A.Importing the necessary Libraries

In [1]:
```python

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.model_selection import train_test_split

# Ignore all warnings
warnings.filterwarnings("ignore")
```

B.Data Loading

In [2]:
```python
df = pd.read_csv("laptopPrice.csv")
df.head(5)
```

Out[2]:

	brand	processor_brand	processor_name	processor_gnrtn	ram_gb	ram_type	ssd	hdd	
0	ASUS	Intel	Core i3	10th	4 GB	DDR4	0 GB	1024 GB	V
1	Lenovo	Intel	Core i3	10th	4 GB	DDR4	0 GB	1024 GB	V
2	Lenovo	Intel	Core i3	10th	4 GB	DDR4	0 GB	1024 GB	V
3	ASUS	Intel	Core i5	10th	8 GB	DDR4	512 GB	0 GB	V
4	ASUS	Intel	Celeron Dual	Not Available	4 GB	DDR4	0 GB	512 GB	V

C.EDA and Data Preprocessing

In [3]:
```python
print(f"The total number of latops given are : {df.shape[0]}")
print(f"The number of feature columns are: {df.shape[1]}")
```

```
The total number of latops given are : 823
The number of feature columns are: 19
```

In [4]:

```
"""Quick summary of dataframe"""
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 823 entries, 0 to 822
Data columns (total 19 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   brand               823 non-null    object
 1   processor_brand     823 non-null    object
 2   processor_name      823 non-null    object
 3   processor_gnrtn     823 non-null    object
 4   ram_gb              823 non-null    object
 5   ram_type            823 non-null    object
 6   ssd                 823 non-null    object
 7   hdd                 823 non-null    object
 8   os                  823 non-null    object
 9   os_bit              823 non-null    object
 10  graphic_card_gb     823 non-null    object
 11  weight              823 non-null    object
 12  warranty            823 non-null    object
 13  Touchscreen         823 non-null    object
 14  msoffice            823 non-null    object
 15  Price               823 non-null    int64 
 16  rating              823 non-null    object
 17  Number of Ratings   823 non-null    int64 
 18  Number of Reviews   823 non-null    int64 
dtypes: int64(3), object(16)
memory usage: 122.3+ KB
```

In [5]:

```
"""
Duplicate rows:
1. Check for duplicate rows
2. If present drop them
"""

print(f"The Number of Duplicated rows are: {df.duplicated().sum()}")
```

```
The Number of Duplicated rows are: 21
```

In [6]:

```
"""Duplicated rows"""
df[df.duplicated()].tail(3)
```

Out[6]:

	brand	processor_brand	processor_name	processor_gnrtn	ram_gb	ram_type	ssd	hdd
605	APPLE	M1	M1	10th	8 GB	DDR4	256 GB	0 GB
616	APPLE	M1	M1	10th	8 GB	DDR4	512 GB	0 GB
622	APPLE	M1	M1	10th	16 GB	DDR4	1024 GB	0 GB

In [7]:
```
df.drop_duplicates(inplace=True)
```

In [8]:
```
"""Shape of dataset after dropping the duplicate rows"""
df.shape
```

Out[8]:
```
(802, 19)
```

In [9]:
```
"""Checking for null values if any available"""
df.isnull().sum()
```

Out[9]:
```
brand                0
processor_brand      0
processor_name       0
processor_gnrtn      0
ram_gb               0
ram_type             0
ssd                  0
hdd                  0
os                   0
os_bit               0
graphic_card_gb      0
weight               0
warranty             0
Touchscreen          0
msoffice             0
Price                0
rating               0
Number of Ratings    0
Number of Reviews    0
dtype: int64
```

In [10]:
```
""" This is showing no null values are present but while going through dat
this column contains 'Not Available' values instead of Null so will try to

df["processor_gnrtn"].replace("Not Available", None , inplace=True)
```

In [11]:
```
df.isnull().sum()
```

Out[11]:
```
brand                    0
processor_brand          0
processor_name           0
processor_gnrtn        224
ram_gb                   0
ram_type                 0
ssd                      0
hdd                      0
os                       0
os_bit                   0
graphic_card_gb          0
weight                   0
warranty                 0
Touchscreen              0
msoffice                 0
Price                    0
rating                   0
Number of Ratings        0
Number of Reviews        0
dtype: int64
```

In [12]:
```
print(f"For {round((224/df.shape[0])*100,3)}% of data values are not prese
"""Based on the analysis, processor_gnrtn column with a high percentage of
df.drop("processor_gnrtn", axis=1, inplace= True)
```

```
For 27.93% of data values are not present
```

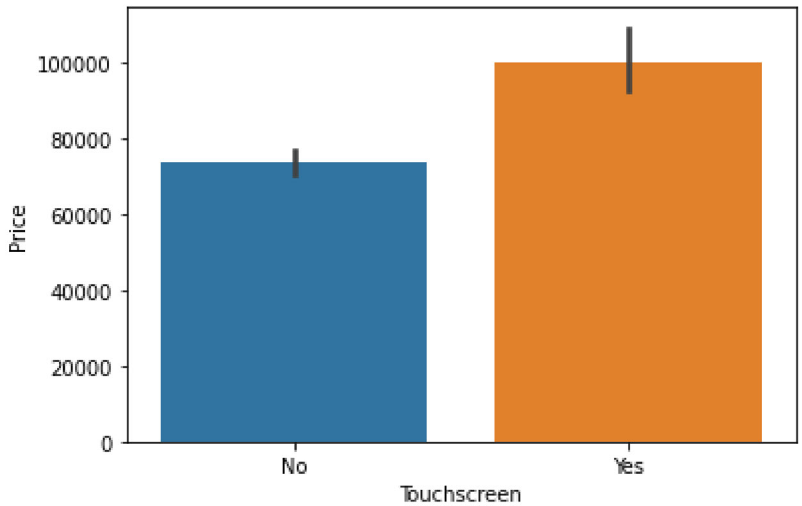
Touch Screen

In [13]:
```
df["Touchscreen"].value_counts()
```

Out[13]:
```
No     706
Yes     96
Name: Touchscreen, dtype: int64
```

In [14]:
```
sns.barplot(x=df['Touchscreen'],y=df['Price'])
```

Out[14]: <Axes: xlabel='Touchscreen', ylabel='Price'>



We can see from the plot, that the average price of laptop is higher with Touchscreen.

In [15]:
```
# converting this Touchscreen object coulmn in to int
df["Touchscreen"]= df["Touchscreen"].apply(lambda x: 1 if x=="Yes" else 0
```

Warranty

In [16]:
```
df["warranty"].value_counts()
```

Out[16]:
```
1 year         498
No warranty    268
2 years         23
3 years         13
Name: warranty, dtype: int64
```

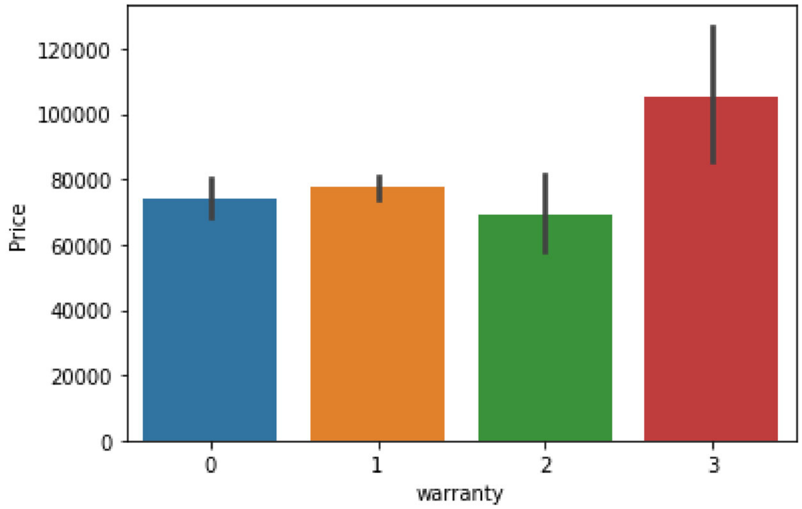
In [17]:
```
"""
We will transform this column into int
No warranty    0
1 year         1
2 years        2
3 years        3
"""
def warranty_tranform(x):
    if x== "No warranty": return 0
    elif x == "1 year": return 1
    elif x == "2 years": return 2
    else: return 3

df["warranty"] = df["warranty"].apply(warranty_tranform)
```

