل‌تر شوند:

ویژگی‌های استفاده از صدک‌ها برای دسته‌بندی:

1. **تعادل بهتر در تعداد نمونه‌ها در هر گروه:**
   - **در روش استفاده از صدک‌ها، داده‌ها بر اساس نسبت‌هایی از کل توزیع تقسیم می‌شوند. به این ترتیب، تعداد نمونه‌ها در هر گروه تقریباً یکسان می‌شود. این روش به ما اجازه داده می شود تا داده‌ها را به‌گونه‌ای دسته‌بندی کنیم که هر گروه حجم متعادلی از نمونه‌ها را در اختیار داشته باشد.**
   - **در روش تقسیم‌بندی با یک مقدار ثابت (مثلاً 2401)، اگر چولگی وجود داشته باشد، ممکن است یک گروه تعداد زیادی از داده‌ها را شامل شود و گروه دیگر بسیار کم باشد. به عنوان مثال، اگر بیشتر قیمت‌ها زیر 2401 باشند، گروه بالایی شامل تعداد کمی نمونه خواهد بود، که منجر به عدم تعادل و کاهش دقت تحلیل می‌شود.**
2. **کنترل بهتری روی چولگی و داده‌های افراطی:(Outliers)**
   - **صدک‌ها به‌طور خودکار چولگی و داده‌های افراطی را مدیریت می‌کنند. برای مثال، اگر قیمت‌ها به‌شدت، چولگی به راست داشته باشند (بیشتر داده‌ها در سمت پایین و تعداد کمی در سمت بالا)، دسته‌بندی بر اساس صدک‌ها می‌تواند این توزیع را متعادل کند و از تأثیر داده‌های پرت جلوگیری کند.**
   - **در مقابل، در روش تقسیم با مقادیر ثابت مانند 2401، داده‌های پرت (outliers) ممکن است باعث ایجاد گروه‌های نامتعادل شوند که تحلیل نتایج را مشکل می‌کند.**
3. **مقاومت در برابر توزیع غیر نرمال:**
   - **صدک‌ها به دلیل وابستگی به ترتیب داده‌ها، نیازی به توزیع نرمال ندارند. یعنی حتی اگر داده‌ها چوله یا به شدت نامتقارن باشند، صدک‌ها همچنان کارایی خود را حفظ می‌کنند. این در حالی است که تقسیم‌بندی با یک مقدار ثابت ممکن است برای داده‌های نرمال مناسب باشد، اما در مواجهه با توزیع‌های غیر نرمال (مانند داده‌های چوله) نتایج نادرستی بدهد.**

4. **انعطاف‌پذیری در تعریف گروه‌های مختلف:**

- **با استفاده از صدک‌ها می‌توان داده‌ها را به تعداد بیشتری از گروه‌ها یا طبقات تقسیم کرد که این تقسیم‌بندی می‌تواند انعطاف‌پذیری بیشتری در تحلیل داده‌ها ایجاد کند. مثلاً می‌توانیم داده‌ها را به چهار یا پنج گروه تقسیم کنیم (بیستک‌ها یا دهک‌ها) و هر گروه یک محدوده خاص از داده‌ها را پوشش دهد.**
- **در مقابل، تقسیم‌بندی با مقادیر ثابت، به یک روش خاص محدود است و انعطاف کمتری دارد.**

چرا صدک‌ها بهتر هستند نسبت به روش مقادیر مطلق (2401)؟

1. **عدم وابستگی به مقدار ثابت و دلخواه:**

- **در روش مقادیر مطلق، انتخاب مقدار ثابت مانند 2401 ممکن است به نوعی دلخواه باشد و لزوماً بر اساس ساختار توزیع داده‌ها صورت نگیرد. این می‌تواند منجر به نتایج نادرست یا عدم تعادل در داده‌ها شود. در حالی که صدک‌ها بر اساس درصدی از کل داده‌ها محاسبه می‌شوند و این باعث می‌شود دسته‌بندی طبیعی‌تر و دقیق‌تری از داده‌ها داشته باشیم.**

2. **گروه‌بندی‌های متعادل‌تر:**

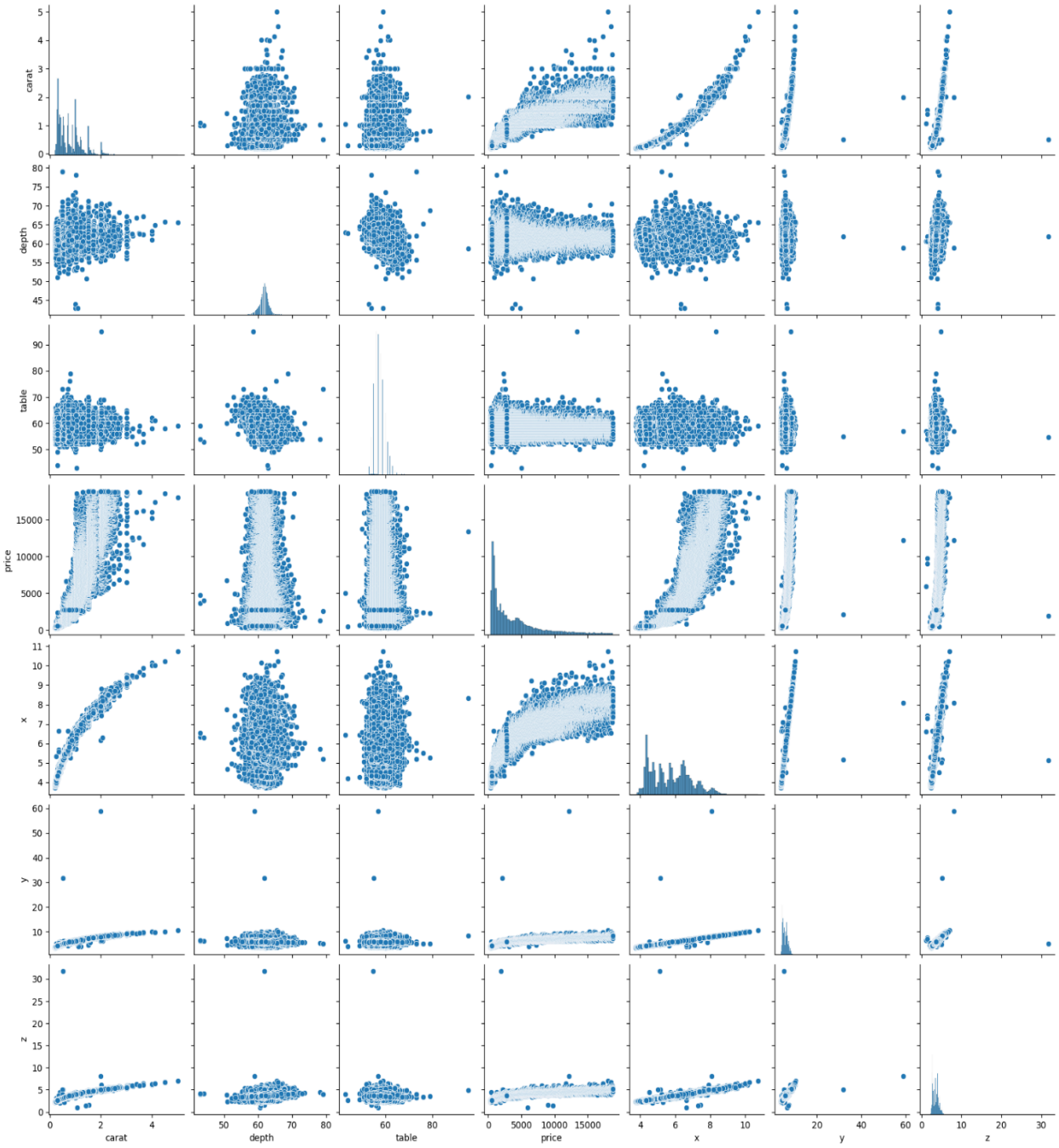
- **در روش صدک‌ها، هر گروه به‌صورت طبیعی بخشی مساوی از داده‌ها را در بر می‌گیرد، بنابراین می‌توانیم مطمئن باشیم که هر گروه تعداد مشابهی از داده‌ها را دارد. اما در روش 2401، ممکن است یک گروه بیش از حد بزرگ و گروه دیگر بسیار کوچک باشد که باعث ایجاد عدم تعادل در تحلیل می‌شود.**

```
3. data['price_category'] = pd.qcut(data['price'], q=4, labels=['Low',
   'Mid', 'High', 'Very High'])
4.
```

```
data.columns
Index(['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'price', 'x', 'y',
    'z', 'price_category'],
   dtype='object')
```

pair plot:

```
sns.pairplot(data)
plt.show()
```



نمودار **pair plot**، به منظور بررسی ارتباط متغیرها رسم شده است. در اینجا می‌توانیم همبستگی بین متغیرهای مختلف را مشاهده کنیم و ببینیم که آیا روابطی خطی یا غیرخطی بین متغیرهای مختلف وجود دارد یا خیر.

ارتباط بین **carat** و **price**

- در این نمودار یک رابطه مثبت و قوی بین وزن الماس **(carat)** و قیمت **(price)** مشاهده می‌شود. به این معنا که با افزایش وزن الماس، قیمت آن به طور قابل توجهی افزایش می‌یابد. این رابطه خطی است و به راحتی قابل مشاهده است.

ارتباط بین x, y, z (ابعاد الماس‌ها) و carat:

متغیرهای x, y، و z که ابعاد الماس‌ها را نشان می‌دهند، با متغیر carat رابطه‌ای قوی و مثبت دارند. این منطقی است، چون هر چه ابعاد الماس بزرگتر باشد، وزن آن نیز بیشتر می‌شود. این رابطه تقریباً خطی به نظر می‌رسد.

