In [ ]:
```
import pandas as pd
import numpy as np
```

In [ ]:
```
data = pd.read_csv("/Users/nishantsingh/Desktop/dataacuu/laptop_cleaned2.csv")
```

In [ ]:
```
data.head()
```

Out[ ]:

	Unnamed: 0	Name	Brand	Price	Rating	Processor_brand	Processor_name	Processor_variant	Processor_gen	Core_per_processor	...	Graphics_name	Graphics_brand	Graphics_GB	Graphic
0	0	HP Victus 15-fb0157AX Gaming Laptop (AMD Ryzen...	HP	50399	4.30	AMD	AMD Ryzen 5	5600H	5.0	6.0	...	AMD Radeon RX 6500M	AMD	4.0	
1	1	Lenovo V15 G4 82YU00W7IN Laptop (AMD Ryzen 3 ...	Lenovo	26690	4.45	AMD	AMD Ryzen 3	7320U	7.0	4.0	...	AMD Radeon Graphics	AMD	NaN	
2	2	HP 15s-fq5007TU Laptop (12th Gen Core i3/ 8GB/...	HP	37012	4.65	Intel	Intel Core i3	1215U	12.0	6.0	...	Intel UHD Graphics	Intel	NaN	
3	3	Samsung Galaxy Book2 Pro 13 Laptop (12th Gen C...	Samsung	69990	4.75	Intel	Intel Core i5	1240P	12.0	12.0	...	Intel Iris Xe Graphics	Intel	NaN	
4	4	Tecno Megabook T1 Laptop (11th Gen Core i3/ 8G...	Tecno	23990	4.25	Intel	Intel Core i3	1115G4	11.0	2.0	...	Intel UHD Graphics	Intel	NaN	

5 rows × 29 columns

In [ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1020 entries, 0 to 1019
Data columns (total 29 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Unnamed: 0              1020 non-null   int64  
 1   Name                    1020 non-null   object 
 2   Brand                   1020 non-null   object 
 3   Price                   1020 non-null   int64  
 4   Rating                  1020 non-null   float64
 5   Processor_brand         1020 non-null   object 
 6   Processor_name          1020 non-null   object 
 7   Processor_variant       996 non-null    object 
 8   Processor_gen           891 non-null    float64
 9   Core_per_processor      1008 non-null   float64
 10  Total_processor         573 non-null    float64
 11  Execution_units         573 non-null    float64
 12  Low_Power_Cores         1020 non-null   float64
 13  Energy_Efficient_Units  1020 non-null   int64  
 14  Threads                 972 non-null    float64
 15  RAM_GB                  1020 non-null   int64  
 16  RAM_type                998 non-null    object 
 17  Storage_capacity_GB     1020 non-null   int64  
 18  Storage_type            1020 non-null   object 
 19  Graphics_name           1018 non-null   object 
 20  Graphics_brand          1018 non-null   object 
 21  Graphics_GB             368 non-null    float64
 22  Graphics_integreted     1018 non-null   object 
 23  Display_size_inches     1020 non-null   float64
 24  Horizontal_pixel        1020 non-null   int64  
 25  Vertical_pixel          1020 non-null   int64  
 26  ppi                     1020 non-null   float64
 27  Touch_screen            1020 non-null   bool   
 28  Operating_system        1020 non-null   object 
dtypes: bool(1), float64(10), int64(7), object(11)
memory usage: 224.2+ KB
```

In [ ]:
```
unique_df = data.drop_duplicates()
num_unique_rows = len(unique_df)
print("Number of unique rows:", num_unique_rows)
```

```
Number of unique rows: 1020
```

In [ ]:
```
data.describe()
```

Out[ ]:

	Unnamed: 0	Price	Rating	Processor_gen	Core_per_processor	Total_processor	Execution_units	Low_Power_Cores	Energy_Efficient_Units	Threads	RAM_GB	Storage_capa
count	1020.000000	1020.000000	1020.000000	891.000000	1008.000000	573.000000	573.000000	1020.000000	1020.000000	972.000000	1020.000000	1020.
mean	509.500000	82063.474510	4.373676	10.450056	8.572421	3.926702	6.998255	0.086275	0.043137	12.817901	13.992157	627.
std	294.592939	66502.150607	0.233295	2.966579	4.375012	1.954429	2.680217	0.406531	0.203266	5.677459	7.189564	316
min	0.000000	8000.000000	3.950000	1.000000	2.000000	1.000000	4.000000	0.000000	0.000000	2.000000	2.000000	32.
25%	254.750000	43990.000000	4.200000	7.000000	6.000000	2.000000	4.000000	0.000000	0.000000	8.000000	8.000000	512.
50%	509.500000	63689.500000	4.350000	12.000000	8.000000	4.000000	8.000000	0.000000	0.000000	12.000000	16.000000	512.
75%	764.250000	94990.000000	4.550000	13.000000	10.000000	6.000000	8.000000	0.000000	0.000000	16.000000	16.000000	512.
max	1019.000000	599990.000000	4.750000	14.000000	24.000000	12.000000	16.000000	2.000000	1.000000	32.000000	64.000000	4000.

In [ ]:
```
categorical_data = data.select_dtypes(include=['object', 'category']).columns
categorical_data
```

Out[ ]:
```
Index(['Name', 'Brand', 'Processor_brand', 'Processor_name',
       'Processor_variant', 'RAM_type', 'Storage_type', 'Graphics_name',
       'Graphics_brand', 'Graphics_integreted', 'Operating_system'],
      dtype='object')
```

In [ ]:
```
continue_data = data.select_dtypes(include=['float64', 'int64']).columns
continue_data
```

Out[ ]:
```
Index(['Unnamed: 0', 'Price', 'Rating', 'Processor_gen', 'Core_per_processor',
       'Total_processor', 'Execution_units', 'Low_Power_Cores',
       'Energy_Efficient_Units', 'Threads', 'RAM_GB', 'Storage_capacity_GB',
       'Graphics_GB', 'Display_size_inches', 'Horizontal_pixel',
       'Vertical_pixel', 'ppi'],
      dtype='object')
```

We now have to prep our data for our model Unnamed column was removed

we know that a laptops price will depend on the hardware features like ram,graphic, display etc. so we will select only that column

the columns will be

1 Brand 1020 non-null object 2 Price 1020 non-null int64
3 Rating 1020 non-null float64 5 Processor_name 1020 non-null object 8 Core_per_processor 1008 non-null float64 11 Low_Power_Cores 1020 non-null float64 12 Energy_Efficient_Units 1020 non-null int64
14 RAM_GB 1020 non-null int64
15 RAM_type 998 non-null object 16 Storage_capacity_GB 1020 non-null int64
17 Storage_type 1020 non-null object
19 Graphics_brand 1018 non-null object 21 Graphics_integreted 1018 non-null object 22 Display_size_inches 1020 non-null float64 23 Horizontal_pixel 1020 non-null int64
24 Vertical_pixel 1020 non-null int64
25 ppi 1020 non-null float64 26 Touch_screen 1020 non-null bool
27 Operating_system 1020 non-null object

these will be the columns which we will work on

In [ ]:
```
selected_columns = [
    'Brand',
    'Price',
    'Rating',
    'Processor_name',
    'Core_per_processor',
    'Low_Power_Cores',
    'Energy_Efficient_Units',
    'RAM_GB',
    'RAM_type',
    'Storage_capacity_GB',
    'Storage_type',
    'Graphics_brand',
    'Graphics_integreted',
    'Display_size_inches',
    'Horizontal_pixel',
    'Vertical_pixel',
    'ppi',
    'Touch_screen',
    'Operating_system'
]
```

In [ ]:
```
laptops = data[selected_columns]
laptops.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1020 entries, 0 to 1019
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Brand                   1020 non-null   object 
 1   Price                   1020 non-null   int64  
 2   Rating                  1020 non-null   float64
 3   Processor_name          1020 non-null   object 
 4   Core_per_processor      1008 non-null   float64
 5   Low_Power_Cores         1020 non-null   float64
 6   Energy_Efficient_Units  1020 non-null   int64  
 7   RAM_GB                  1020 non-null   int64  
 8   RAM_type                998 non-null    object 
 9   Storage_capacity_GB     1020 non-null   int64  
 10  Storage_type            1020 non-null   object 
 11  Graphics_brand          1018 non-null   object 
 12  Graphics_integreted     1018 non-null   object 
 13  Display_size_inches     1020 non-null   float64
 14  Horizontal_pixel        1020 non-null   int64  
 15  Vertical_pixel          1020 non-null   int64  
 16  ppi                     1020 non-null   float64
 17  Touch_screen            1020 non-null   bool   
 18  Operating_system        1020 non-null   object 
dtypes: bool(1), float64(5), int64(6), object(7)
memory usage: 144.6+ KB
```

In [ ]:
```
laptops.describe()
```

Out[ ]:

	Price	Rating	Core_per_processor	Low_Power_Cores	Energy_Efficient_Units	RAM_GB	Storage_capacity_GB	Display_size_inches	Horizontal_pixel	Vertical_pixel	ppi
count	1020.000000	1020.000000	1008.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000
mean	82063.474510	4.373676	8.572421	0.086275	0.043137	13.992157	627.733333	15.163775	2035.512745	1214.019608	157.178265
std	66502.150607	0.233295	4.375012	0.406531	0.203266	7.189564	316.911679	1.001537	409.209289	306.863086	33.585713
min	8000.000000	3.950000	2.000000	0.000000	0.000000	2.000000	32.000000	11.600000	1080.000000	768.000000	100.450000
25%	43990.000000	4.200000	6.000000	0.000000	0.000000	8.000000	512.000000	14.000000	1920.000000	1080.000000	141.210000
50%	63689.500000	4.350000	8.000000	0.000000	0.000000	16.000000	512.000000	15.600000	1920.000000	1080.000000	141.210000
75%	94990.000000	4.550000	10.000000	0.000000	0.000000	16.000000	512.000000	15.600000	1920.000000	1200.000000	161.730000
max	599990.000000	4.750000	24.000000	2.000000	1.000000	64.000000	4000.000000	18.000000	3840.000000	2560.000000	337.930000

In [ ]:
```
rows_with_null_ram_type = laptops[laptops['RAM_type'].isnull()]
rows_with_null_ram_type.shape
```

Out[ ]: (22, 19)

In [ ]:
```
laptops = laptops.dropna()
laptops.head()
```

Out[ ]:

	Brand	Price	Rating	Processor_name	Core_per_processor	Low_Power_Cores	Energy_Efficient_Units	RAM_GB	RAM_type	Storage_capacity_GB	Storage_type	Graphics_brand	Graphics_integretec
0	HP	50399	4.30	AMD Ryzen 5	6.0	0.0	0	8	DDR4	512	SSD	AMD	False
1	Lenovo	26690	4.45	AMD Ryzen 3	4.0	0.0	0	8	LPDDR5	512	SSD	AMD	False
2	HP	37012	4.65	Intel Core i3	6.0	0.0	0	8	DDR4	512	SSD	Intel	False
3	Samsung	69990	4.75	Intel Core i5	12.0	0.0	0	16	LPDDR5	512	SSD	Intel	False
4	Tecno	23990	4.25	Intel Core i3	2.0	0.0	0	8	LPDDR4	512	SSD	Intel	False

Getting a visualization of our data

In [ ]:
```
import matplotlib.pyplot as plt
```

In [ ]:
```
# let us understand how different features lead to the price our laptops
```

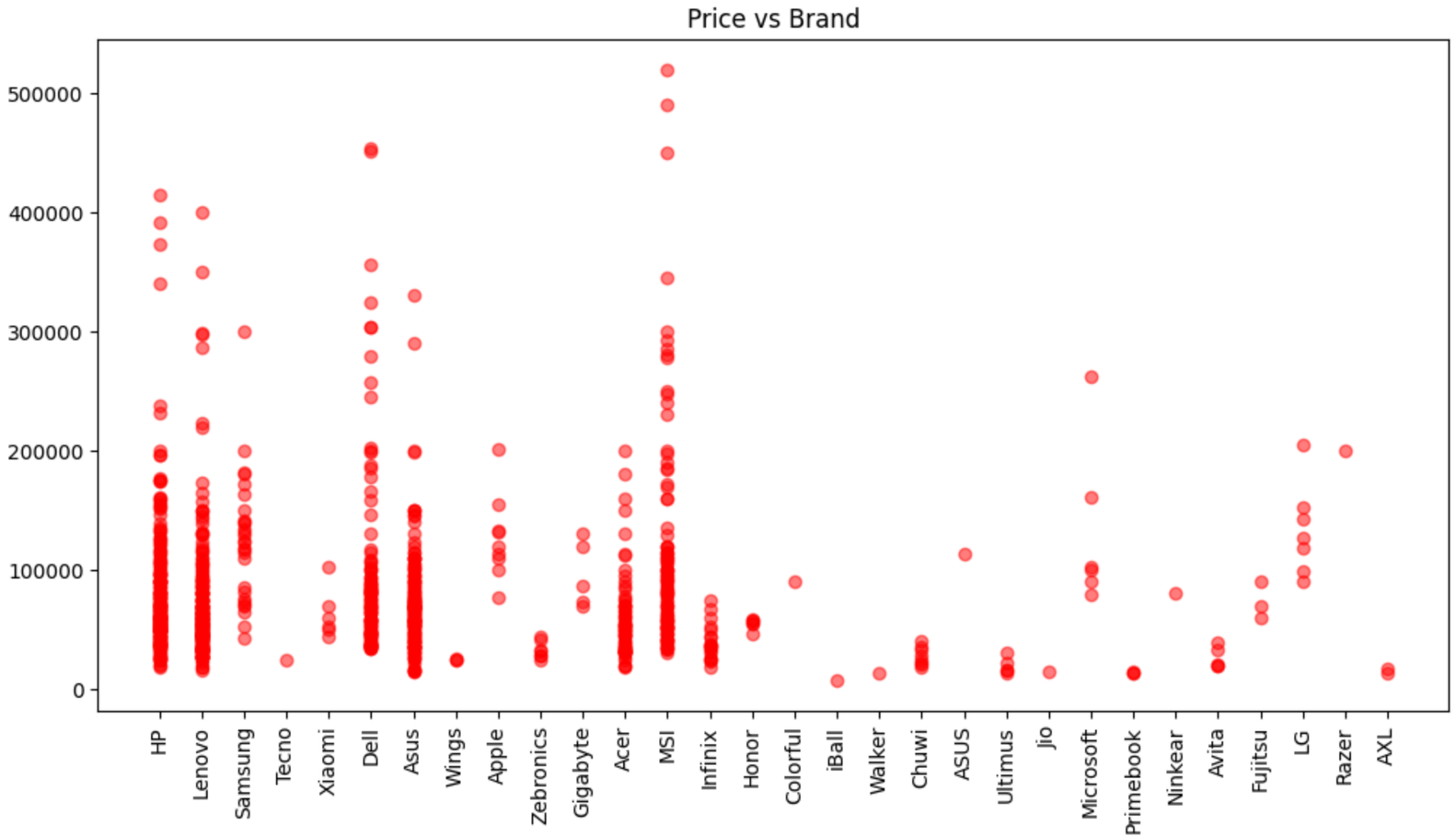
'Brand', 'Price', 'Rating', 'Processor_name', 'Core_per_processor', 'Low_Power_Cores', 'Energy_Efficient_Units', 'RAM_GB', 'RAM_type', 'Storage_capacity_GB', 'Storage_type', 'Graphics_brand', 'Graphics_integreted', 'Display_size_inches', 'Horizontal_pixel', 'Vertical_pixel', 'ppi', 'Touch_screen', 'Operating_system'