In [18]:

```
""" We will check the afftect of warranty duration on Laptop Price"""
sns.barplot(x=df["warranty"],y=df['Price'])
```

Out[18]: <Axes: xlabel='warranty', ylabel='Price'>



MSOffice

In [19]:

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

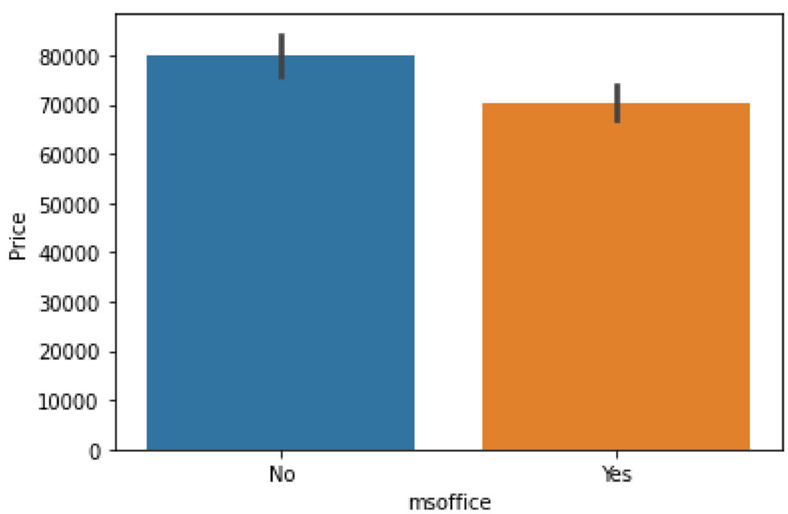
Out[19]:

```
No     522
Yes    280
Name: msoffice, dtype: int64
```

In [20]:

```
sns.barplot(x=df["msoffice"],y=df['Price'])
```

Out[20]: <Axes: xlabel='msoffice', ylabel='Price'>



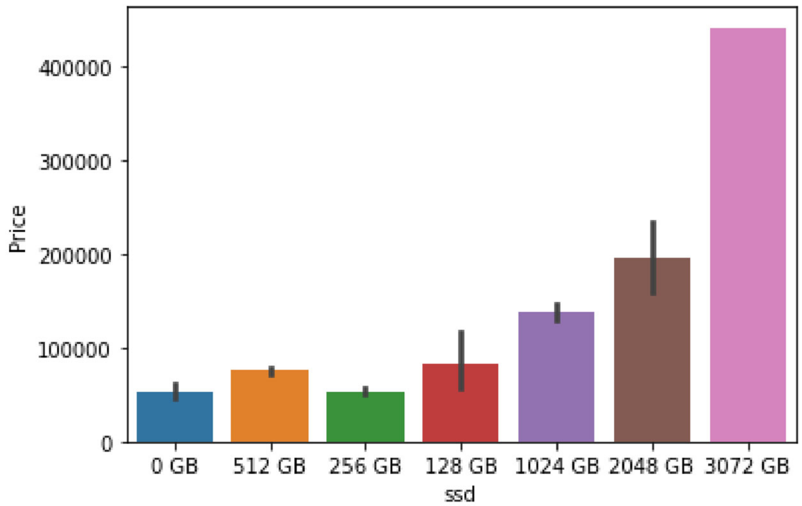
The bar plot indicates that laptops with MS Office have a lower average price compared to laptops without MS Office.

In [21]:
```
# converting this msoffice object coulmn in to int
df["msoffice"]= df["msoffice"].apply(lambda x: 1 if x=="Yes" else 0)
```

SSD

In [22]:
```
""" Comparing SSD with Price """
sns.barplot(x=df["ssd"],y=df['Price'])
```

Out[22]: <Axes: xlabel='ssd', ylabel='Price'>



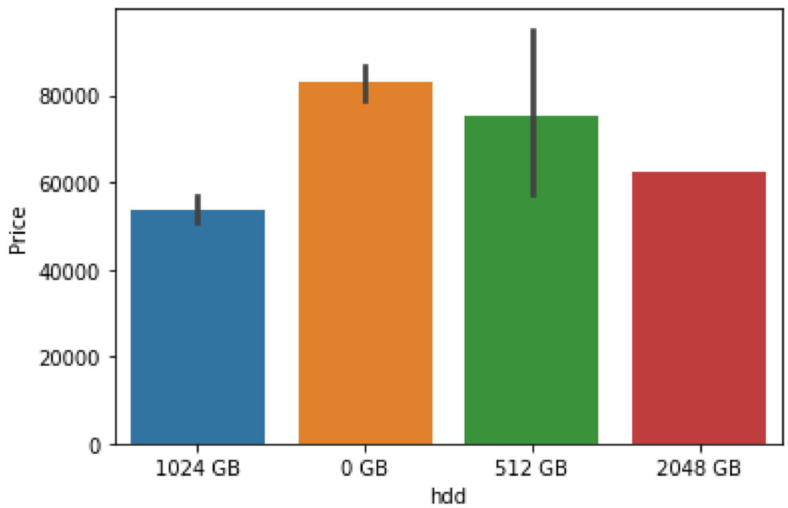
It is observed from the plot, with increase in SSD the laptop price increases.

In [23]:
```
""" Remove GB from SSD column values and convert it from object to int."""
df["ssd"] = df["ssd"].str.replace("GB","")
df["ssd"] = df["ssd"].astype('int')
```

HDD

In [24]:
```
""" Comparing HDD with Price """
sns.barplot(x=df["hdd"],y=df['Price'])
```

Out[24]: <Axes: xlabel='hdd', ylabel='Price'>

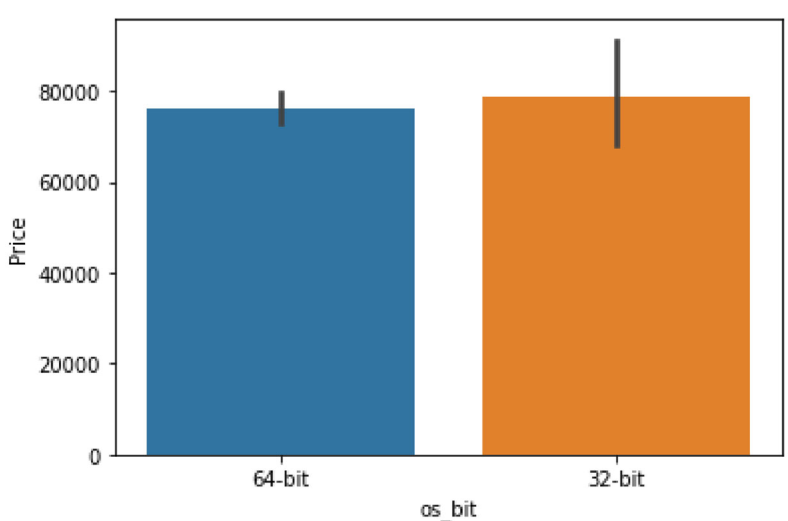


In [25]:
```
""" Remove GB from SSD column values and convert it from object to int."""
df["hdd"] = df["hdd"].str.replace("GB","")
df["hdd"] = df["hdd"].astype('int')
```