ارتباط بین table و price:

- **ارتباط بین table(اندازه سطح بالای الماس) و priceبسیار ضعیف‌تر است. به نظر نمی‌رسد که افزایش یا کاهش اندازه‌ی table تأثیر زیادی بر قیمت داشته باشد. نمودار نقاط پراکنده و بدون الگوی مشخص است.**

ارتباط بین depth و price:

- **مشابه table، رابطه قوی‌ای بین عمق (depth) و قیمت دیده نمی‌شود. به نظر می‌رسد که در بعضی محدوده‌های عمق، قیمت‌ها مشابه هستند و این نشان می‌دهد که عمق به تنهایی عامل تعیین‌کننده‌ای برای قیمت نیست.**

ارتباط بین carat و ابعاد:(x, y, z)

- **رابطه بین وزن الماس (carat) و ابعاد آن (x, y, z) بسیار قوی و خطی است. هرچه وزن افزایش پیدا کند، ابعاد نیز افزایش می‌یابد. این نشان‌دهنده این است که افزایش وزن مستقیماً به افزایش ابعاد منجر می‌شود.**

نکات مهم تحلیل:

همبستگی مثبت قوی:

بین carat وprice ، همچنین بین carat و ابعادx, y, z ، همبستگی مثبت و قوی وجود دارد که به صورت رابطه خطی دیده می‌شود.

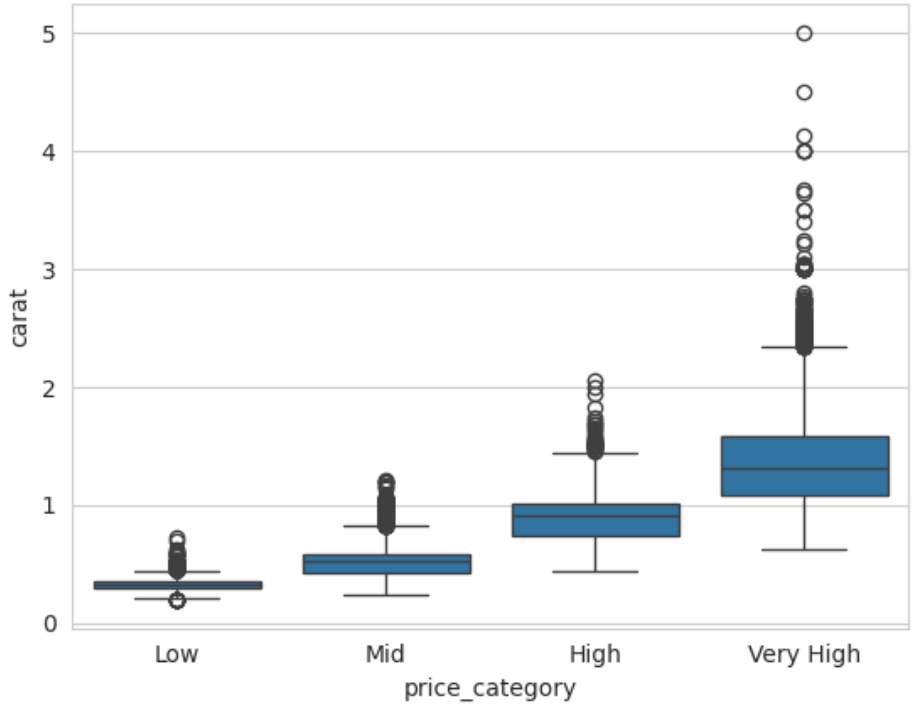
همبستگی ضعیف یا بدون رابطه:

متغیرهای depth و table همبستگی ضعیفی با قیمت دارند، به این معنی که این ویژگی‌ها به تنهایی تأثیر زیادی بر قیمت ندارند.

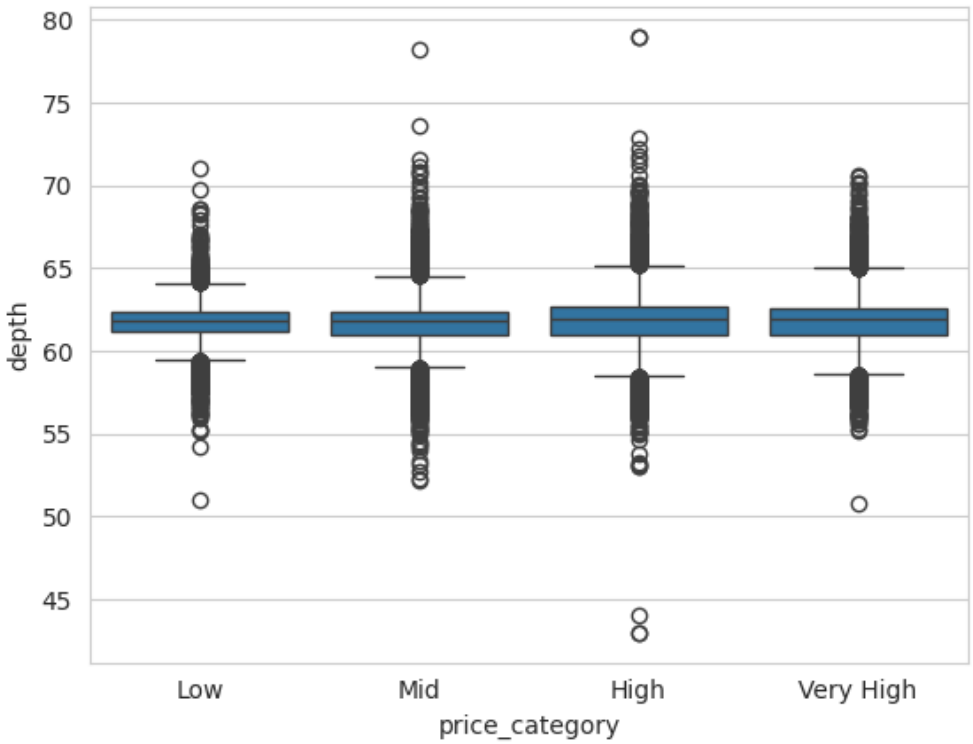
وجودoutlier ها:

- **در برخی از جفت‌ها، نقاط دور از نمودار اصلی (outliers) مشاهده می‌شود، که می‌تواند نشان‌دهنده داده‌های نادرست یا موارد نادر در دیتاست باشد. این داده‌ها می‌توانند تحلیل‌های آماری را تحت تأثیر قرار دهند و باید در صورت لزوم بررسی و حذف شوند.**
-

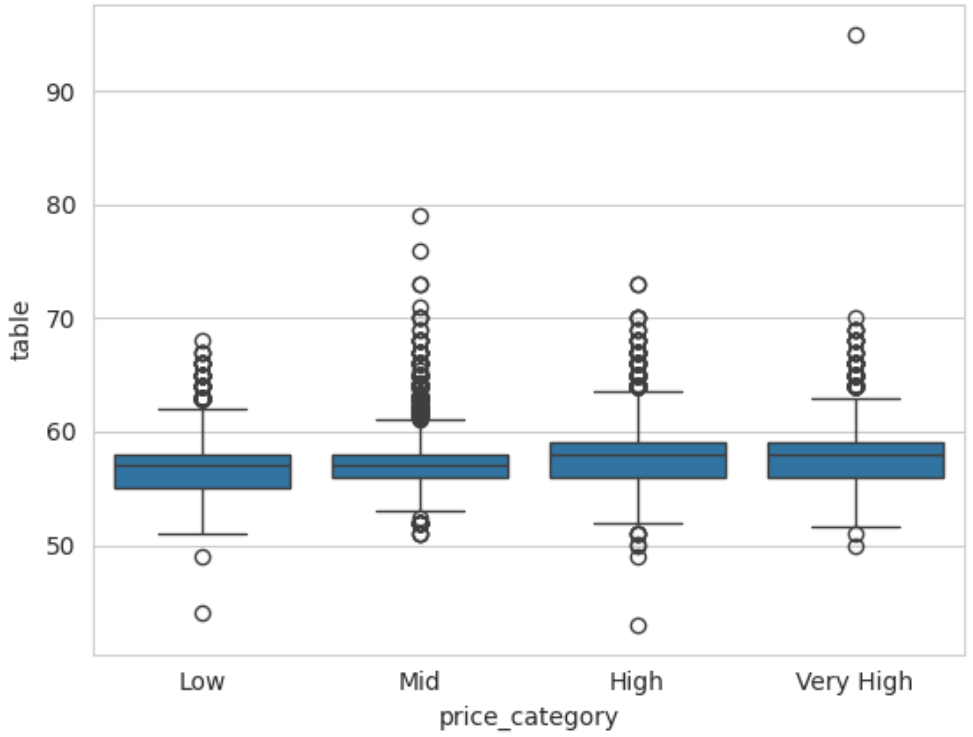
```
sns.boxplot(data, x='price_category', y='carat')
```




```
sns.boxplot(data, x='price_category', y='depth')
```

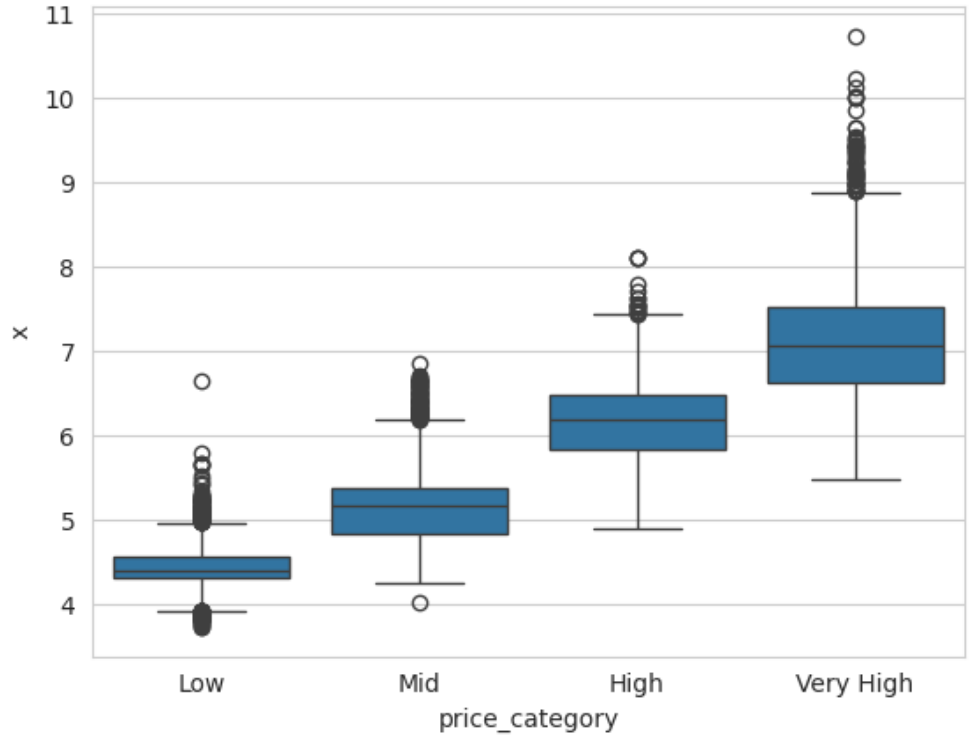



```
sns.set_style("whitegrid")
sns.boxplot(data, x='price_category', y='table')
```

