Project :

Laptop Price Predictor

So, The Main Objective of this project is Predicting The price of Laptops. Using The past data, it Undergoes The Data preprocessing and Exploratory Data analysis. and We Use "Machine learning" algorithms to predict The model.

The output (dependent variable) is "Continous data" say what.! data is in 'Numeric data', so, we appl Regression models. For prediction.

So, on observation we can see that if we remove "GB" from RAM, i can make it as integer value Then after, now same goes with "memory", "weight" for weight i can classify it as floating variable using

The Str.replace(). 8GB ⟹ 8.

- df["Ram"] = df["Ram"].str.replace("GB", "")

Error: can only use .Str accessor with string values!

- df["Ram"] = df["Ram"].astype(str).str.replace("GB", "")

datatypes changed

- df["Weight"] = df["Weight"].astype(str).str.replace("kg", "")

#converting From string = integer for "Ram" column

- df["Ram"] = df["Ram"].astype("int32")

Converting from string = float for Weight column

df["weight"] = df["weight"].astype("float32")

df.head()

Ram	weight
8	1.37
8	1.34

Old value : 8 GB ..... 1.34 kg.

- df.info()

Ram = int32. , old info : object

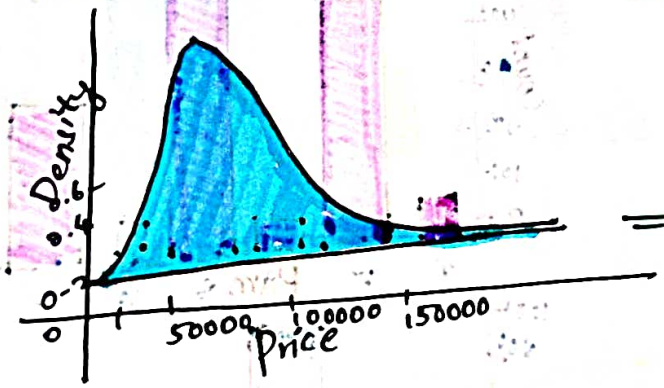
Weight = float32 , old info : object

- **EDA**

sns.distplot(df["Price"])

↳ output variable

dhi distribution plot edi



→ Right Skewed.

plotting Countplots for "Categorical variables"

⇓

* Company
* TypeName
* Ram
* OpSys

Object = Categorical data

function ← name

- def drawplot (col): → column

Edi Rayaka pothe Arm ayethadi

plt.figure (figsize = (15, 7)) ⇒ (7, 3)) ← eyakuva chinna size ki

→ only last figure ehy print avuthadi

sns.countplot (df [col])

plt.xticks (rotation = "vertical") → Edi easy ga ardam ayethadi

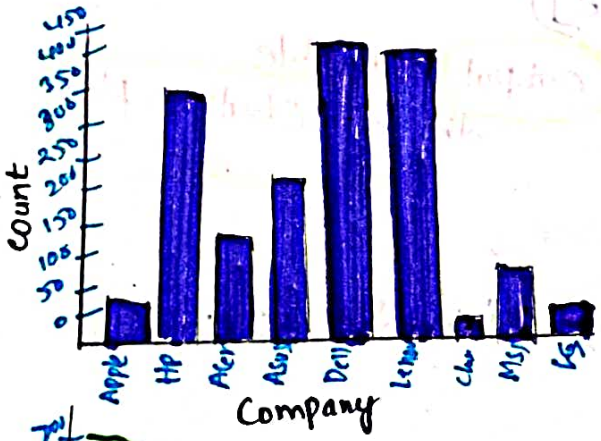
↳ place vundadhu pratidhaniki, ---- (0) !!!

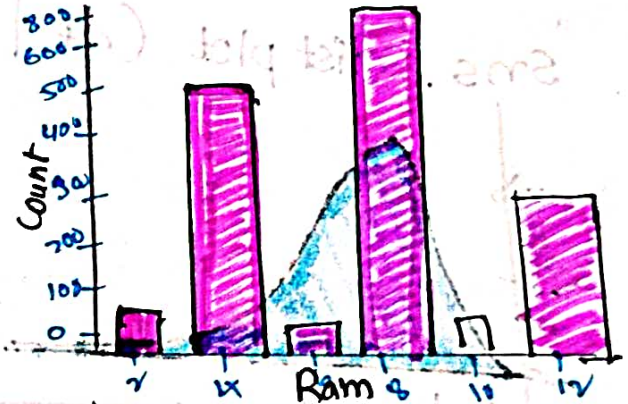
to view = ["Company", "TypeName", "Ram", "OpSys"]

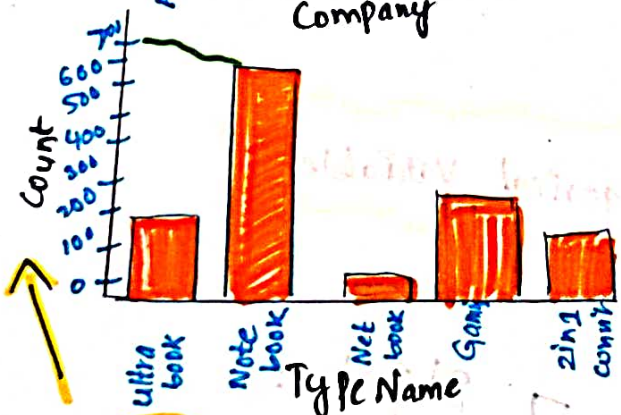
for col in to view : → categorical data (object):

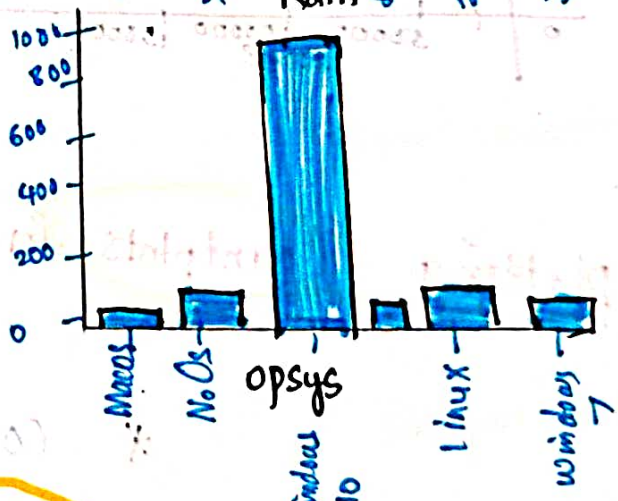
drawplot (col)

→ draw countplot. ani ardham.









df ["TypeName"] . value_counts()

Out: Notebook : 727
Gaming : 205
2 in 1 = 129
Workstation : 29
Netbook : 25

highest 700

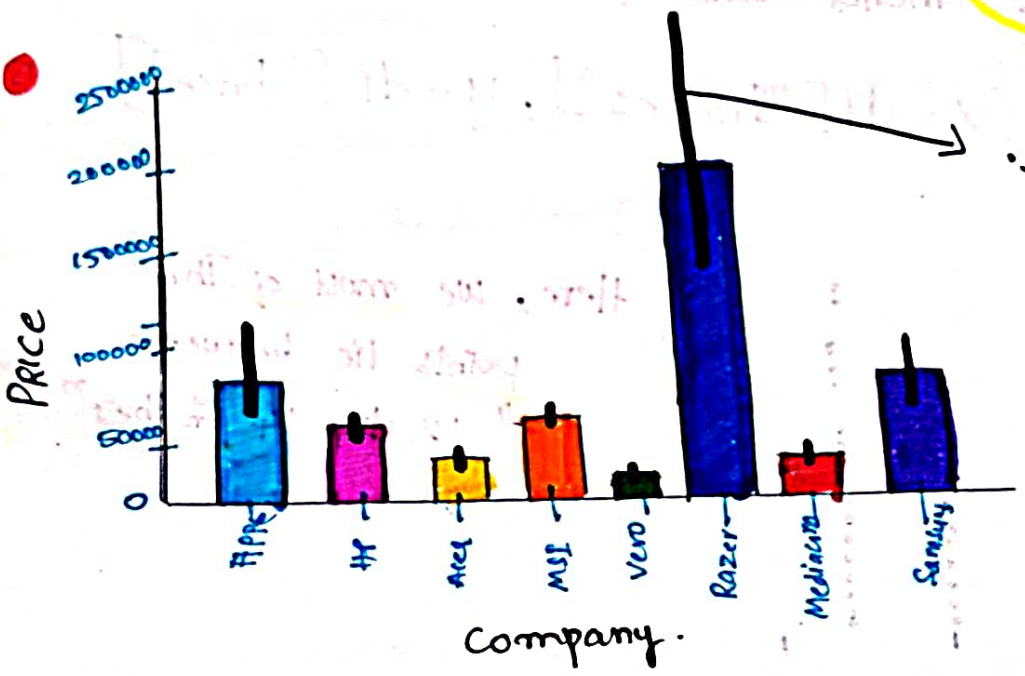
sns.lo visual ga represent cheytharu

average price For each of the laptop brands

This will say us insights that as per company the price of laptop vary. ⇒ Barplot ⇒ discrete Data

```
plt.figure(figsize = (7,3))
sns.barplot(x = df["Company"], y = df["Price"])
plt.xticks(rotation = "Vertical")
plt.show()
```