```
    lowcore
```

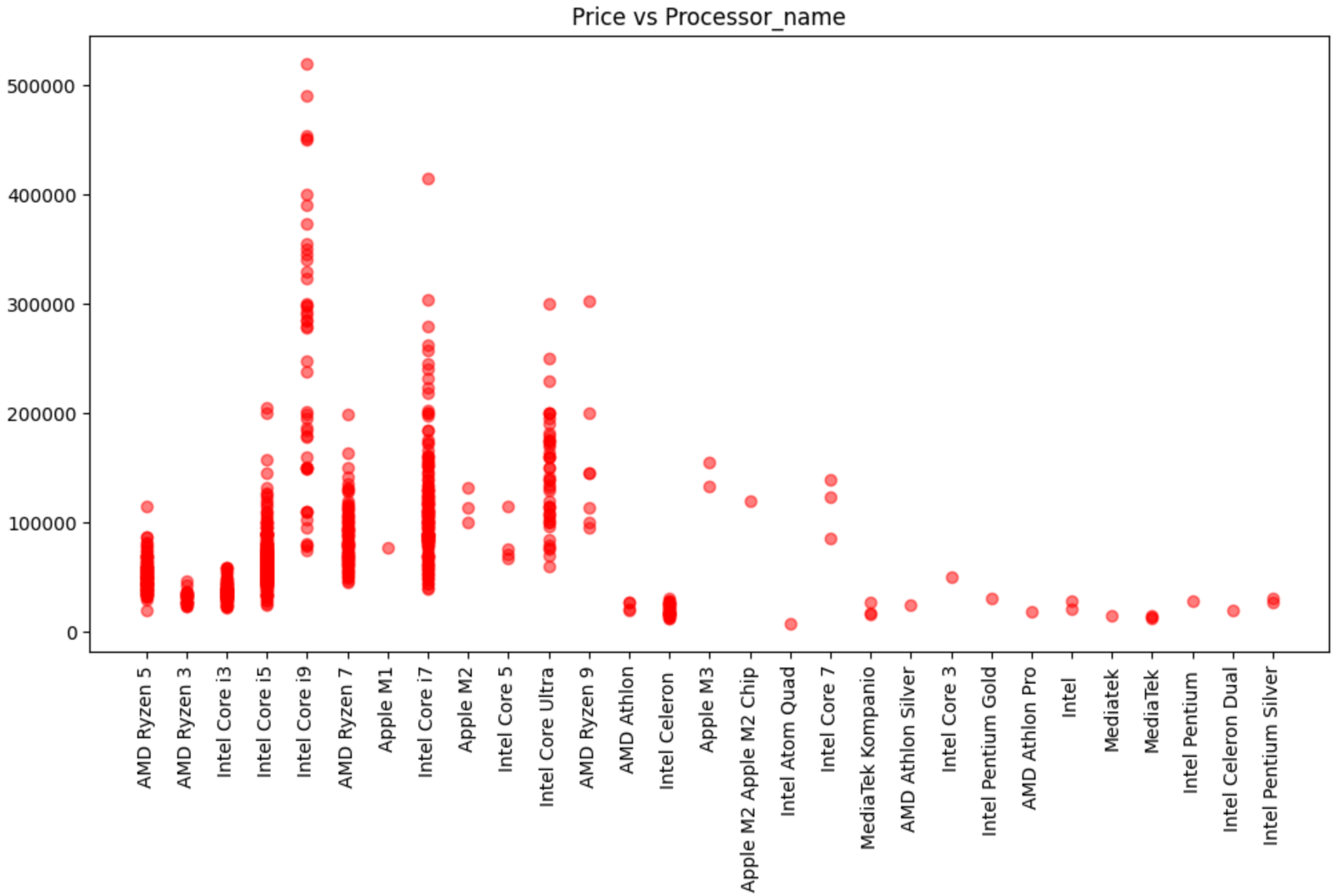
energry effi graphic touch

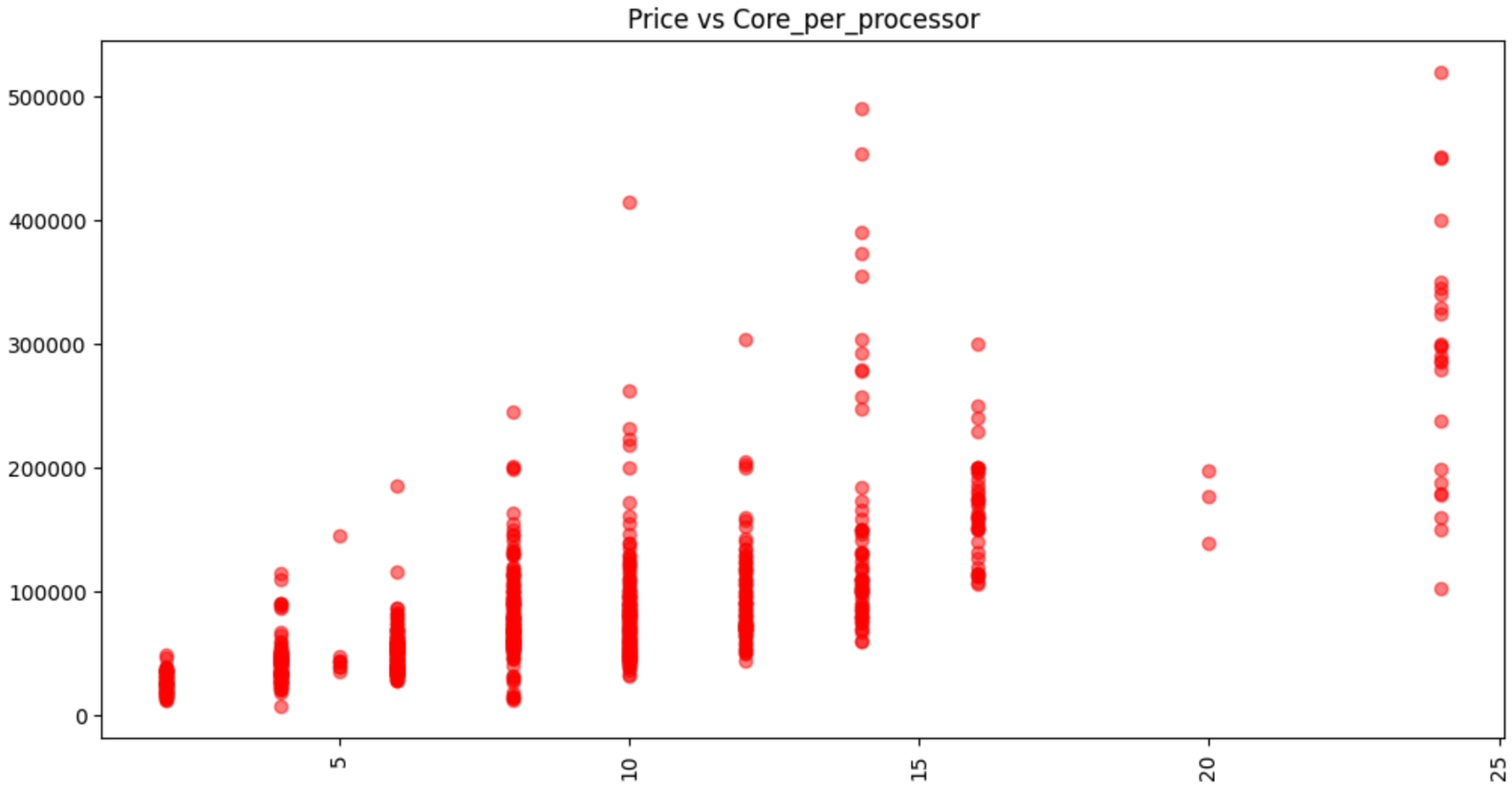
In [ ]:
```
no_scatter_plot = ["Low_Power_Cores","Energy_Efficient_Units","Graphics_integreted","Touch_screen"]
for cols in selected_columns:
```

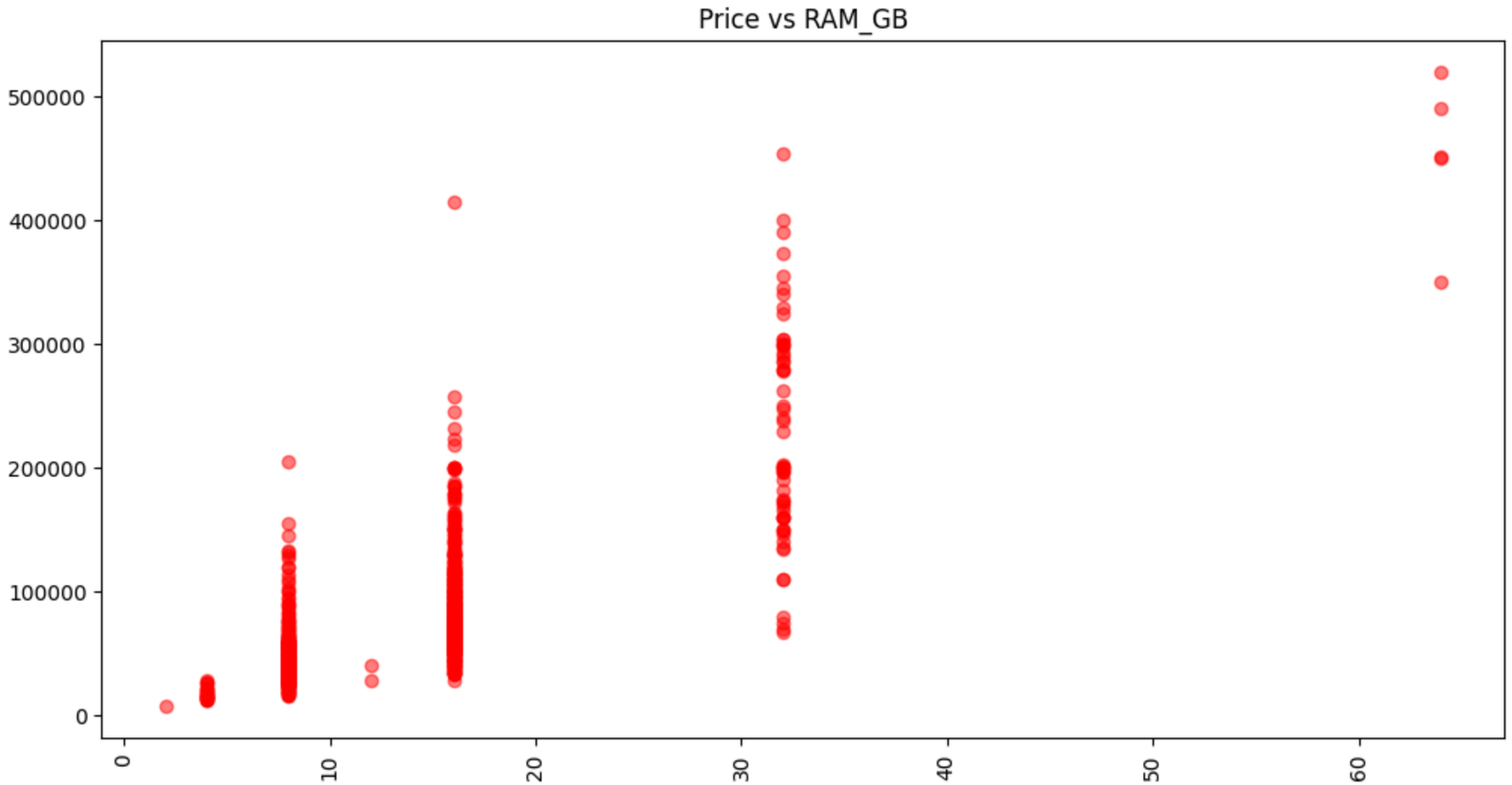
```
    if (cols not in  no_scatter_plot) & (cols !="Price"):
        plt.figure(figsize=(12, 6))
        plt.scatter(laptops[cols], laptops["Price"],alpha=0.5,color='red')
        plt.xticks(rotation=90)
        plt.title(f"Price vs {cols}")
        plt.show()
```

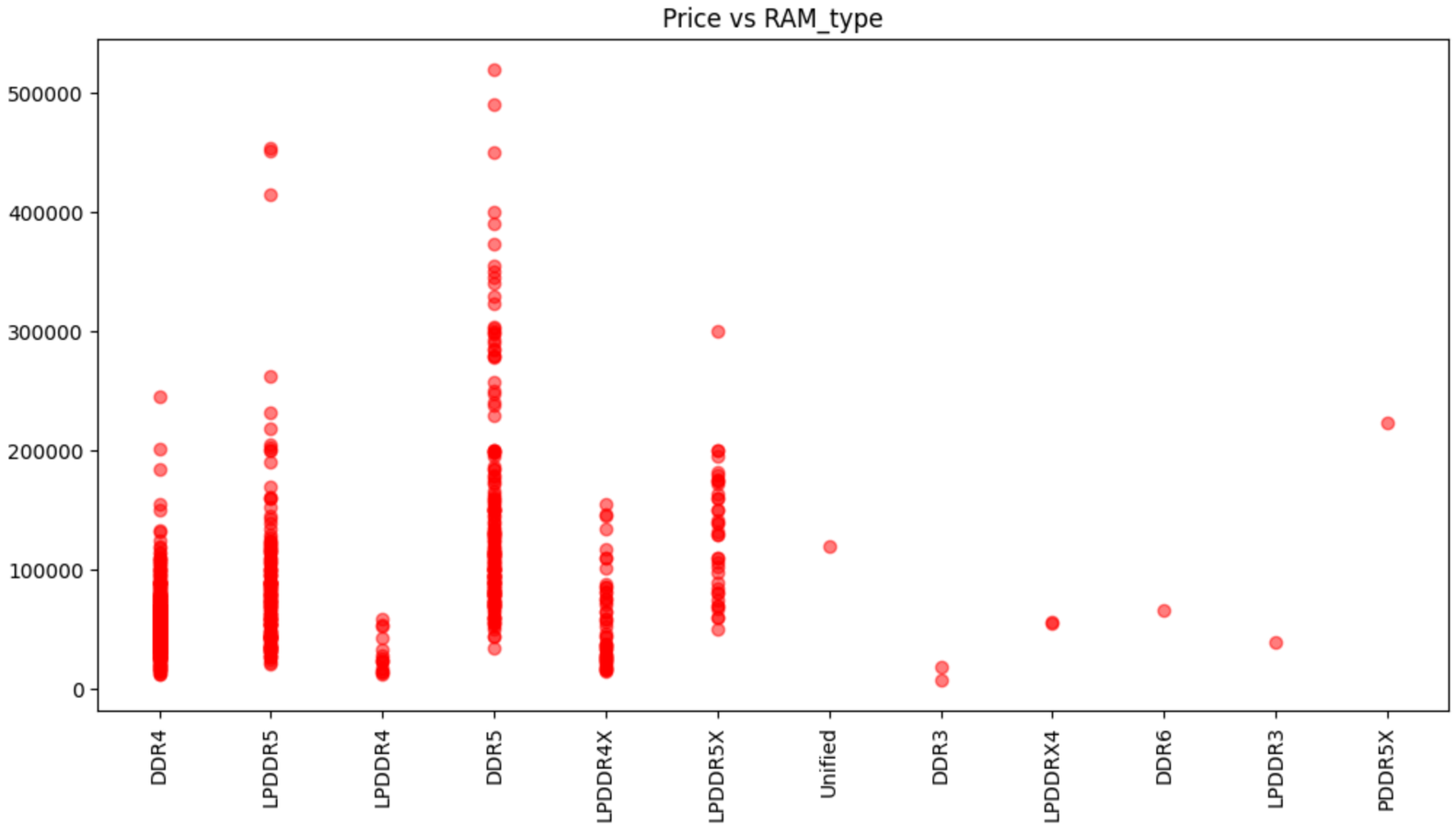


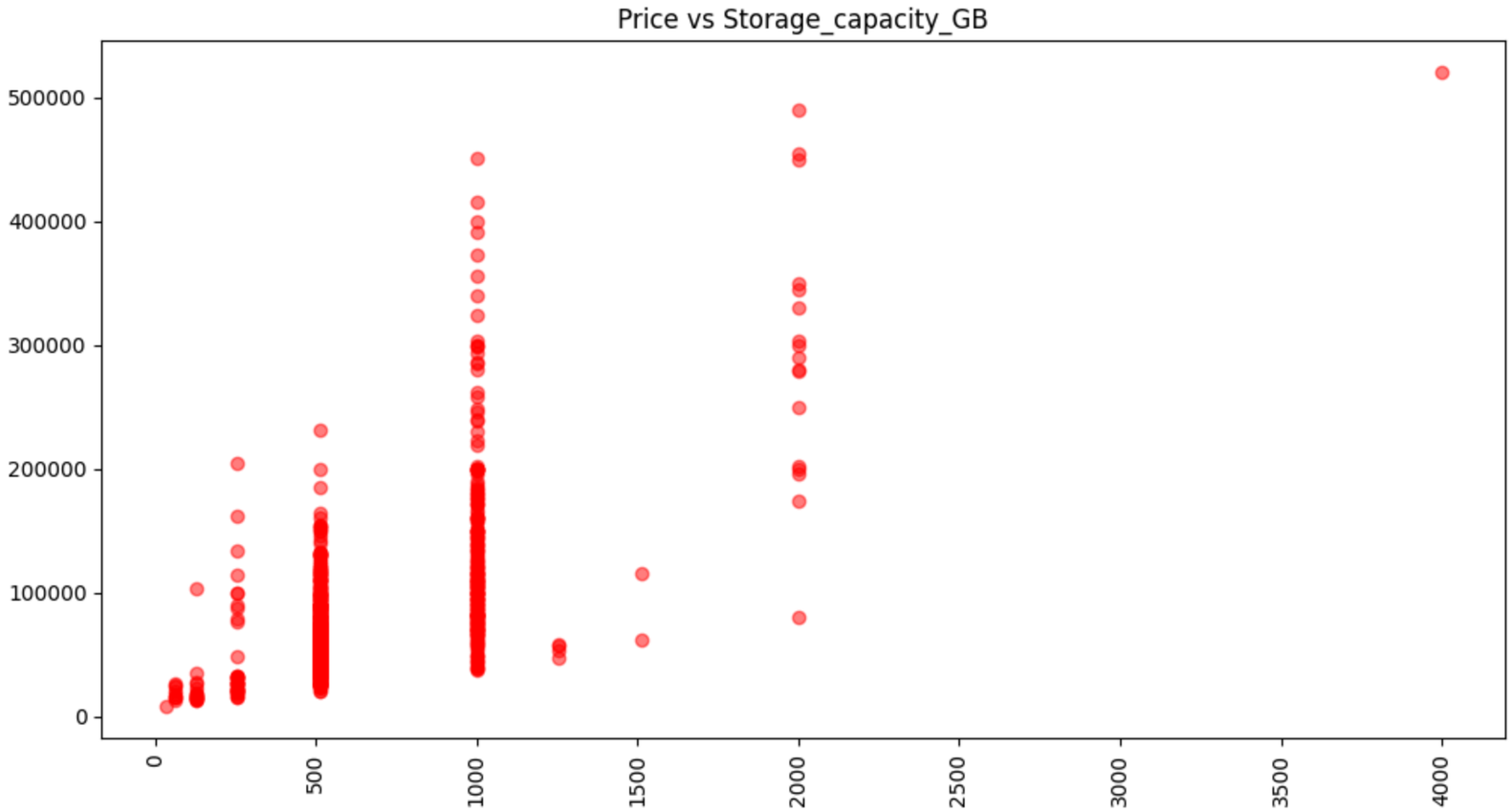

Price vs Rating
500000
400000
300000
200000
100000
0
4.0
4.1
4.2
4.3
4.4
4.5
4.6
4.7

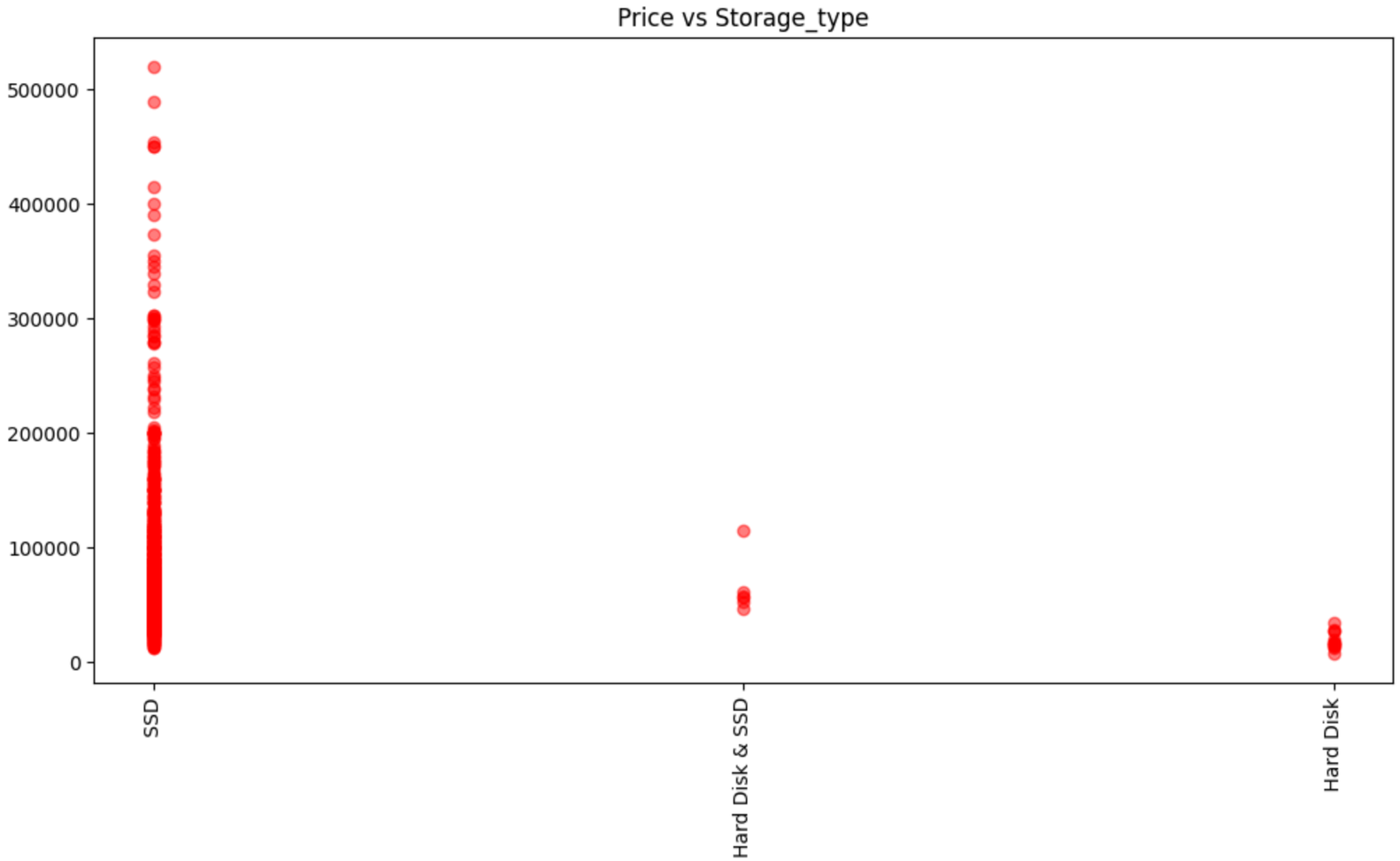

Price vs Processor_name
500000
400000
300000
200000
100000
0
AMD Ryzen 5
AMD Ryzen 3
Intel Core i3
Intel Core i5
Intel Core i9
AMD Ryzen 7
Apple M1
Intel Core i7
Apple M2
Intel Core 5
Intel Core Ultra
AMD Ryzen 9
AMD Athlon
Intel Celeron
Apple M3
Apple M2 Apple M2 Chip
Intel Atom Quad
Intel Core 7
MediaTek Kompanio
AMD Athlon Silver
Intel Core 3
Intel Pentium Gold
AMD Athlon Pro
Intel
Mediatek
MediaTek
Intel Pentium
Intel Celeron Dual
Intel Pentium Silver

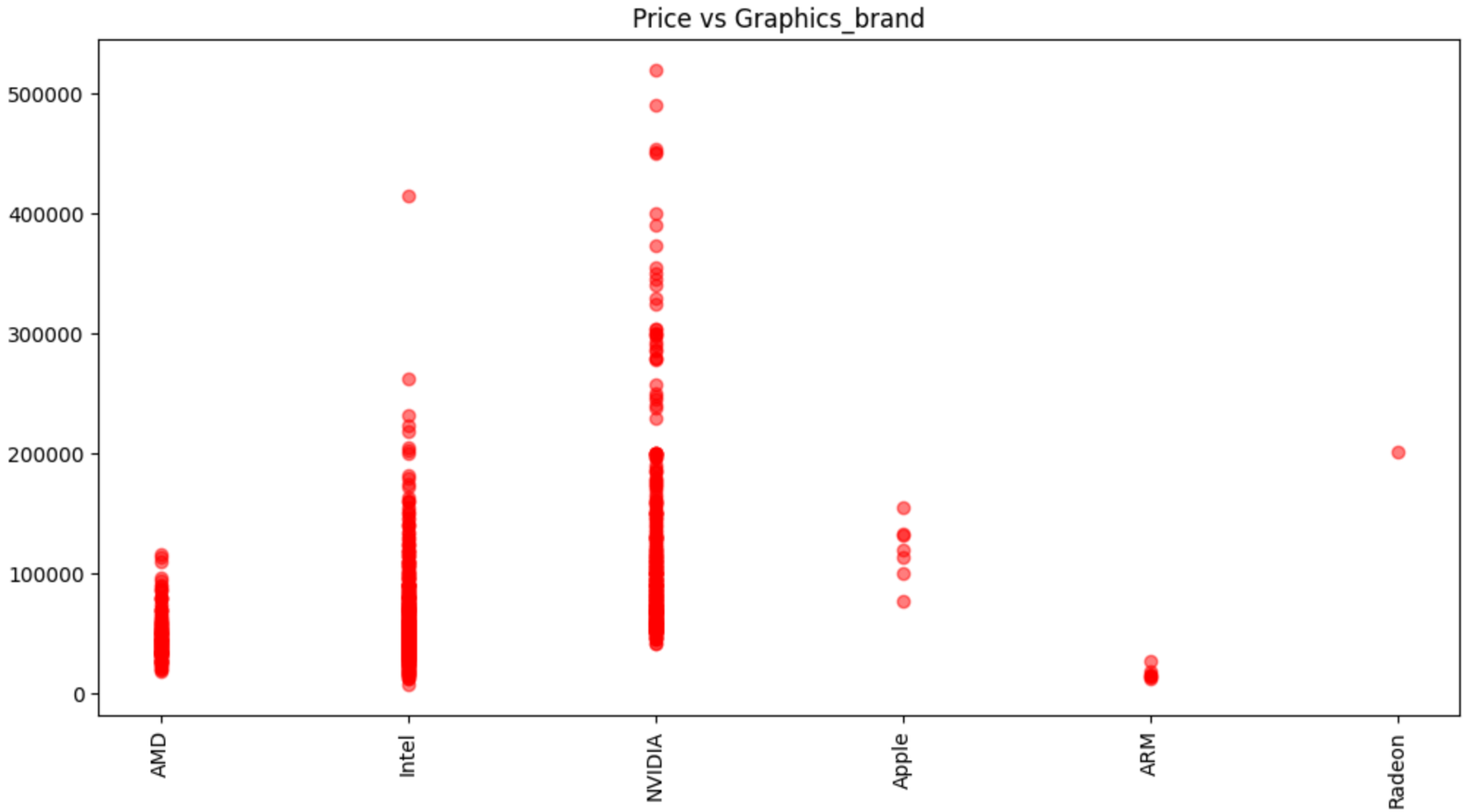

Price vs Core_per_processor
500000
400000
300000
200000
100000
0
5
10
15
20
25

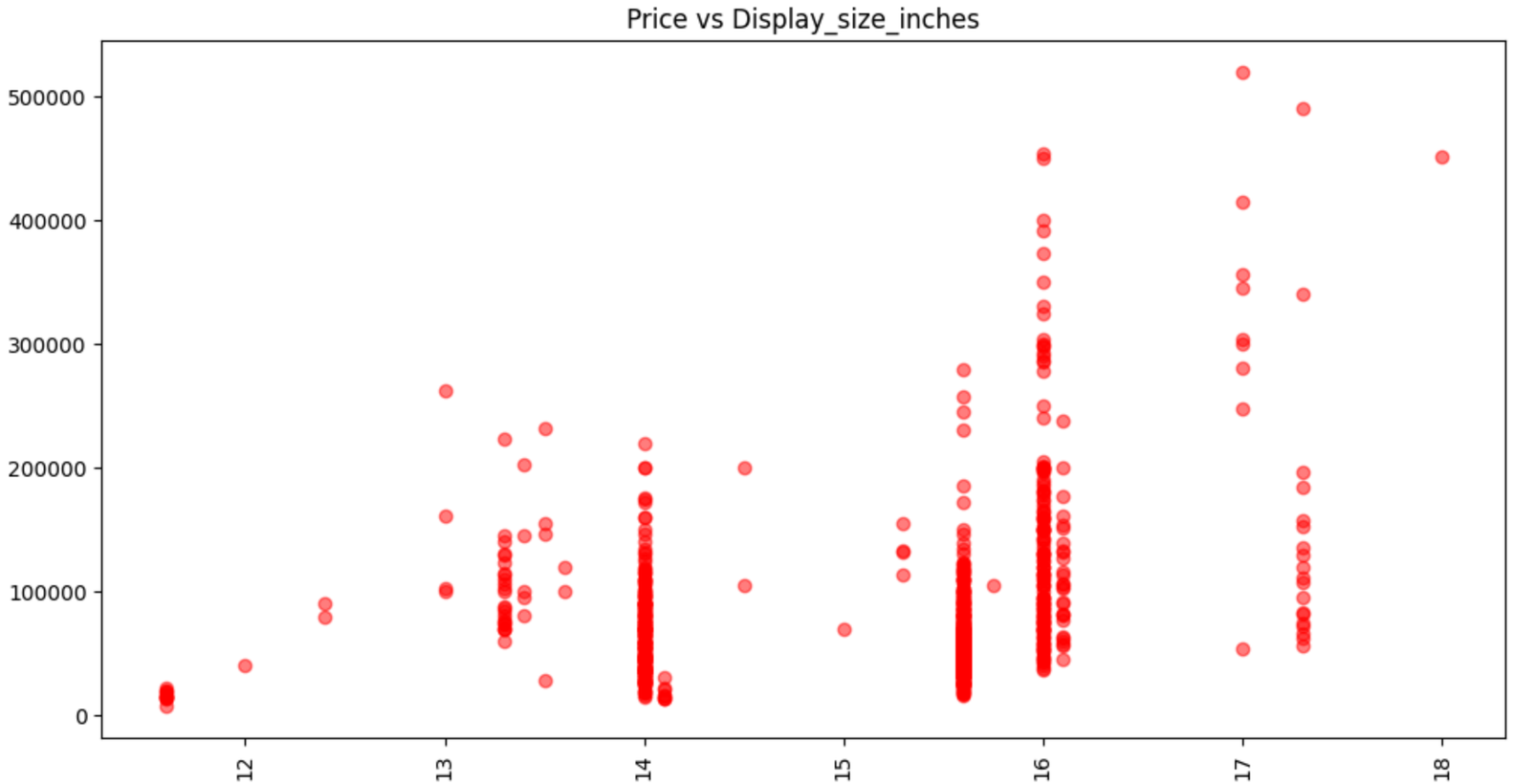

Price vs RAM_GB
500000
400000
300000
200000
100000
0
0
10
20
30
40
50
60


Price vs RAM_type
500000
400000
300000
200000
100000
0
DDR4
LPDDR5
LPDDR4
DDR5
LPDDR4X
LPDDR5X
Unified
DDR3
LPDDRX4
DDR6
LPDDR3
PDDR5X

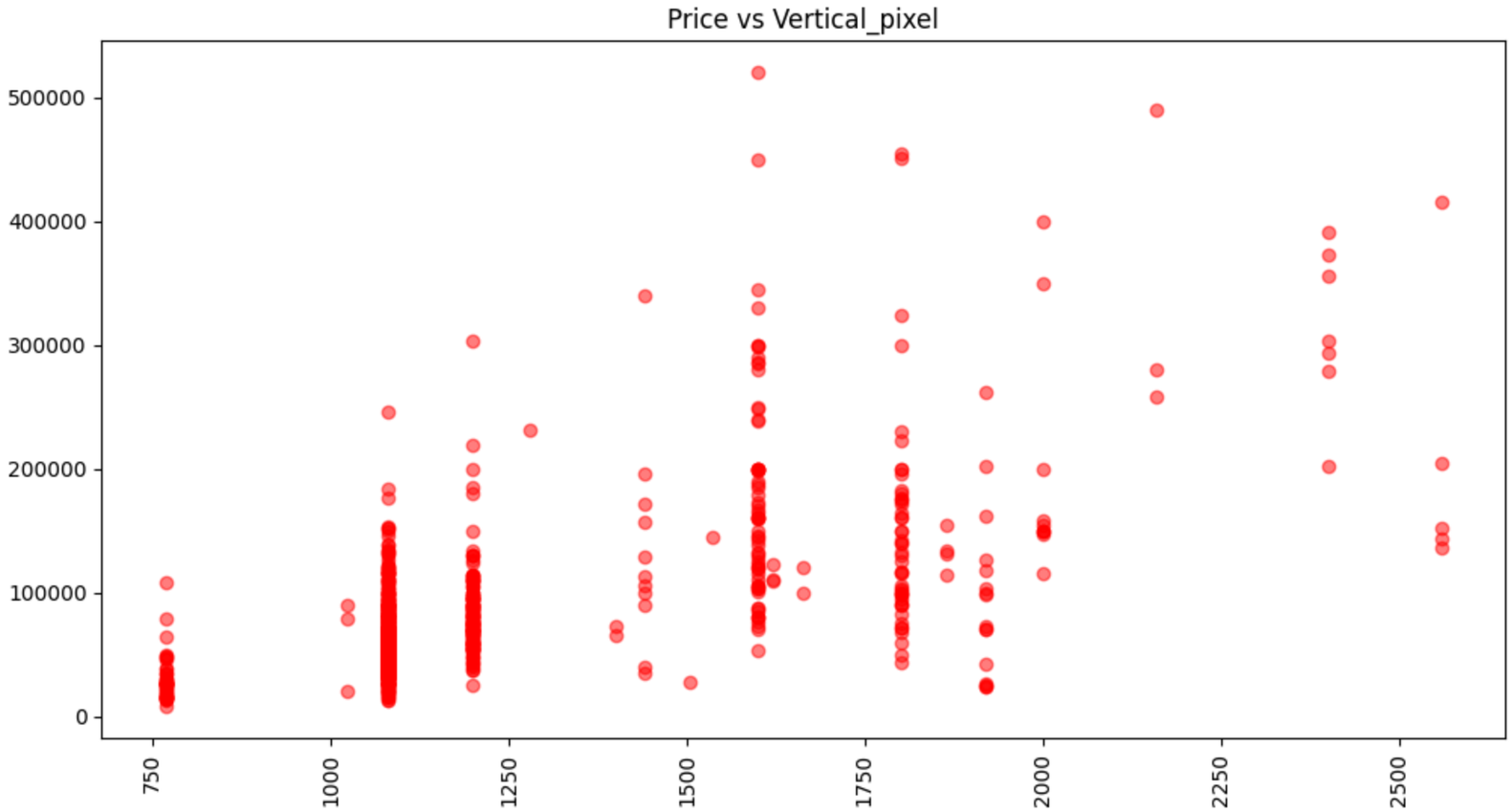

Price vs Storage_capacity_GB
500000
400000
300000
200000
100000
0
0
500
1000
1500
2000
2500
3000
3500
4000


Price vs Storage_type
500000
400000
300000
200000
100000
0
SSD
Hard Disk & SSD
Hard Disk

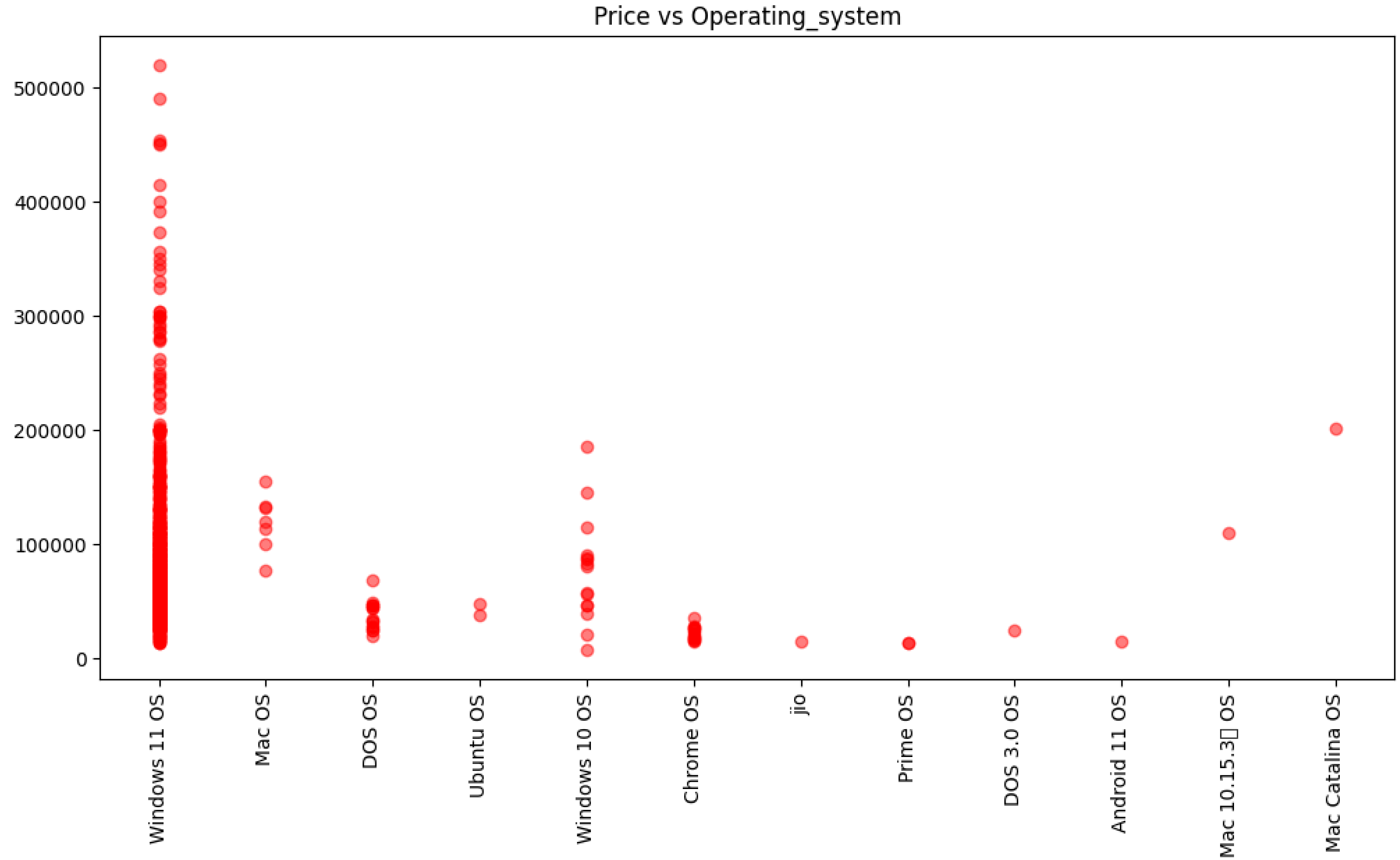

Price vs Graphics_brand
500000
400000
300000
200000
100000
0
AMD
Intel
NVIDIA
Apple
ARM
Radeon


Price vs Display_size_inches
500000
400000
300000
200000
100000
0
12
13
14
15
16
17
18


Price vs Horizontal_pixel
500000
400000
300000
200000
100000
0
1000
1500
2000
2500
3000
3500

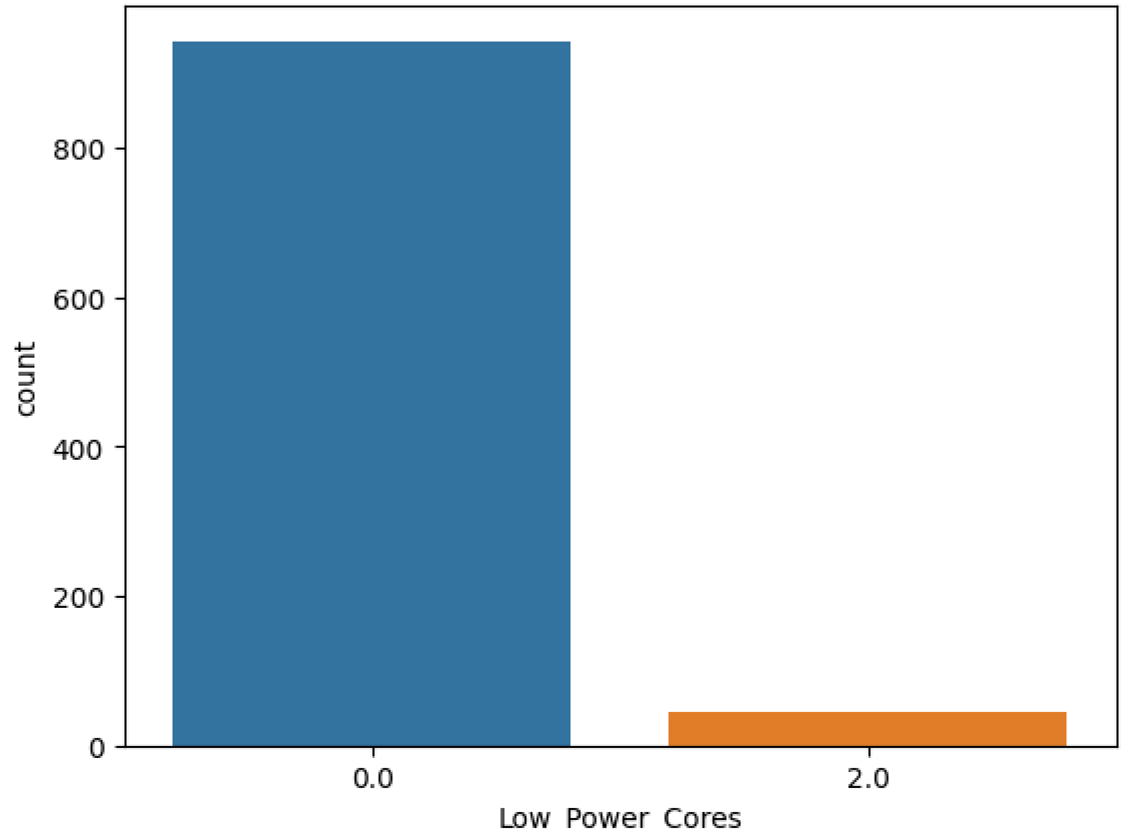

Price vs Vertical_pixel
500000
400000
300000
200000
100000
0
750
1000
1250
1500
1750
2000
2250
2500