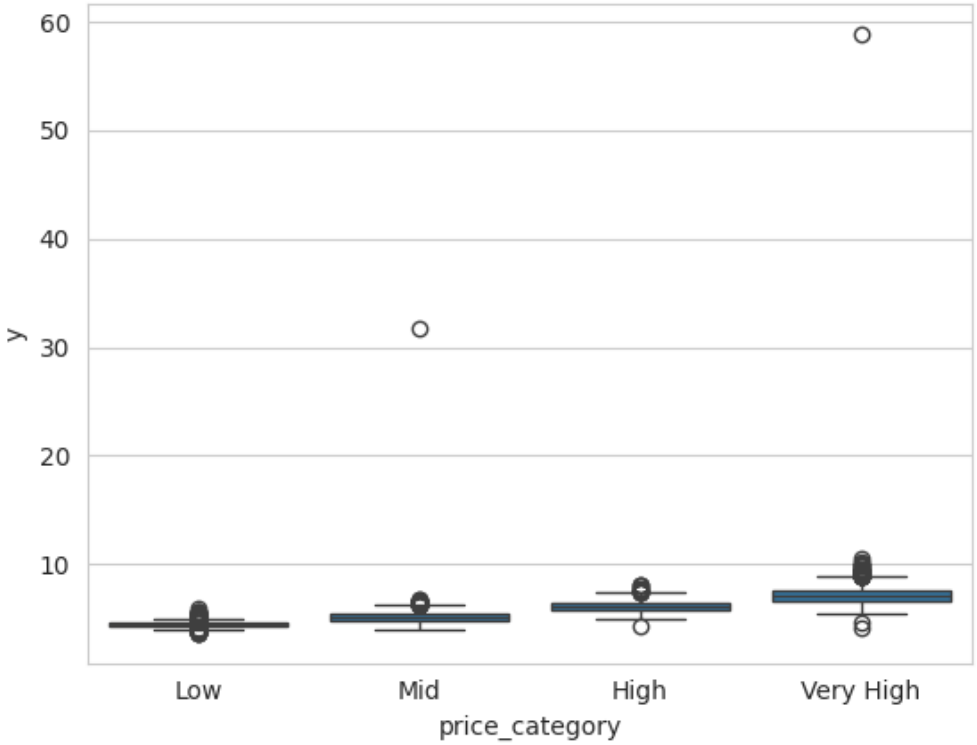



```
sns.set_style("whitegrid")
```

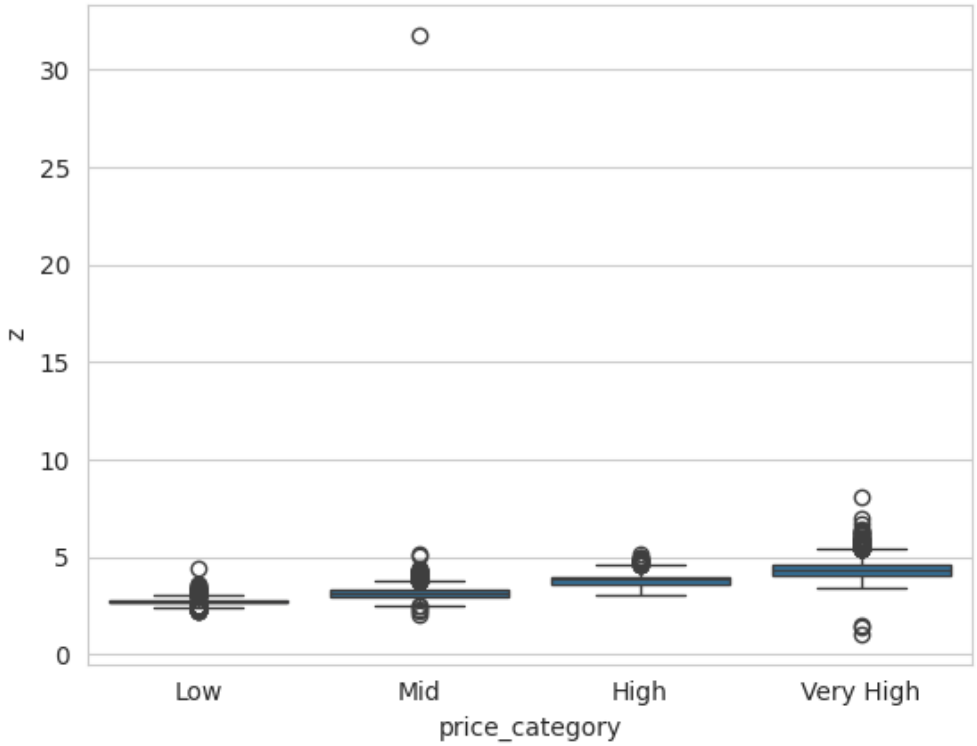
```
sns.boxplot(data, x='price_category', y='x')
```




```
sns.set_style("whitegrid")
sns.boxplot(data, x='price_category', y='y')
```



```
sns.set_style("whitegrid")
sns.boxplot(data, x='price_category', y='z')
```



همانطور که می بینیم بر اساس **(pair plot)**و **(box plot)**که قبلاً ارائه شد، "**x**" و "**carat**" دو معیار مفید برای ما هستند.

چرا "**x**" معیار مفیدی است:

- در **(pair plot)**، همبستگی بین **x**(طول الماس بر حسب میلی‌متر) و قیمت به وضوح مشاهده می‌شود. با افزایش مقدار **x**، قیمت الماس نیز معمولاً افزایش می‌یابد. این موضوع نشان می‌دهد که اندازه‌ی الماس (که با **x**نشان داده می‌شود) در تعیین قیمت آن نقش مهمی دارد.

چرا "**carat**" معیار مفیدی است:

- نمودار جعبه‌ای که ارائه کردیم، به وضوح نشان می‌دهد که **carat**(قیراط یا وزن الماس) رابطه قوی با دسته‌بندی قیمت دارد. هرچه قیراط الماس بیشتر باشد، آن الماس به دسته‌های قیمتی بالاتری تعلق می‌گیرد.
- همچنین، در **pair plot** هم ارتباط میان **carat**و قیمت به وضوح قابل مشاهده است. الماس‌های با قیراط بیشتر معمولاً با قیمت‌های بالاتری همراه هستند.

Scatter plot :

colors={'Low':'blue','Mid':'green','High':'red','VeryHigh':'purple'}

forcategory,colorincolors.items():

subset=data[data['price_category']==category]

plt.scatter(subset['carat'],subset['price'],label=category,color=color)

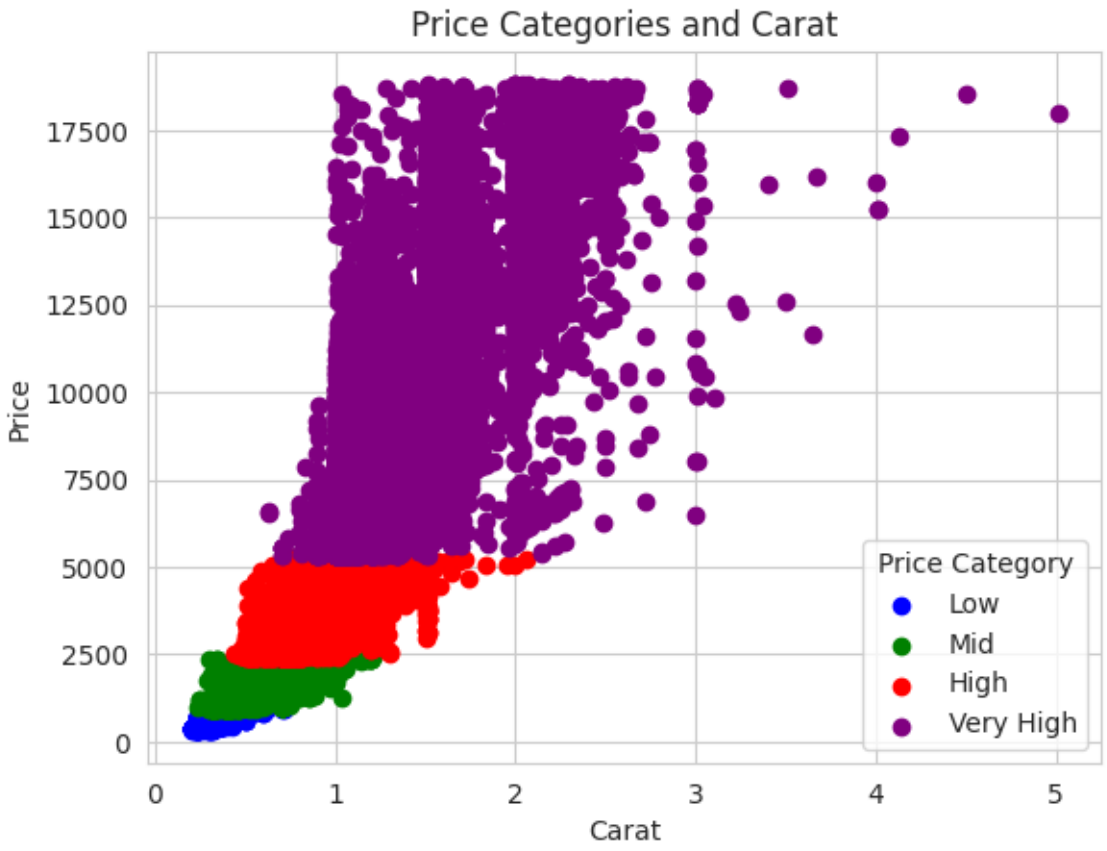
plt.xlabel('Carat')

plt.ylabel('Price')

plt.title('PriceCategoriesandCarat')

