In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import math
%matplotlib inline
import warnings

warnings.filterwarnings("ignore")
%matplotlib inline
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor


from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
```

In [2]:
```
pwd
```

Out[2]:
```
'C:\\Users\\chetna'
```

Download the dataset from above link and load it into your Python environment.

Problem Statement: Predicting the Price Range of Mobile Phones Based on Their Specifications

The goal of this project is to build a machine learning model that predicts the price range of mobile phones based on their specifications. Given a set of features such as battery power, RAM, Rear and Frount camera quality, screen size, and other technical specifications, we aim to predict the price category of

the mobile phone.

In [5]:
```
data=pd.read_csv("Cellphone.csv")
data.head()
```

Out[5]:

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
0	203	2357	10	135.0	5.2	424	8	1.35	16.0	3.000	13.00	8.0	2610	7.4
1	880	1749	10	125.0	4.0	233	2	1.30	4.0	1.000	3.15	0.0	1700	9.9
2	40	1916	10	110.0	4.7	312	4	1.20	8.0	1.500	13.00	5.0	2000	7.6
3	99	1315	11	118.5	4.0	233	2	1.30	4.0	0.512	3.15	0.0	1400	11.0
4	880	1749	11	125.0	4.0	233	2	1.30	4.0	1.000	3.15	0.0	1700	9.9

In [7]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 161 entries, 0 to 160
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Product_id    161 non-null    int64
 1   Price         161 non-null    int64
 2   Sale          161 non-null    int64
 3   weight        161 non-null    float64
 4   resoloution   161 non-null    float64
 5   ppi           161 non-null    int64
 6   cpu core      161 non-null    int64
 7   cpu freq      161 non-null    float64
 8   internal mem  161 non-null    float64
 9   ram           161 non-null    float64
 10  RearCam       161 non-null    float64
 11  Front_Cam     161 non-null    float64
 12  battery       161 non-null    int64
 13  thickness     161 non-null    float64
dtypes: float64(8), int64(6)
memory usage: 17.7 KB
```

In [9]:
```
data.shape
```

Out[9]: (161, 14)

In [11]:
```
data.columns
```

Out[11]:
```
Index(['Product_id', 'Price', 'Sale', 'weight', 'resoloution', 'ppi',
       'cpu core', 'cpu freq', 'internal mem', 'ram', 'RearCam', 'Front_Cam',
       'battery', 'thickness'],
      dtype='object')
```

In [13]:
```
data.isnull().sum()
```

Out[13]:
```
Product_id      0
Price           0
Sale            0
weight          0
resoloution     0
ppi             0
cpu core        0
cpu freq        0
internal mem    0
ram             0
RearCam         0
Front_Cam       0
battery         0
thickness       0
dtype: int64
```

In [15]:
```
data.duplicated().sum()
```

Out[15]:
```
0
```

In [17]:
```
pd.options.display.float_format = '{:.2f}'.format
data.describe()
```

Out[17]:

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
count	161.00	161.00	161.00	161.00	161.00	161.00	161.00	161.00	161.00	161.00	161.00	161.00	161.00	161.00
mean	675.56	2215.60	621.47	170.43	5.21	335.06	4.86	1.50	24.50	2.20	10.38	4.50	2842.11	8.92
std	410.85	768.19	1546.62	92.89	1.51	134.83	2.44	0.60	28.80	1.61	6.18	4.34	1366.99	2.19
min	10.00	614.00	10.00	66.00	1.40	121.00	0.00	0.00	0.00	0.00	0.00	0.00	800.00	5.10
25%	237.00	1734.00	37.00	134.10	4.80	233.00	4.00	1.20	8.00	1.00	5.00	0.00	2040.00	7.60
50%	774.00	2258.00	106.00	153.00	5.15	294.00	4.00	1.40	16.00	2.00	12.00	5.00	2800.00	8.40
75%	1026.00	2744.00	382.00	170.00	5.50	428.00	8.00	1.88	32.00	3.00	16.00	8.00	3240.00	9.80
max	1339.00	4361.00	9807.00	753.00	12.20	806.00	8.00	2.70	128.00	6.00	23.00	20.00	9500.00	18.50

In [19]:

```
pd.options.display.float_format = '{:.2f}'.format
data.describe(include=["int64","float64"]).T
```

Out[19]:

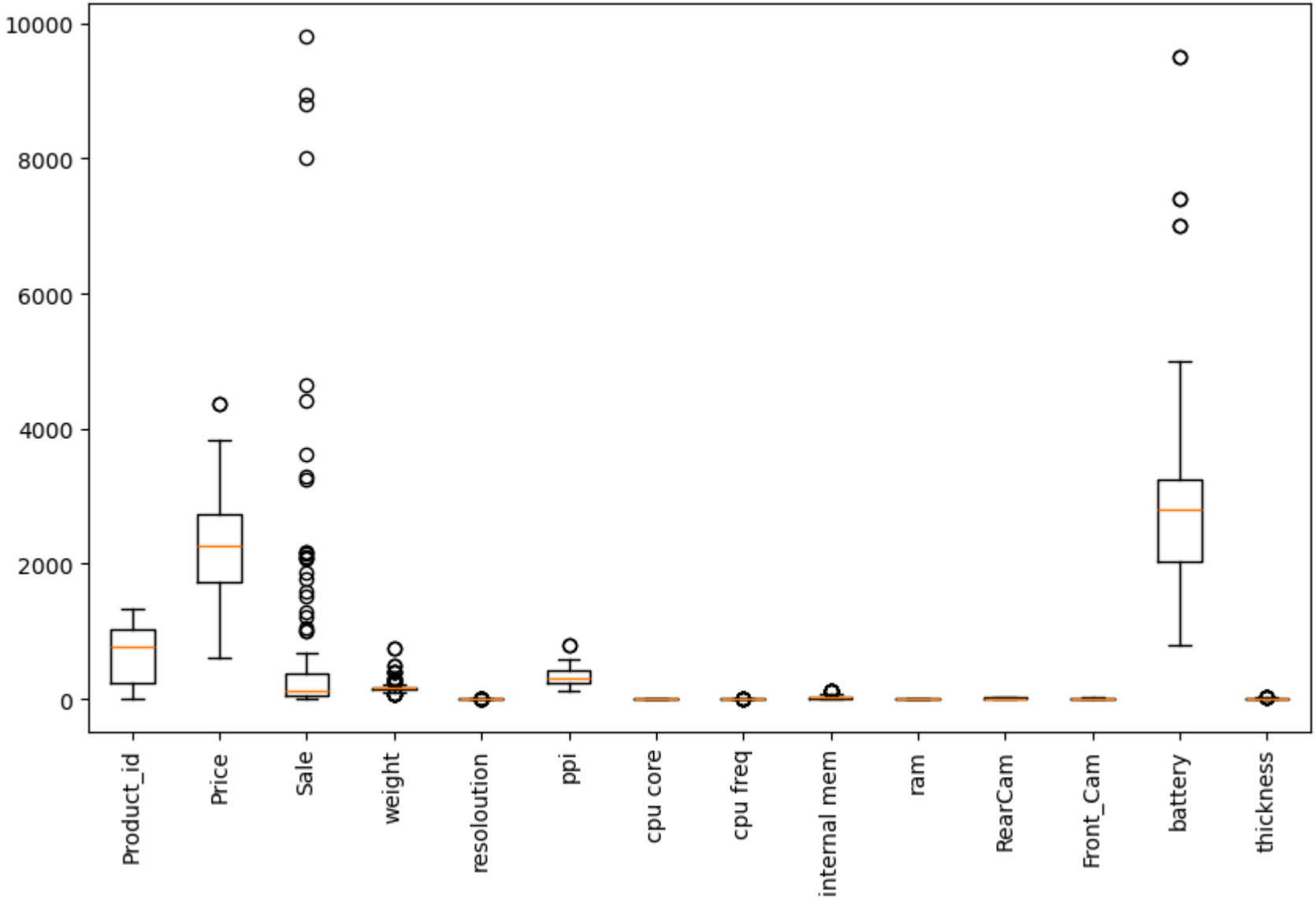
	count	mean	std	min	25%	50%	75%	max
Product_id	161.00	675.56	410.85	10.00	237.00	774.00	1026.00	1339.00
Price	161.00	2215.60	768.19	614.00	1734.00	2258.00	2744.00	4361.00
Sale	161.00	621.47	1546.62	10.00	37.00	106.00	382.00	9807.00
weight	161.00	170.43	92.89	66.00	134.10	153.00	170.00	753.00
resoloution	161.00	5.21	1.51	1.40	4.80	5.15	5.50	12.20
ppi	161.00	335.06	134.83	121.00	233.00	294.00	428.00	806.00
cpu core	161.00	4.86	2.44	0.00	4.00	4.00	8.00	8.00
cpu freq	161.00	1.50	0.60	0.00	1.20	1.40	1.88	2.70
internal mem	161.00	24.50	28.80	0.00	8.00	16.00	32.00	128.00
ram	161.00	2.20	1.61	0.00	1.00	2.00	3.00	6.00
RearCam	161.00	10.38	6.18	0.00	5.00	12.00	16.00	23.00
Front_Cam	161.00	4.50	4.34	0.00	0.00	5.00	8.00	20.00
battery	161.00	2842.11	1366.99	800.00	2040.00	2800.00	3240.00	9500.00
thickness	161.00	8.92	2.19	5.10	7.60	8.40	9.80	18.50

In [ ]:

```
data.describe(include="object")
#No categorical data is present in the dataset
```

In [21]:

```
plt.figure(figsize=(10, 6))
plt.boxplot(data,labels=data.columns)
plt.xticks(rotation=90)
plt.show()
```



ii. Perform the EDA and do the visualizations.