barplot :-
relationship b/w numeric and data
categoric
price
comp

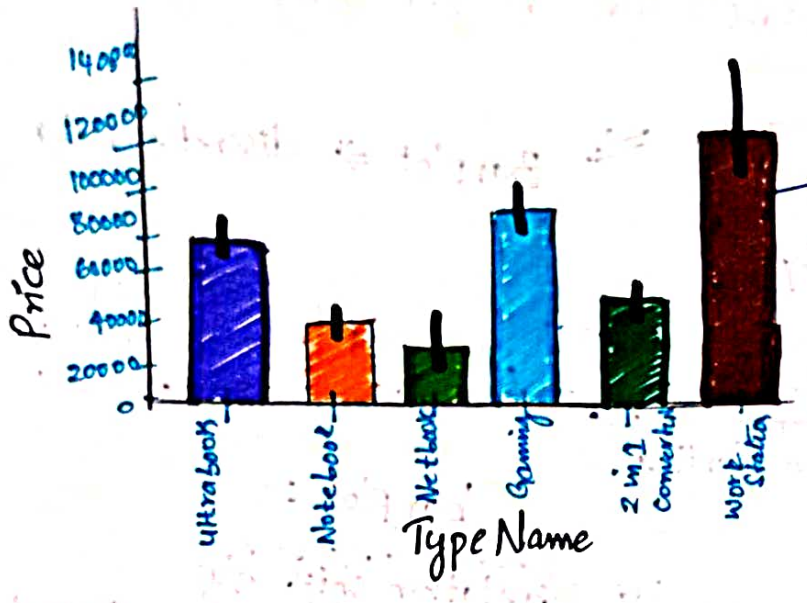


indicates so many values are over ride.
antey chala values kalisi vunnaye ani

laptop type and variation about price

```
sns.barplot(x = df["TypeName"], y = df["Price"])
plt.xticks(rotation = "vertical")
```

Graph.

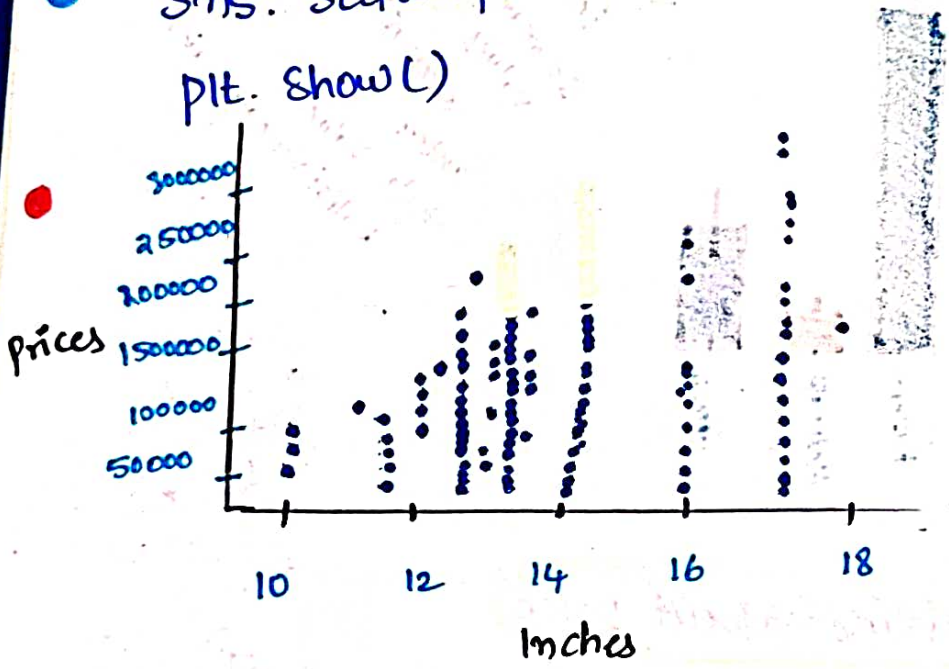


Here, Work station cost or) price is more compare to others, because very few buys these laptops.

Thats, why value_counts have 29 only.

- Variation Towards "Inches" and "Price".

```
sns.scatterplot(x = df["Inches"], y = df["Price"]
plt.show()
```



Here, we most of the points lie between "12 to 14 inches"

For Screen resolution column we have many types of Screen resolutions out there as shown. "Touch Screen" and "Normal" and IPS panel are 3 parts on basis of which we can Seperate the Things.

df["ScreenResolution"].value_counts()

Full HD 1920x1080	507
1366x768	281
IPS panel Full HD	230
IPS panel / Touch Screen	53
⋮	⋮
Touch screen / Full HD 1920x1080	1
Touch screen / Quad HD+ 3200	1
Touch screen / 4K ultra	1

→ Here, We see, repeated Words in Screen Resolution...

* Full HD
* IPS panel
* Touch screen

Create a New column, TouchScreen if the value is "1" that Laptop is Touch Screen.

Take no. of arguments in one expression.

df["TouchScreen"] = df["ScreenResolution"].apply(lambda element : 1 if "Touch screen" in element else 0)

New column.

(lambda element : 1 if "Touch screen" in element else 0)

ardham

Lambda function, For loop la [element = Full HD 1920x1080]

Value_counts lo. ekkada Touch Screen

Vuntadho. akkada "1" ani create chesthadi..

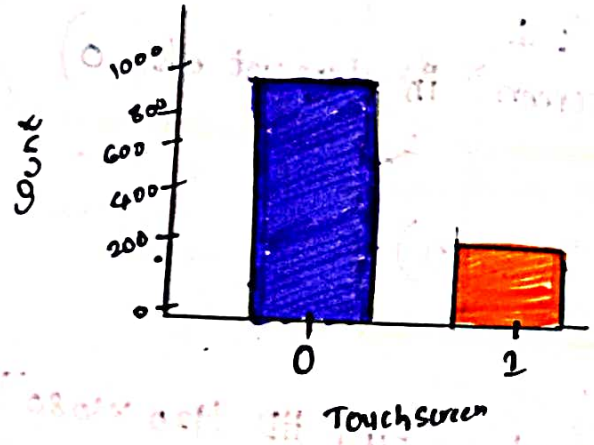
else (lekapothey) "0" ani df["TouchScreen"] = column ki

Save ayethadhi.

Ekkada "Touch screen", Vunte akkada → 1 lekapothe "0"

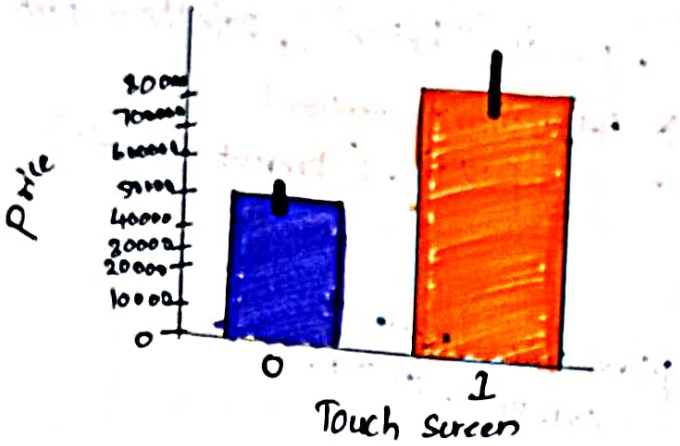
	Company	Type Name	Inches	Screen Resolution	Cpu	Ram	Memo.	....	Price	Touchscreen
0				IPS Panel Retina Display						0
1298				IPS Panel Full HD / Touchscreen						1
1299				IPS Panel Quad / Touchscreen						1
1302				1366x768						0

- df["Touchscreen"].value_counts()
- 0 = 1111
  1 = 192
- sns.countplot(df["Touchscreen"])



Touch Screen Comparision with price of laptop

- sns.barplot(x = df["Touch Screen"], y = df["Price"])
- Graph.