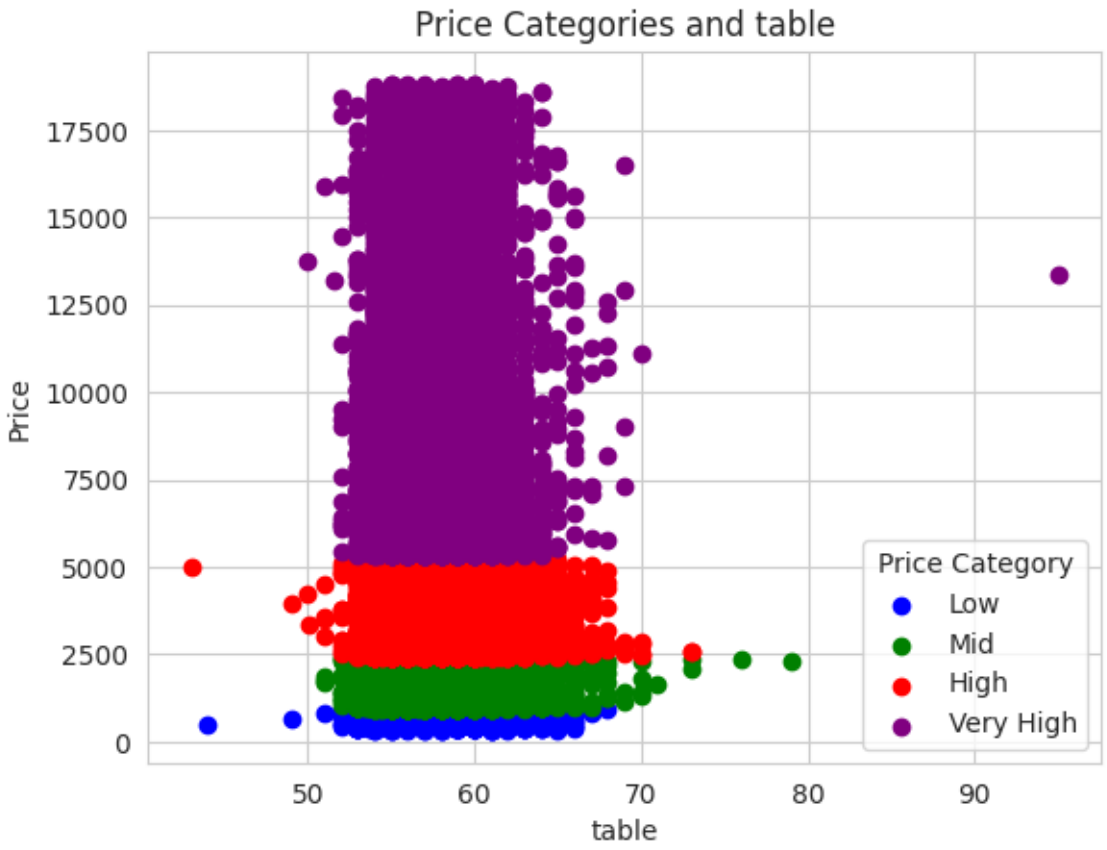
plt.legend(title='PriceCategory')

plt.show()




```
colors = {'Low': 'blue', 'Mid': 'green', 'High': 'red', 'Very High':
'purple'}

for category, color in colors.items():
```

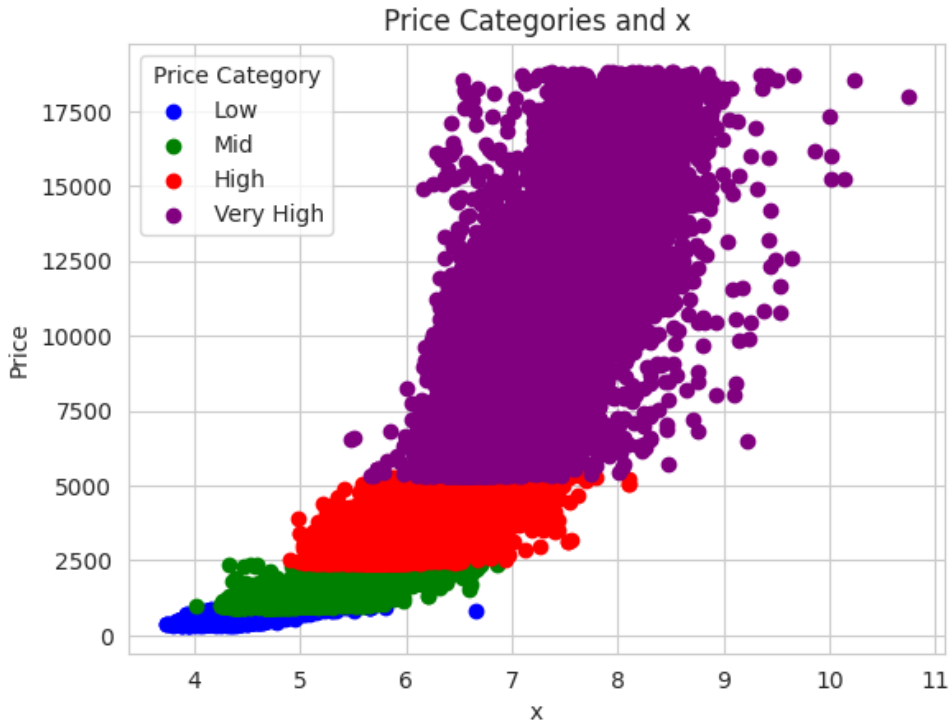
```
    subset = data[data['price_category'] == category]
    plt.scatter(subset['table'], subset['price'], label=category,
color=color)

plt.xlabel('table')
plt.ylabel('Price')
plt.title('Price Categories and table')
plt.legend(title='Price Category')

plt.show()
```




```
colors = {'Low': 'blue', 'Mid': 'green', 'High': 'red', 'Very High':
'purple'}

for category, color in colors.items():
    subset = data[data['price_category'] == category]
    plt.scatter(subset['y'], subset['price'], label=category, color=color)

plt.xlabel('y')
plt.ylabel('Price')
plt.title('Price Categories and y')
```

```
plt.legend(title='Price Category')

plt.show()
```

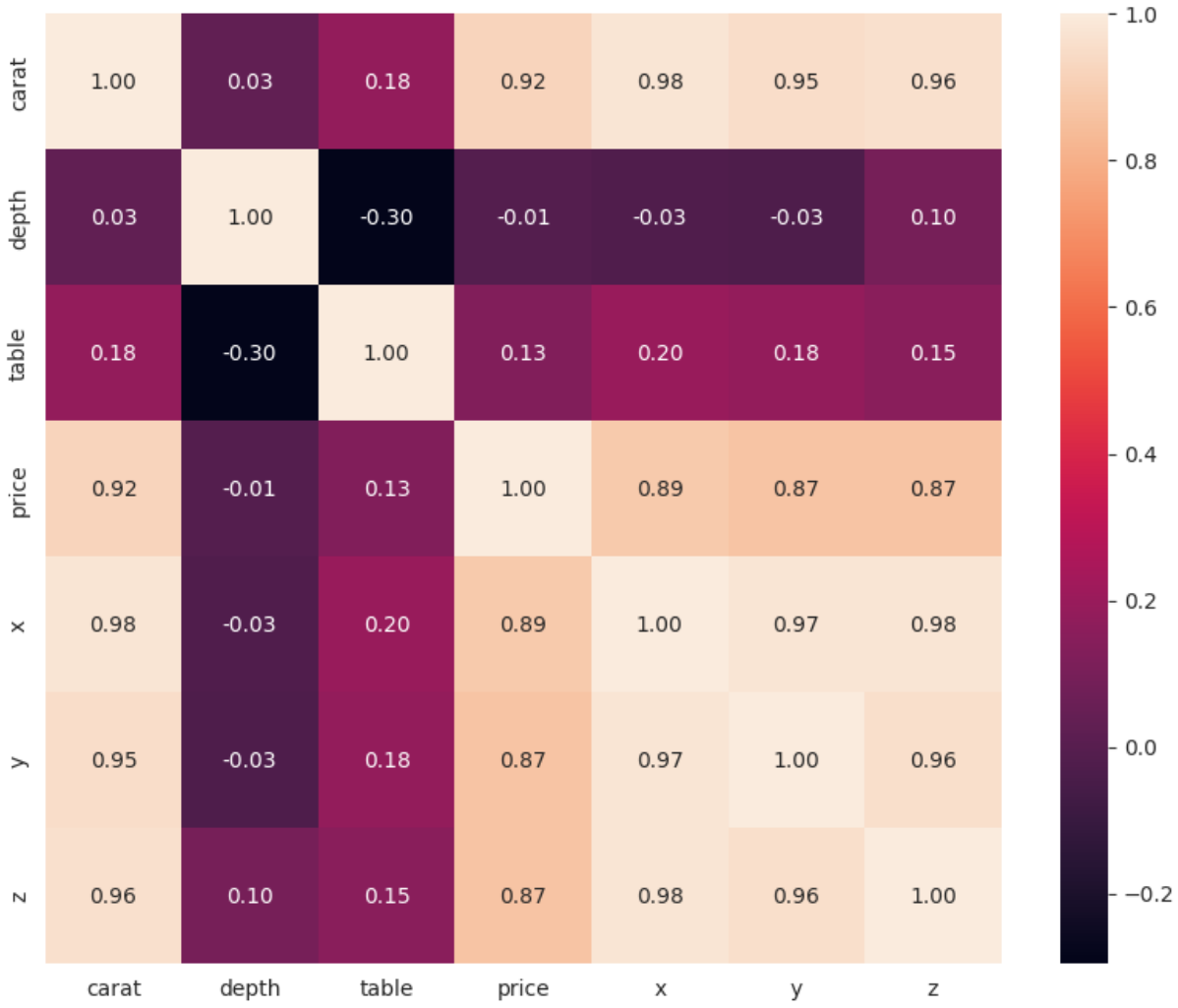

```
colors = {'Low': 'blue', 'Mid': 'green', 'High': 'red', 'Very High':
'purple'}

for category, color in colors.items():
    subset = data[data['price_category'] == category]
    plt.scatter(subset['x'], subset['price'], label=category, color=color)

plt.xlabel('x')
plt.ylabel('Price')
plt.title('Price Categories and x')
plt.legend(title='Price Category')

plt.show()
```




```
colors = {'Low': 'blue', 'Mid': 'green', 'High': 'red', 'Very High':
'purple'}

for category, color in colors.items():
    subset = data[data['price_category'] == category]
    plt.scatter(subset['z'], subset['price'], label=category, color=color)

plt.xlabel('z')
plt.ylabel('Price')
plt.title('Price Categories and z')
plt.legend(title='Price Category')

plt.show()
```

نمودار Heatmap نشان‌دهنده‌ی همبستگی بین متغیرهای مختلف یک مجموعه داده است. در اینجا هر سلول از ماتریس به یک ضریب همبستگی (بین −1 تا +1) اشاره دارد که رابطه‌ی بین دو متغیر را مشخص می‌کند. رنگ‌های تیره‌تر (مثل بنفش) نشان‌دهنده‌ی همبستگی‌های کمتر یا حتی منفی هستند، در حالی که رنگ‌های روشن‌تر (مثل نارنجی یا سفید) نشان‌دهنده‌ی همبستگی‌های مثبت قوی‌تر هستند .