```
/Users/nishantsingh/Library/Python/3.11/lib/python/site-packages/IPython/core/pylabtools.py:152: UserWarning: Glyph 9 ( ) missing from current font.
  fig.canvas.print_figure(bytes_io, **kw)
```

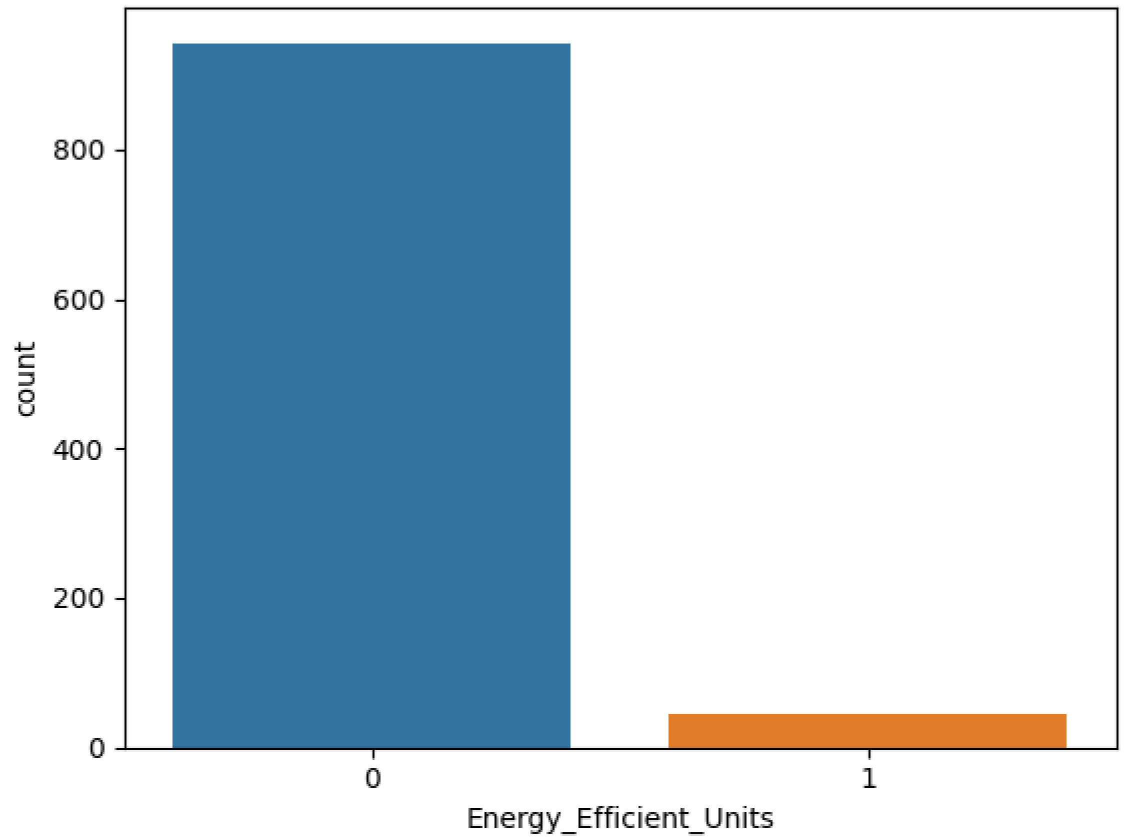


In [ ]:
```
bar_plot = ["Low_Power_Cores","Energy_Efficient_Units","Graphics_integreted","Touch_screen"]

import seaborn as sns
for cols in bar_plot:
    sns.countplot(x=cols, data=laptops)
    plt.show()
```

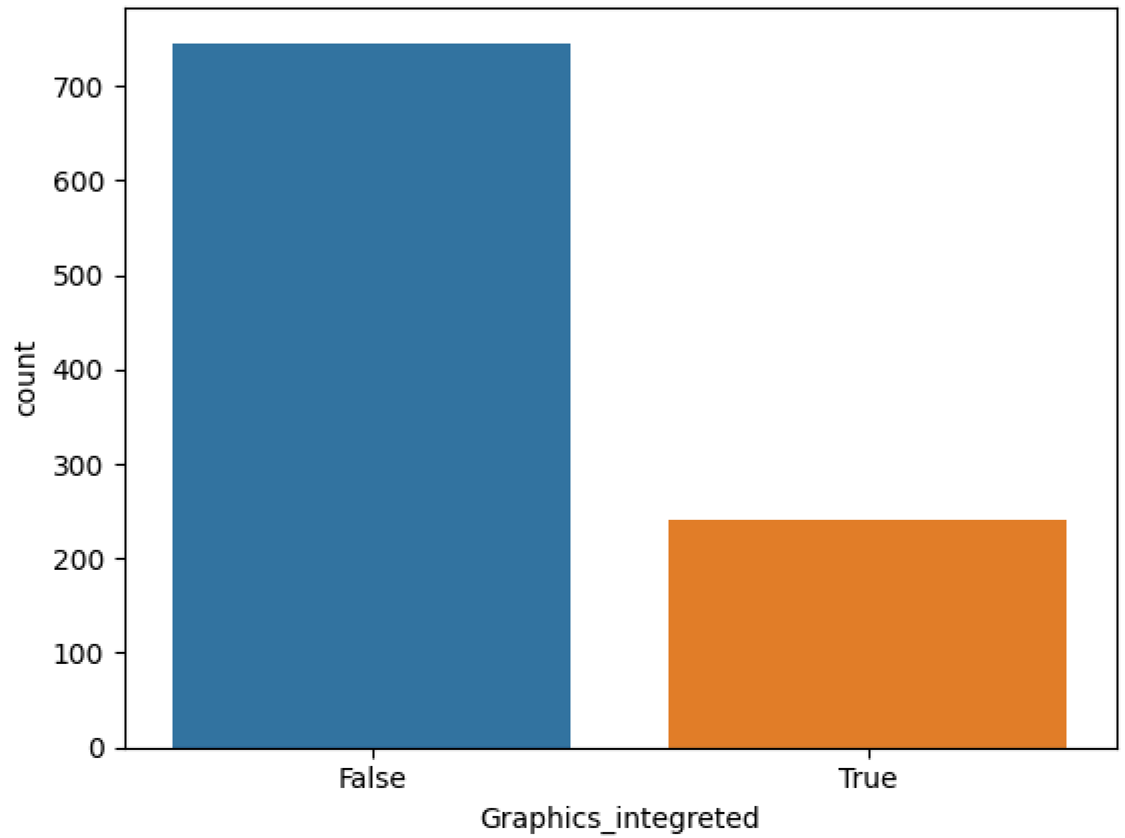
```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



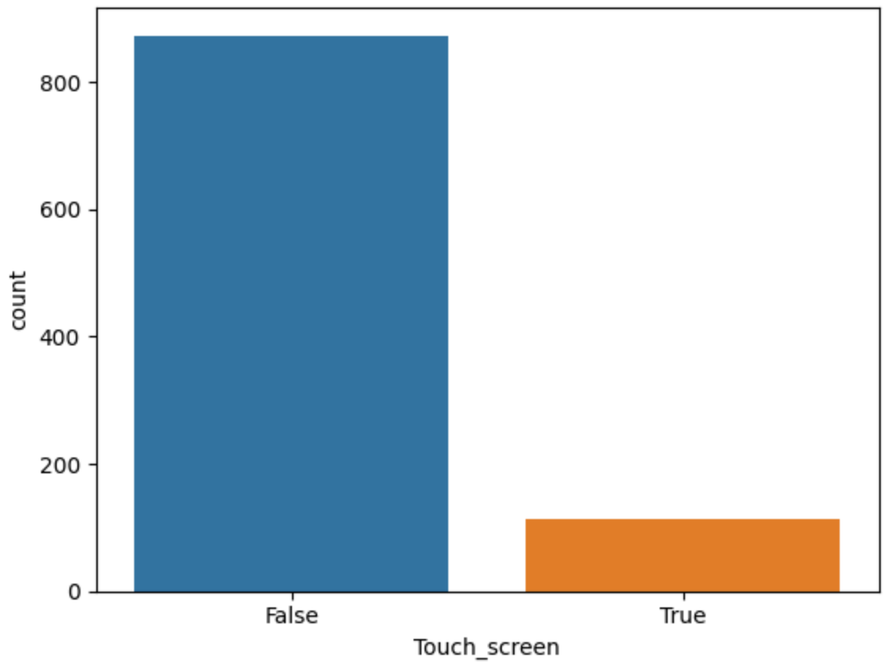

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```




```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```




