```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
data = pd.read_csv('/content/sample_data/mission 4.csv')
```

```
data.head()
```

	Brand	Processor Type	RAM Size (GB)	Storage (GB)	Screen Size (inches)	Operating System	Price ($)
0	Acer	Intel Core i5	16	256	17.3	macOS	1808.865225
1	Acer	Intel Core i7	32	2048	13.3	macOS	2020.923055
2	Apple	AMD Ryzen 5	32	512	17.3	Linux	1152.453189
3	HP	AMD Ryzen 5	8	512	15.6	macOS	1884.457406
4	Lenovo	AMD Ryzen 7	64	256	17.3	macOS	2780.779164

```
data.isna().sum()
```

	0
Brand	0
Processor Type	0
RAM Size (GB)	0
Storage (GB)	0
Screen Size (inches)	0
Operating System	0
Price ($)	0

dtype: int64

```
data.columns
```

```
Index(['Brand', 'Processor Type', 'RAM Size (GB)', 'Storage (GB)',
       'Screen Size (inches)', 'Operating System', 'Price ($)'],
      dtype='object')
```

```
from sklearn.preprocessing import LabelEncoder
encode_cols = ['Brand', 'Processor Type', 'Operating System']
le = {}
for col in encode_cols:
  le[col] = LabelEncoder()
  data[col] = le[col].fit_transform(data[col])
```

```
data.head()
```

	Brand	Processor Type	RAM Size (GB)	Storage (GB)	Screen Size (inches)	Operating System	Price ($)
0	0	2	16	256	17.3	2	1808.865225
1	0	3	32	2048	13.3	2	2020.923055
2	1	0	32	512	17.3	0	1152.453189
3	3	0	8	512	15.6	2	1884.457406
4	4	1	64	256	17.3	2	2780.779164

```
scale_Cols = ['RAM Size (GB)', 'Storage (GB)', 'Screen Size (inches)']
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data[scale_Cols] = scaler.fit_transform(data[scale_Cols])
```

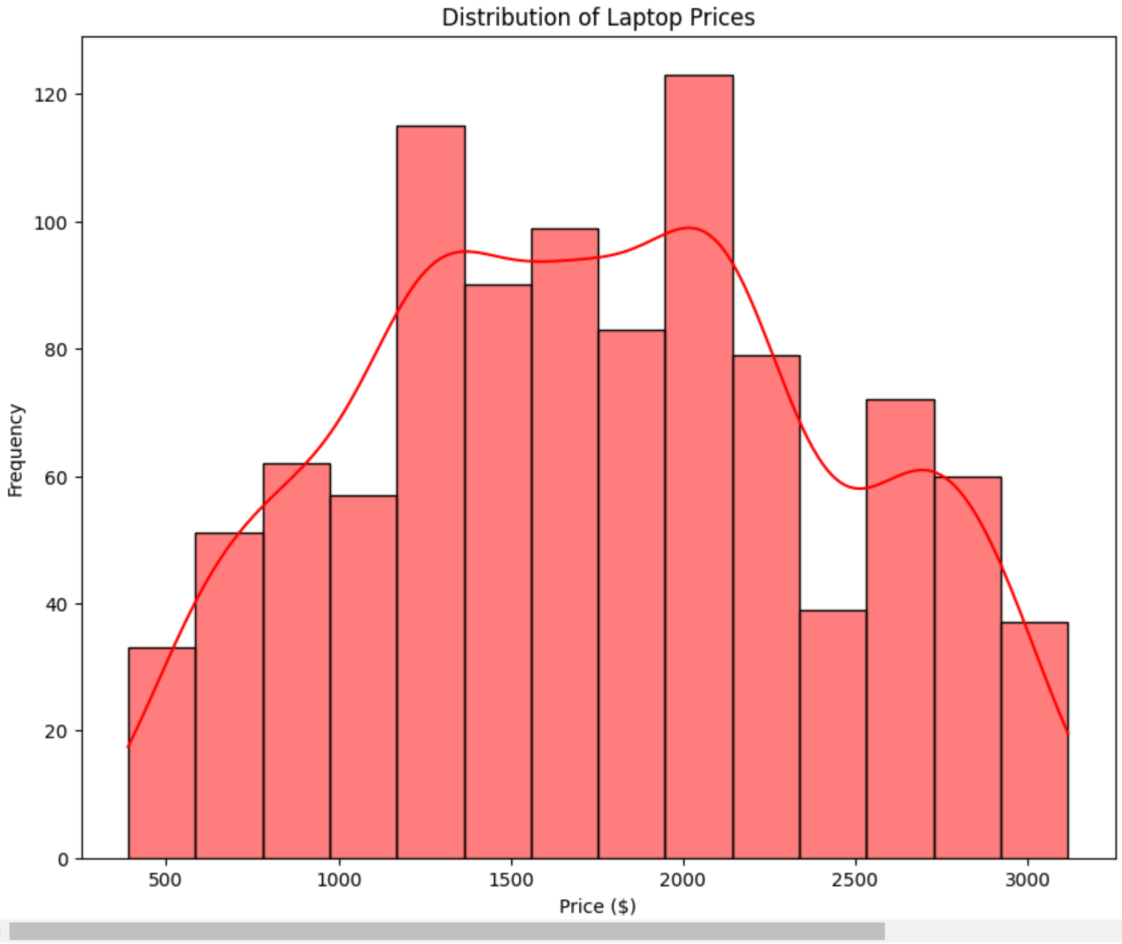
```
data.head()
```

```
   Brand  Processor Type  RAM Size (GB)  Storage (GB)  Screen Size (inches)  Operating System  Price ($)
0      0               2      -0.653598     -1.023895              1.413918                 2  1808.865225
1      0               3       0.083266      1.515887             -1.146020                 2  2020.923055
2      1               0       0.083266     -0.661069              1.413918                 0  1152.453189
3      3               0      -1.022029     -0.661069              0.325944                 2  1884.457406
4      4               1       1.556992     -1.023895              1.413918                 2  2780.779164
```

```
data.describe()
```