OS-Bit

In [26]:
```
""" Comparing OS-Bit with Price """
sns.barplot(x=df["os_bit"],y=df['Price'])
```

Out[26]: <Axes: xlabel='os_bit', ylabel='Price'>

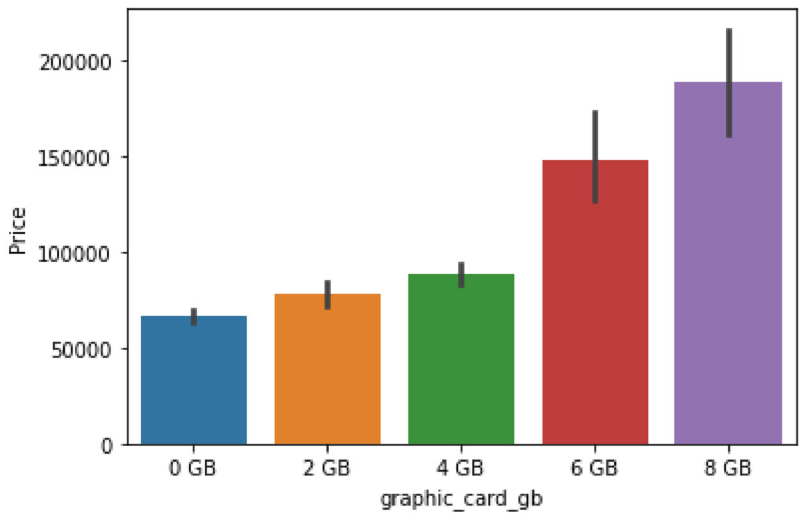


There is almost no effect of OS-Bit on price of Laptop

Graphic_Card_GB

In [27]:
```
""" Comparing graphic_card_gb with Price """
sns.barplot(x=df["graphic_card_gb"],y=df['Price'])
```

Out[27]: <Axes: xlabel='graphic_card_gb', ylabel='Price'>



It can be clearly seen that as graphic_card_gb size increases laptop price increases linearly

In [28]:
```
""" Remove GB from 'graphic_card_gb' column values and convert it from obj
df["graphic_card_gb"] = df["graphic_card_gb"].str.replace("GB","")
df["graphic_card_gb"] = df["graphic_card_gb"].astype('int')
```

In [29]:
```
df.corr()['Price']
```

Out[29]:
```
ssd                  0.628734
hdd                 -0.251266
graphic_card_gb      0.467499
warranty             0.057953
Touchscreen          0.191227
msoffice            -0.103783
Price                1.000000
Number of Ratings   -0.152553
Number of Reviews   -0.156791
Name: Price, dtype: float64
```

In [30]:
```
df
```

Out[30]:

	brand	processor_brand	processor_name	ram_gb	ram_type	ssd	hdd	os	os_bi
0	ASUS	Intel	Core i3	4 GB	DDR4	0	1024	Windows	64-bi
1	Lenovo	Intel	Core i3	4 GB	DDR4	0	1024	Windows	64-bi
2	Lenovo	Intel	Core i3	4 GB	DDR4	0	1024	Windows	64-bi
3	ASUS	Intel	Core i5	8 GB	DDR4	512	0	Windows	32-bi
4	ASUS	Intel	Celeron Dual	4 GB	DDR4	0	512	Windows	64-bi
...	...	...	...	...	...	...	...	...	..
818	ASUS	AMD	Ryzen 9	4 GB	DDR4	1024	0	Windows	64-bi
819	ASUS	AMD	Ryzen 9	4 GB	DDR4	1024	0	Windows	64-bi
820	ASUS	AMD	Ryzen 9	4 GB	DDR4	1024	0	Windows	64-bi
821	ASUS	AMD	Ryzen 9	4 GB	DDR4	1024	0	Windows	64-bi
822	Lenovo	AMD	Ryzen 5	8 GB	DDR4	512	0	DOS	64-bi

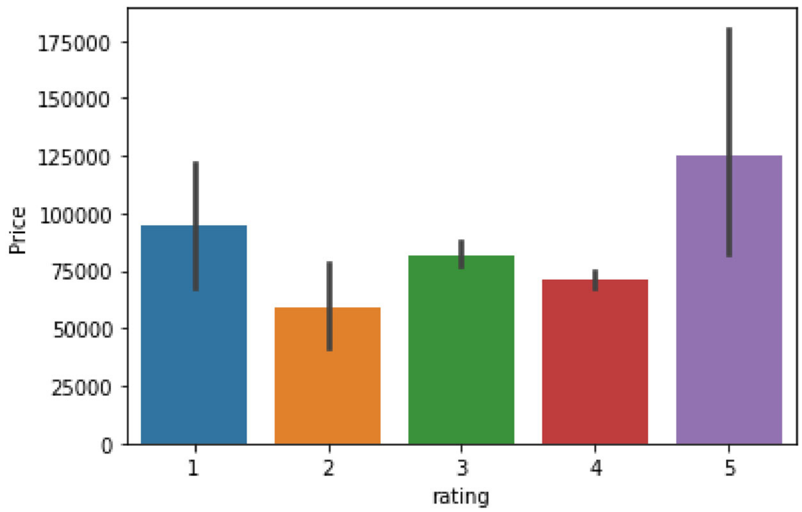
802 rows × 18 columns

Rating

In [31]:
```
""" Rating values are in range of 1 to 5 i.e. worst to good. Remove stars
pattern = '|'.join(['stars', 'star'])
df['rating'] = df['rating'].str.replace(pattern, '')
df["rating"] = df["rating"].astype('int')
```

In [32]:
```
""" Comparing Rating with Price """
sns.barplot(x=df["rating"],y=df['Price'])
```

Out[32]: <Axes: xlabel='rating', ylabel='Price'>



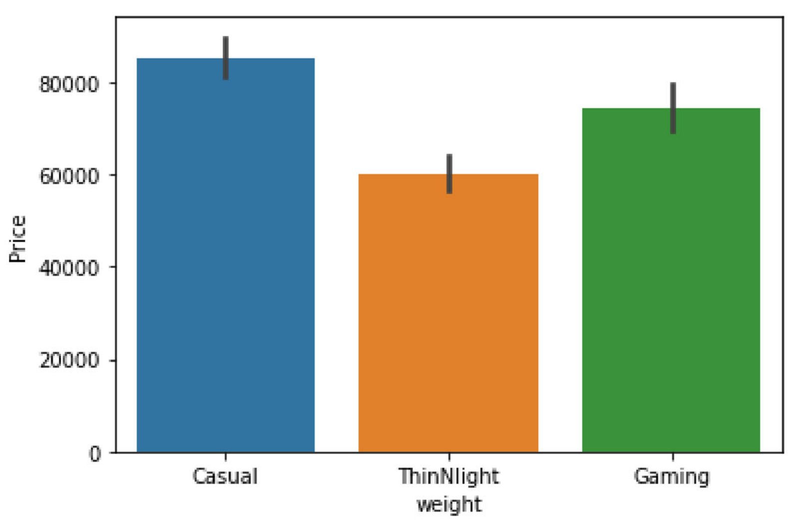
Weight

In [33]:
```
df["weight"].value_counts()
```

Out[33]:
```
Casual        509
ThinNlight    254
Gaming         39
Name: weight, dtype: int64
```