```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



In [ ]:
```
selected_columns_2 = [
    'Price',
    'Rating',
    'Core_per_processor',
    'Low_Power_Cores',
    'Energy_Efficient_Units',
    'RAM_GB',
    'Storage_capacity_GB',
    'Storage_type',
    'Graphics_integreted',
    'Display_size_inches',
    'Horizontal_pixel',
    'Vertical_pixel',
    'ppi',
    'Touch_screen',
    'Operating_system'
]
```

In [ ]:
```
df = laptops[selected_columns_2]
df.head()
```

Out[ ]:

	Price	Rating	Core_per_processor	Low_Power_Cores	Energy_Efficient_Units	RAM_GB	Storage_capacity_GB	Storage_type	Graphics_integreted	Display_size_inches	Horizontal_pixel	Vertical_pixel	
0	50399	4.30	6.0	0.0	0	8	512	SSD	False	15.6	1920	1080	
1	26690	4.45	4.0	0.0	0	8	512	SSD	False	15.6	1920	1080	
2	37012	4.65	6.0	0.0	0	8	512	SSD	False	15.6	1920	1080	
3	69990	4.75	12.0	0.0	0	16	512	SSD	False	13.3	1080	1920	1
4	23990	4.25	2.0	0.0	0	8	512	SSD	False	15.6	1920	1080	

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

encoding_labels = ['Storage_type','Operating_system']

for cols in encoding_labels:
    df[cols] = label_encoder.fit_transform(df[cols])

df['Graphics_integreted'] = df['Graphics_integreted'].astype(int)
df['Touch_screen'] = df['Touch_screen'].astype(int)
```

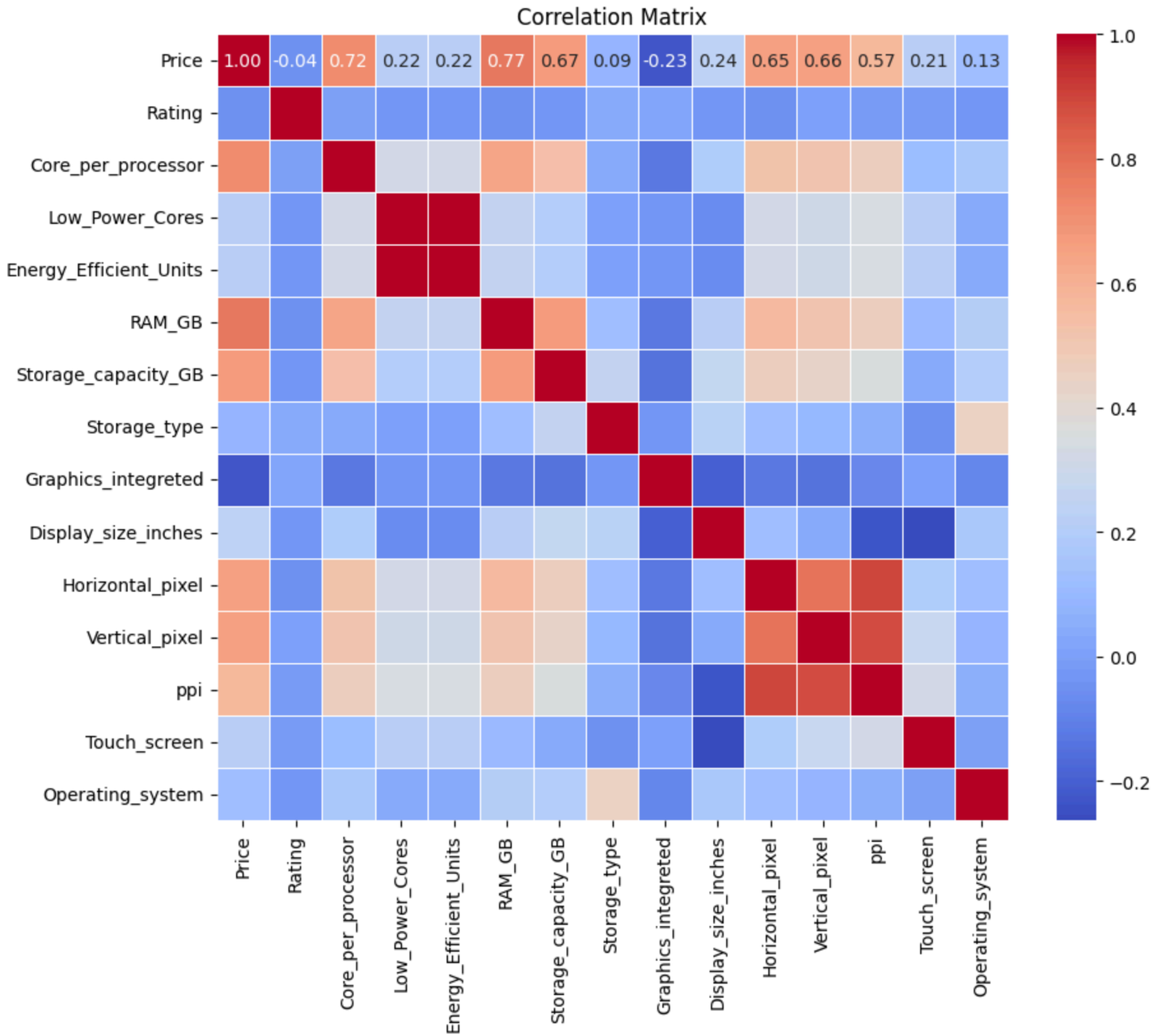
```
/var/folders/kn/x036pv7d2sgg8wy4n1b8614m0000gn/T/ipykernel_11764/2353218521.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[cols] = label_encoder.fit_transform(df[cols])
/var/folders/kn/x036pv7d2sgg8wy4n1b8614m0000gn/T/ipykernel_11764/2353218521.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[cols] = label_encoder.fit_transform(df[cols])
/var/folders/kn/x036pv7d2sgg8wy4n1b8614m0000gn/T/ipykernel_11764/2353218521.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Graphics_integreted'] = df['Graphics_integreted'].astype(int)
/var/folders/kn/x036pv7d2sgg8wy4n1b8614m0000gn/T/ipykernel_11764/2353218521.py:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Touch_screen'] = df['Touch_screen'].astype(int)
```

In [ ]:
```
df.corr()
```

Out[ ]:

	Price	Rating	Core_per_processor	Low_Power_Cores	Energy_Efficient_Units	RAM_GB	Storage_capacity_GB	Storage_type	Graphics_integreted	Display_size_inches	Horiz
Price	1.000000	-0.041996	0.719818	0.219841	0.219841	0.774043	0.671881	0.086042	-0.234395	0.242106	
Rating	-0.041996	1.000000	-0.004586	-0.040597	-0.040597	-0.042779	-0.028134	0.042342	0.016629	-0.036497	
Core_per_processor	0.719818	-0.004586	1.000000	0.314600	0.314600	0.641045	0.532406	0.036563	-0.132618	0.192844	
Low_Power_Cores	0.219841	-0.040597	0.314600	1.000000	1.000000	0.249824	0.202761	0.012319	-0.031354	-0.061839	
Energy_Efficient_Units	0.219841	-0.040597	0.314600	1.000000	1.000000	0.249824	0.202761	0.012319	-0.031354	-0.061839	
RAM_GB	0.774043	-0.042779	0.641045	0.249824	0.249824	1.000000	0.671089	0.130616	-0.133784	0.222448	
Storage_capacity_GB	0.671881	-0.028134	0.532406	0.202761	0.202761	0.671089	1.000000	0.264789	-0.150166	0.272962	
Storage_type	0.086042	0.042342	0.036563	0.012319	0.012319	0.130616	0.264789	1.000000	-0.033963	0.226965	
Graphics_integreted	-0.234395	0.016629	-0.132618	-0.031354	-0.031354	-0.133784	-0.150166	-0.033963	1.000000	-0.196917	
Display_size_inches	0.242106	-0.036497	0.192844	-0.061839	-0.061839	0.222448	0.272962	0.226965	-0.196917	1.000000	
Horizontal_pixel	0.651431	-0.046182	0.524051	0.320568	0.320568	0.558447	0.468446	0.126722	-0.130271	0.128469	
Vertical_pixel	0.656980	0.007940	0.511406	0.303999	0.303999	0.519890	0.426760	0.100602	-0.146771	0.040142	
ppi	0.571938	-0.011371	0.459006	0.343733	0.343733	0.468720	0.362385	0.058470	-0.073102	-0.239137	
Touch_screen	0.210706	-0.021116	0.118444	0.215280	0.215280	0.105867	0.042563	-0.046663	0.010431	-0.264039	
Operating_system	0.126446	-0.026248	0.164709	0.046271	0.046271	0.205819	0.202309	0.456387	-0.090805	0.172370	

In [ ]:
```
correlation_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



In [ ]:
```
columns=['Rating', 'Core_per_processor', 'RAM_GB', 'Storage_capacity_GB',
       'Graphics_integrated', 'Display_size_inches', 'Horizontal_pixel',
       'Vertical_pixel', 'ppi', 'Touch_screen']
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
```

```
# Create a StandardScaler object
scaler = StandardScaler()

# Fit and transform your data
scaled_data = scaler.fit_transform(df)
```

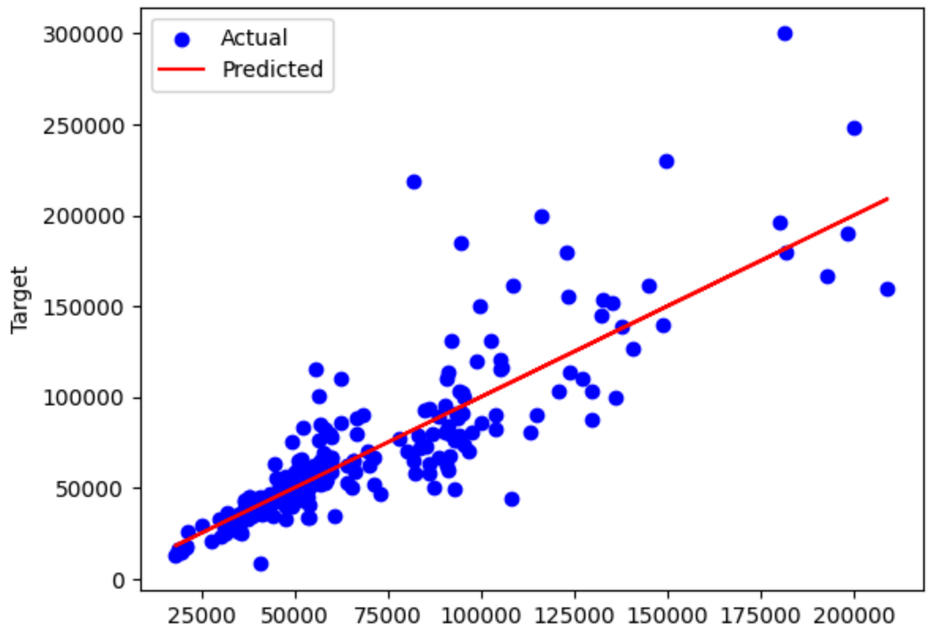
In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.metrics import r2_score

y = df.iloc[:, 0]
X = df.iloc[:, 1:]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

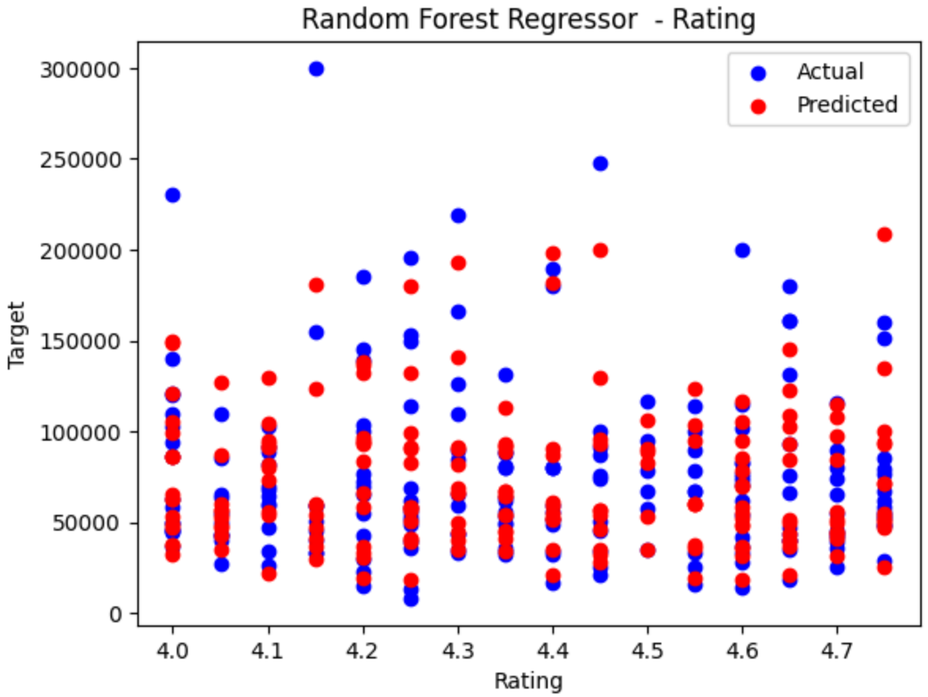
In [ ]:
```
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared Score (R^2):", r2)
```

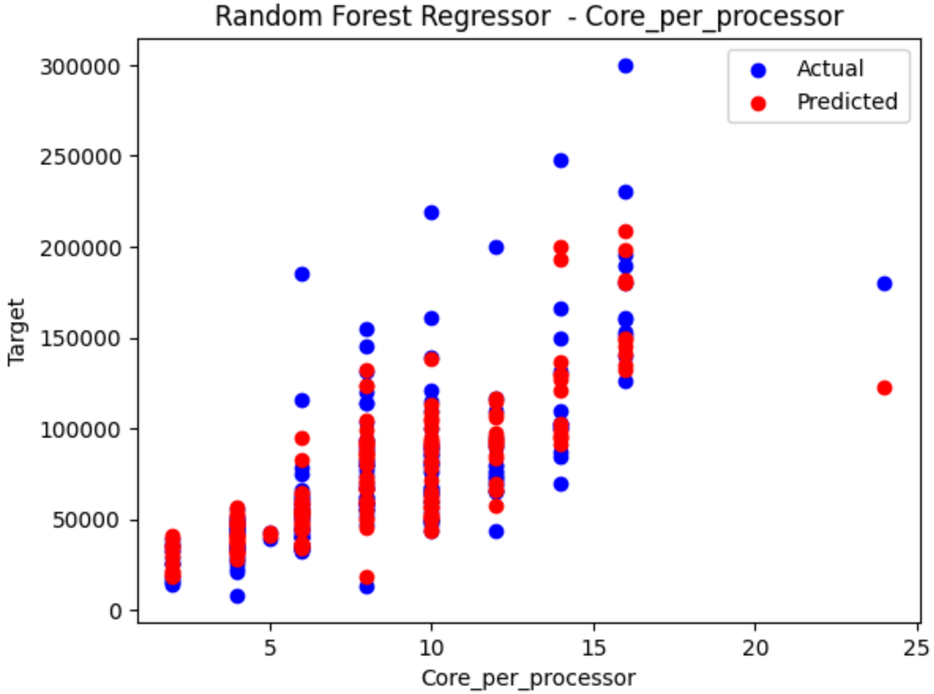
```
Root Mean Squared Error (RMSE): 24602.169847940197
R-squared Score (R^2): 0.720612527683302
```

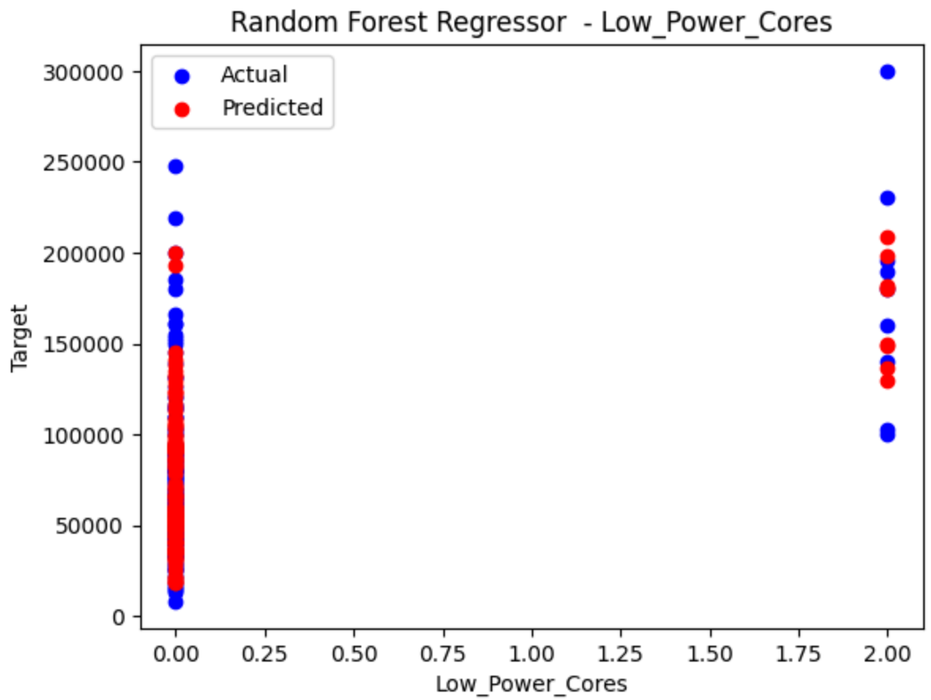
In [ ]:
```
plt.scatter(y_pred, y_test, label='Actual', color='blue')
plt.plot(y_pred, y_pred, label='Predicted', color='red')
plt.ylabel('Target')
plt.legend()
plt.show()
```

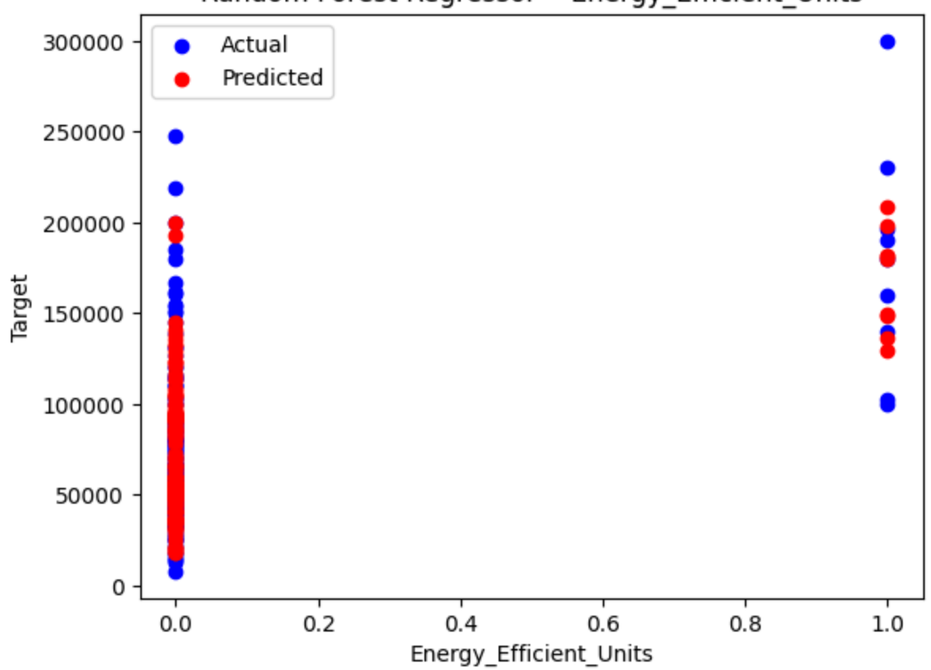


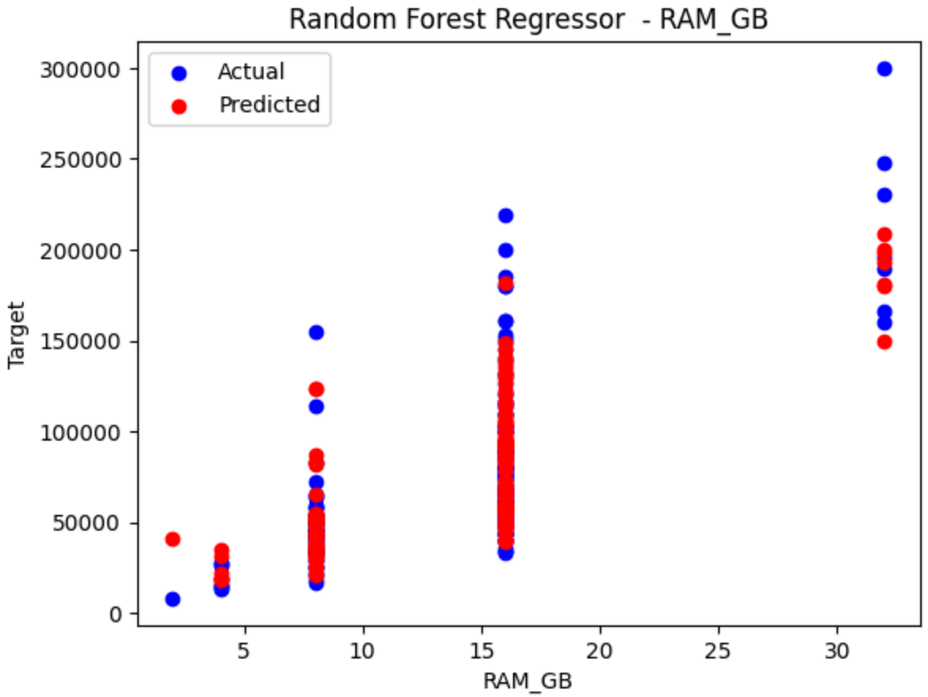
In [ ]:
```
for column in X_test.columns:
    plt.scatter(X_test[column], y_test, label='Actual', color='blue')
    plt.scatter(X_test[column], y_pred, label='Predicted', color='red')
    plt.xlabel(column)
    plt.ylabel('Target')
    plt.title('Random Forest Regressor  - {}'.format(column))
    plt.legend()
    plt.show()
```