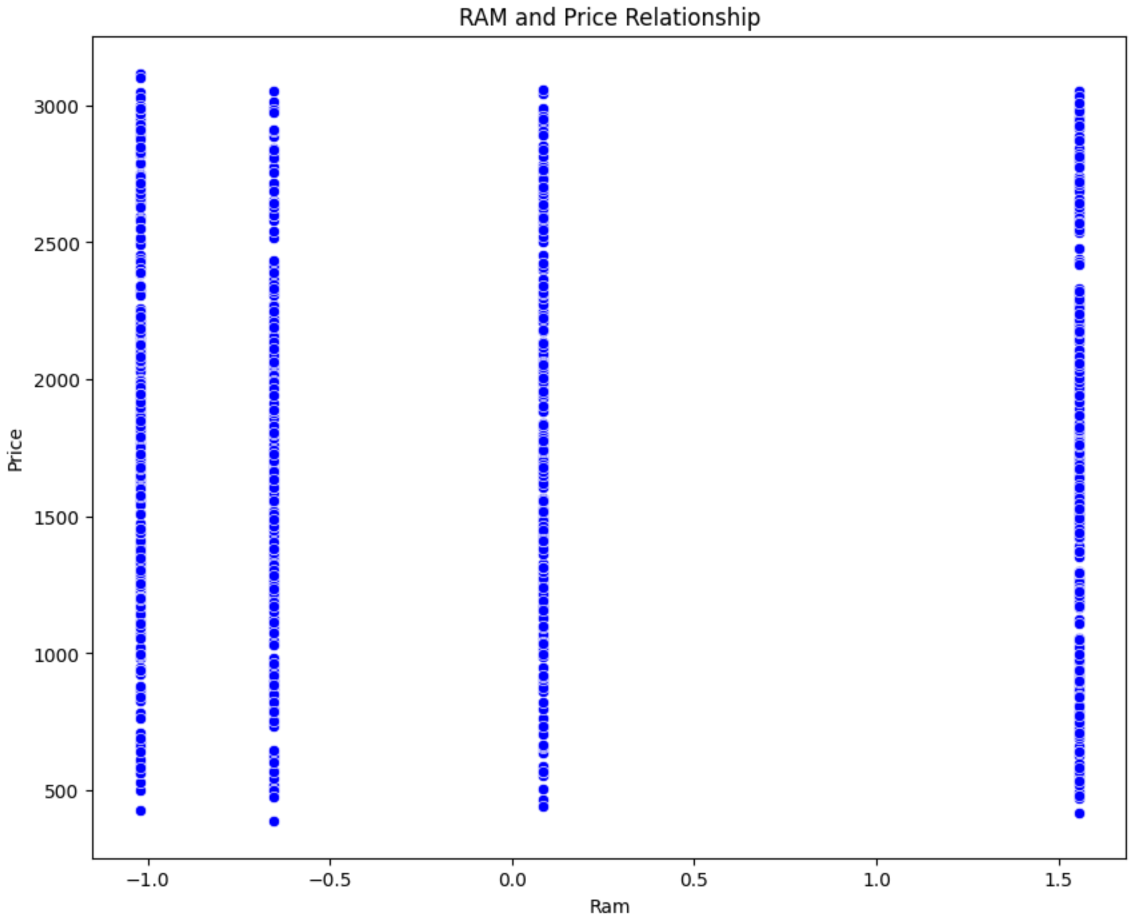
```
            Brand  Processor Type  RAM Size (GB)  Storage (GB)  Screen Size (inches)  Operating System    Price ($)
count  1000.00000     1000.000000   1.000000e+03  1.000000e+03          1.000000e+03       1000.000000  1000.000000
mean      1.89000        1.340000  -3.197442e-17 -2.042810e-17          1.945111e-16          1.120000  1752.871053
std       1.36373        1.116085   1.000500e+00  1.000500e+00          1.000500e+00          0.816251   671.389528
min       0.00000        0.000000  -1.022029e+00 -1.023895e+00         -1.146020e+00          0.000000   389.808247
25%       1.00000        0.000000  -7.457056e-01 -6.610691e-01         -1.146020e+00          0.000000  1233.948071
50%       2.00000        1.000000  -6.535977e-01 -6.610691e-01          3.259442e-01          1.000000  1738.449367
75%       3.00000        2.000000   1.556992e+00  1.515887e+00          1.413918e+00          2.000000  2215.303742
max       4.00000        3.000000   1.556992e+00  1.515887e+00          1.413918e+00          2.000000  3117.210896
```

```
plt.figure(figsize=(10, 8))
sns.histplot(data['Price ($)'], kde=True, color="red")
plt.xlabel('Price ($)')
plt.ylabel('Frequency')
plt.title('Distribution of Laptop Prices')
plt.show()
```