In [34]:
```
""" Comparing weight with Price """
sns.barplot(x=df["weight"],y=df['Price'])
```

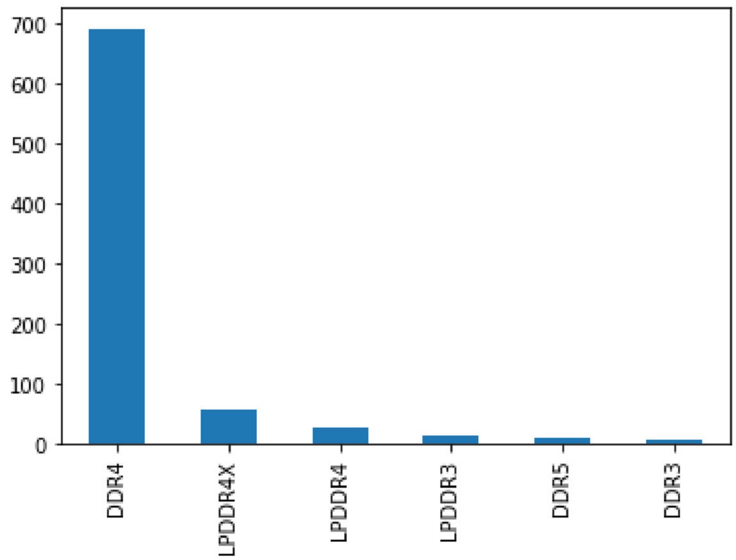
Out[34]: <Axes: xlabel='weight', ylabel='Price'>



Ram_type

In [35]:
```
df['ram_type'].value_counts().plot(kind='bar')
```

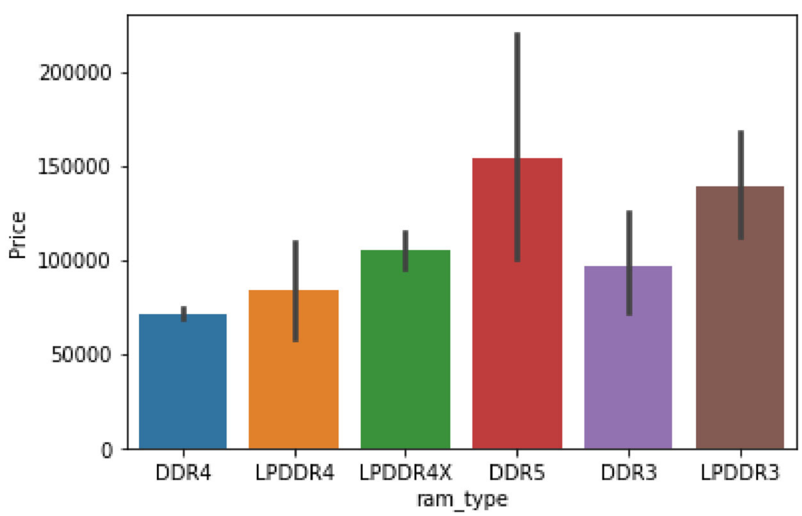
Out[35]: <Axes: >



observation: most of the laptops are of DDR4 type

In [36]:
```
""" Comparing ram_type with Price """
sns.barplot(x=df["ram_type"],y=df['Price'])
```

Out[36]: <Axes: xlabel='ram_type', ylabel='Price'>



Based on the observed plot, it is evident that the "ram_type" column has an impact on the price of laptops. Considering the nature of this categorical column as ordinal, we can handle it accordingly.

In [37]:
```
# Define the mapping for ordinal encoding
mapping = {
    'DDR4':    1,
    'LPDDR4X': 4,
    'LPDDR4':  2,
    'LPDDR3':  5,
    'DDR5':    6,
    'DDR3':    3,
}
# Perform ordinal encoding
df['ram_type'] = df['ram_type'].map(mapping)

```

In [38]:
```
df.corr()["Price"]
```

Out[38]:
```
ram_type              0.316037
ssd                   0.628734
hdd                  -0.251266
graphic_card_gb       0.467499
warranty              0.057953
Touchscreen           0.191227
msoffice             -0.103783
Price                 1.000000
rating               -0.040564
Number of Ratings    -0.152553
Number of Reviews    -0.156791
Name: Price, dtype: float64
```

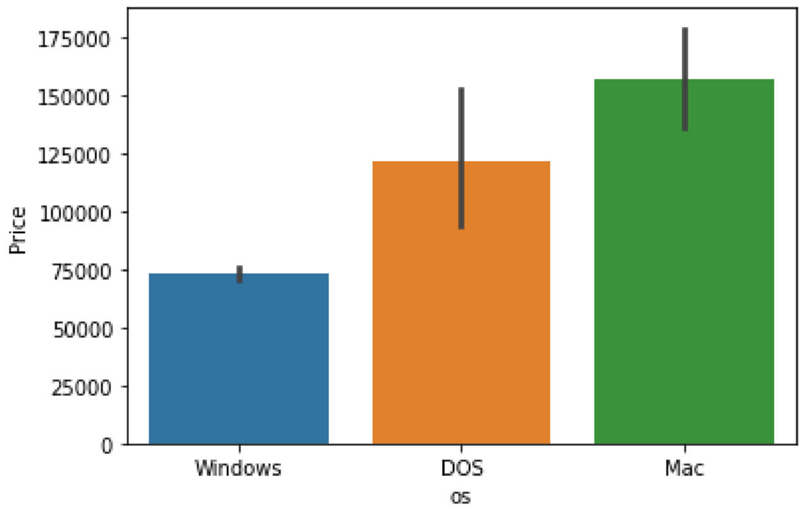
OS

In [39]:
```
df["os"].value_counts()
```

Out[39]:
```
Windows    763
Mac         23
DOS         16
Name: os, dtype: int64
```

In [40]:
```
""" Comparing os with Price """
sns.barplot(x=df["os"],y=df['Price'])
```

Out[40]: <Axes: xlabel='os', ylabel='Price'>



In [ ]:
```

```

In [41]:
```
df
```

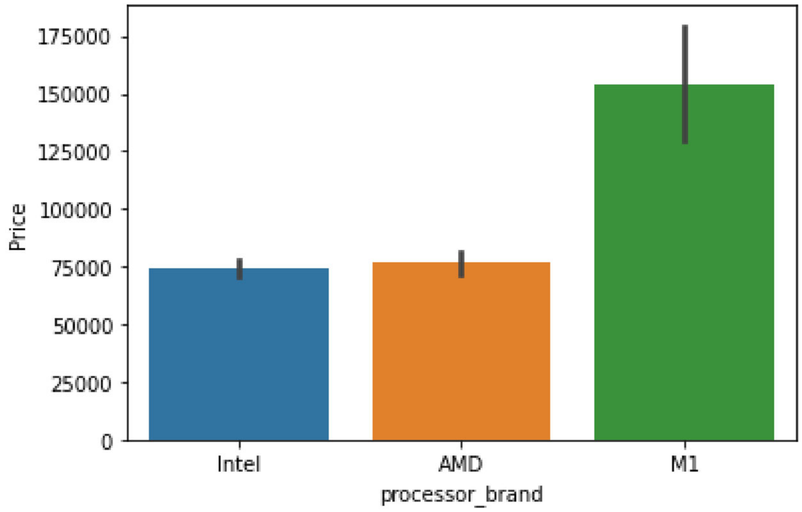
Out[41]:

	brand	processor_brand	processor_name	ram_gb	ram_type	ssd	hdd	os	os_bi
0	ASUS	Intel	Core i3	4 GB	1	0	1024	Windows	64-bi
1	Lenovo	Intel	Core i3	4 GB	1	0	1024	Windows	64-bi
2	Lenovo	Intel	Core i3	4 GB	1	0	1024	Windows	64-bi
3	ASUS	Intel	Core i5	8 GB	1	512	0	Windows	32-bi
4	ASUS	Intel	Celeron Dual	4 GB	1	0	512	Windows	64-bi
...	...	...	...	...	...	...	...	...	..
818	ASUS	AMD	Ryzen 9	4 GB	1	1024	0	Windows	64-bi
819	ASUS	AMD	Ryzen 9	4 GB	1	1024	0	Windows	64-bi
820	ASUS	AMD	Ryzen 9	4 GB	1	1024	0	Windows	64-bi
821	ASUS	AMD	Ryzen 9	4 GB	1	1024	0	Windows	64-bi
822	Lenovo	AMD	Ryzen 5	8 GB	1	512	0	DOS	64-bi