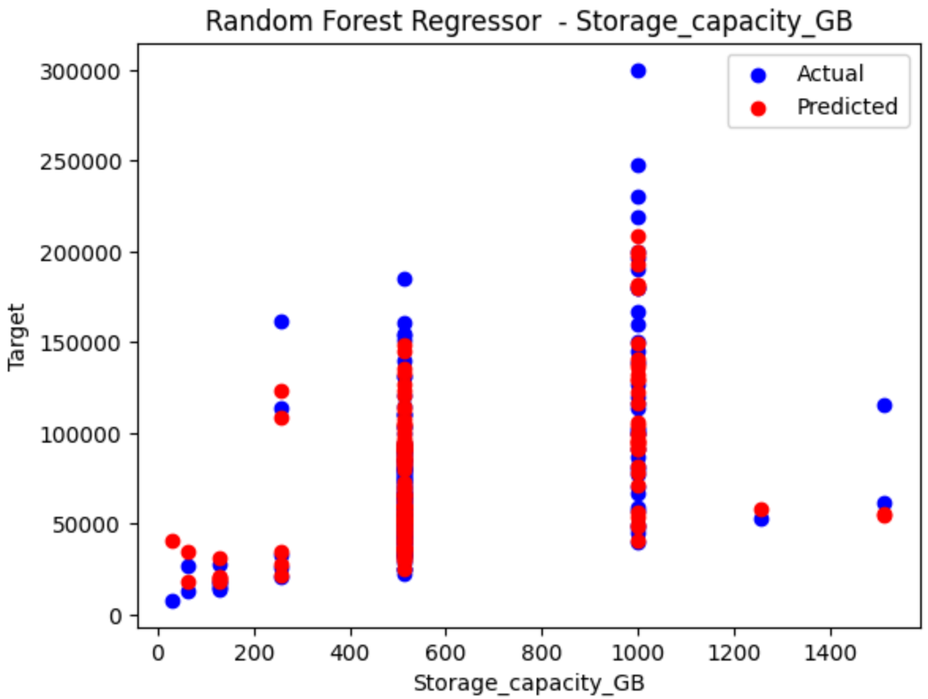

Random Forest Regressor - Rating
Actual
Predicted
Target
Rating
300000
250000
200000
150000
100000
50000
0
4.0
4.1
4.2
4.3
4.4
4.5
4.6
4.7

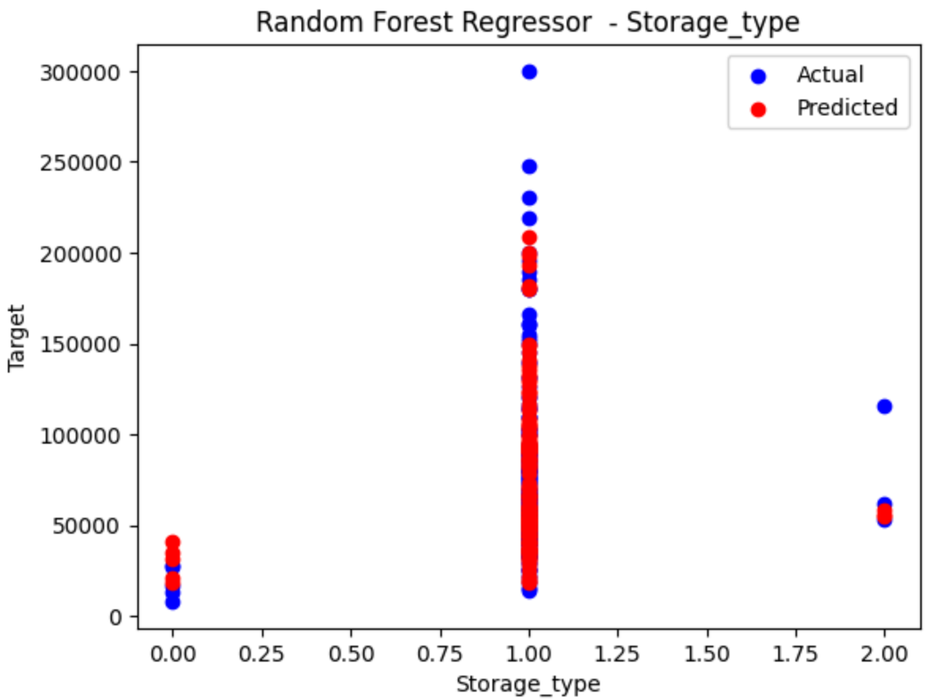

Random Forest Regressor - Core_per_processor
Actual
Predicted
Target
Core_per_processor
300000
250000
200000
150000
100000
50000
0
5
10
15
20
25

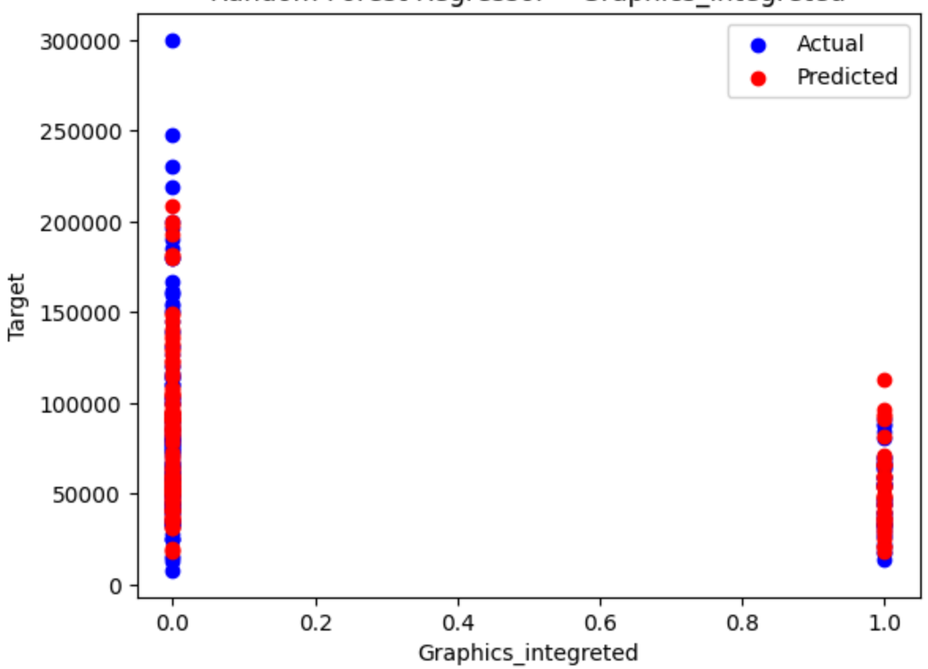

Random Forest Regressor - Low_Power_Cores
Actual
Predicted
Target
Low_Power_Cores
0
50000
100000
150000
200000
250000
300000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

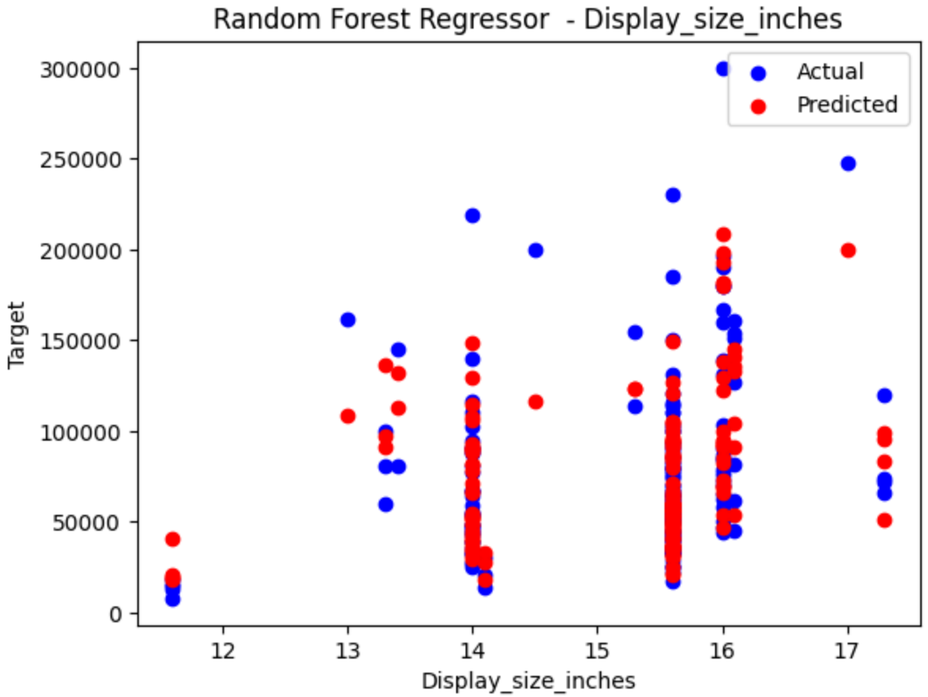

Random Forest Regressor - Energy_Efficient_Units
Actual
Predicted
Target
Energy_Efficient_Units
0
50000
100000
150000
200000
250000
300000
0.0
0.2
0.4
0.6
0.8
1.0

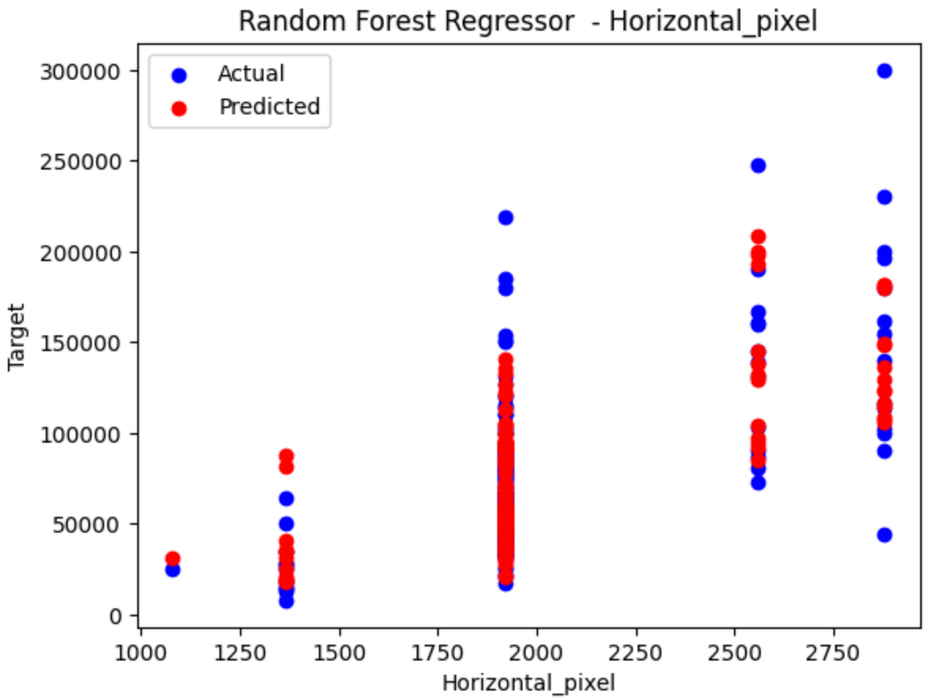

Random Forest Regressor - RAM_GB
Actual
Predicted
Target
RAM_GB
0
50000
100000
150000
200000
250000
300000
5
10
15
20
25
30

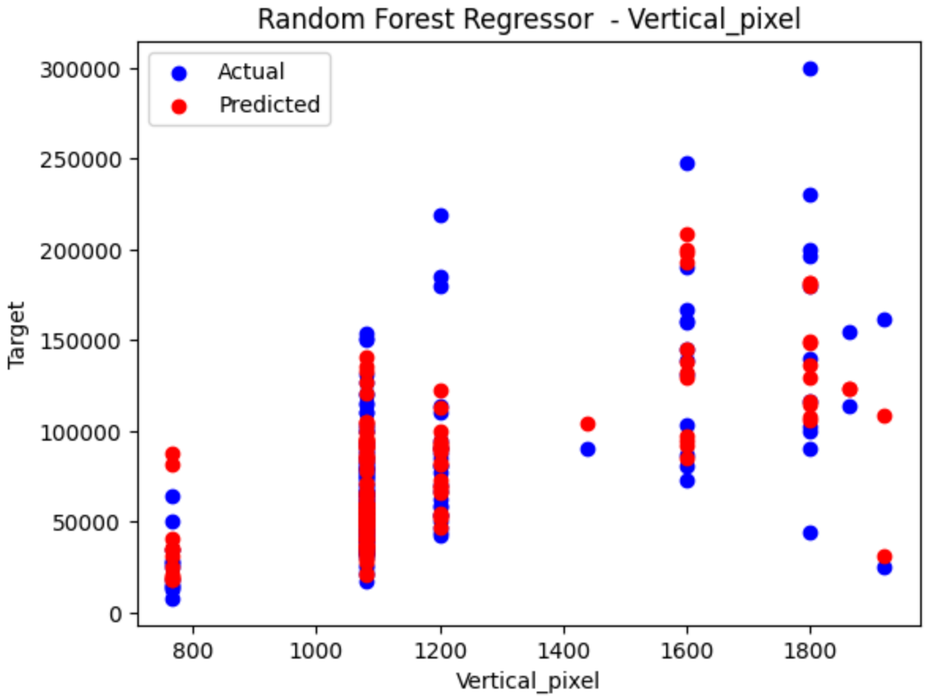

Random Forest Regressor - Storage_capacity_GB
Actual
Predicted
Target
Storage_capacity_GB
0
50000
100000
150000
200000
250000
300000
0
200
400
600
800
1000
1200
1400

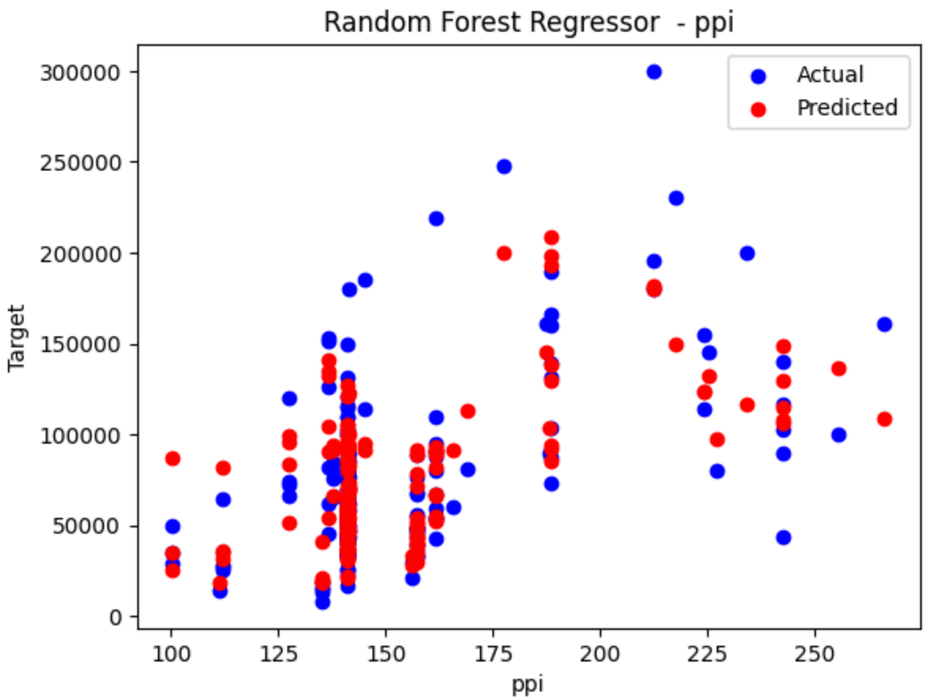

Random Forest Regressor - Storage_type
Actual
Predicted
Target
Storage_type
300000
250000
200000
150000
100000
50000
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

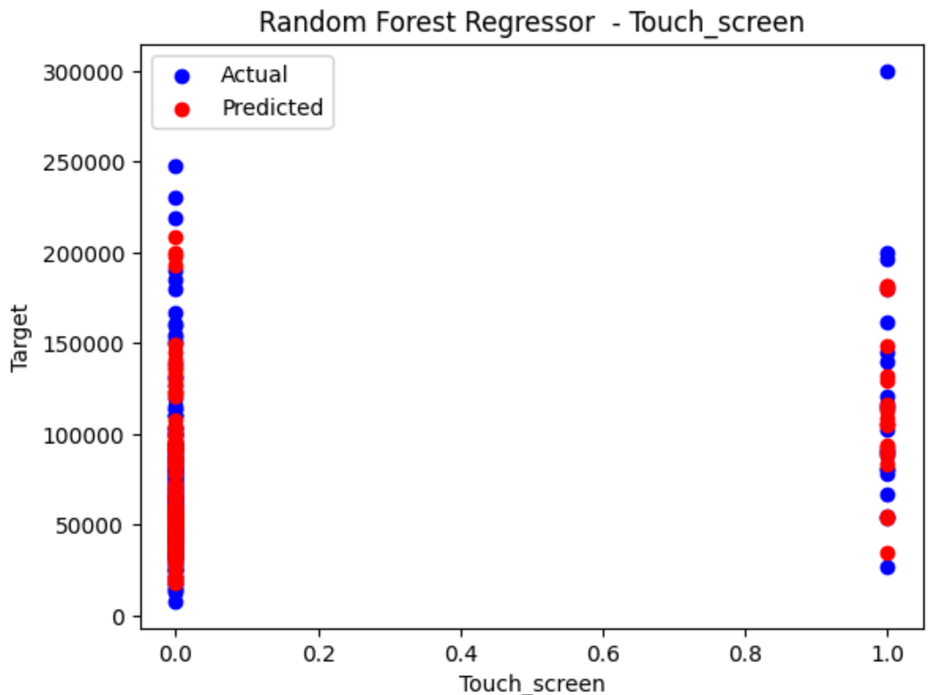

Random Forest Regressor - Graphics_integreted
Actual
Predicted
Target
Graphics_integreted
300000
250000
200000
150000
100000
50000
0
0.0
0.2
0.4
0.6
0.8
1.0

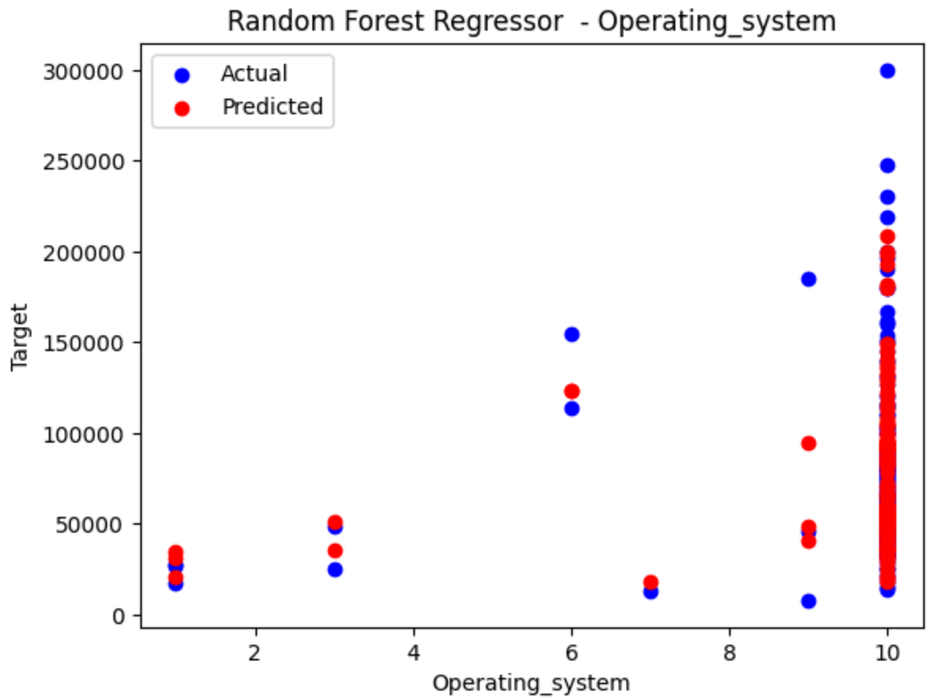

Random Forest Regressor - Display_size_inches
Actual
Predicted
Target
Display_size_inches
300000
250000
200000
150000
100000
50000
0
12
13
14
15
16
17


Random Forest Regressor - Horizontal_pixel
Actual
Predicted
Target
Horizontal_pixel
300000
250000
200000
150000
100000
50000
0
1000
1250
1500
1750
2000
2250
2500
2750


Random Forest Regressor - Vertical_pixel
Actual
Predicted
Target
Vertical_pixel
0
50000
100000
150000
200000
250000
300000
800
1000
1200
1400
1600
1800


Random Forest Regressor - ppi
Actual
Predicted
Target
ppi
0
50000
100000
150000
200000
250000
300000
100
125
150
175
200
225
250





In [ ]:
```
pd.DataFrame({'actual':y_test,'prediction':y_pred})
```

Out[ ]:

	actual	prediction
322	102099	94847.773333
144	119990	98841.040000
518	33990	53811.467560
885	40300	53869.762833
91	57990	86152.061038
...	...	...
685	32990	35040.472083
654	51980	71193.458831
474	149990	99556.569167
558	82588	58266.870000
309	179990	181703.049333