تفسیر : Heatmap




```
data.head(5)
```

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

```
le = LabelEncoder()
data['cut'] = le.fit_transform(data['cut'])
data['color'] = le.fit_transform(data['color'])
data['clarity'] = le.fit_transform(data['clarity'])
data.head()
```

	carat	cut	color	clarity	depth	table	price	x	y	z	price_category
0	0.23	2	1	3	61.5	55.0	326	3.95	3.98	2.43	Low
1	0.21	3	1	2	59.8	61.0	326	3.89	3.84	2.31	Low
2	0.23	1	1	4	56.9	65.0	327	4.05	4.07	2.31	Low
3	0.29	3	5	5	62.4	58.0	334	4.20	4.23	2.63	Low
4	0.31	1	6	3	63.3	58.0	335	4.34	4.35	2.75	Low

از **LabelEncoder**برای تبدیل ویژگی‌های متنی (دسته‌ای) به مقادیر عددی استفاده می‌کند. در اینجا، ستون‌های **'cut'**، **'color'**و **'clarity'**از دیتافریم **data**به صورت عددی تبدیل می‌شوند.

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	2	1	3	61.5	55.0	326	3.95	3.98	2.43
1	0.21	3	1	2	59.8	61.0	326	3.89	3.84	2.31
2	0.23	1	1	4	56.9	65.0	327	4.05	4.07	2.31
3	0.29	3	5	5	62.4	58.0	334	4.20	4.23	2.63
4	0.31	1	6	3	63.3	58.0	335	4.34	4.35	2.75

```
x = data.iloc[:,0:10]
x.head(5)
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	2	1	3	61.5	55.0	326	3.95	3.98	2.43
1	0.21	3	1	2	59.8	61.0	326	3.89	3.84	2.31
2	0.23	1	1	4	56.9	65.0	327	4.05	4.07	2.31
3	0.29	3	5	5	62.4	58.0	334	4.20	4.23	2.63
4	0.31	1	6	3	63.3	58.0	335	4.34	4.35	2.75

```
y = data.iloc[:,10]
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2,
random_state = 1)
```

x_train: داده‌های ورودی برای مجموعه آموزش.

x_test: داده‌های ورودی برای مجموعه تست.

y_train: برچسب‌ها یا هدف‌ها برای مجموعه آموزش.

y_test: برچسب‌ها یا هدف‌ها برای مجموعه تست.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
print(x_train)
```

```
0.38888777- 0.42023876- ... 0.93741257- 0.54107221- 0.52156219-]]
[0.43141975-
1.11021871  1.17808884  ... 1.52467046- 0.43303843  1.06017528 ]
[0.99838758
0.81845629- 0.69704409- ... 0.82436108  0.43303843  0.77464018-]
[0.74915471-
...
0.85352311- 0.89348659- ... 0.35015469- 0.54107221- 0.83790968-]
[0.86469469-
0.25984837  0.35660203  ... 1.99887684  0.43303843  0.21658197 ]
[0.40624515
0.92611791  0.93700032  ... 0.93741257- 1.40714907  0.87036679 ]
[[1.05615757
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier
KNN_classifier = KNeighborsClassifier(n_neighbors = 5, metric =
'minkowski', p = 2)
KNN_classifier.fit(x_train, y_train)
```

ایجاد مدل:KNN

در اینجا، یک شیء از نوع KNeighborsClassifier **به نام** KNN_classifier **ایجاد می‌کنیم. پارامترهای زیر به مدل پاس داده می‌شوند:**

n_neighbors = 5: **این پارامتر تعداد همسایه‌های نزدیک برای مدل** KNN **را مشخص می‌کند. در اینجا، مدل برای پیش‌بینی هر نقطه داده جدید،** 5 **نزدیک‌ترین همسایه را در نظر می‌گیرد.**

metric='minkowski': **این پارامتر متریک فاصله مورد استفاده را تعیین می‌کند. در اینجا از متریک مینکوفسکی استفاده می‌شود که یک متریک عمومی برای محاسبه فاصله است.**

p=2: **این پارامتر در صورتی که** metric='minkowski' **باشد، به مدل می‌گوید که از چه نوع فاصله‌ای استفاده کند. زمانی که** p=2 **است، مینکوفسکی به فاصله اقلیدسی تبدیل می‌شود (معمول‌ترین متریک فاصله).**

آموزش مدل :KNN

این خط مدل KNN **را با استفاده از داده‌های آموزشی** x_train **و** y_train **آموزش می‌دهد.**

- x_train: **داده‌های ورودی آموزش، که شامل ویژگی‌ها** (features) **است.**
- y_train: **برچسب‌های هدف مرتبط با داده‌های ورودی، که مدل بر اساس آن‌ها آموزش می‌بیند.**

```
KNeighborsClassifier?i
KNeighborsClassifier(n_neighbors=5)
```

```
y_pred = KNN_classifier.predict(x_test)
y_pred
```

```
array(['budget', 'mid_range', 'luxury', ..., 'mid_range', 'mid_range',
       'mid_range'], dtype=object)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
print("accuracy = ", accuracy_score(y_test,y_pred))
```

```
[[2487    0  142  102]
 [   0 2523  182    0]
 [ 164  256 2244    2]
 [ 148    0    0 2534]]
accuracy =  0.9076409495548962
```

```
from sklearn.metrics import ConfusionMatrixDisplay
y = KNN_classifier.fit(x_train, y_train)
ConfusionMatrixDisplay.from_estimator(y, x_test, y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x781172509cf0>
```

در الگوریتم‌های KNN ، پارامتر metric تعیین می‌کند که چگونه فاصله بین نقاط داده محاسبه شود. به غیر از 'minkowski' ، برخی از متریک‌های معمول برای محاسبه فاصله شامل موارد زیر هستند:

- **'euclidean': فاصله اقلیدسی را محاسبه می‌کند (L2 norm) که فاصله معمولی بین دو نقطه است.**

فرمول:

$$\sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

این یک حالت خاص از فاصله Minkowski است که در آن p=2p = 2p=2.

- **'manhattan':که به آن فاصله L1 نیز گفته می‌شود. مجموع تفاوت‌های مطلق بین مختصات را اندازه‌گیری می‌کند.**

فرمول:

$$\sum_{i=1}^{n}|y_i - x_i|$$

- **'hamming':برای داده‌های دسته‌ای (categorical) استفاده می‌شود و تعداد موقعیت‌هایی که عناصر متناظر متفاوت هستند را شمارش می‌کند.**
- **'cosine':زاویه بین دو بردار غیر صفر را اندازه‌گیری می‌کند. اغلب در داده‌های متنی یا فضاهای با ابعاد بالا استفاده می‌شود.**

فرمول:

$$\frac{xy}{\|x\|\|y\|} - 1$$

- **'mahalanobis':فاصله بین یک نقطه و توزیع داده‌ها را اندازه‌گیری می‌کند و به همبستگی بین متغیرها توجه دارد.**

برای محاسبه این متریک به ماتریس کوواریانس نیاز است.

Hamming Metric :

```
KNN_classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'hamming')
KNN_classifier.fit(x_train, y_train)
```

```
KNeighborsClassifier?i
KNeighborsClassifier(metric='hamming')
```

```
y_predict = KNN_classifier.predict(x_test)
y_predict
array(['Low', 'Mid', 'Low', ..., 'Very High', 'Low', 'Low'], dtype=object)
cm = confusion_matrix(y_test, y_predict)
```

```
print(cm)
print("accuracy = ", accuracy_score(y_test,y_predict))
```

```
[[2069  274  282  106]
 [   5 2532  168    0]
 [ 204  286 2173    3]
 [ 427  662  429 1164]]
accuracy =  0.7360905044510386
```

Mahalanobis:

```
cov_matrix = np.cov(x_train, rowvar=False)
inv_cov_matrix = np.linalg.inv(cov_matrix)

KNN_classifier = KNeighborsClassifier(n_neighbors=5, metric='mahalanobis',
metric_params={'V': inv_cov_matrix})
KNN_classifier.fit(x_train, y_train)
```

KNeighborsClassifier?i

```
KNeighborsClassifier(metric='mahalanobis',
                     metric_params={'V': array([[ 3.96955439e+01,
8.87325731e-02, -1.92135166e+00,
         1.19224953e+00, -7.85055705e-01, -7.87580774e-01,
        -1.13353241e+01, -1.86549578e+01,  8.55440720e-01,
        -1.01031307e+01],
       [ 8.87325731e-02,  1.05624054e+00, -1.93644063e-02,
        -1.15044078e-02,  2.38254862e-01, -1.20118226e-01,
        -1.61798096e-01,  6.95555762e-01...
         1.76009419e+00,  2.71864294e+02,  1.42008279e+01,
        -2.69556790e+02],
       [ 8.55440720e-01, -4.98630544e-02, -5.46263082e-02,
         3.00685361e-02,  4.93154208e+00,  6.41233923e-02,
        -3.09176542e-01,  1.42008279e+01,  2.57439424e+01,
        -4.01166239e+01],
       [-1.01031307e+01, -5.87731926e-01,  3.60256222e-01,
        -3.43827298e-01, -3.79402313e+01,  1.03186247e+00,
         1.44462151e+00, -2.69556790e+02, -4.01166239e+01,
         3.18956684e+02]])})
```