802 rows × 18 columns

In [42]:
```
""" Comparing processor_brand with Price """
sns.barplot(x=df["processor_brand"],y=df['Price'])
```

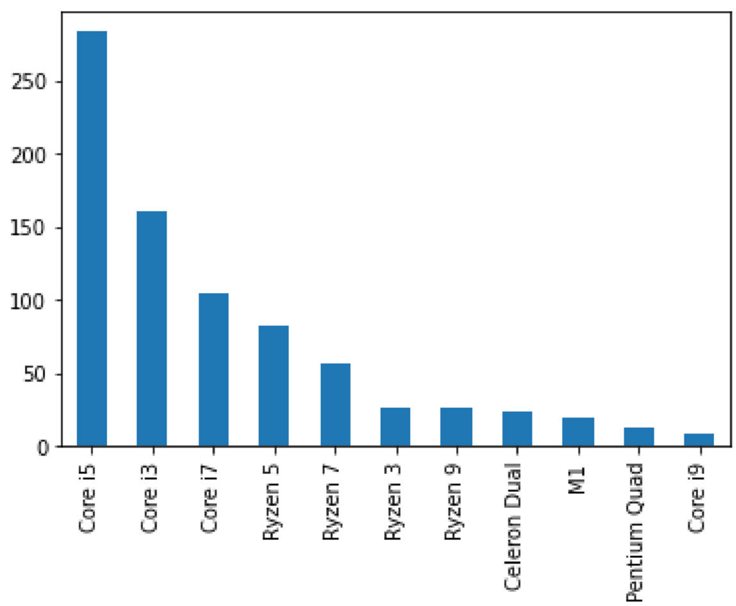
Out[42]: <Axes: xlabel='processor_brand', ylabel='Price'>



Processor Name

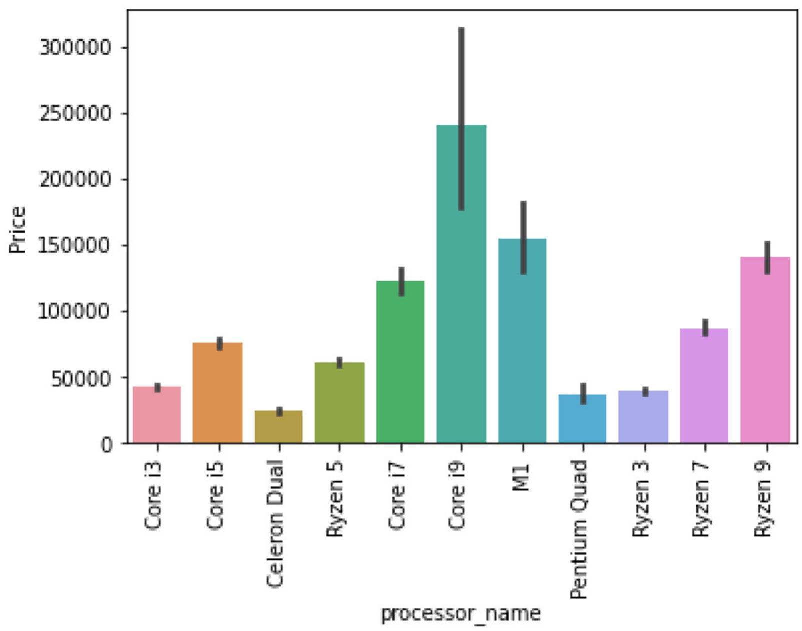
In [43]:
```
"""Laptop data distribution based on processor_name"""
df['processor_name'].value_counts().plot(kind='bar')
```

Out[43]: <Axes: >



In [44]:
```
""" Comparing processor_name with Price """
sns.barplot(x=df["processor_name"],y=df['Price'])
plt.xticks(rotation = 90)
```

Out[44]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
 [Text(0, 0, 'Core i3'),
  Text(1, 0, 'Core i5'),
  Text(2, 0, 'Celeron Dual'),
  Text(3, 0, 'Ryzen 5'),
  Text(4, 0, 'Core i7'),
  Text(5, 0, 'Core i9'),
  Text(6, 0, 'M1'),
  Text(7, 0, 'Pentium Quad'),
  Text(8, 0, 'Ryzen 3'),
  Text(9, 0, 'Ryzen 7'),
  Text(10, 0, 'Ryzen 9')])
```



In [45]:
```
df["processor_name"].value_counts()
```

Out[45]:
```
Core i5         284
Core i3         161
Core i7         104
Ryzen 5          82
Ryzen 7          56
Ryzen 3          26
Ryzen 9          26
Celeron Dual     23
M1               19
Pentium Quad     13
Core i9           8
Name: processor_name, dtype: int64
```

The bar plot clearly indicates that the "Processor Brand" feature is an ordinal categorical variable, as the version of the processor directly impacts its price.

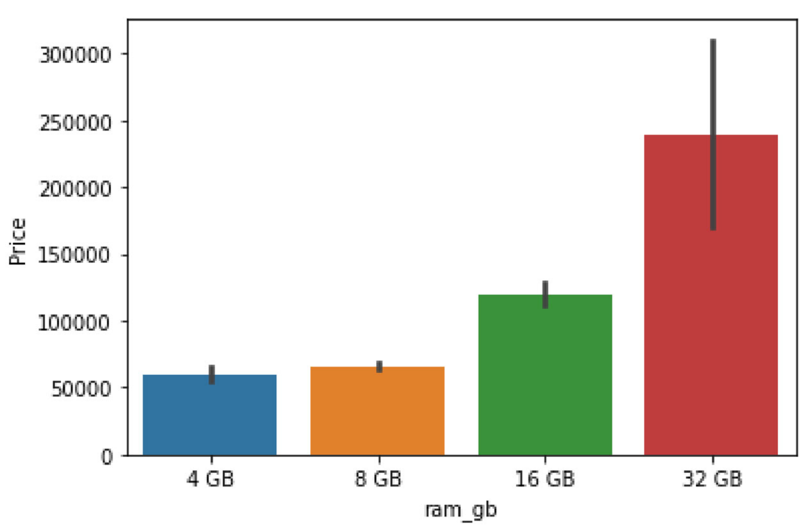
In [46]:
```
"""
The processor_name feature column will be transformed into an ordinal cate
"""
def processor_name_transform(x):
    if x == "Celeron Dual":
        return 1
    elif x == "Pentium Quad" or x == "Ryzen 3":
        return 2
    elif x == "Core i3":
        return 3
    elif x == "Ryzen 5":
        return 4
    elif x == "Core i5":
        return 5
    elif x == "Ryzen 7":
        return 6
    elif x == "Core i7":
        return 7
    elif x == "Ryzen 9":
        return 8
    elif x == "M1":
        return 9
    elif x == "Core i9":
        return 10