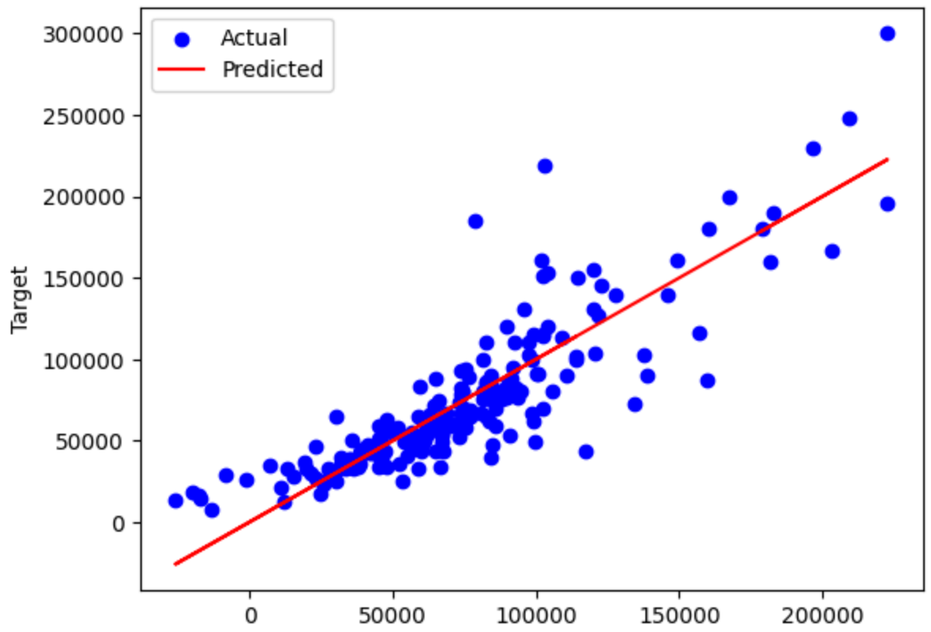
198 rows × 2 columns

In [ ]:
```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

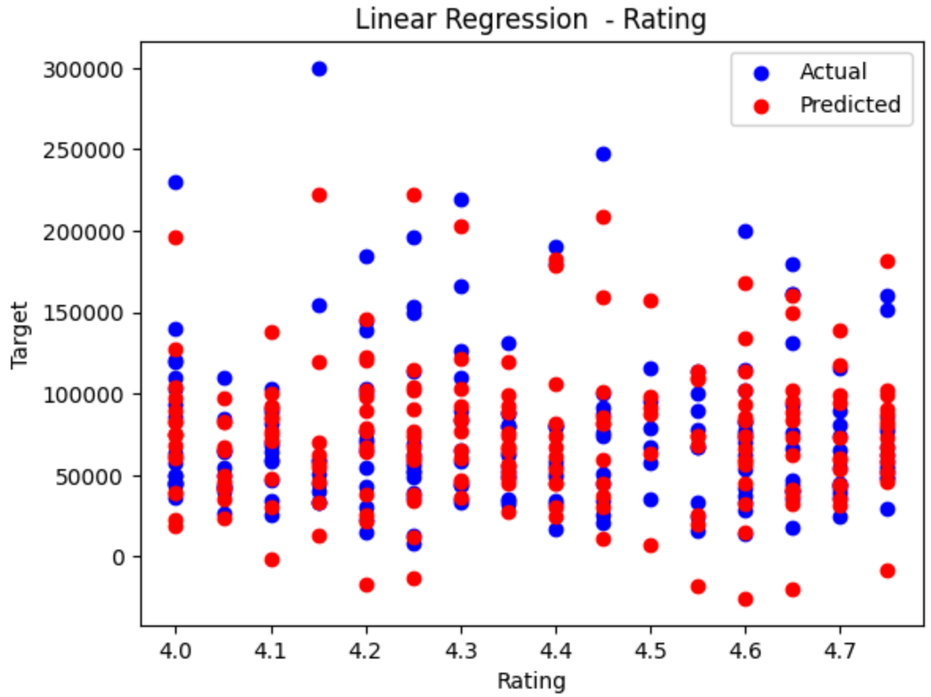
In [ ]:
```
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared Score (R^2):", r2)
```

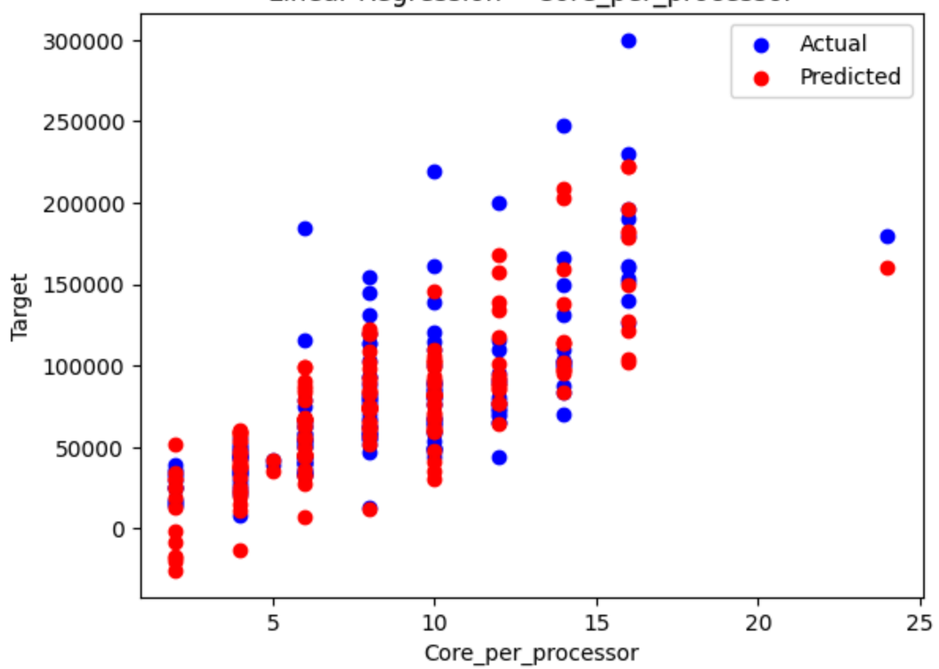
```
Root Mean Squared Error (RMSE): 23901.44784708819
R-squared Score (R^2): 0.7363009759228678
```

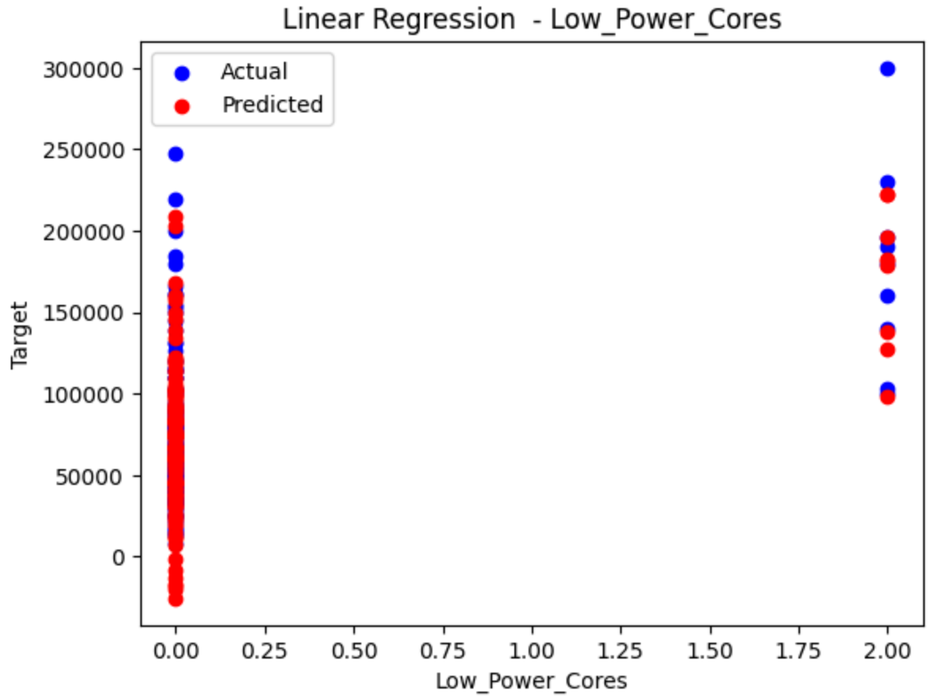
In [ ]:
```
plt.scatter(y_pred, y_test, label='Actual', color='blue')
plt.plot(y_pred, y_pred, label='Predicted', color='red')
plt.ylabel('Target')
plt.legend()
plt.show()
```

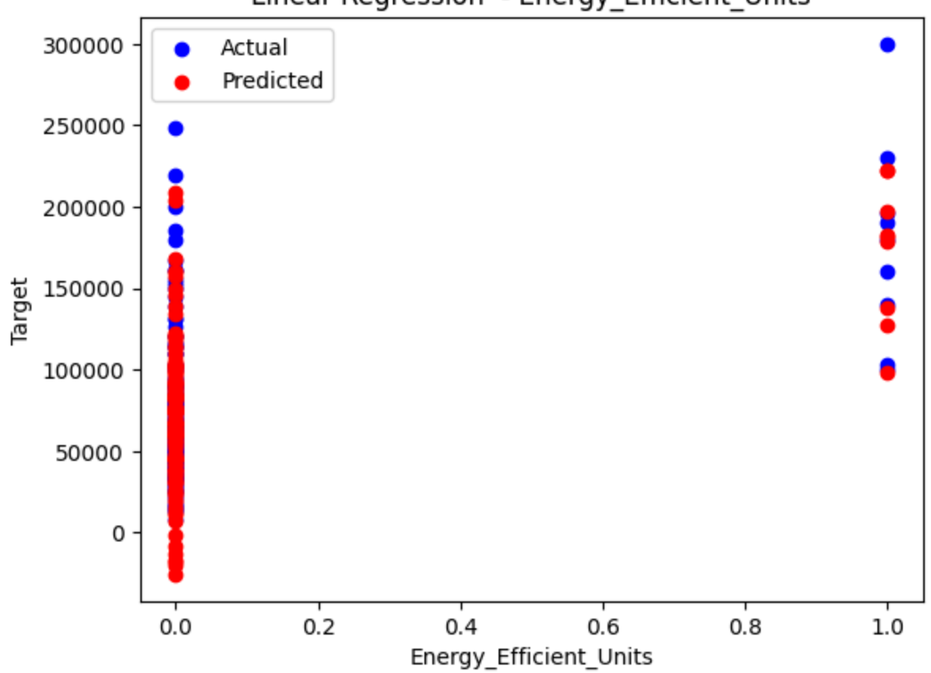


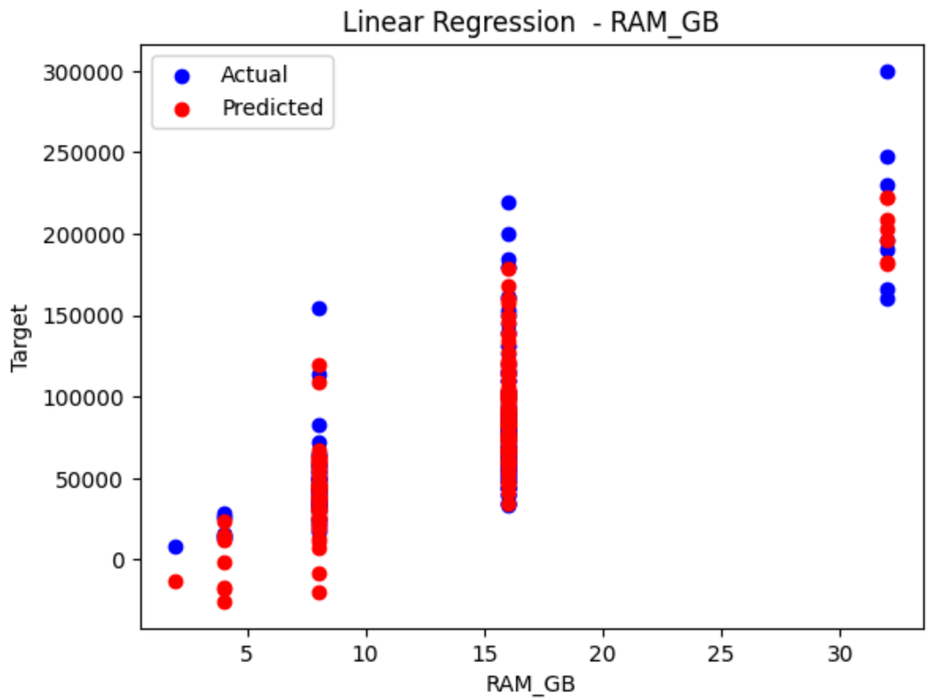
In [ ]:
```
for column in X_test.columns:
    plt.scatter(X_test[column], y_test, label='Actual', color='blue')
    plt.scatter(X_test[column], y_pred, label='Predicted', color='red')
    plt.xlabel(column)
    plt.ylabel('Target')
    plt.title('Linear Regression  - {}'.format(column))
    plt.legend()
    plt.show()
```

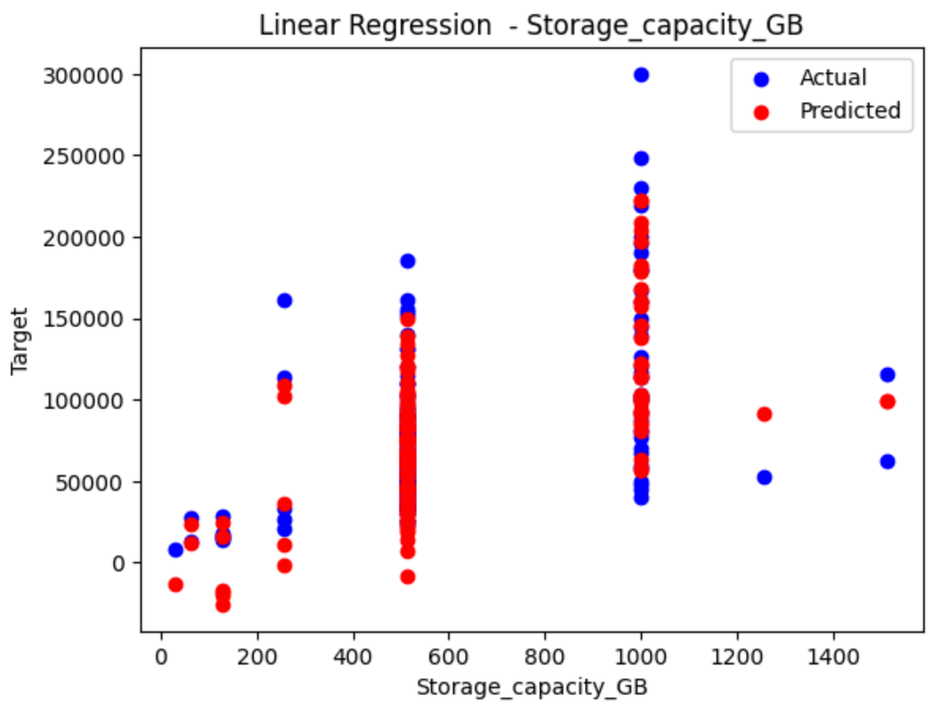

Linear Regression - Rating
Actual
Predicted
Target
Rating
300000
250000
200000
150000
100000
50000
0
4.0
4.1
4.2
4.3
4.4
4.5
4.6
4.7

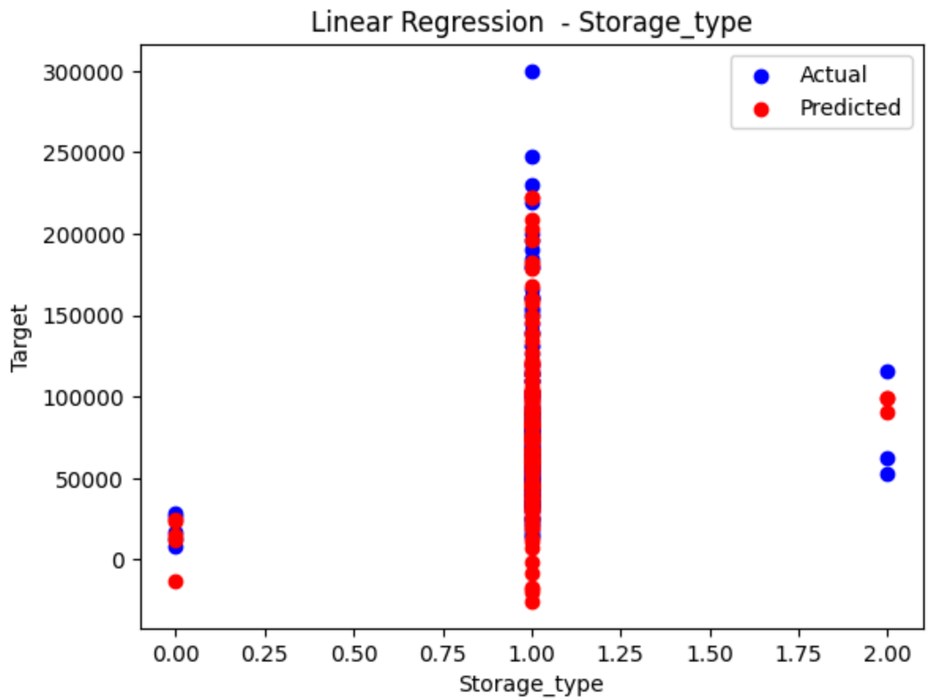

Linear Regression - Core_per_processor
Actual
Predicted
Target
Core_per_processor
300000
250000
200000
150000
100000
50000
0
5
10
15
20
25

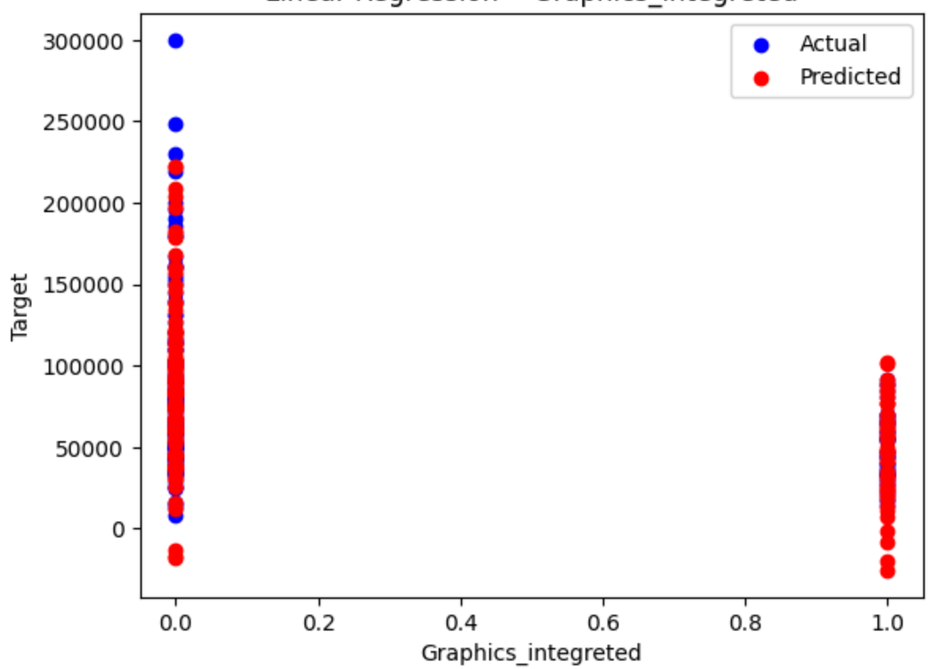

Linear Regression - Low_Power_Cores
Actual
Predicted
Target
Low_Power_Cores
300000
250000
200000
150000
100000
50000
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

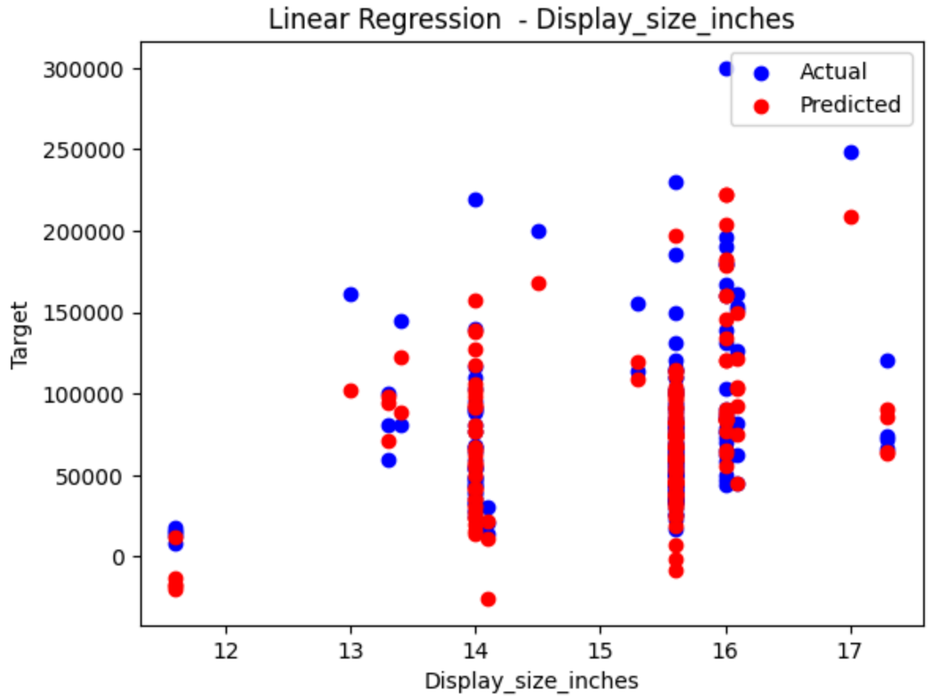

Linear Regression - Energy_Efficient_Units
Actual
Predicted
Target
Energy_Efficient_Units
300000
250000
200000
150000
100000
50000
0
0.0
0.2
0.4
0.6
0.8
1.0

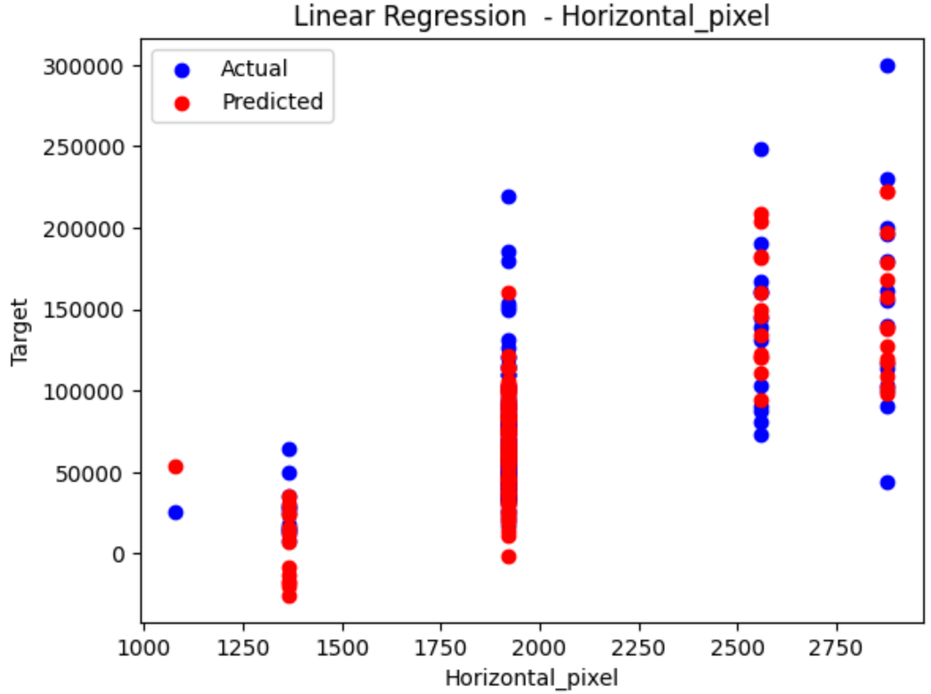

Linear Regression - RAM_GB
Actual
Predicted
Target
RAM_GB
300000
250000
200000
150000
100000
50000
0
5
10
15
20
25
30

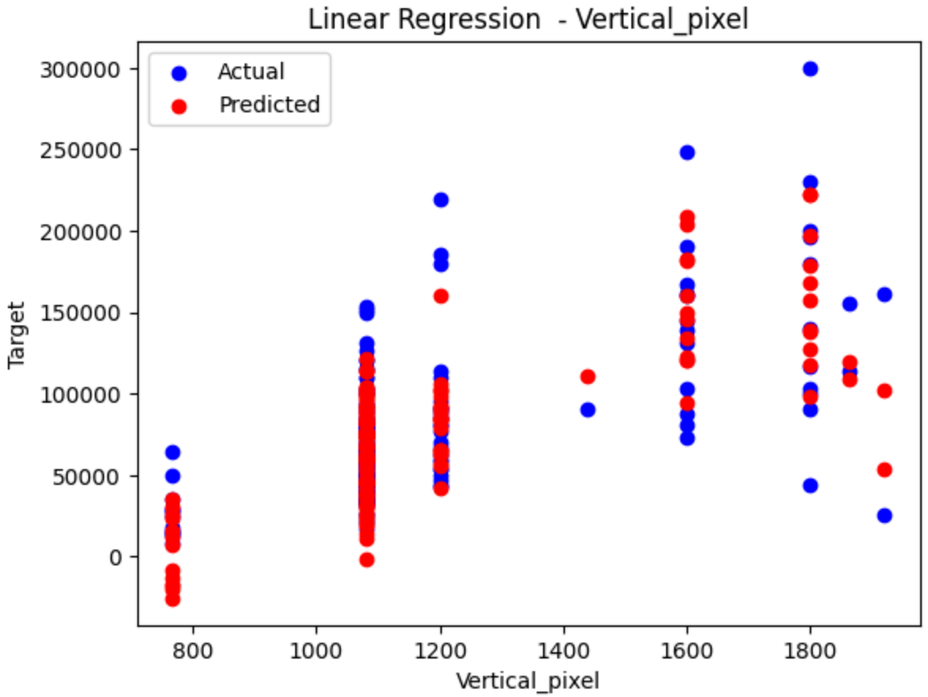

Linear Regression - Storage_capacity_GB
Actual
Predicted
Target
Storage_capacity_GB
300000
250000
200000
150000
100000
50000
0
0
200
400
600
800
1000
1200
1400

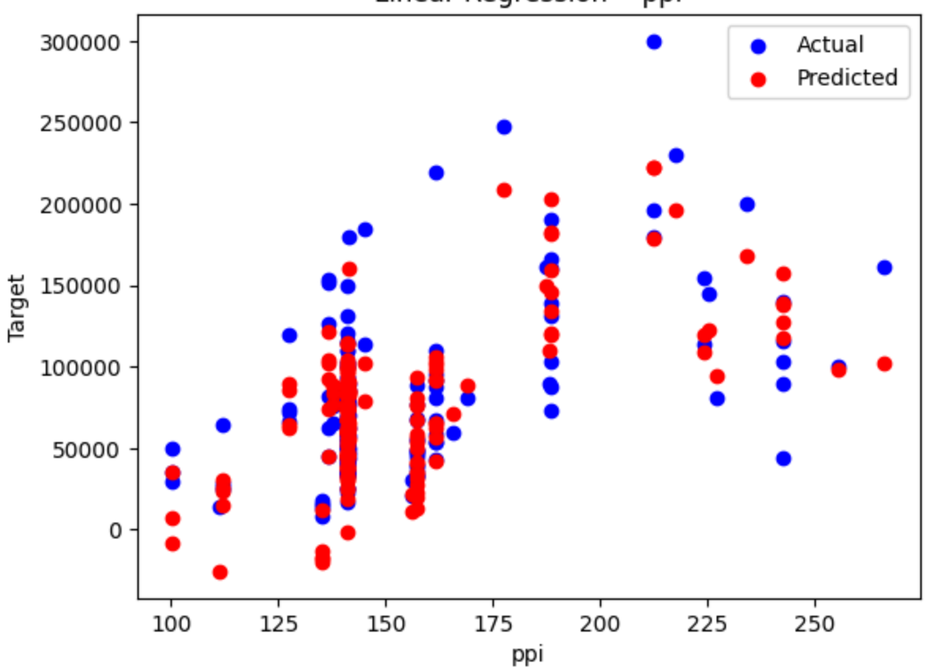

Linear Regression - Storage_type
Actual
Predicted
Target
Storage_type
300000
250000
200000
150000
100000
50000
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

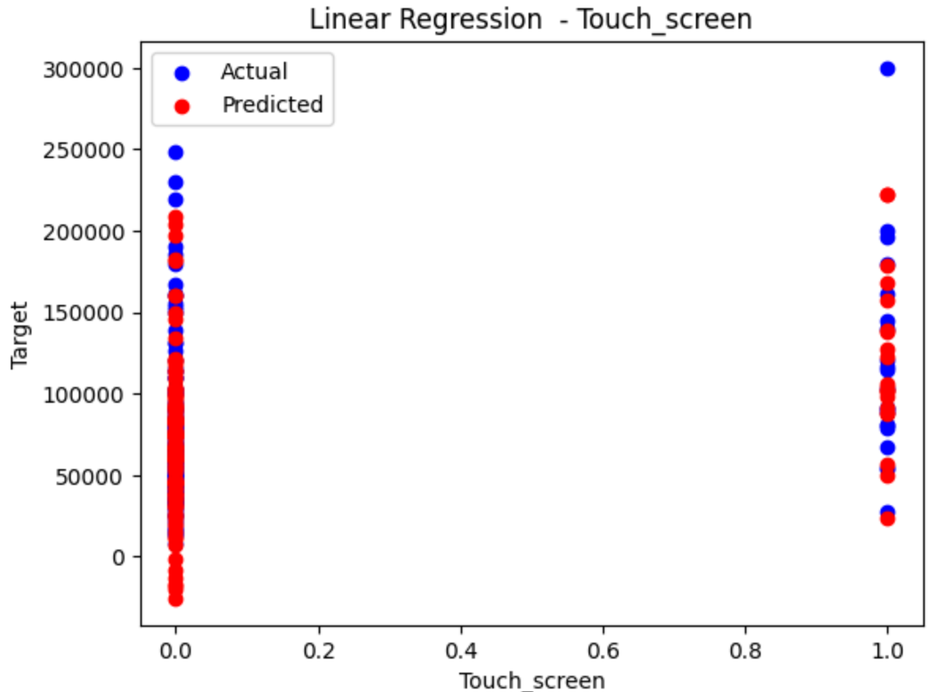

Linear Regression - Graphics_integreted
Actual
Predicted
Target
Graphics_integreted
300000
250000
200000
150000
100000
50000
0
0.0
0.2
0.4
0.6
0.8
1.0

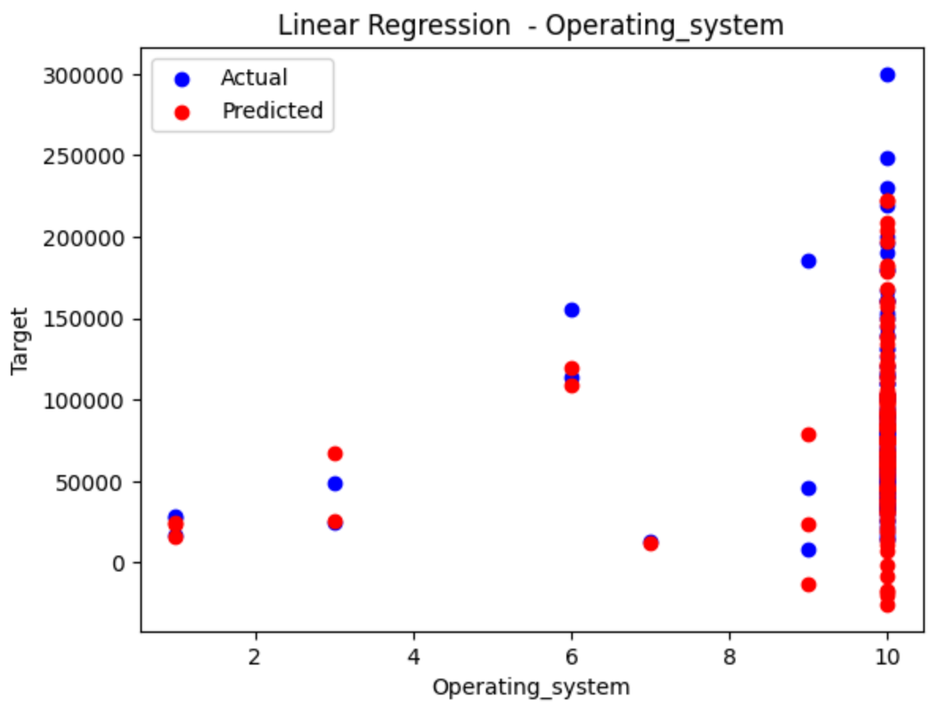

Linear Regression - Display_size_inches
Actual
Predicted
Target
Display_size_inches
300000
250000
200000
150000
100000
50000
0
12
13
14
15
16
17


Linear Regression - Horizontal_pixel
Actual
Predicted
Target
Horizontal_pixel
300000
250000
200000
150000
100000
50000
0
1000
1250
1500
1750
2000
2250
2500
2750


Linear Regression - Vertical_pixel
Actual
Predicted
Target
Vertical_pixel
300000
250000
200000
150000
100000
50000
0
800
1000
1200
1400
1600
1800


Linear Regression - ppi
Actual
Predicted
Target
ppi
300000
250000
200000
150000
100000
50000
0
100
125
150
175
200
225
250





In [ ]:
```
pd.DataFrame({'actual':y_test,'prediction':y_pred})
```

Out[ ]:

	actual	prediction
322	102099	113720.801968
144	119990	89910.135318
518	33990	45315.910473
885	40300	54993.952549
91	57990	75245.653761
...	...	...
685	32990	36015.110443
654	51980	73213.739844
474	149990	114669.028463
558	82588	73620.122627
309	179990	178778.816933

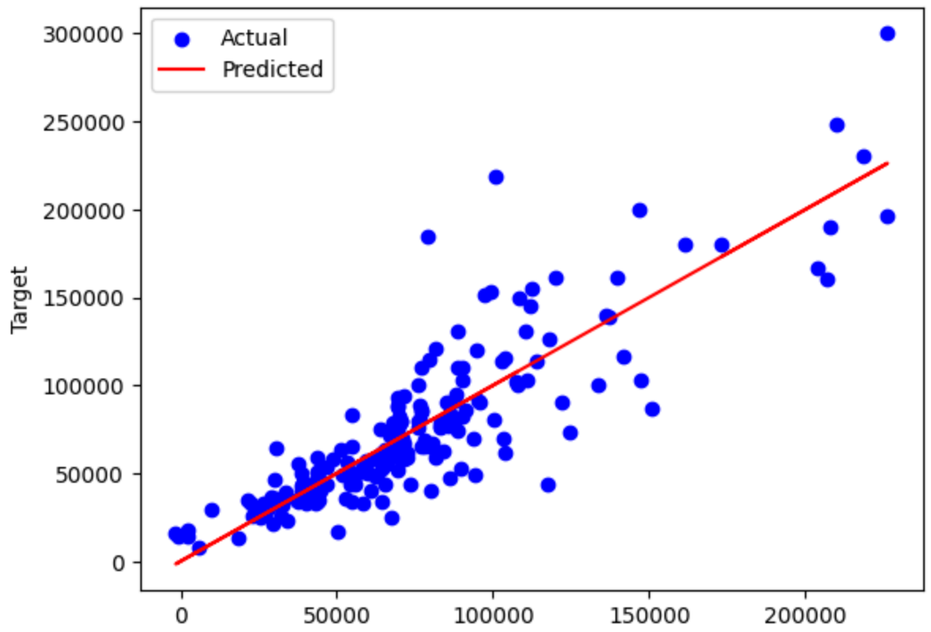
198 rows × 2 columns

In [ ]:
```
from sklearn.neural_network import MLPRegressor