From, Here, we can say, The Price for The Touch screen is more Than normal Desktops

```
df["Ips"] = df["ScreenResolution"].apply(lambda element : 1 if "Ips" in element else 0)
```

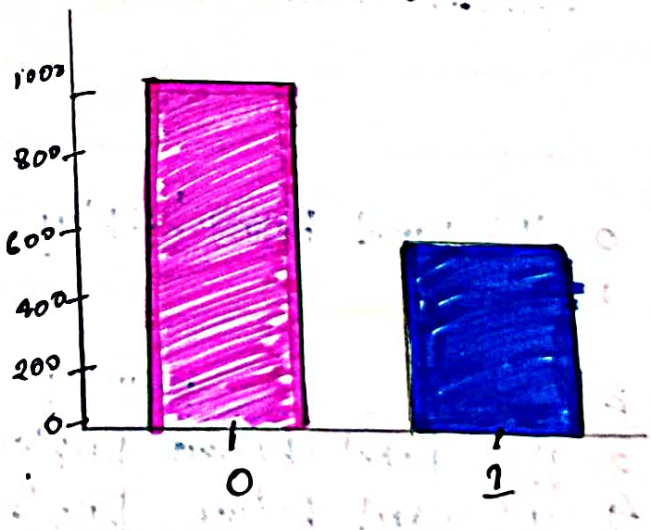
```
df.sample(5)
```

	Company	Type Name	Inches	ScreenResolution	.....	Price	Touch screen	Ips
263				Full HD			0	0
596				Ips panel			0	1
731				1366x768			0	0
221				Ips panel Full HD			0	1
712				Full HD			0	0

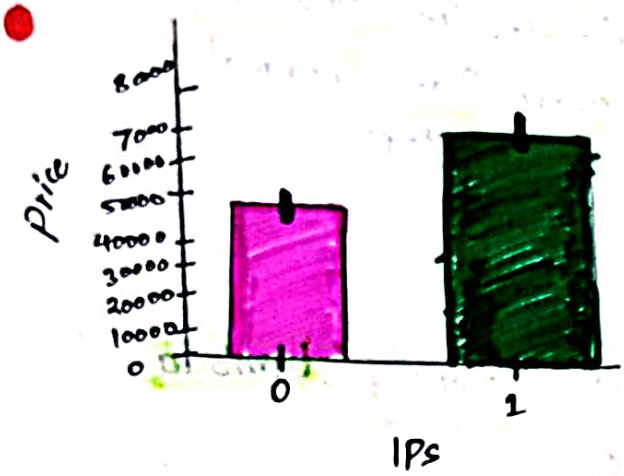
```
df["Ips"].value_counts()
```

```
0    938
1    365
```

```
sns.countplot(df["Ips"])
```



- `sns.barplot(x=df["Ips"], y=df["Price"], palette="dark")`
  - Ips → intrusion prevention system. (prevent threats)



Max. price - 8,000
Avg. price - 70-75 k.

Extracting The X Resolution and Y Resolution.

We will Split The Text at the "x" letter and Seperate The 2 parts

From this we can Observe That One of The col is Y resolution

We need to do Feature Engineering on The X res column.

Ex:- 1280 x 1600
1280 <before> "x" 1600 <after>

Some Feature Engineering

- `Splitdf = df["ScreenResdution"].str.split("x", n=1, expand=True)`
- `splitdf.head()`

		0	1
0	IPS panel Retina Display	2560	1600
1	Balanc	1440	900
2	Full HD	1920	1080
3	IPS Panel Retina Display	2880	1800
4	IPS panel Retina Display	2560	1600

Ex:

Whatever, it may be before "x" = 1440
Take That into consideration. even if " " also balance space Consider.

Split df .= df["ScreenResolution"].str.split("x", n=1, expand=True)

Fit df["X_res"] = split df[0]

df["Y_res"] = split df[1]

df.head()

Extract chesmaam.

	Compan	Type Name	Inche	ScreenResolution	....	...	Price	Touch screen	IPS	X_res	Y_res
0				IPS Panel Retina Display 2560 x 1600						IPS Panel 2560	1600
1				1440 x 900						1440	900
2				Full HD 1920 x 1080						Full HD 1920	1080
3				IPS panel Retina 2880 x 1800						IPS 2880	1800
4				IPS Panel Retina 2560x1600						IPS 2560	1600

So, basically From that Whole text of the "X_res" col, We need to extract the digits from it, but the problem is the numbers are ~~scrated~~ scattered in Some Cases, that is the reason why iam using regex, if we use this we will exactly get the numbers which we are looking for

So, firstly replace all the "," with "" and Then find all numbers from that string as "\d+\.?\d

\d means That integer

\.? all numbers which Come afeter an number

\d+ The string must end with number

• Ekkada manam only number consider chesi, text ignore chesam

```
df["x_res"] = df["x_res"].str.replace(",","")
              .str.findall(r'(\d+\.?\d+)').apply(lambda x:x[0])
```

Sorting "x" lo x[0] anty first dhi ani artham.

Regular Expression [RegEx]

First dhanimeda apply chestham

Ex:- IPS Panel Retina Display 2880 → started with number, ended with integer number

First, we look for number (\d)+(\.?)(\d+) and ends with integer

There is (.) dot en not in the text data.

df.head()

Company	Type	Inches	Screen Resolution	-----	Price	Touch Screen	Ips	X_res	Y_res
			IPS Panel Retina Display 2560x1600					2560	1600
								1440	900
								1920	1080
								2880	1800
								2560	1600

Ela vunna Data ni

Ela marcham.

Convert [x_res][y_res] column into "int" Data.

```
df["x_res"] = df["x_res"].astype("int")
df["y_res"] = df["y_res"].astype("int")
```

```
df.info()
```

	Data columns	Dtype
0	Company	object
⋮	⋮	⋮
13	x_res	int32
14	Y_res	int32

old : object

old : object

Correlation

```
sns.heatmap(df.corr(), annot=True, cmap="plasma")
plt.figure(fig_size=(15,7))
```

Display number

	Inches	Ram	Weight	Price	Touch screen	IPS	X_res	Y_res
Inches	1	0.24	0.83	0.068	-0.36	-0.11	-0.071	-0.095
Ram	0.24	1	0.38	0.74	0.12	0.21	0.43	0.42
Weight	0.83	0.38	1	0.21	-0.29	0.017	-0.033	-0.054
Price	0.068	0.74	0.21	1	0.19	0.25	0.56	0.55
Touch screen	-0.36	0.12	-0.29	0.19	1	0.15	0.35	0.36
IPS	-0.11	0.21	0.017	0.25	0.15	1	0.28	0.29
X_res	-0.071	0.43	-0.033	0.56	0.35	0.28	1	0.99
Y_res	-0.095	0.42	-0.054	0.55	0.36	0.29	0.99	1

1.0 0.8 0.6 0.4 0.2 0.0 -0.2

```
df.corr()["Price"]
```

Inches	0.068197
Ram	0.743007
weight	0.210370
price	1.000000
Touchscreen	0.191226
IPS	0.252208

X res	0.556529
Y_res	0.552809.

From the correlation plot we observed that the X_res and Y_res is increasing, the price of the laptop is also increasing, so X_res & Y_res are positively correlated and they are giving much information so that is the reason. why I had spitted resolution column into X_res & Y_res columns respectively.

So, to make things good, we can create a new column named PPI [Pixel Per Inch], Now as we saw from correlation plot that the X_res and Y_res are having much collinearity, so why not combine them with inches. which is having "less collinearity", so, we will combine them as follows, so here formula

To Calculate PPI [Pixel Per Inch]

$$PPI\ [pixel\ per\ inch] = \frac{\sqrt{X_{resolution}^2 + Y_{resolution}^2}}{Inches}$$

To combine X_Xes, Y_res and Inches.
They must have same datatypes

* Inches = Float64 →
* X_res = Int
* Y_res = int

```
df["PPI"] = ( (df["X_res"]**2 + df["Y_res"]**2)**0.5 / df["Inches"] )
```

PPI = $\sqrt{(X\_res)^2 + (Y\_res)^2}$ / Inches

`df.head()`

	Company	TypeName	Inches	ScreenResolution	...	X_res	Y_res	PPI
0			13.3			2560	1600	226.983
1			13.3			1440	900	127.6779
2			15.6			1920	1080	141.2119
3			15.4			2880	1800	220.5346
4			13.3			2560	1600	226.98300

Ex:- $\frac{(2560^2 + 1600^2)^{0.5}}{13.3} = 226.983$

`df.corr()["Price"]`

Inches	0.0681
⋮	⋮
X_res	0.556529
Y_res	0.552809
PPI	0.473487

We Combined these three columns with which gives Lets on Good correlation.