df["processor_name"] = df["processor_name"].apply(processor_name_transforr
```

Ram

In [47]:
```
""" Comparing ram_gb with Price """
sns.barplot(x=df["ram_gb"],y=df['Price'])
```

Out[47]: <Axes: xlabel='ram_gb', ylabel='Price'>



Here also, the bar plot clearly indicates that the "Ram Size" feature is an ordinal categorical variable, as the size of the ram directly impacts its price.

In [48]:
```
""" Remove GB from ram_gb column values and convert it from object to int
df["ram_gb"] = df["ram_gb"].str.replace("GB","")
df["ram_gb"] = df["ram_gb"].astype('int')
```

In [49]:
```
df.corr()["Price"]
```

Out[49]:
```
processor_name       0.745720
ram_gb               0.516454
ram_type             0.316037
ssd                  0.628734
hdd                 -0.251266
graphic_card_gb      0.467499
warranty             0.057953
Touchscreen          0.191227
msoffice            -0.103783
Price                1.000000
rating              -0.040564
Number of Ratings   -0.152553
Number of Reviews   -0.156791
Name: Price, dtype: float64
```

In [50]:
```
df
```

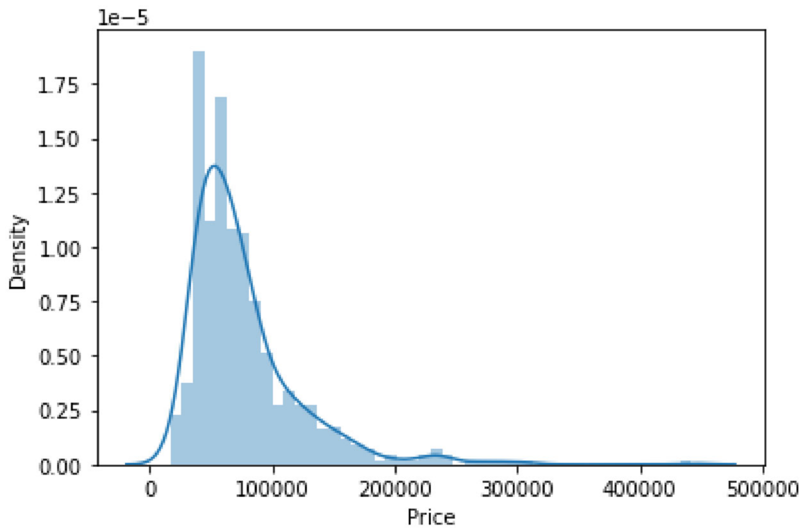
Out[50]:

	brand	processor_brand	processor_name	ram_gb	ram_type	ssd	hdd	os	os_bi
0	ASUS	Intel	3	4	1	0	1024	Windows	64-bi
1	Lenovo	Intel	3	4	1	0	1024	Windows	64-bi
2	Lenovo	Intel	3	4	1	0	1024	Windows	64-bi
3	ASUS	Intel	5	8	1	512	0	Windows	32-bi
4	ASUS	Intel	1	4	1	0	512	Windows	64-bi
...	...	...	...	...	...	...	...	...	..
818	ASUS	AMD	8	4	1	1024	0	Windows	64-bi
819	ASUS	AMD	8	4	1	1024	0	Windows	64-bi
820	ASUS	AMD	8	4	1	1024	0	Windows	64-bi
821	ASUS	AMD	8	4	1	1024	0	Windows	64-bi
822	Lenovo	AMD	4	8	1	512	0	DOS	64-bi

802 rows × 18 columns

In [51]:
```
"""Checking the distribtion of price column"""
sns.distplot(df['Price'])
```

Out[51]: <Axes: xlabel='Price', ylabel='Density'>



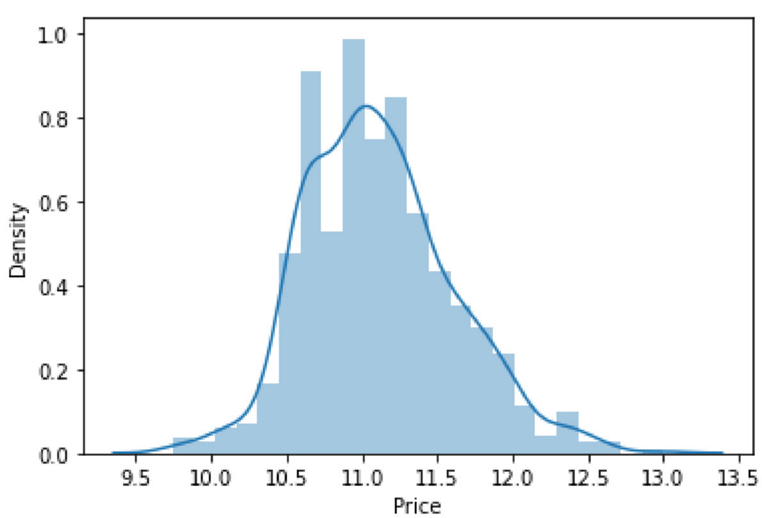
The distribution plot reveals a right-skewed pattern, indicating a higher concentration of data towards the lower values.

Applying a log transformation to a right-skewed distribution can help in achieving a more symmetrical or normally distributed shape, which is often preferred in statistical modeling and analysis.

This transformation can be beneficial because many statistical techniques assume normality, allowing for more accurate and reliable analysis results.

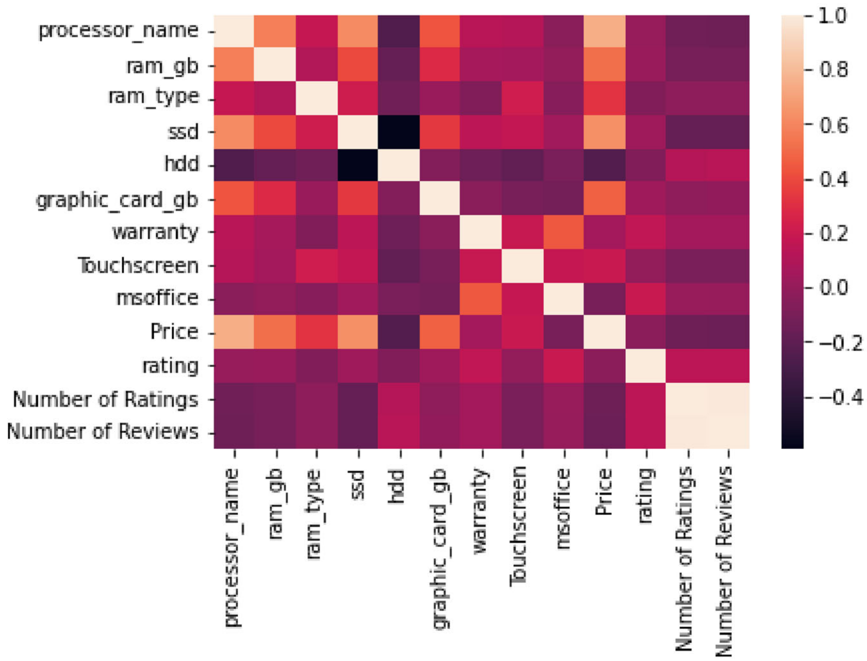
In [52]:
```
sns.distplot(np.log(df['Price']))
```

Out[52]: <Axes: xlabel='Price', ylabel='Density'>



In [53]:
```
"""Heatmap Analysis"""
sns.heatmap(df.corr())
```

Out[53]: <Axes: >



The varying intensities of the heatmap highlight the strength and direction of these correlations, aiding in identifying key factors that influence the pricing of laptops.

The features having high correlations with the price of laptop are:

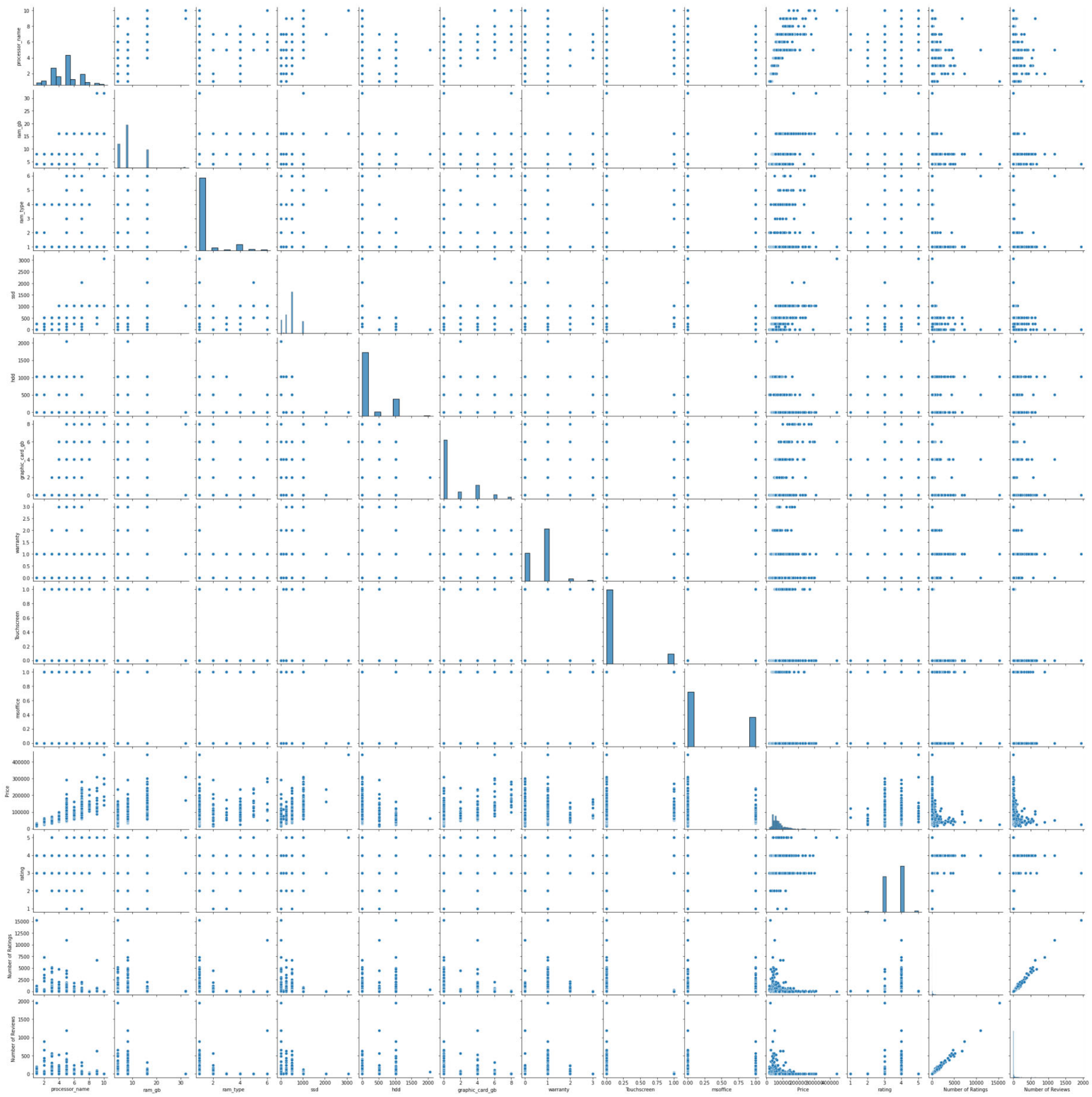
1. processor_name
2. ram_gb
3. ssd
4. ram_type
5. graphic_card_gb

In [54]:
```
df.corr()["Price"]
```

Out[54]:
```
processor_name       0.745720
ram_gb               0.516454
ram_type             0.316037
ssd                  0.628734
hdd                 -0.251266
graphic_card_gb      0.467499
warranty             0.057953
Touchscreen          0.191227
msoffice            -0.103783
Price                1.000000
rating              -0.040564
Number of Ratings   -0.152553
Number of Reviews   -0.156791
Name: Price, dtype: float64
```

In [66]:
```
sns.pairplot(df)
```

Out[66]:
```
<seaborn.axisgrid.PairGrid at 0x19c46942e20>
```



D. Input Variables and Target seperation and Splitting into train and test data

In [55]:
```
""" Seperating the input variables and Target into X and y repectively"""
X = df.drop(columns=['Price'])
y = np.log(df['Price'])
```

In [56]:
```
"""Splitting our dataset into Train and Validation"""
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.15,rando
```

E.Applying Machine Learning Model for Laptop Price

E1. Linear Regression Model

In [57]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import r2_score,mean_absolute_error
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import RandomizedSearchCV,GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor
from scipy.stats import randint
```