# Assuming X_train, X_test, y_train, y_test are your training and testing data
model = MLPRegressor(hidden_layer_sizes=(100, 50), max_iter=5000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

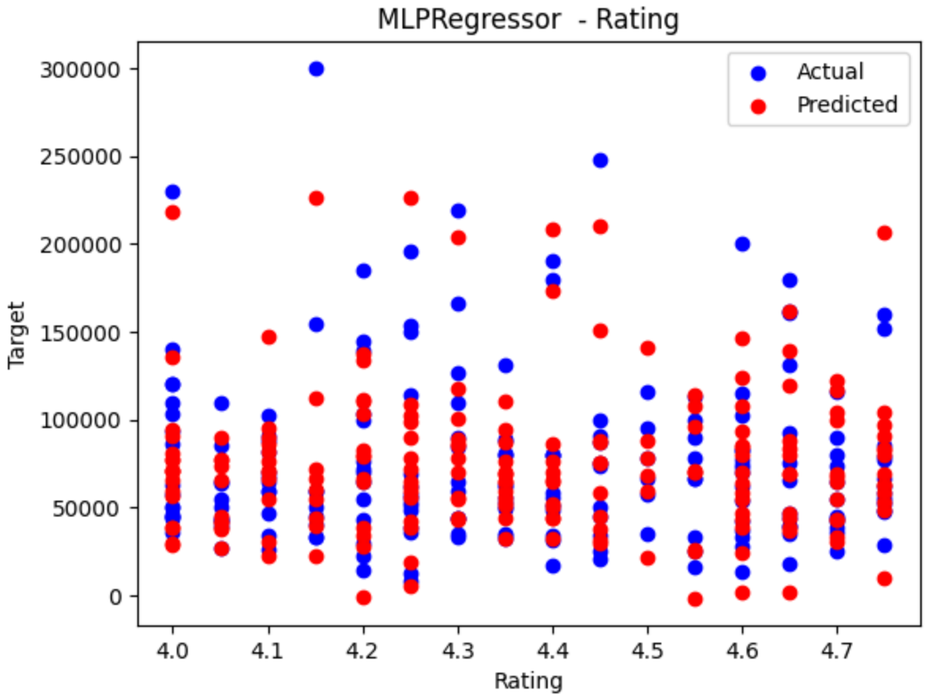
In [ ]:
```
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared Score (R^2):", r2)
```

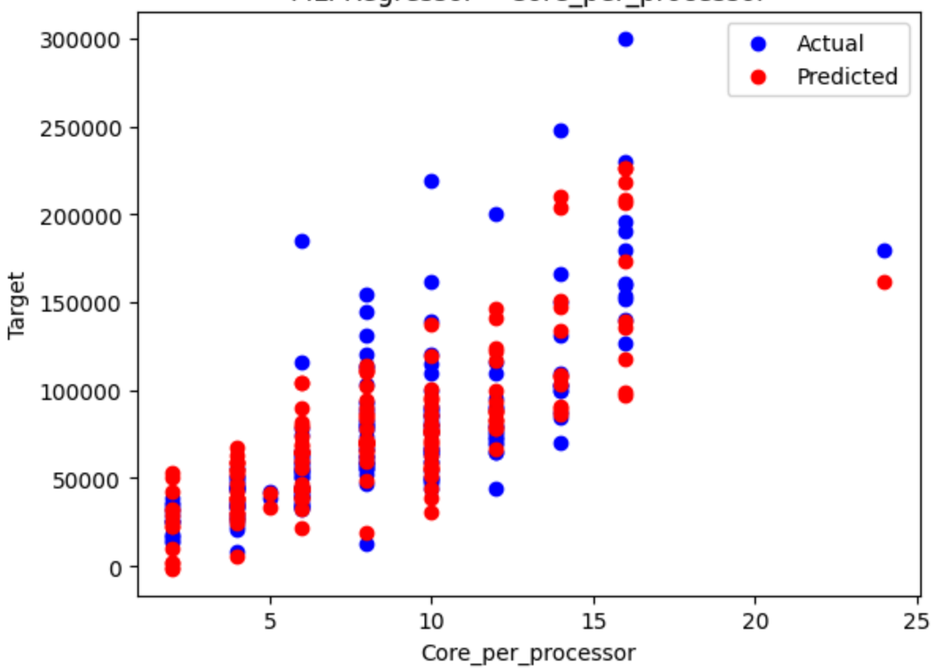
```
Root Mean Squared Error (RMSE): 23572.58488186279
R-squared Score (R^2): 0.7435075888573958
```

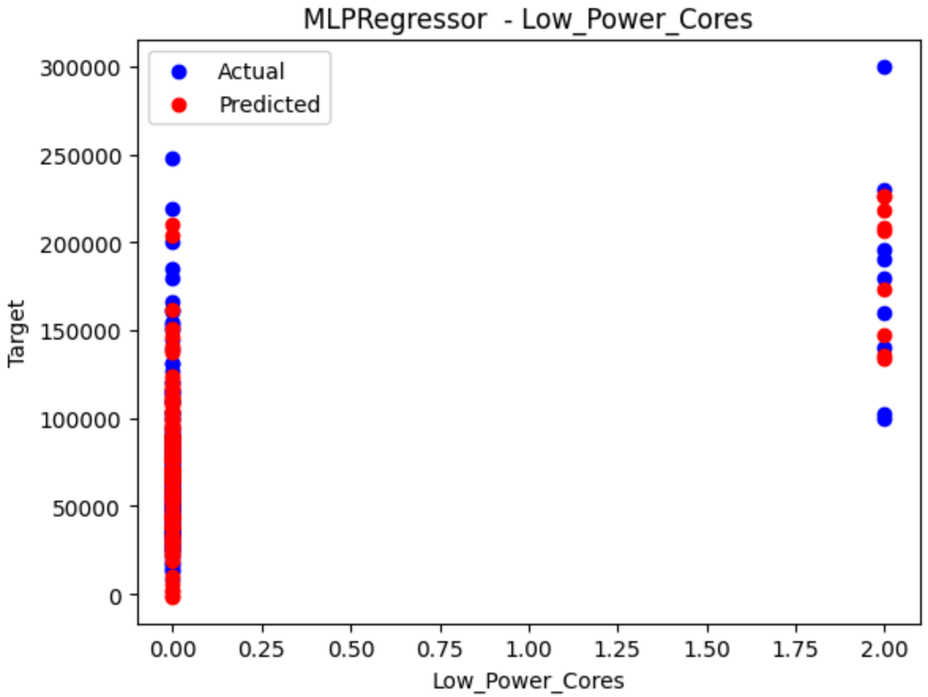
In [ ]:
```
plt.scatter(y_pred, y_test, label='Actual', color='blue')
plt.plot(y_pred, y_pred, label='Predicted', color='red')
plt.ylabel('Target')
plt.legend()
plt.show()
```

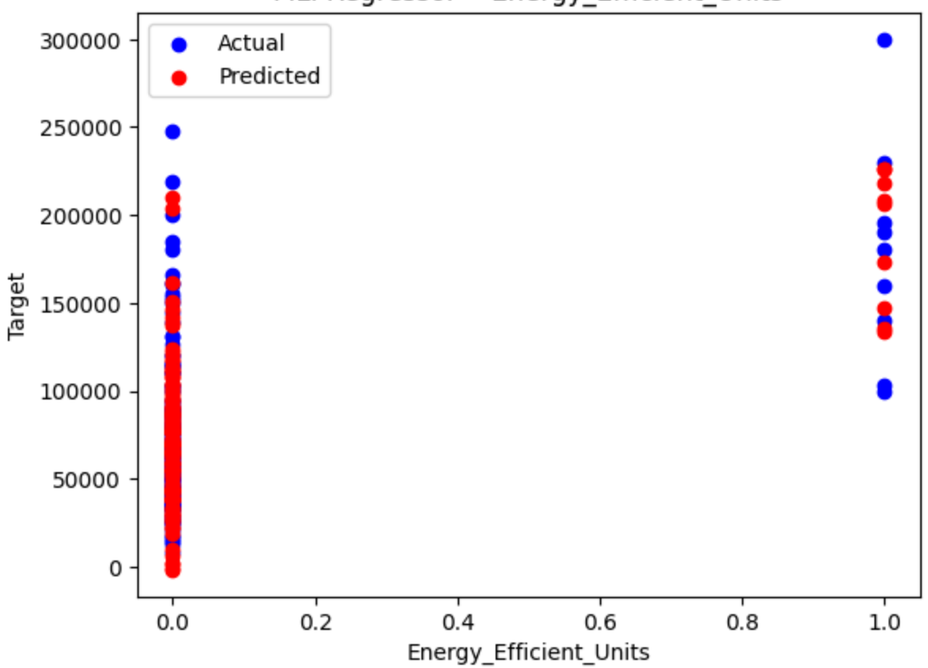


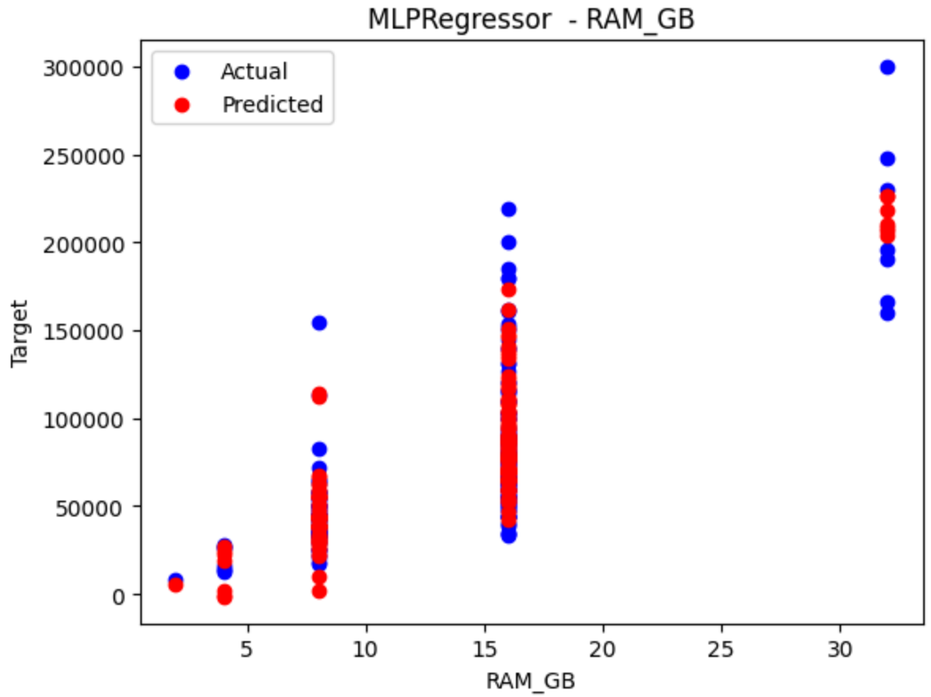
In [ ]:
```
for column in X_test.columns:
    plt.scatter(X_test[column], y_test, label='Actual', color='blue')
    plt.scatter(X_test[column], y_pred, label='Predicted', color='red')
    plt.xlabel(column)
    plt.ylabel('Target')
    plt.title('MLPRegressor  - {}'.format(column))
    plt.legend()
    plt.show()
```