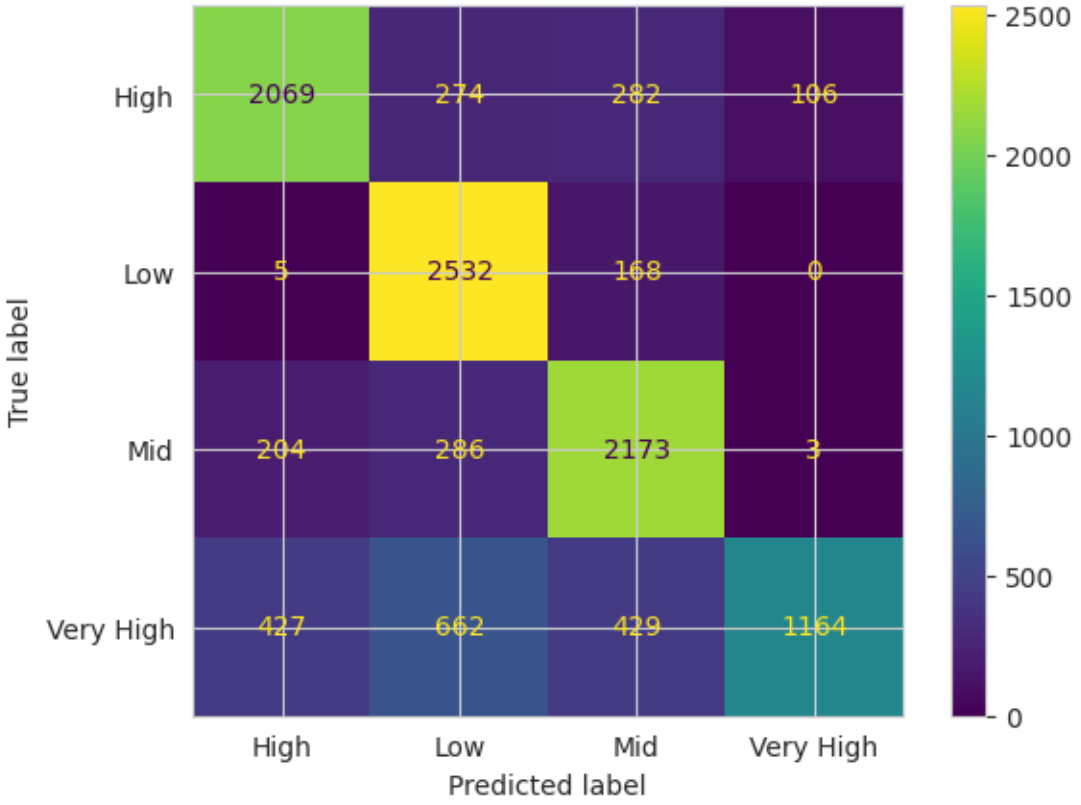
```
cm = confusion_matrix(y_test, y_predict)
print(cm)
print("accuracy = ", accuracy_score(y_test,y_predict))
```

```
[[2491    0  124  116]
 [   0 2496  209    0]
 [ 144  237 2283    2]
 [ 124    0    0 2558]]
```

```
accuracy =  0.9113501483679525
```

نتیجه : بهترین **metric** با بالاترین دقت = **91.13٪** **Mahalanobis**

```
from sklearn.metrics import ConfusionMatrixDisplay
y = KNN_classifier.fit(x_train, y_train)
ConfusionMatrixDisplay.from_estimator(y, x_test, y_test)
```




```
knn_classifier = KNeighborsClassifier(n_neighbors=5)

knn_classifier.fit(x_train, y_train)
y_pred_knn = knn_classifier.predict(x_test)

print(classification_report(y_test, y_pred_knn))
```

```
              precision    recall  f1-score   support

        High       0.89      0.91      0.90      2731
         Low       0.91      0.93      0.92      2705
         Mid       0.87      0.84      0.86      2666
   Very High       0.96      0.94      0.95      2682

    accuracy                           0.91     10784
   macro avg       0.91      0.91      0.91     10784
weighted avg       0.91      0.91      0.91     10784
```

Decision tree

```
diamondTree = DecisionTreeClassifier(criterion = "gini" , max_depth =
None)
model = diamondTree.fit(x_train, y_train)
y_predict = diamondTree.predict(x_test)
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_predict)
accuracy_score(y_test, y_predict)
1.0
```

1.ساخت مدل درخت تصمیم:

- **criterion="gini"** از معیار **Gini** برای تقسیم داده‌ها در هر گره استفاده می‌کند (که یکی از معیارهای معروف برای انتخاب بهترین تقسیم است).
- **max_depth=None** به این معنا است که عمق درخت هیچ محدودیتی ندارد و تا جایی که بتواند داده‌ها را کامل تقسیم کند، پیش می‌رود.

fit(x_train, y_train): این خط مدل را با استفاده از داده‌های آموزشی **x_train** و **y_train** آموزش می‌دهد.

2.پیش‌بینی بر روی داده‌های تست:

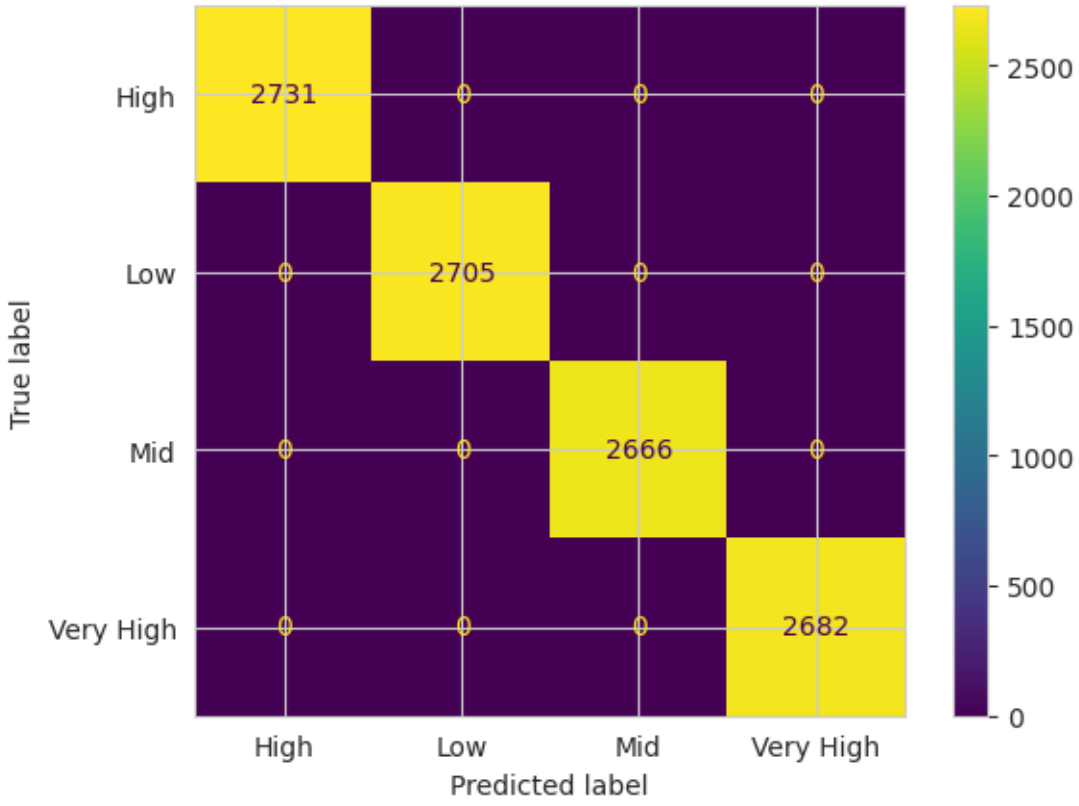
- **predict(x_test):** با استفاده از مدل آموزش‌دیده، مقادیر **y_test** پیش‌بینی می‌شود.

3.محاسبه دقت و confusion matrix:

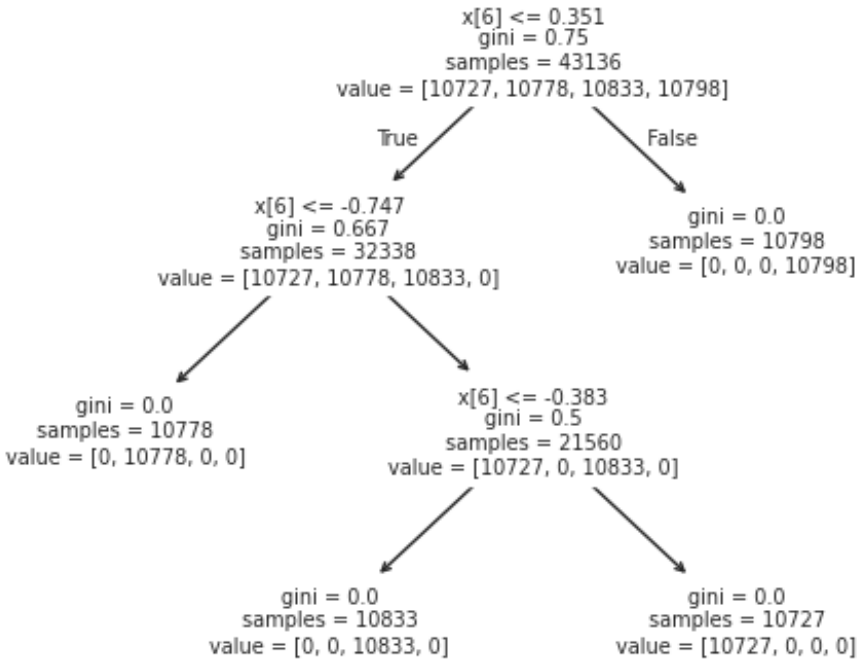
accuracy_score(y_test, y_predict): دقت مدل محاسبه می‌شود و برابر با **1.0 (100%)** است.

```
from sklearn.metrics import ConfusionMatrixDisplay
```

```
ConfusionMatrixDisplay.from_estimator(model, x_test, y_test)
```




```
from sklearn import tree
tree.plot_tree(model)
plt.show()
```



تحلیل درخت تصمیم :

برای تفسیر این درخت، به هر گره (نود) نگاه می‌کنیم که چگونه داده‌ها را بر اساس یک ویژگی و مقدار خاص تقسیم می‌کند.