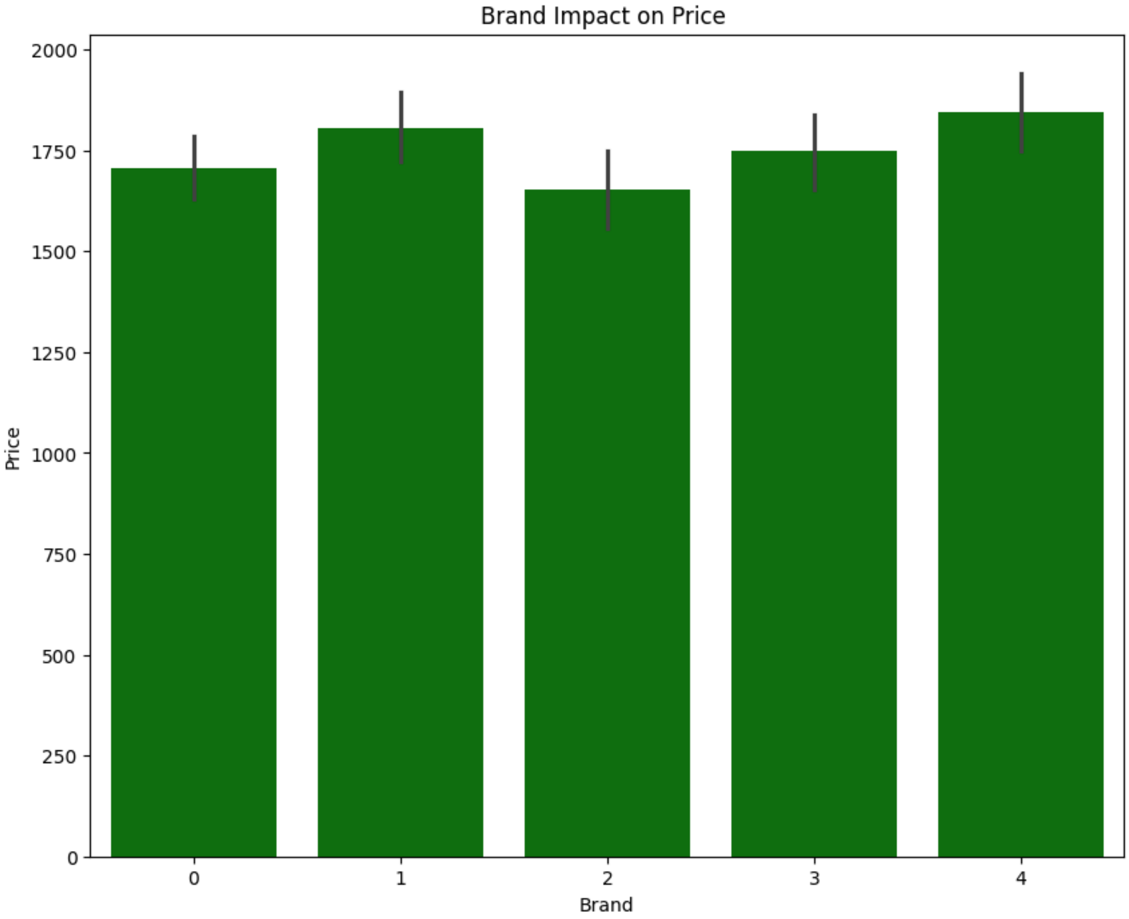
```
plt.figure(figsize=(10,8))
sns.scatterplot(x=data['RAM Size (GB)'], y=data['Price ($)'], color='Blue')
plt.xlabel('Ram')
plt.ylabel('Price')
plt.title('RAM and Price Relationship')
```

```
Text(0.5, 1.0, 'RAM and Price Relationship')
```



```
plt.figure(figsize=(10,8))
sns.barplot(x=data['Brand'], y=data['Price ($)'], color='Green')
plt.xlabel('Brand')
plt.ylabel('Price')
plt.title('Brand Impact on Price ')
```

```
Text(0.5, 1.0, 'Brand Impact on Price ')
```



```
from sklearn.model_selection import train_test_split
x = data.drop('Price ($)', axis=1)
y = data['Price ($)']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
x_train
```