So, we drop these 3 columns along with "screen resolut column.

because, we extracted total data From

```
df.drop(columns = ["ScreenResolution", "Inches", "x_res",
                   "y_res"], inplace = True)
```

```
df.head()
```

Company	Type Name	Cpu	Ram	Memory	Gpu	Opsys	weight	Price	Touch Screen	Ips	ppi

↳ Remaining Columns.

We will Work on Cpu Column

```
df["Cpu"].value_counts()
```

```
Intel Core i5 7200U 2.5GHz     190
Intel Core i7 7700HQ 2.8GHz    146
Intel Core i7 7500U 2.7GHz     134
.....                          1
AMD A6-Series
```

Most common processors are made by intel right, so we will be clustering their processor into different categories. i5, i7, other → processors of intel which do not have i3, i5, i7 attached to it. Completely different so, that's the reason i will cluster them into other and other category is "AMD" which is different category in whole.

So, if we observe we need to extract the first 3 words of Cpu column. as first 3 words of every row under cpu column of the cpu. we will using them.

Text = "Intel Core i5 7200U 2.5GHz"

Text.spit() (split chesinam)

["Intel", :"Core", "i5", "7200U", "2.5GHz"]

For Every Text in Cpu apply the condition (or) Logic

Text.split()[:3] (It Gives only upto 2)

["Intel", "Core", "i5"] (dhani only first three matrame select) chesam Text ni.

" ".join(Text.split()[:3])

"Intel Core i5" Tarwata join chesam.

"_".join(Text.split()[:3])

"Intel_Core_i5" → dheni ardham.. Prati dhaniki space echam.

Step By Step Exceution (Breaking code into parts)

df["cpu_name"] = df["Cpu"].apply(lambda text : " ".join(text.split()[:3]))

Company	Type name	Cpu		Cpu_name
		Intel Core i5 2.3GHz		intel Core i5
		Intel core i5 1.8GHz		intel core i5
		⋮		⋮
		intel core i5 3.1 GHz		intel core i5

Problem Here :- Some rows Have in Cpu_name Have [556.intel Atom x5-Z8350, 204.intel Xeon E3-1505M]

So, Manaki only i5, i3, i7 matrame Ekkuva Vunnaye. Kakapothey. Ekkada "intel" lo kuda rakalu unn ayethey vati simple ga "Intel" rasi, migithaye "AMD" Processor" anirradham.

Function :

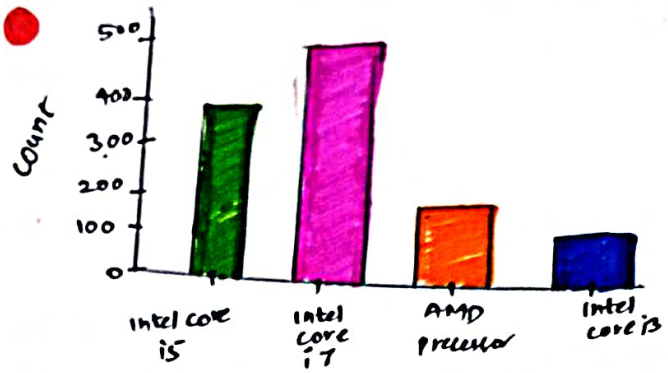
```
def processor_type (text) :
    if text == "Intel Core i7" or text == "Intel Core i5"
            or text == "Intel Core i3":
        return text
    else :
        if text.split()[0] == "Intel":
            return "Other Intel processor"
        else :
            return "AMD processor"
df["cpu_name"] = df["cpu_name"].apply(lambda text :
                    processor_type(text))
```

df.sample(5)

Company	Type Name	cpu	....	cpu_name
		Intel Celeron Dual core		AMD processor
		AMD A9-Seri		AMD Precessor
		intel core i7 7700 HQ		intel core i7
		intel celeron Dual core		AMD Precessor
		Intel Core i7		intel core i7

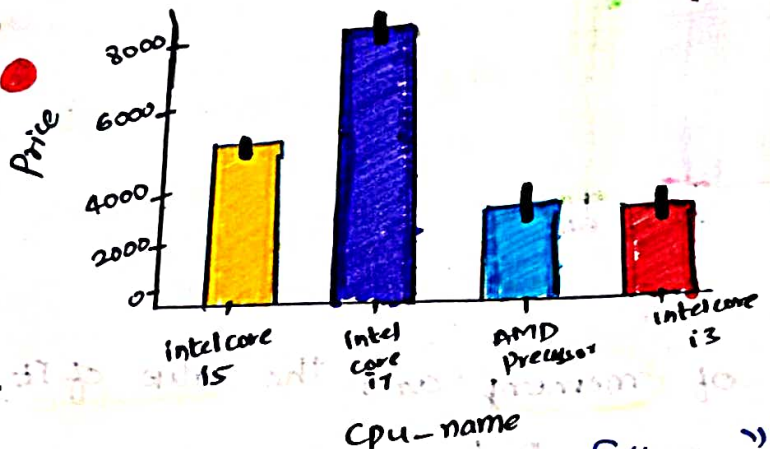
- `sns.countplot(df["cpu_name"])`

∴ Issue [other processor] among column was thaled ⚠



Variation between "price" & "cpu_name"

- `sns.barplot(x=df["cpu_name"], y=df["Price"])`



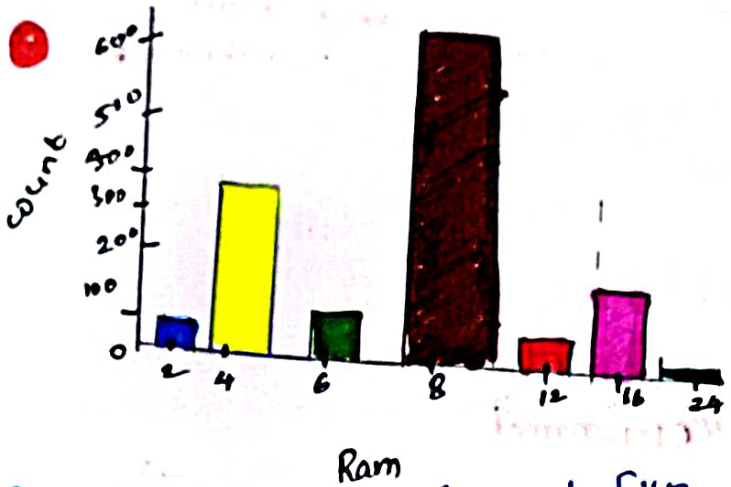
- `df.drop(columns=["cpu"], inplace=True)`

`df.head()`

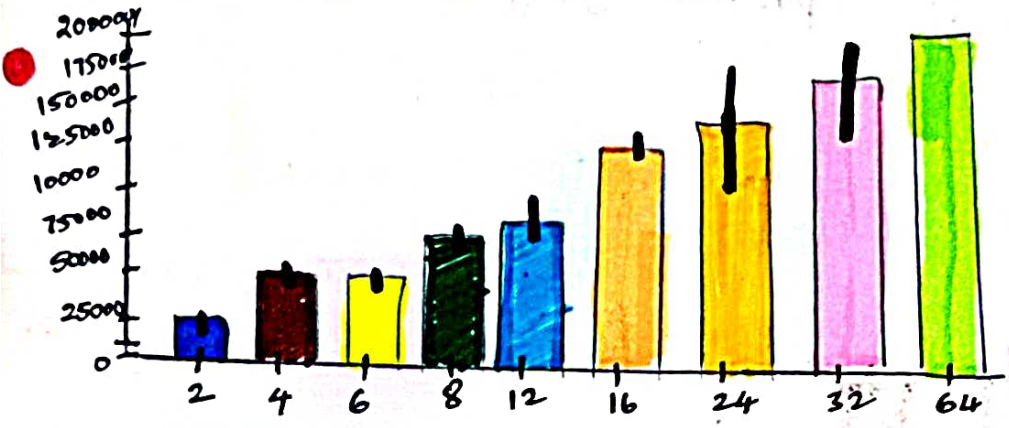
Company	Type Name	Ram	Memory	Gpu	OpSys	Weight	Price	Touch Screen	Ips	PPI	Cpu ha

RAM Column

- sns.countplot(df["Ram"])



- sns.barplot(x=df["Ram"], y=df["Price"])



Memory Column

We will Seperate the type of memory and the value of it, just similar to one done in previous part.