In [58]:
```
"""Converting Nominal Categorical Fetures into One Hot Encodings"""

col_tnf = ColumnTransformer(transformers=[
    ('col_tnf',OneHotEncoder(sparse=False,drop='first'),[0,1,7,8,10])
],remainder='passthrough')

X_train_transformed = col_tnf.fit_transform(X_train)
X_test_transformed = col_tnf.transform(X_test)
```

In [59]:
```
LinearRegressionModel = LinearRegression()

# fit and prediction
LinearRegressionModel.fit(X_train_transformed,y_train)
y_pred = LinearRegressionModel.predict(X_test_transformed)

# Calculate R2 score and MAE
print('R2 score:',r2_score(y_test,y_pred))
print('MAE:',mean_absolute_error(y_test,y_pred))
```

```
R2 score: 0.8492439253870319
MAE: 0.1459487523971702
```

E2. KNN Regression Model

In [60]:

```
KNNModel = KNeighborsRegressor()
# Create a parameter grid for randomized search
param_grid = {
    'n_neighbors': np.arange(1, 20),
}
# Randomized Search CV
rs_cv = RandomizedSearchCV(
    estimator=KNNModel,
    param_distributions=param_grid,
    scoring='r2',
    n_iter=10,  # Number of parameter settings that are sampled
    cv=5,       # Number of folds in cross-validation
    random_state=42
)

# fit and prediction
rs_cv.fit(X_train_transformed, y_train)
y_pred = rs_cv.predict(X_test_transformed)

# Calculate R2 score and MAE
print('R2 score:',r2_score(y_test,y_pred))
print('MAE:',mean_absolute_error(y_test,y_pred))
```

```
R2 score: 0.5743328756286931
MAE: 0.2554921623082943
```