	Brand	Processor Type	RAM Size (GB)	Storage (GB)	Screen Size (inches)	Operating System
29	0	1	-0.653598	-1.023895	-0.698031	1
535	4	0	-0.653598	1.515887	1.413918	0
695	0	0	0.083266	1.515887	0.325944	2
557	1	3	1.556992	1.515887	-0.698031	0
836	1	1	1.556992	-0.661069	-0.698031	0
...	...	...	...	...	...	...
106	4	0	0.083266	0.064583	-1.146020	1
270	4	0	0.083266	-1.023895	1.413918	2
860	1	3	1.556992	-0.661069	-1.146020	2
435	0	2	1.556992	1.515887	1.413918	0
102	4	3	-1.022029	0.064583	1.413918	1

800 rows × 6 columns

```
y_train
```

	Price ($)
29	2807.865134
535	1181.476005
695	1275.795790
557	747.523015
836	2774.574566
...	...
106	2285.394987
270	1981.674481
860	1566.686975
435	1351.000000
102	2253.110885

800 rows × 1 columns

dtype: float64

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

```
lr = LinearRegression()
lr.fit(x_train, y_train)
dt = DecisionTreeRegressor()
dt.fit(x_train, y_train)
rf = RandomForestRegressor()
rf.fit(x_train, y_train)
y_pred_lr = lr.predict(x_test)
y_pred_dt = dt.predict(x_test)
y_pred_rf = rf.predict(x_test)
```

```
print("Linear Regression:")
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_lr))
print("R-squared:", r2_score(y_test, y_pred_lr))
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred_lr))

print("Decision Tree Regression:")
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_dt))
print("R-squared:", r2_score(y_test, y_pred_dt))
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred_dt))

print("Random Forest Regression:")
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_rf))
print("R-squared:", r2_score(y_test, y_pred_rf))
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred_rf))
```

```
Linear Regression:
Mean Squared Error: 459090.8989545152
R-squared: 0.05026750906647026
Mean Absolute Error: 566.2744459316656
Decision Tree Regression:
Mean Squared Error: 574158.2116455722
R-squared: -0.18777503491768366
Mean Absolute Error: 502.8410628307085
Random Forest Regression:
Mean Squared Error: 379689.36189567775
R-squared: 0.2145273969156346
Mean Absolute Error: 464.2130240587683
```

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
random_params = {
    'n_estimators': [50, 100, 200, 250,300],
    'max_depth': [None, 10, 20, 30,35,40],

}
```

```
random_model = RandomizedSearchCV(RandomForestRegressor(), random_params, cv=5)
random_model.fit(x_train, y_train)
```

```
print("Best Parameters:", random_model.best_params_)
print("Best Score:", random_model.best_score_)
```

```
Best Parameters: {'n_estimators': 300, 'max_depth': 10}
Best Score: 0.3227072956323328
```