This part involves various steps, we don't have memo as a Complete we have it in different dimension as 128GB SSD +1TB HDD, so In order to for it come in Same dimension, we need to do some modification.

We have different Categories and also different variation

```
df["Memory"].value_counts()
```

```
256 GB SSD                412
1 TB HDD                  223
128 GB SSD + 1 TB HDD      94
1.0 TB Hybrid               9
128 GB Flash Storage        4
```

We have 4 common variants : HDD, SSD, Flash, Hybrid

This Expression remove decimal space for Ex: 1.0 TB will be 1TB

```
df["Memory"] = df["Memory"].astype(str).replace('\.0', '', regex=True)
```

Ex:- 128GB SSD + 1TB HDD

- 128 SSD + 1 TB HDD
- 128 SSD + 1000 HDD

what ever coming after decimal point Delete it. Ex:- 1.0 TB = 1TB

apply to Every Tuple

convert it in string.

```
# replace GB with " "
df["Memory"] = df["Memory"].str.replace("GB", "")
```

```
# replace TB with "000"
df["Memory"] = df["Memory"].str.replace("TB", "000")
```

```
# Split the word across "+" character
newdf = df["Memory"].str.split("+", n=1, expand=True)
```

Removed

- 128 SSD + 1000 HDD

["128 SSD", "1000 HDD"]

- new df
- new df.sample(5)

0	1
128 SSD	None
128 Flash storage	None
256 SSD	None
512 SSD	None
⋮	
1000 HDD	None
500 HDD	None

0	1
"	None
"	1000 HDD
"	None
"	None
1000 SSD	2000 HDD

Endulo kuda values vunnaye

1000 SSD + 2000 HDD :

We will strip up all the "white spaces", basically Eliminating white space.

```
df["first"] = new df[0]   # Here we considered only [0]. but ignore [1]
df["first"] = df["first"].str.strip()
df.head()
```

string ga marchi ⇒ strip() apply cheyali

⇓

white space motam pothaye.

Company	Type Name	Inche	Memory	...	first
			1000 HDD		1000 HDD
			512 SSD		512 SSD
			1000 HDD		1000 HDD
			256 HDD		256 HDD
			512 HDD		512 HDD

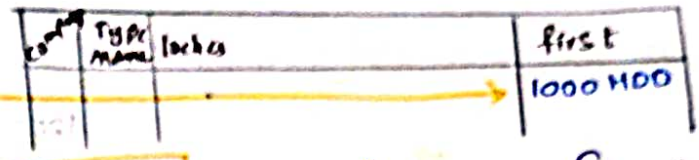
↓ with white space

↓ without white space

```
def apply_changes(value):
    df["Layer1" + value] = df["first"].apply(lambda x: 1 if value in x else 0)

List_to_apply = ["HDD", "SSD", "Hybrid", "FlashStorage"]
for value in List_to_apply:
    apply_changes(values)
```

Detail explanation




```
def apply changes (value)
    df["Layer1" + value] = df["first"].apply(lambda x: 1 if value in x else 0)
```

important

Layer1 + 1000HDD → "Vunna dhaggara 1" vasthadi

```
list to apply = ["HDD", "SSD", "Hybrid", "Flash storage"]
```

lambda function dheni meda apply cheyali

```
for value in list_to_apply:
    apply changes (values)
```

motam lambda tho check chesthadi

ni

def Function

df.head()

Company	Type Name	Ram	Memory		first	Layer 1 HDD	Layer 1 SSD	Layer 1 Hybrid	Layer 1 Flash Storage
					128 SSD	0	1	0	0
					128 Flash Stora	0	0	0	1
					1000 HDD	1	0	0	0
					128 Hybrid	0	0	1	0

remove all characters just keep the numbers

```
df["first"] = df["first"].str.replace(r"/D", "")
```

remove all text

Ex:- 128 SSD

A:- 128

```
df["first"].value_counts()
```

256 → 508
1000 → 250

128 → 177
512 → 140
8 → 1

508 → 1
240 → 1

we consider Second part of newdf[0][1].

1200 SSD records

1200 SSD + 120 HDD records

- `df["second"] = newdf[1]`

`df.head()`

Company	Type Name	Inches	...	First	Layer1 SSD	Layer1 HDD	Layer1 Hybrid	Layer1 Flash	Second
				128					None
				128					1000 HDD
				256					None
				512					None
				256					2000 HDD

We have apply same as "0".

```
def applychanges1(value):
    df["Layer2"+Value] = df["second"].apply(lambda x: 1 if Value in x else 0)
List to apply1 ["HDD", "SSD", "Hybrid", "Flashstorage"]
df["second"] = df["second"].fillna("0")
for Value in List to apply1:
    applychanges1(value)
df["second"] = df["second"].str.replace("GB","")
df["second"].value_counts()
```

0	→ 1095	2000	— 15
1000	→ 187	500	2
256	→ 8	512	1

Change datatypes of "first" and "second" to int

```
df["first"] = df["first"].astype("int")
df["second"] = df["second"].astype("int")
df.tail()
df.info()
```

Company : object

first : int — old : first ⇒ object

second : int — old. second ⇒ object

multiplying the elements and storing the result in Subsequent Columns.

```
df["HDD"] = (df["first"] * df["layer1 HDD"] +
             df["second"] * df["layer 2 HDD"])
```

Ex:- [128 × 0] + [0 × 0] = 0

• df["HDD"] = 0

[1000 × 1] + [0 × 0] = 1000

• df["HDD"] = 1000

```
df["SSD"] = (df["first"] * df["layer1SSD"] +
             df["second"] * df["layer2SSD"])

df["Hybrid"] = (df["first"] * df["layer1Hybrid"] +
                df["second"] * df["layer2SSD"])

df["Flash_Storage"] = df["first"] * df["layer1Flash_storage"]
                    + df["second"] * df["layer2Flash_storage"]

# dropping of Unnecessary columns
df.drop(columns = ["first", "Second", "layer1HDD",
        "layer1SSD", "layer1Hybrid", "layer1Flash-
        storage", "layer2HDD", "layer2SSD",
        "layer2Hybrid", "layer2Flashstorage"],
        inplace = True)

df.sample(5)
```

	Company	TypeName	Ram	Memory	....	HDD	SSD	Hybrid	Flash_Storage
27	Dell	Notebook	8	256SSD		0	256	0	0
680	MSI	Gaming	8	256SSD		0	256	0	0
431	Lenovo	Notebook	8	256SSD		0	256	0	0
907	Acer	Notebook	4	16Flash Storage		0	0	0	0
467	Dell	Notebook	8	2000HDD		2000	0	0	0

```
df.drop(columns=["Memory"], inplace=True)
df.head()
```

Column will be deleted (Malla Evadu Rayali ☺)

```
df..corr()["Price"]
```

Ram	0.743
Weight	0.210
Price	1.000
Touchscreen	0.191
Ips	0.252
PPI	0.473
HDD	-0.096 ✓
SSD	0.670 ✓
Hybrid	0.007
Flash_Storage	NaN

* not contributing to price column. (or) Very less contributing.
* We can Drop These two columns. Hybrid & Flash_storage.
* Where SSD & HDD having good correlation

* But, HDD has (-ve) Negative relation with price,
* The Price of Laptop is increasing There is more probability that t
  Laptop is gonna Use SSD instead of HDD and vice versa

- `df.drop(columns = "Hybrid", "Flash_storage"], inplace = True)`

Company	Type Name	Ram	Gpu	opSys	Weight	Price	Touch Screen	IPS	PPI	CPU_name	HDD	SSD

Gpu column.

- `df.Gpu.Value_Counts()`

```
intel HD Graphics 620      281
intel HD Graphics 520      185
        ⋮                   ⋮
ARM Mali T860 MP4            1
```

Here as we are having less data regarding the laptops, its better that we focus on Gpu brands. instead focusing on the values which are present there beside them, we will focus on the brands.