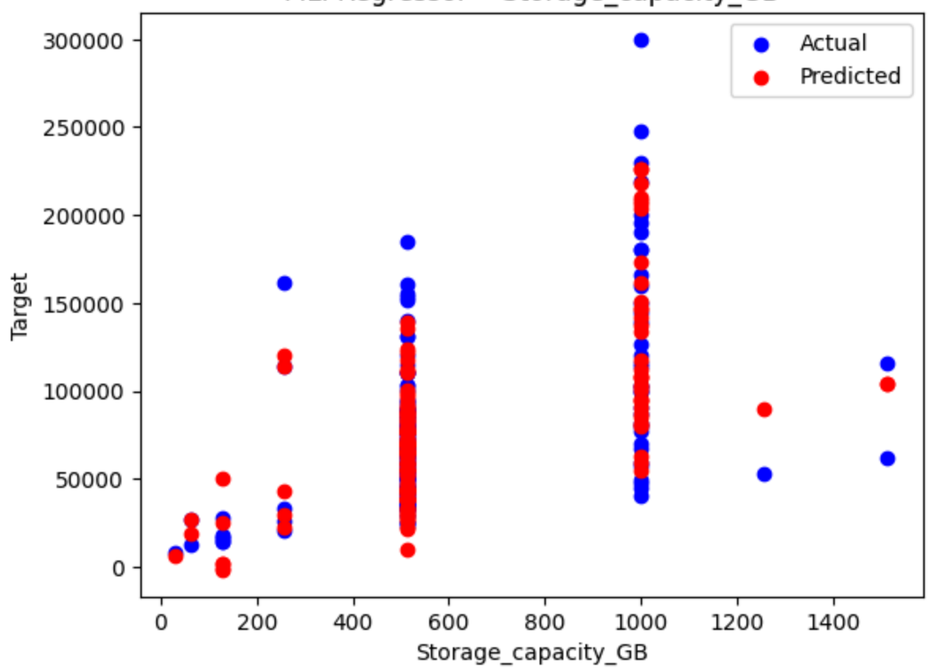

MLPRegressor - Rating
Actual
Predicted
Target
Rating
300000
250000
200000
150000
100000
50000
0
4.0
4.1
4.2
4.3
4.4
4.5
4.6
4.7

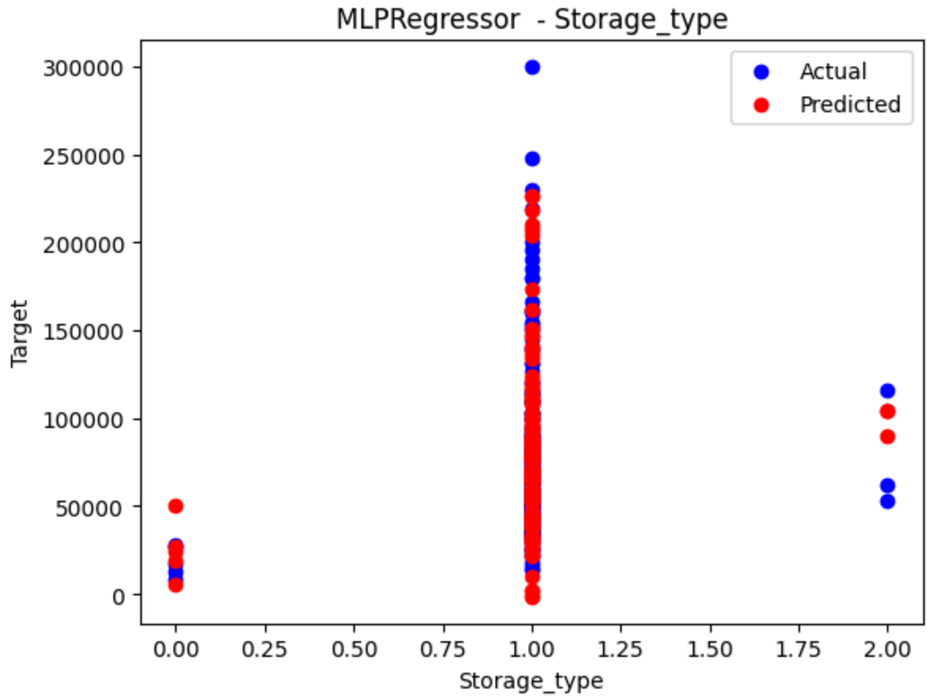

MLPRegressor - Core_per_processor
Actual
Predicted
Target
Core_per_processor
300000
250000
200000
150000
100000
50000
0
5
10
15
20
25

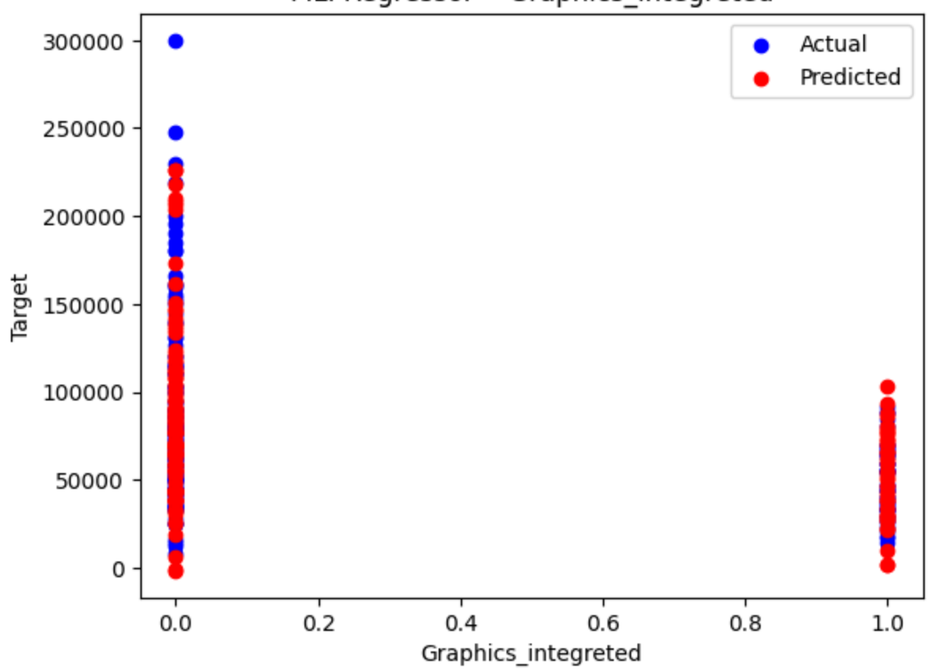

MLPRegressor - Low_Power_Cores
Actual
Predicted
Target
Low_Power_Cores
0
50000
100000
150000
200000
250000
300000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

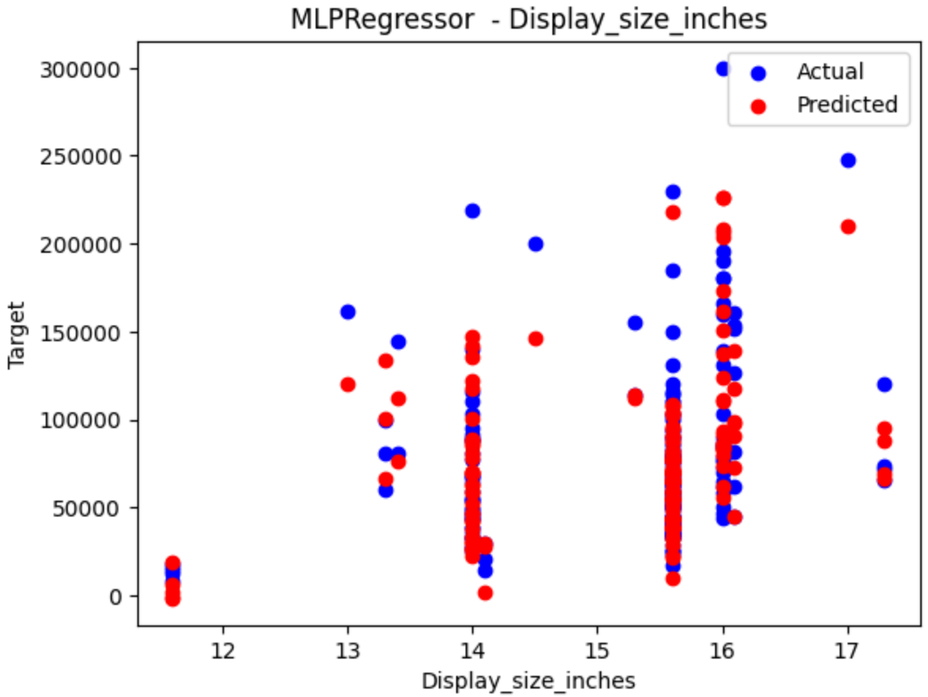

MLPRegressor - Energy_Efficient_Units
Actual
Predicted
Target
Energy_Efficient_Units
0
50000
100000
150000
200000
250000
300000
0.0
0.2
0.4
0.6
0.8
1.0

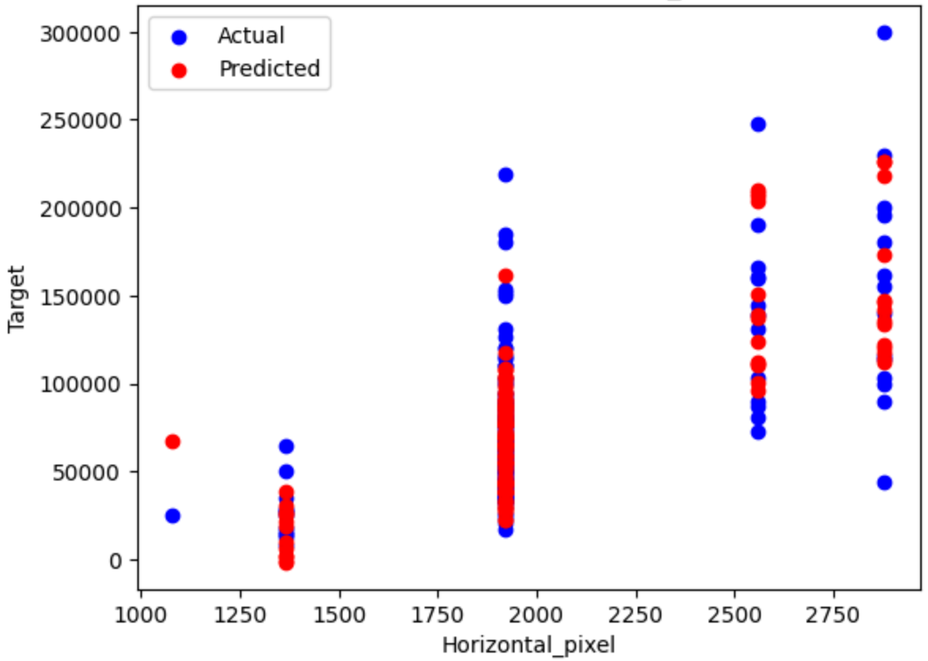

MLPRegressor - RAM_GB
Actual
Predicted
Target
RAM_GB
0
50000
100000
150000
200000
250000
300000
5
10
15
20
25
30

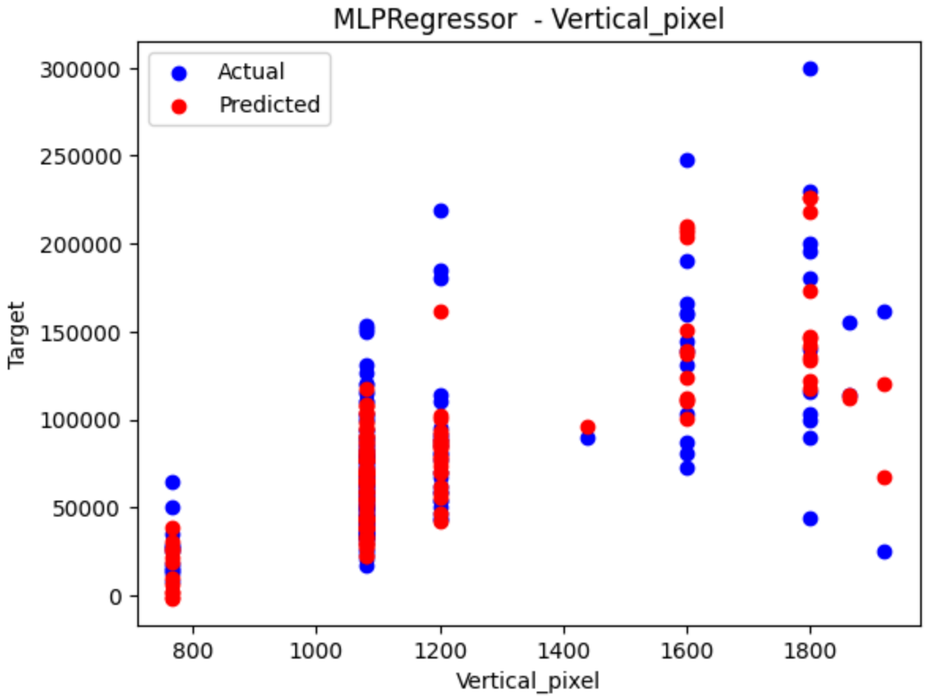

MLPRegressor - Storage_capacity_GB
Actual
Predicted
Target
Storage_capacity_GB
0
50000
100000
150000
200000
250000
300000
0
200
400
600
800
1000
1200
1400

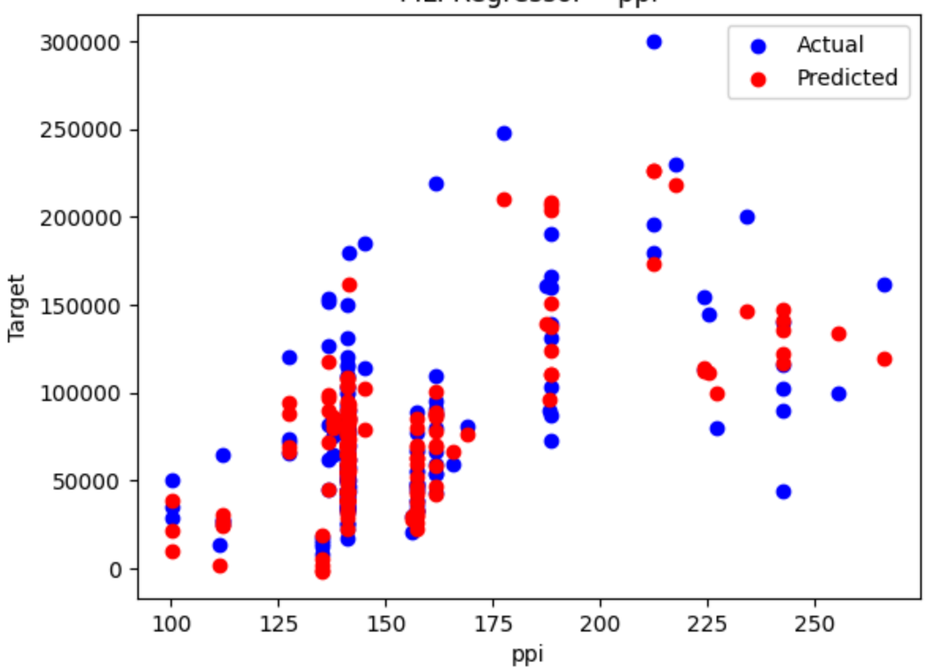

MLPRegressor - Storage_type
Actual
Predicted
Target
Storage_type
300000
250000
200000
150000
100000
50000
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

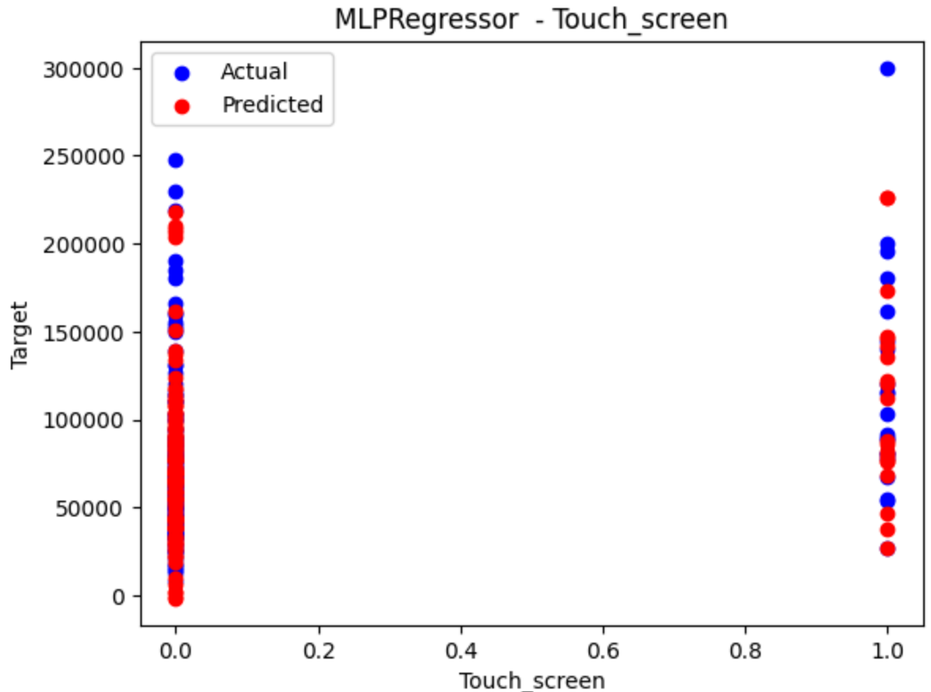

MLPRegressor - Graphics_integreted
Actual
Predicted
Target
Graphics_integreted
300000
250000
200000
150000
100000
50000
0
0.0
0.2
0.4
0.6
0.8
1.0

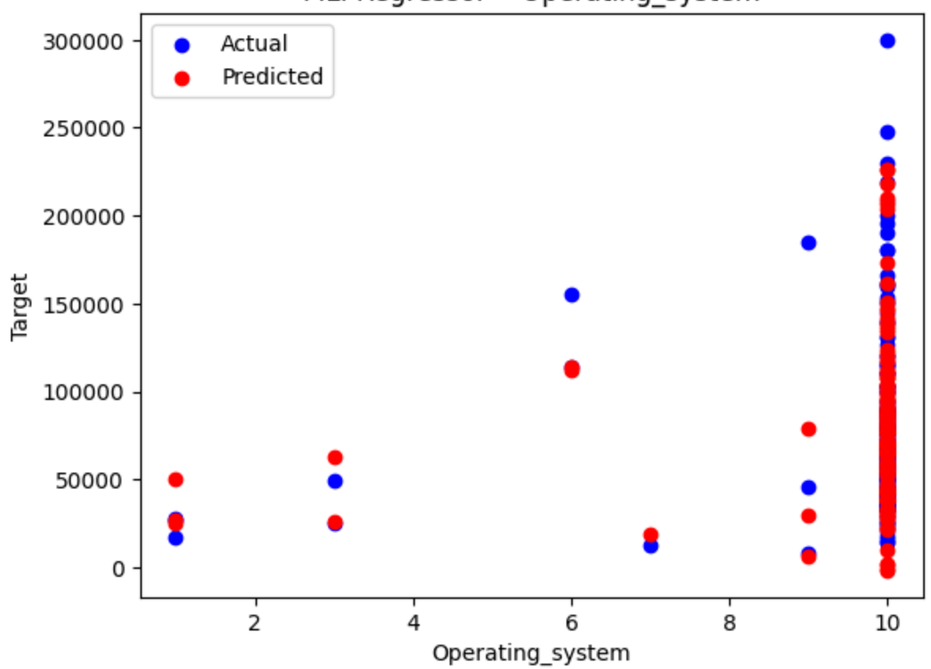

MLPRegressor - Display_size_inches
Actual
Predicted
Target
Display_size_inches
300000
250000
200000
150000
100000
50000
0
12
13
14
15
16
17


MLPRegressor - Horizontal_pixel
Actual
Predicted
Target
Horizontal_pixel
300000
250000
200000
150000
100000
50000
0
1000
1250
1500
1750
2000
2250
2500
2750


MLPRegressor - Vertical_pixel
Actual
Predicted
Target
Vertical_pixel
0
50000
100000
150000
200000
250000
300000
800
1000
1200
1400
1600
1800


MLPRegressor - ppi
Actual
Predicted
Target
ppi
0
50000
100000
150000
200000
250000
300000
100
125
150
175
200
225
250





In [ ]:
```
pd.DataFrame({'actual':y_test,'prediction':y_pred})
```

Out[ ]:

	actual	prediction
322	102099	107471.520114
144	119990	94825.959023
518	33990	43908.637932
885	40300	60743.945626
91	57990	71267.180675
...	...	...
685	32990	43291.334124
654	51980	69402.273444
474	149990	108565.545032
558	82588	69775.254890
309	179990	173184.490433

198 rows × 2 columns