E3.Decision Tree Regression Model

In [61]:

```
dtr = DecisionTreeRegressor()
# Create a parameter grid for randomized search
param_grid = {
    'max_depth': np.arange(1, 10),
    'min_samples_split': np.arange(2, 11),
    'min_samples_leaf': np.arange(1, 11)
}
# Randomized Search CV
rs_cv = RandomizedSearchCV(
    estimator=dtr,
    param_distributions=param_grid,
    scoring='r2',
    n_iter=10,  # Number of parameter settings that are sampled
    cv=5,       # Number of folds in cross-validation
    random_state=42
)
rs_cv.fit(X_train_transformed, y_train)
best_params = rs_cv.best_params_
dtr_best = DecisionTreeRegressor(**best_params)

# fit and prediction
dtr_best.fit(X_train_transformed, y_train)
y_pred = dtr_best.predict(X_test_transformed)

# Calculate R2 score and MAE
print('R2 score:',r2_score(y_test,y_pred))
print('MAE:',mean_absolute_error(y_test,y_pred))
print('Best parameters:', best_params)

```

```
R2 score: 0.8573330917319021
MAE: 0.1454562949028369
Best parameters: {'min_samples_split': 4, 'min_samples_leaf': 4, 'max_depth': 7}
```

E4. Gradient Boosting Model

In [62]:

```
param_grid = {
    'n_estimators': randint(100, 1000),
    'learning_rate': [0.1, 0.01, 0.001],
    'max_depth': randint(3, 6)
}

GradientBoostingRegressorModel = GradientBoostingRegressor()

rs_cv = RandomizedSearchCV(GradientBoostingRegressorModel, param_distribu
rs_cv.fit(X_train_transformed, y_train)
best_params = rs_cv.best_params_
best_model = GradientBoostingRegressor(**best_params)

# fit and prediction
best_model.fit(X_train_transformed, y_train)
y_pred = best_model.predict(X_test_transformed)

# Calculate R2 score and MAE
print('R2 score:', r2_score(y_test, y_pred))
print('MAE:', mean_absolute_error(y_test, y_pred))
print('Best parameters:', best_params)

```

```
R2 score: 0.9107112564822367
MAE: 0.11991196744780869
Best parameters: {'learning_rate': 0.01, 'max_depth': 4, 'n_estimators': 869}
```

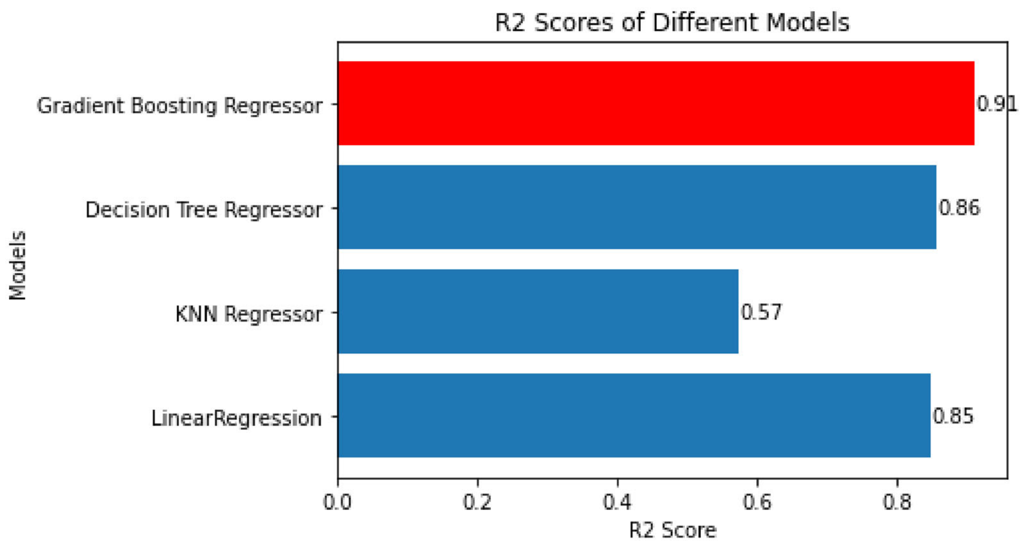
F. Plots of Different tried models results for analysis.

In [63]:

```
r2_scores = [0.8492, 0.5743, 0.8573, 0.9116]
model_names = ['LinearRegression', 'KNN Regressor', 'Decision Tree Regres
highest_index = np.argmax(r2_scores)
plt.barh(model_names, r2_scores)
for i, score in enumerate(r2_scores):
    plt.text(score, i, f'{score:.2f}', ha='left', va='center')
plt.barh(highest_index, r2_scores[highest_index], color='red')

plt.xlabel('R2 Score')
plt.ylabel('Models')
plt.title('R2 Scores of Different Models')
plt.show()

```



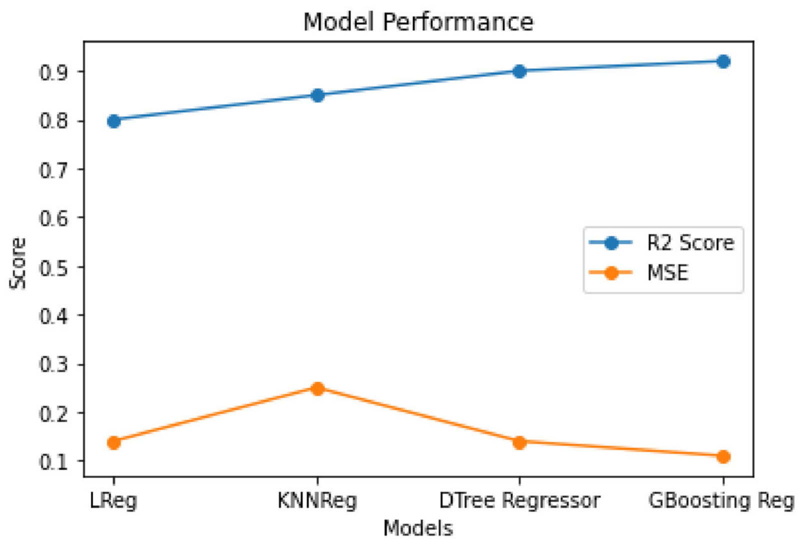
The R2 score measures the goodness of fit of a regression model. It ranges from 0 to 1, A score of 1 indicates that the model perfectly predicts the observed data.

As we can see from the plot Gradient Boosting Regressor model is performing best amongst the 4 models with highest R2 : 0.91

In [64]:
```
import matplotlib.pyplot as plt
r2_scores = [0.8, 0.85, 0.9, 0.92]
mse_values = [0.14, 0.25, 0.14, 0.11]
model_names = ['LReg', 'KNNReg', 'DTree Regressor', 'GBoosting Reg']
plt.plot(model_names, r2_scores, marker='o', label='R2 Score')
plt.plot(model_names, mse_values, marker='o', label='MSE')

plt.xlabel('Models')
plt.ylabel('Score')
plt.title('Model Performance')
plt.legend()
plt.show()

```



In [65]:
```
# G. Inference
```

We have selected Linear Regression (LR) as our first model because of its simplicity and interpretability, it is giving an r2 around 0.8 . LR models performance is limited when dealing with complex replationhips or non-linearities in the data.

To get more better results we have opted KNN Regressor as our second model, as it can capture non-linear patterns by considering the proximity of data points. Despite attempting various values for the n_neighbors parameter in the range of "1-20" for the KNN Regressor, we were still unable to achieve satisfactory results. it's possible that it might struggling with high-dimensional data or large datasets due to computational limitations.

As decision trees have ability to capture non-linear relationships, handle complex interactions between variables, be robust to outliers, and handle mixed data types effectively. We selected is as our next model. After performing Randomized Search CV on the Decision Tree Regressor (DTR) model, the best parameter combination that we got the best results was:

min_samples_split: 4 min_samples_leaf: 4 max_depth: 7

Type *Markdown* and LaTeX: α^2