This is what we will be doing, Extracting the brands.

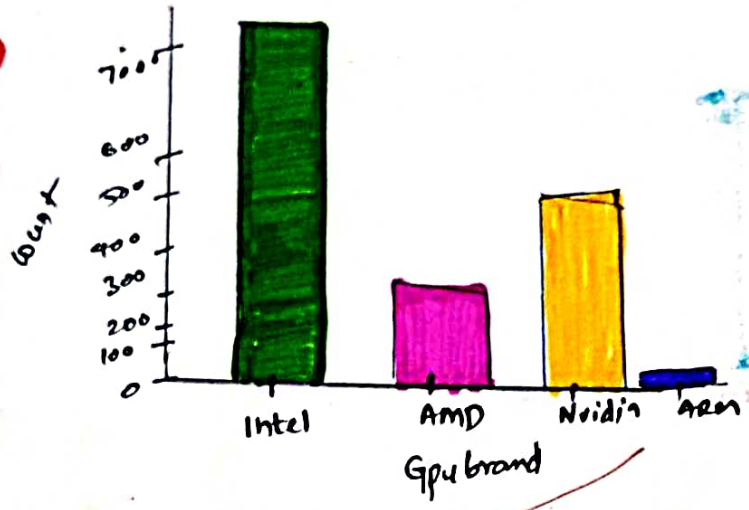
- `a = df["Gpu"].iloc[0]` (index (0))

  `Print(a.split()[0].)`

- Intel

df["Gpu_brand"] = df["Gpu"].apply(lambda x : x.split()[0])

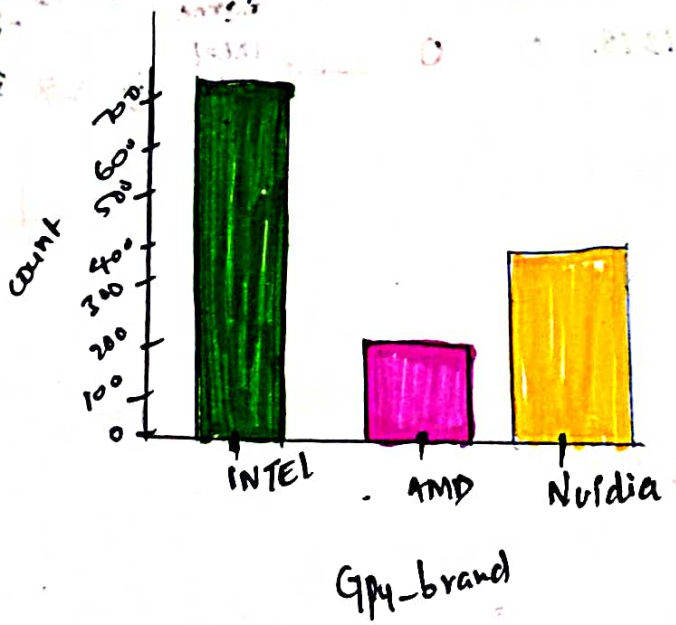
sns.countplot(df["Gpu_brand"], palette = "plasma")



#Remove "ARM" Tuple

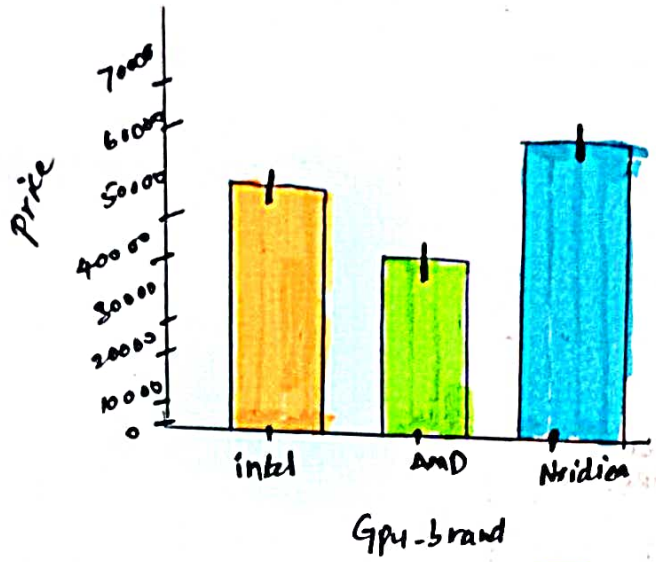
df=df[df["Gpu_brand"] != "ARM"]

sns.countplot(df["Gpu_brand"])



price - Gpu analysis, i used np.median in order to check if there is any impact of outlier or not

- sns.barplot(df["Gpu_brand"], df["Price"], estimator = np.mean)



- df = df.drop(["Gpu"], columns =)

df.head

Company	TypeName	Ram	OpSys	Weight	Price	Touchscreen	Ips	PPI	Cpu_name	HDD	SSD	Gpu_brand
Apple	ultra book	8	MacOS	1.37	71378.	0	0	226.9	Intel core i5	0	128	Intel

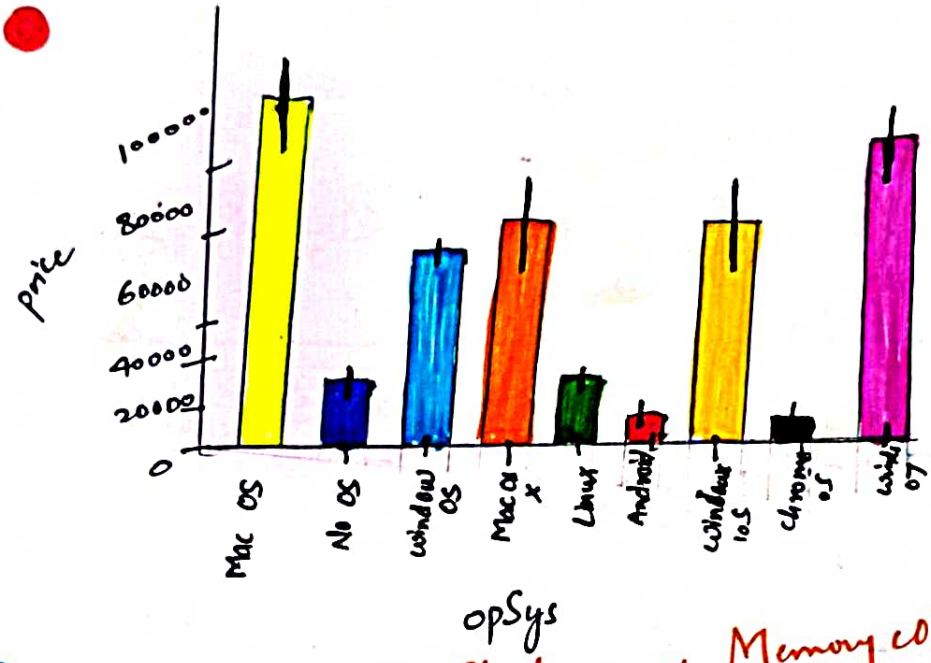
OpSys Column

- df["OpSys"].value_counts()

Windows 10	1072
No OS	66
Linux	62
⋮	
Android	8

Variation factor b/w "price" and "opSys" column.

```
sns.barplot(df["opSys"], df["price"])
plt.xticks(rotation = "Vertical")
plt.show()
```



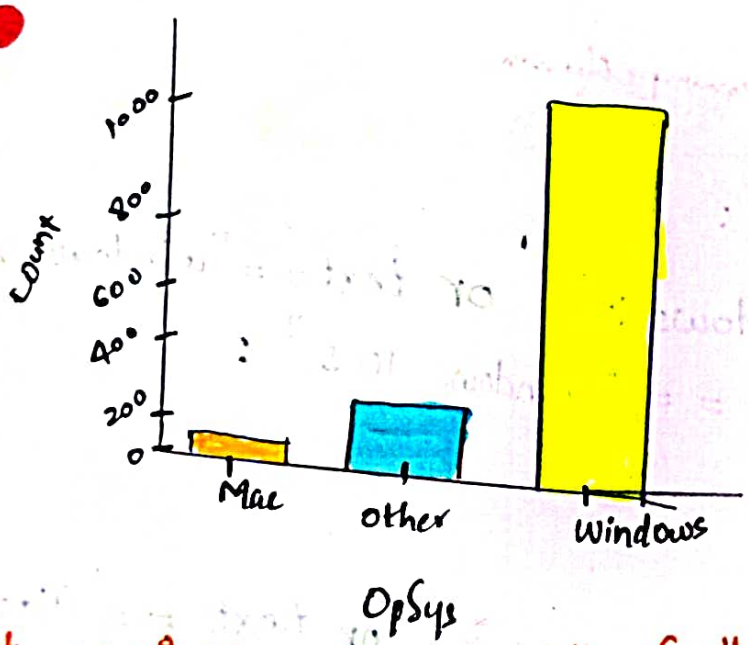
We apply same Strategy as Memory column.

```
def SetCategory(text):
    if text == "Windows 10" or text == "Windows 7"
        or text == "Windows 10 S":
        return "Windows"
    elif text == "Mac OS X" or text == "Mac
        return "Mac"
    else:
        return "Other"
```

```
df["opSys"] = df["opSys"].apply(lambda x : set category(x))
df.head(5)
```