توضیحات گره‌ها:

1. **گره ریشه:(Root Node)**
   - **شرط x[6] <= 0.351 :**
   - **تعداد نمونه‌ها: 43136**
   - **شاخص جینی (gini) برای این گره: 0.75 (هرچه شاخص جینی بزرگ‌تر باشد، نشان‌دهنده ناهمگونی بیشتر است).**
   - **توزیع نمونه‌ها [10727, 10778, 10833, 10798] :این توزیع نشان می‌دهد که چگونه نمونه‌ها در چهار کلاس یا گروه تقسیم شده‌اند.**

2. **شاخه چپ:(True)**
   - **شرط: x[6] <= -0.747**
   - **تعداد نمونه‌ها: 32338**
   - **شاخص جینی: 0.667 (مقداری کمتر از گره ریشه، بنابراین این گروه داده‌ها تا حدودی همگن‌تر هستند).**
   - **توزیع نمونه‌ها: [10727, 10778, 10833, 0] اینجا می‌بینیم که تمام نمونه‌ها در سه کلاس اول هستند و هیچ نمونه‌ای در کلاس چهارم نیست.**
3. **شاخه راست:(False)**
   - **شرط: x[6] > 0.351**
   - **تعداد نمونه‌ها: 10798**
   - **شاخص جینی: 0.0 (تمام نمونه‌ها به طور کامل در یک کلاس هستند).**
   - **توزیع نمونه‌ها: [0, 0, 0, 10798] همه نمونه‌ها به کلاس چهارم تعلق دارند.**
4. **گره نهایی:(Leaf Nodes)**
   - **وقتی به گره‌های نهایی (با شاخص جینی 0) می‌رسیم، داده‌ها به طور کامل به یک کلاس خاص تخصیص یافته‌اند. به عنوان مثال:**
     - **شاخه چپ نهایی با 10778 نمونه همگی در کلاس دوم قرار دارند.**
     - **شاخه راست با 10833 نمونه همگی در کلاس سوم قرار دارند.**

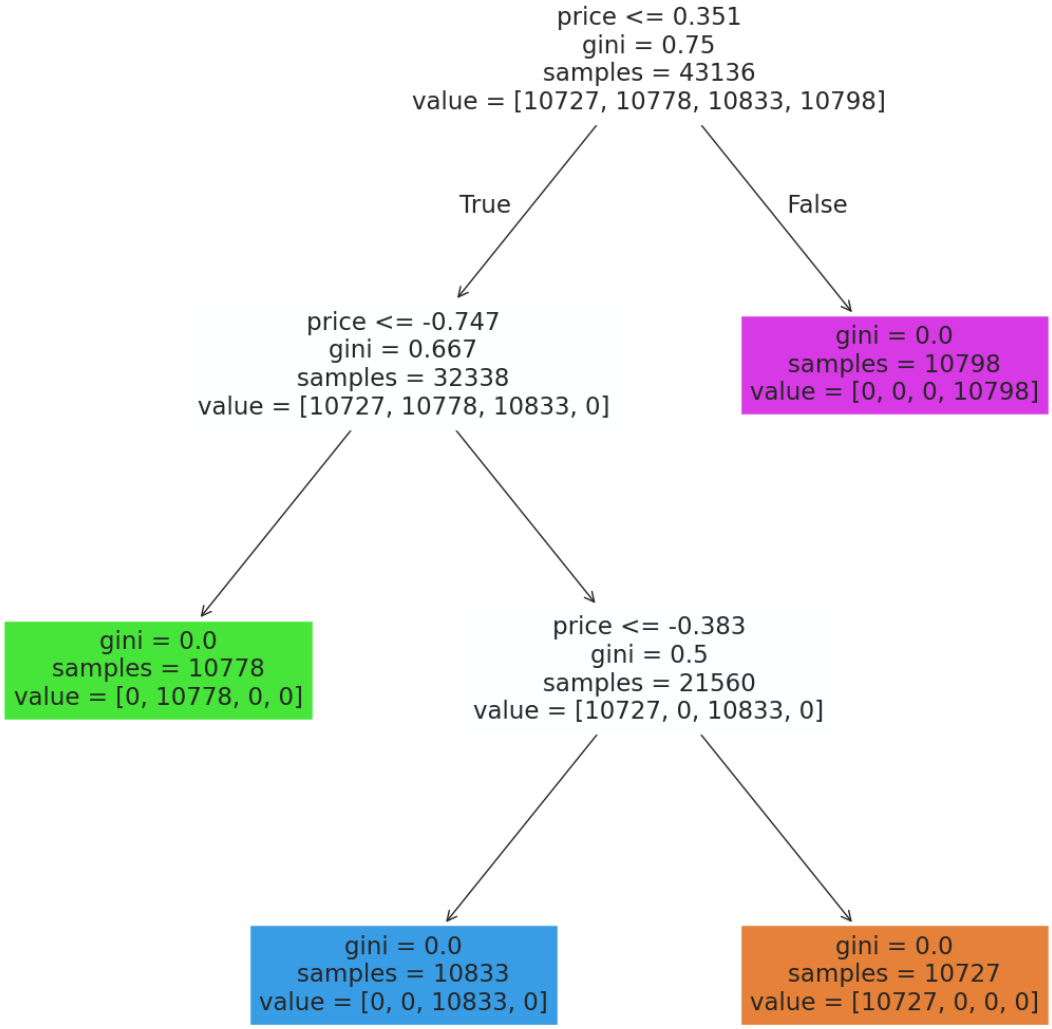
نتیجه‌گیری:

- **در هر مرحله از درخت تصمیم، الگوریتم داده‌ها را بر اساس ویژگی‌های مختلف) در اینجا (x[6] تقسیم می‌کند تا نمونه‌ها را به گروه‌هایی که بیشترین تشابه را با یکدیگر دارند (کمترین مقدار شاخص جینی) تخصیص دهد.**
- **در انتها، شاخه‌هایی که به گره‌های نهایی ختم می‌شوند نشان‌دهنده گروه‌هایی هستند که داده‌ها کاملاً همگن شده‌اند و تمام نمونه‌های آن‌ها به یک کلاس تعلق دارند.**

```
decision_tree = DecisionTreeClassifier(random_state=42)
decision_tree.fit(x_train, y_train)

y_pred = decision_tree.predict(x_test)

print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9997
Confusion Matrix:
[[2730    0    1    0]
 [   0 2703    2    0]
 [   0    0 2666    0]
 [   0    0    0 2682]]
Classification Report:
              precision    recall  f1-score   support

        High       1.00      1.00      1.00      2731
         Low       1.00      1.00      1.00      2705
         Mid       1.00      1.00      1.00      2666
   Very High       1.00      1.00      1.00      2682

    accuracy                           1.00     10784
   macro avg       1.00      1.00      1.00     10784
weighted avg       1.00      1.00      1.00     10784
```

```
      fig = plt.figure(figsize=(15,15))

_ = tree.plot_tree(model,
                   feature_names= x.columns,
                   filled=True)
```



سبزها MID ، ابی ها LOW، قرمز ها HIGH ، بنفش ها همان VERY HIGH هستند.

```
      from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators = 100, criterion =
'gini', random_state = 42)
classifier.fit(x_train, y_train)
```

RandomForestClassifier?i

```
RandomForestClassifier(random_state=0)
```

```
y_predict = classifier.predict(x_test)
cm = confusion_matrix(y_test, y_predict)
accuracy_score(y_test, y_predict)
```

0.9997218100890207

Logistic Regression:

```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(max_iter=1000).fit(x_train,y_train)
y_predict = LR.predict(x_test)
y_predict
```

```
array(['budget', 'mid_range', 'luxury', ..., 'mid_range', 'mid_range',
       'mid_range'], dtype=object)
```

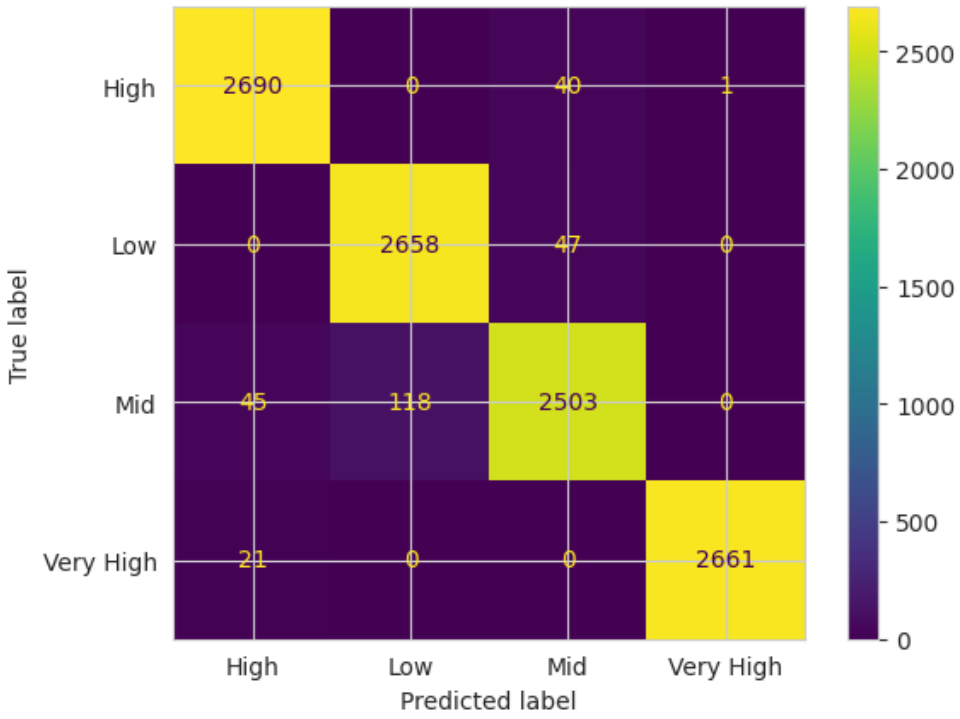
```
y_Predict_prob = LR.predict_proba(x_test)
y_Predict_prob
```

```
array([[1.05040015e-07, 9.19828271e-01, 8.01716241e-02, 2.15001055e-19],
       [2.27217350e-10, 1.40430968e-65, 2.33931382e-27, 1.00000000e+00],
       [2.09974213e-16, 1.49118298e-88, 3.88514528e-39, 1.00000000e+00],
       ...,
       [1.53996882e-09, 2.94153208e-64, 4.01134969e-26, 9.99999998e-01],
       [2.24523036e-09, 9.94883698e-01, 5.11629968e-03, 1.07082402e-21],
       [2.99071865e-02, 5.41721002e-05, 9.70038641e-01, 2.39376672e-11]])
```

```
from sklearn.metrics import accuracy_score
print("accuracy = ", accuracy_score(y_test,y_predict))
```

```
accuracy =  0.9747774480712166
```

```
from sklearn.metrics import ConfusionMatrixDisplay
y = LogisticRegression(max_iter=500).fit(x_train, y_train)
ConfusionMatrixDisplay.from_estimator(y, x_test, y_test)
```