```
pip install gradio
```

```
Collecting gradio
  Downloading gradio-5.12.0-py3-none-any.whl.metadata (16 kB)
Collecting aiofiles<24.0,>=22.0 (from gradio)
  Downloading aiofiles-23.2.1-py3-none-any.whl.metadata (9.7 kB)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (3.7.1)
Collecting fastapi<1.0,>=0.115.2 (from gradio)
  Downloading fastapi-0.115.6-py3-none-any.whl.metadata (27 kB)
Collecting ffmpy (from gradio)
  Downloading ffmpy-0.5.0-py3-none-any.whl.metadata (3.0 kB)
Collecting gradio-client==1.5.4 (from gradio)
  Downloading gradio_client-1.5.4-py3-none-any.whl.metadata (7.1 kB)
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.10/dist-packages (from gradio) (0.28.1)
Requirement already satisfied: huggingface-hub>=0.25.1 in /usr/local/lib/python3.10/dist-packages (from gradio) (0.27.1)
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (3.1.5)
Collecting markupsafe~=2.0 (from gradio)
  Downloading MarkupSafe-2.1.5-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.0 kB)
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (1.26.4)
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (3.10.13)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from gradio) (24.2)
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (2.2.2)
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (11.1.0)
Requirement already satisfied: pydantic>=2.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (2.10.4)
Collecting pydub (from gradio)
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting python-multipart>=0.0.18 (from gradio)
  Downloading python_multipart-0.0.20-py3-none-any.whl.metadata (1.8 kB)
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (6.0.2)
Collecting ruff>=0.2.2 (from gradio)
  Downloading ruff-0.9.1-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (25 kB)
Collecting safehttpx<0.2.0,>=0.1.6 (from gradio)
  Downloading safehttpx-0.1.6-py3-none-any.whl.metadata (4.2 kB)
Collecting semantic-version~=2.0 (from gradio)
  Downloading semantic_version-2.10.0-py2.py3-none-any.whl.metadata (9.7 kB)
Collecting starlette<1.0,>=0.40.0 (from gradio)
  Downloading starlette-0.45.2-py3-none-any.whl.metadata (6.3 kB)
Collecting tomlkit<0.14.0,>=0.12.0 (from gradio)
  Downloading tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.10/dist-packages (from gradio) (0.15.1)
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (4.12.2)
Collecting uvicorn>=0.14.0 (from gradio)
  Downloading uvicorn-0.34.0-py3-none-any.whl.metadata (6.5 kB)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from gradio-client==1.5.4->gradio) (2024.10.0)
Requirement already satisfied: websockets<15.0,>=10.0 in /usr/local/lib/python3.10/dist-packages (from gradio-client==1.5.4->gradio) (
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (from anyio<5.0,>=3.0->gradio) (3.10)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-packages (from anyio<5.0,>=3.0->gradio) (1.3.1)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<5.0,>=3.0->gradio) (1.2.2)
Collecting starlette<1.0,>=0.40.0 (from gradio)
  Downloading starlette-0.41.3-py3-none-any.whl.metadata (6.0 kB)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from httpx>=0.24.1->gradio) (2024.12.14)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.10/dist-packages (from httpx>=0.24.1->gradio) (1.0.7)
Requirement already satisfied: h11<0.15,>=0.13 in /usr/local/lib/python3.10/dist-packages (from httpcore==1.*->httpx>=0.24.1->gradio)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.25.1->gradio) (3.16.1)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.25.1->gradio) (2.32.3)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.25.1->gradio) (4.67.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas<3.0,>=1.0->gradio) (2.8
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas<3.0,>=1.0->gradio) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas<3.0,>=1.0->gradio) (2024.2)
```

```
import gradio as gr
def predict_laptop_price(brand, processor_type, ram_size, storage, screen, os):
  try:
    input_data = pd.DataFrame(
        {
            'Brand': [brand],
            'Processor Type': [processor_type],
            'RAM Size (GB)': [ram_size],
            'Storage (GB)': [storage],
            'Screen Size (inches)': [screen],
            'Operating System': [os]

        }
    )

    for col in encode_cols:
      input_data[col] = le[col].transform(input_data[col])
    input_data[scale_Cols] = scaler.transform(input_data[scale_Cols])
    prediction = random_model.best_estimator_.predict(input_data)
    return prediction[0]
  except Exception as e:
    return str(e)
gr.Interface(
      inputs=[
          gr.Dropdown(choices=list(data['Brand'].unique()), label='Brand'),
          gr.Dropdown(choices=list(data['Processor Type'].unique()), label='Processor Type'),
          gr.Number(label='RAM Size (GB)'),
          gr.Number(label='Storage (GB)'),
          gr.Number(label='Screen Size (inches)'),
          gr.Dropdown(choices=list(data['Operating System'].unique()), label='Operating System')
      ]
      , outputs=gr.Textbox(label='Predicted Price ($)'),
      fn=predict_laptop_price,
      title='Laptop Price Prediction',
      description='Enter the details of the laptop to predict its price.'
  ).launch()
```

```
Running Gradio in a Colab notebook requires sharing enabled. Automatically setting `share=True` (you can turn this off by setting `share

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://780f9f0b14c28fdedf.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working d
```

gradio

No interface is running right now