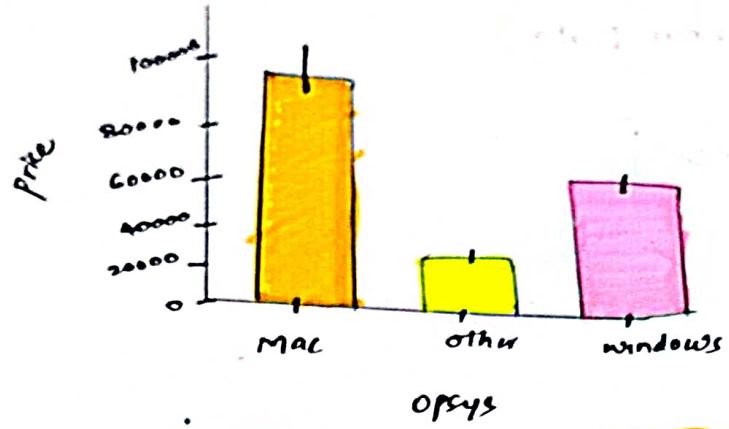
Company	TypeName	Ram	opSys		. . . . .	Gpu brand
Apple			Mac			
Apple			Mac			
Hp			other			
Apple			Mac			
Apple			Mac			

```
sns.countplot(df["opSys"])
```



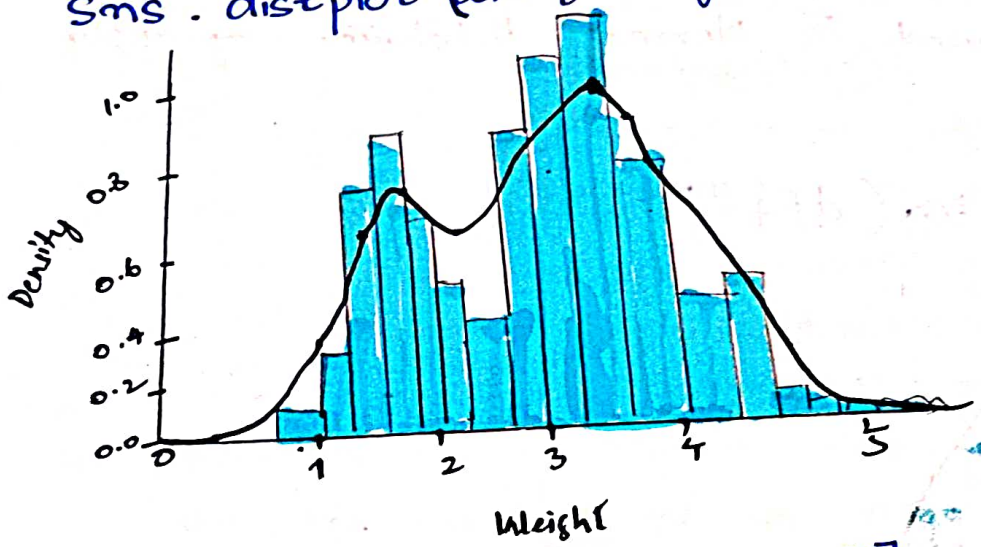
Variation between "opSys" & "Price"

```
sns.barplot(x = df["opsys"], y = df["Price"])
plt.xticks(rotation = "vertical")
```

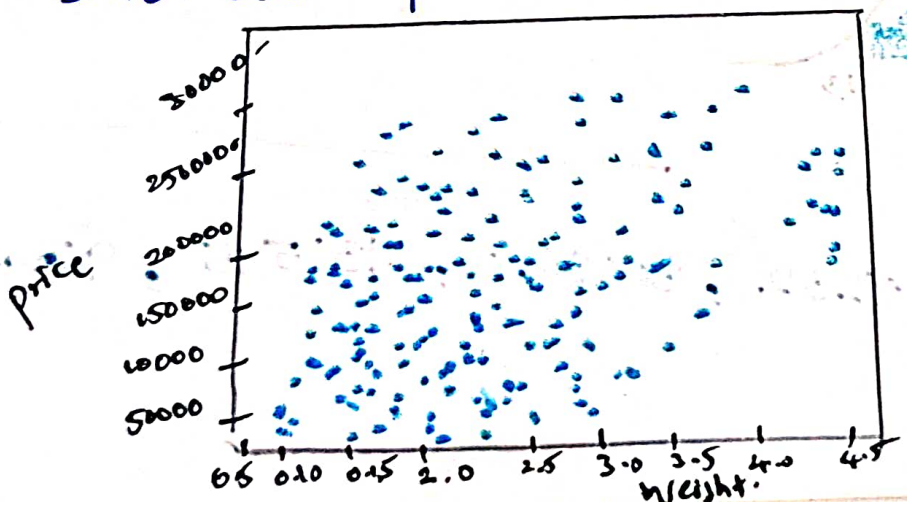


Weight Analysis → Continuos Data

Sns . distplot (df ["weight"])

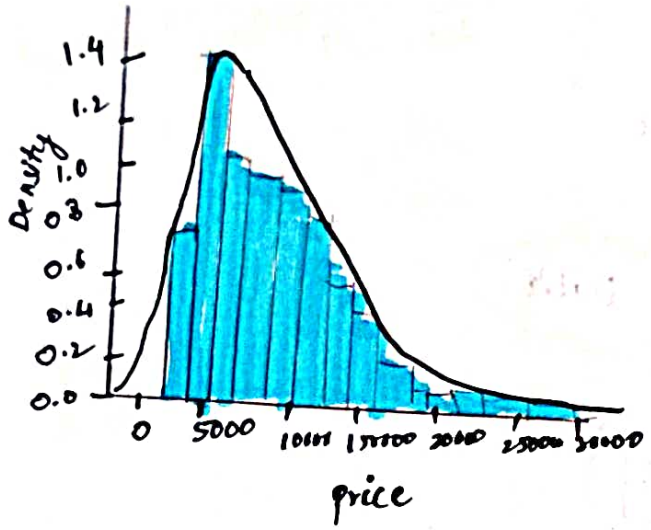


Sns. scatterplot (df ["weight"], df ["price"])



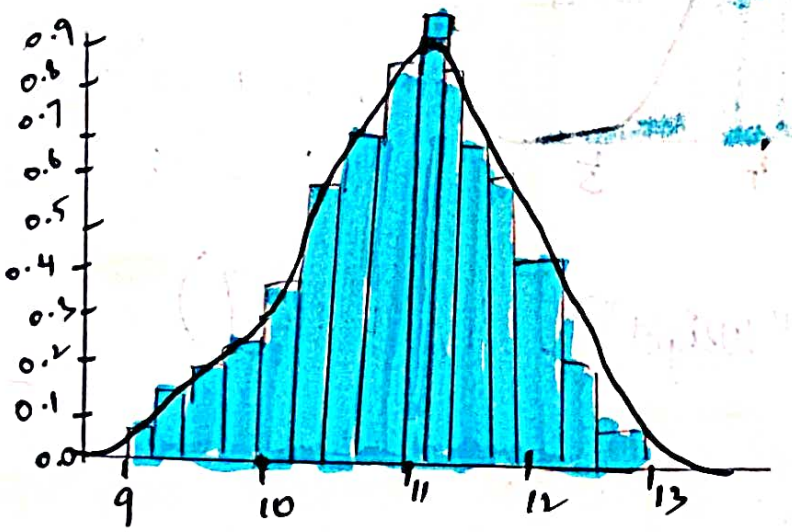
Price Analysis → Continous Data

- sns.distplot(df["price"])



We convert right skewed To Normal distribution by apply "log" to it.