SVM:

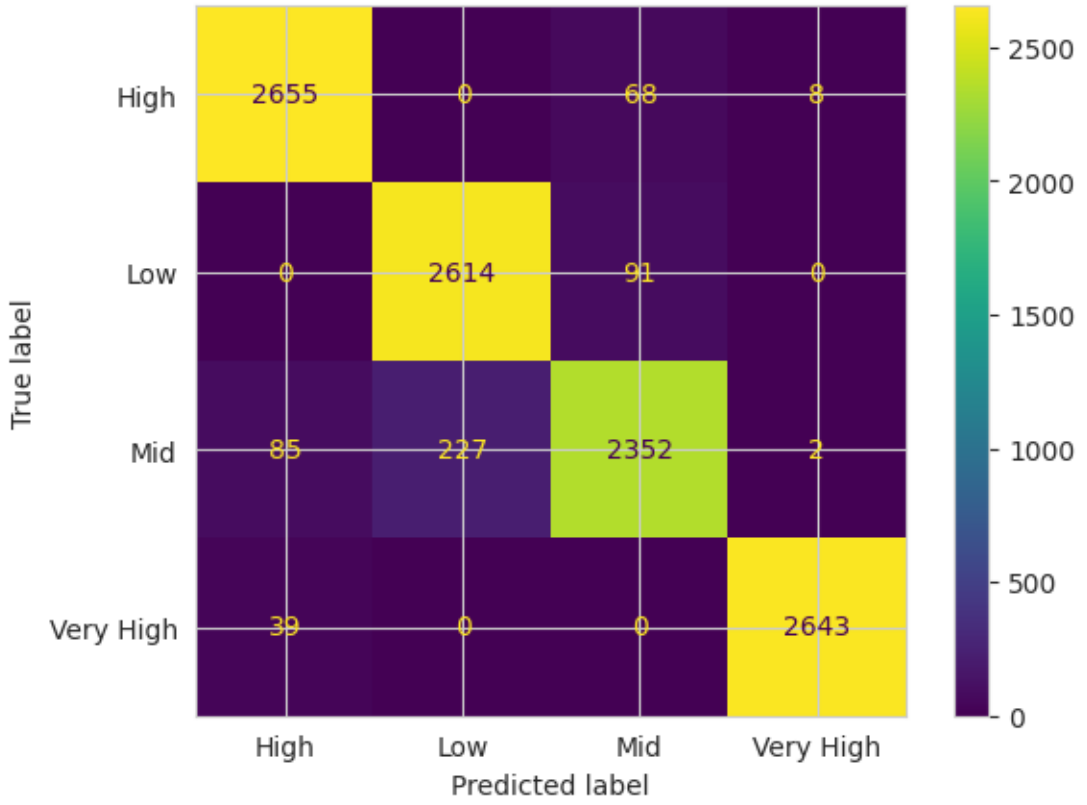
```
from sklearn import svm
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
ksvm = svm.SVC(kernel='rbf',
               gamma=0.1,
               C=1.0)
ksvm.fit(x_train, y_train)

accuracy = ksvm.score(x_test, y_test)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.951780415430267
```

```
from sklearn.metrics import classification_report

y_pred = ksvm.predict(x_test)
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
High	0.96	0.97	0.96	2731
Low	0.92	0.97	0.94	2705
Mid	0.94	0.88	0.91	2666
Very High	1.00	0.99	0.99	2682
accuracy			0.95	10784
macro avg	0.95	0.95	0.95	10784
weighted avg	0.95	0.95	0.95	10784

```
from sklearn.metrics import ConfusionMatrixDisplay


ConfusionMatrixDisplay.from_estimator(ksvm, x_test, y_test)
```




```
ksvm = svm.SVC(kernel='linear',
               C=1.0)

ksvm.fit(x_train, y_train)

accuracy = ksvm.score(x_test, y_test)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.9896142433234422
```

```
from sklearn.model_selection import cross_val_score


cv_scores = cross_val_score(ksvm, x_train, y_train, cv=5)
print("Cross-validation scores:", cv_scores)
print("Mean cross-validation score:", cv_scores.mean())
```

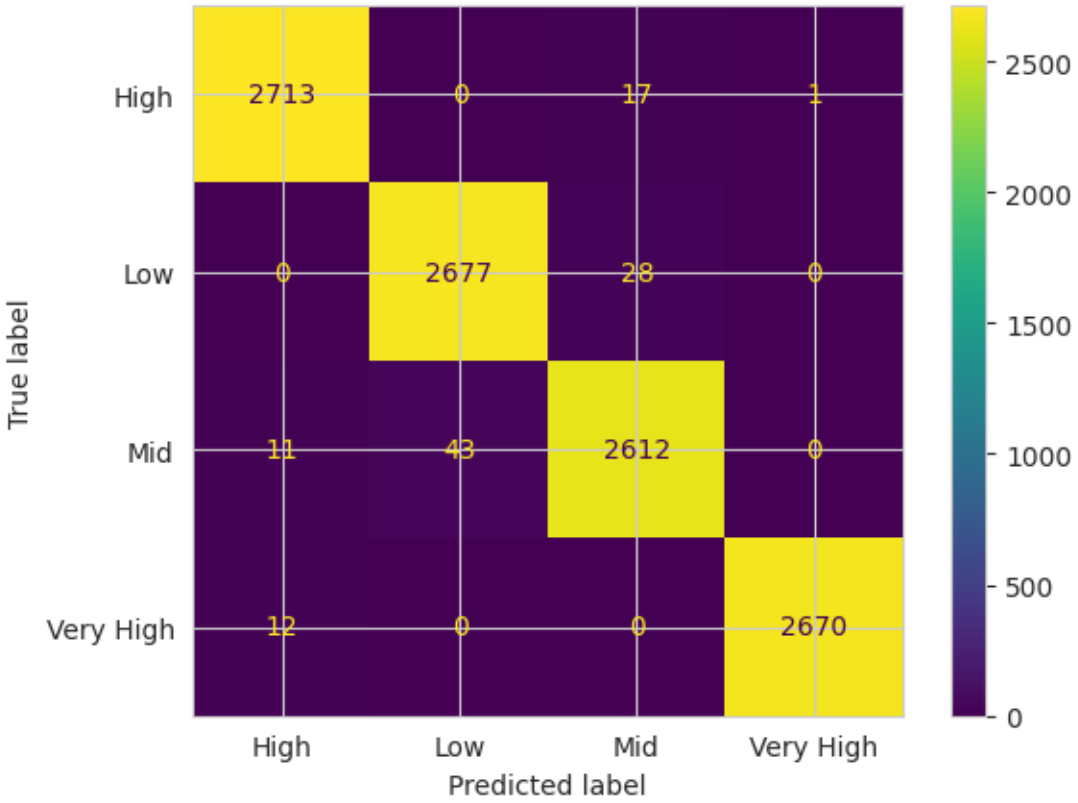
```
Cross-validation scores: [0.98597589 0.98748116 0.98597427 0.98690159
0.98609018]
```

```
Mean cross-validation score: 0.9864846186184666
```

```
y_pred = ksvm.predict(x_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

      budget       0.98      0.99      0.99      2846
      luxury       1.00      1.00      1.00      1014
   mid_range       0.99      0.99      0.99      5018
     premium       1.00      0.99      0.99      1910

    accuracy                           0.99     10788
   macro avg       0.99      0.99      0.99     10788
weighted avg       0.99      0.99      0.99     10788
```

```
from sklearn.metrics import ConfusionMatrixDisplay

ConfusionMatrixDisplay.from_estimator(ksvm, x_test, y_test)
```



Neural network:

❖ MLP

```
❖ from sklearn.neural_network import MLPClassifier
❖
❖ mlp = MLPClassifier(hidden_layer_sizes=(100,), max_iter=500,
  activation='relu', solver='adam', random_state=42)
❖
❖ mlp.fit(x_train, y_train)
❖
❖ y_pred = mlp.predict(x_test)
❖
❖ cm = confusion_matrix(y_test, y_pred)
❖ print("Confusion Matrix:\n", cm)
❖
❖ accuracy = accuracy_score(y_test, y_pred)
❖ print("Accuracy:", accuracy)
❖
❖ print("Classification Report:\n", classification_report(y_test,
  y_pred))
❖
```

```
❖ Confusion Matrix:
```

```
 [[2709    0   12   10]
 [   0 2669   36    0]
 [   4    9 2653    0]
 [   2    0    0 2680]]
Accuracy: 0.9932307121661721
Classification Report:
              precision    recall  f1-score   support

        High       1.00      0.99      0.99      2731
         Low       1.00      0.99      0.99      2705
         Mid       0.98      1.00      0.99      2666
   Very High       1.00      1.00      1.00      2682

    accuracy                           0.99     10784
   macro avg       0.99      0.99      0.99     10784
weighted avg       0.99      0.99      0.99     10784
```

```python
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(100,), max_iter=500,
activation='tanh', solver='adam', random_state=42)

mlp.fit(x_train, y_train)

y_pred = mlp.predict(x_test)

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

print("Classification Report:\n", classification_report(y_test,
y_pred))
```

```
Confusion Matrix:
 [[2721    0    8    2]
 [   0 2684   21    0]
 [  13   19 2634    0]
 [   6    0    0 2676]]
Accuracy: 0.9936016320474778
Classification Report:
              precision    recall  f1-score   support

        High       0.99      1.00      0.99      2731
         Low       0.99      0.99      0.99      2705
```

```
              Mid       0.99      0.99      0.99      2666
        Very High       1.00      1.00      1.00      2682

         accuracy                           0.99     10784
        macro avg       0.99      0.99      0.99     10784
     weighted avg       0.99      0.99      0.99     10784
```

جدول مقایسه ی تمامی روش ها :

Classifier	Accuracy	RECALL	Precision	F1-score
KNN	91.13	0.91	0.91	0.91
DECISION TREE	99.97	100	100	100
LOGISTIC REGRESSION	97.47	97	97	97
SVM(rbf)	95.17	0.95	0.95	0.95
SVM(linear)	98.96	99	99	99
Neural network (MLP)	99.36	99	99	99