- sns.distplot(np.log(df["Price"]))



- list(df["Price"])[:5]

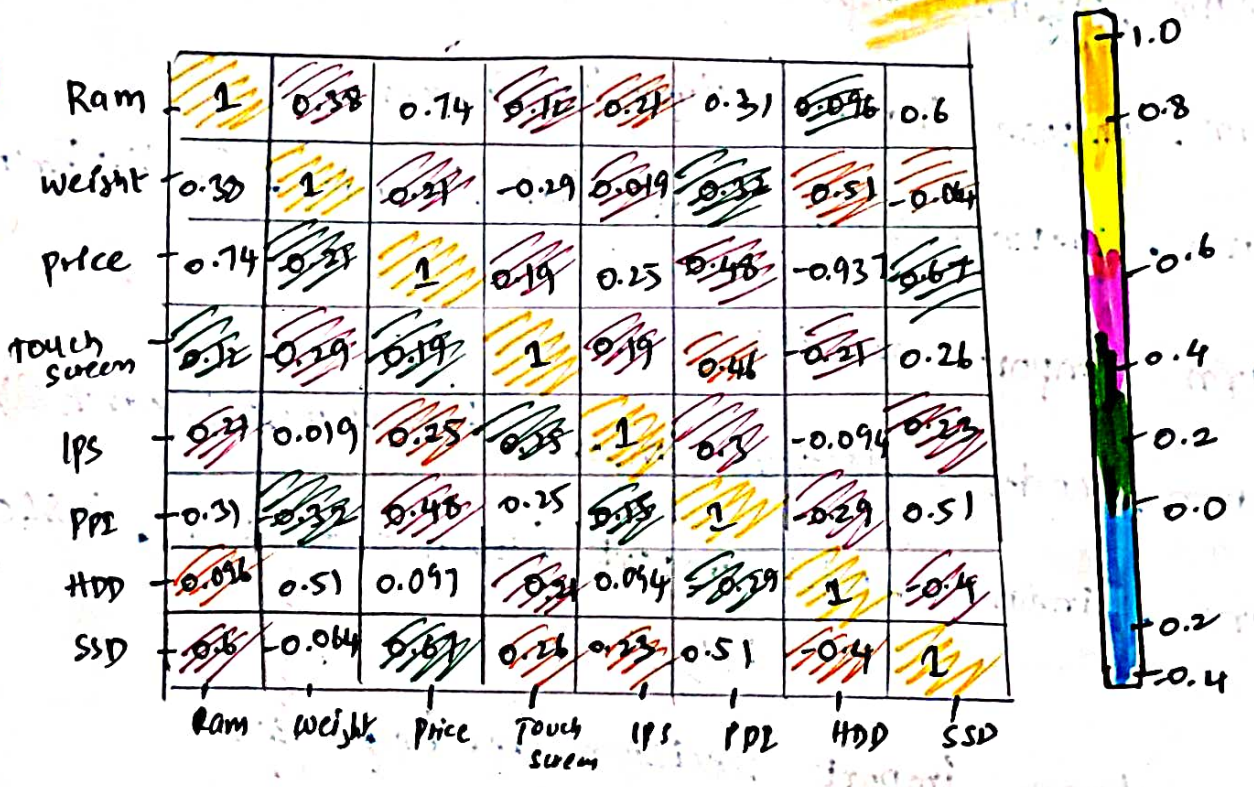
[7137.8.6382, 47895.5232, 30636.0, 131595, 96095.8

correlation with price.

- df.corr()["price"]

Ram	0.74290
weight	0.2098
price	1.0000
Touchscreen	0.192917
IPS	0.253320
PPI	0.475
HDD	-0.0968
SSD	0.670660

- plt.figure (figsize = (10,5))
- sns.heatmap (df.corr(), annot = True, cmap = "plasma")



Continuation of Converting "log" of price

- np.log (71378.6832)
- 11.175754549
- np.exp (11.175754549)
- 71378.6832

MODEL BUILDING

- test = np.log(df["Price"])
  train = df.drop(["Price"], axis = 1)

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScale
from sklearn.pipline import pipeline → pipe
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import LabelEncoder, OneHotEncode
from sklearn import metrics
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import LinearRegression, Lasso,
                                 Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor,
                             GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
```

```
from sklearn. import tree
```

Train Test split

```
X_train , X_test , y_train , y_test = train_test_split
                                      (train , test , test_size = 0.15 ,
                                                  random_state = 2)
```

```
X_train.shape , X_test.shape
```

```
((1106, 12), (196, 12))
```

There's a class which we imported named as ColumnTrans former

We use this widely while building our models using pipelines

so for this we have to get the index numbers of the columns which having categorical variables.

```
mapper = { i : Value for i , Value in enumerate
                                    (X_train.columns) }
```

n kinda pattern

```
mapper
```

0 "company" (Catgorical)
1 "TypeName" (cat)
2 "Ram"
3 "OpSys" (cat)
4 "Weight"
5 "Touchscreen"
6 "Ips"
7 "PPI"
8 "cpu_name" (cat)
9 "HDD"
10 "SSD"
11 "Gpu_brand" (cat)

Linear Regression

Here, we apply OneHot Encoding for columns :- 0, 1, 3, 11

The Remainder we keep as "pass through", i.e no other col must get effected.

except the ones undergoing Transformation.

→ Converting Catgorical variable To Numeric Variable

```
Step 1 = ColumnTransformer (transformers = [("col_tnf", OneHotEncoder (sparse = False, drop = "first", handle_unknown = "ignore"), [0,1,3,8,11])], remainder = "pass through")
```

Just name

simply pass through & make some changes.

We have lots of "0" in the matrixes
Ex:- [[1,0,0,0,0,0,0,0,0,0]
An identity matrix of dimension).
[100 crore x 1000] diagonals are 1. rest all are zero.

model

```
Step 2 = LinearRegression()
```

capital

```
Pipe = Pipeline([("step1", step1), ("step2", step2)])
```

Label (ColumnTransfor)

model

```
Pipe.fit(x_train, y_train)
```

fitting the data

Predict

```
y_pred = Pipe.predict(x_test)
Print("R2 score", metrics.r2_score(y_test, y_pred))
Print("MAE", metrics.mean_absolute_error(y_test, y_pred))
```

R2score 0.8073 → 0.80 features are giving good results.

accuracy of

MAE 0.21017

Giving Error of 0.21%.

```
np.exp(0.21)
1.2336780
```

```
71378.6832 - np.exp(0.21017)
71377.449301
```

it is giving 1.233 difference.

Ridge Regression

```
step1 = ColumnTransformer(transformers = [["col_tnf",
        OneHotEncoder(sparse = False, drop = "first",
    handle_Unknown = "Ignore"), [0,1,3,8,11]])]
    remainder = "passthrough")

step2 = Ridge(alpha = 10)

pipe = pipeline([("step1", step1), ("step2", step2)])

pipe.fit(x_train, y_train)  # fitting data

y_pred = pipe.predict(x_test)  → on test Data

Print("R2 Score", Metrics.r2_score(y_test, y_pred))

Print("MAE", Metrics.mean_squared_error(y_test,
    y_pred))
```

Predicting on "price" column

R2_score 0.8120403

MAE 0.21034714

Lasso Regression

```
step1 = ColumnTransformer(transformers=[("col_tnf",
        OneHotEncoder(sparse=False, drop="first",
        handle_unknown="ignore"),[0,1,3,8,11])],
        remainder="passthrough")

step2 = Lasso(alpha=0.001)

pipe = pipline[[("step1",step1),("step2",step2)]])

y_pred = pipe.Predict(x_test)

print("R2_score", Metrics.r2_score(y_test,y_pred))

print("MAE", Metrics.mean_squared_error(y_test,
                                        y_pred))
```

R2 score 0.80733

MAE 0.2101708

Decision Tree Regressor

- Step 1 = ColumnTransformer (transformer = ([("col_tnf", OneHotEncoder (sparse = False, drop = "first", handle_error = "ignore"), [0, 1, 3, 8, 11])], remainder = "passthrough")
- Step 2 = DecisionTreeRegressor (max_depth = 8)
- pipe = Pipeline ("step1", step1), ("step2", step2)
- pipe . fit (x_train, y_train)
- y_pred = pipe . predict (x_test)
- Print ("R2_score", metrics. R2_score (y_pred, y_test)
- Print ("MAE", metrics . mean_absolute_error (y_pred, y_test).

- R2_score : 0.848598

mean_absolute Error : 0.1781070

26/03/23