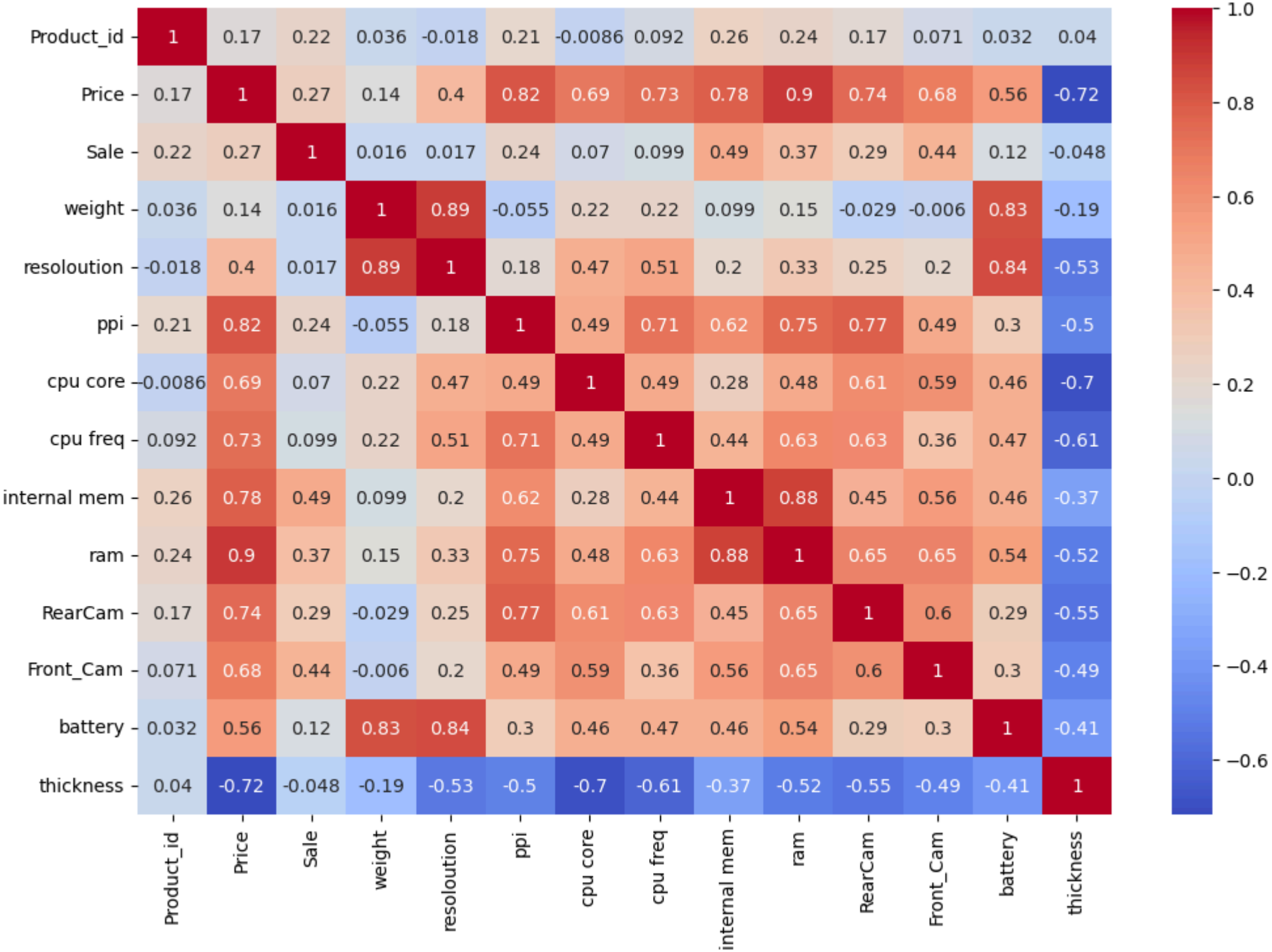
In [21]:
```
corr_matrix=data.corr()
corr_matrix
```

Out[21]:

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
Product_id	1.00	0.17	0.22	0.04	-0.02	0.21	-0.01	0.09	0.26	0.24	0.17	0.07	0.03	0.04
Price	0.17	1.00	0.27	0.14	0.40	0.82	0.69	0.73	0.78	0.90	0.74	0.68	0.56	-0.72
Sale	0.22	0.27	1.00	0.02	0.02	0.24	0.07	0.10	0.49	0.37	0.29	0.44	0.12	-0.05
weight	0.04	0.14	0.02	1.00	0.89	-0.05	0.22	0.22	0.10	0.15	-0.03	-0.01	0.83	-0.19
resoloution	-0.02	0.40	0.02	0.89	1.00	0.18	0.47	0.51	0.20	0.33	0.25	0.20	0.84	-0.53
ppi	0.21	0.82	0.24	-0.05	0.18	1.00	0.49	0.71	0.62	0.75	0.77	0.49	0.30	-0.50
cpu core	-0.01	0.69	0.07	0.22	0.47	0.49	1.00	0.49	0.28	0.48	0.61	0.59	0.46	-0.70
cpu freq	0.09	0.73	0.10	0.22	0.51	0.71	0.49	1.00	0.44	0.63	0.63	0.36	0.47	-0.61
internal mem	0.26	0.78	0.49	0.10	0.20	0.62	0.28	0.44	1.00	0.88	0.45	0.56	0.46	-0.37
ram	0.24	0.90	0.37	0.15	0.33	0.75	0.48	0.63	0.88	1.00	0.65	0.65	0.54	-0.52
RearCam	0.17	0.74	0.29	-0.03	0.25	0.77	0.61	0.63	0.45	0.65	1.00	0.60	0.29	-0.55
Front_Cam	0.07	0.68	0.44	-0.01	0.20	0.49	0.59	0.36	0.56	0.65	0.60	1.00	0.30	-0.49
battery	0.03	0.56	0.12	0.83	0.84	0.30	0.46	0.47	0.46	0.54	0.29	0.30	1.00	-0.41
thickness	0.04	-0.72	-0.05	-0.19	-0.53	-0.50	-0.70	-0.61	-0.37	-0.52	-0.55	-0.49	-0.41	1.00

In [23]:

```
#plotting the correlation
plt.figure(figsize=(12, 8))
sns.heatmap(data.corr(),annot=True,cmap='coolwarm')
plt.show()
```



Observation:

Strong Positive correlatio

Their seems to be a strong positive correlation of price of cellphone with the RAM, ppi, cpu freq, internal memory, RearCam and Front_Cam.

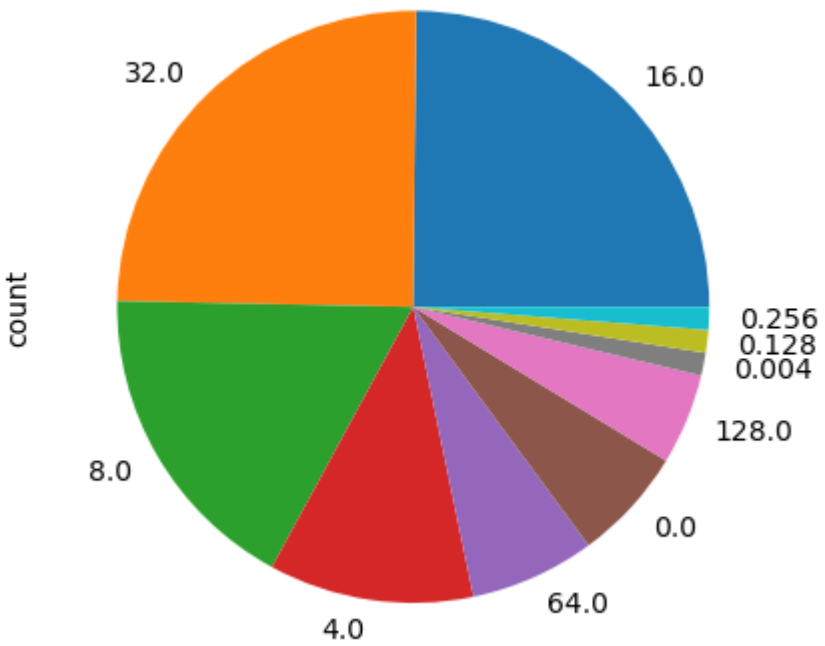
Moderate Correlati on: Battery and cpu core both have a positive relation impact on price.

Negative Correla tion: The thickness has a negative correlation with price showing that lesser thickness is preferred with increase in

Very Low Correlations: Price has very low correlation with sale, weight and resolution.price.

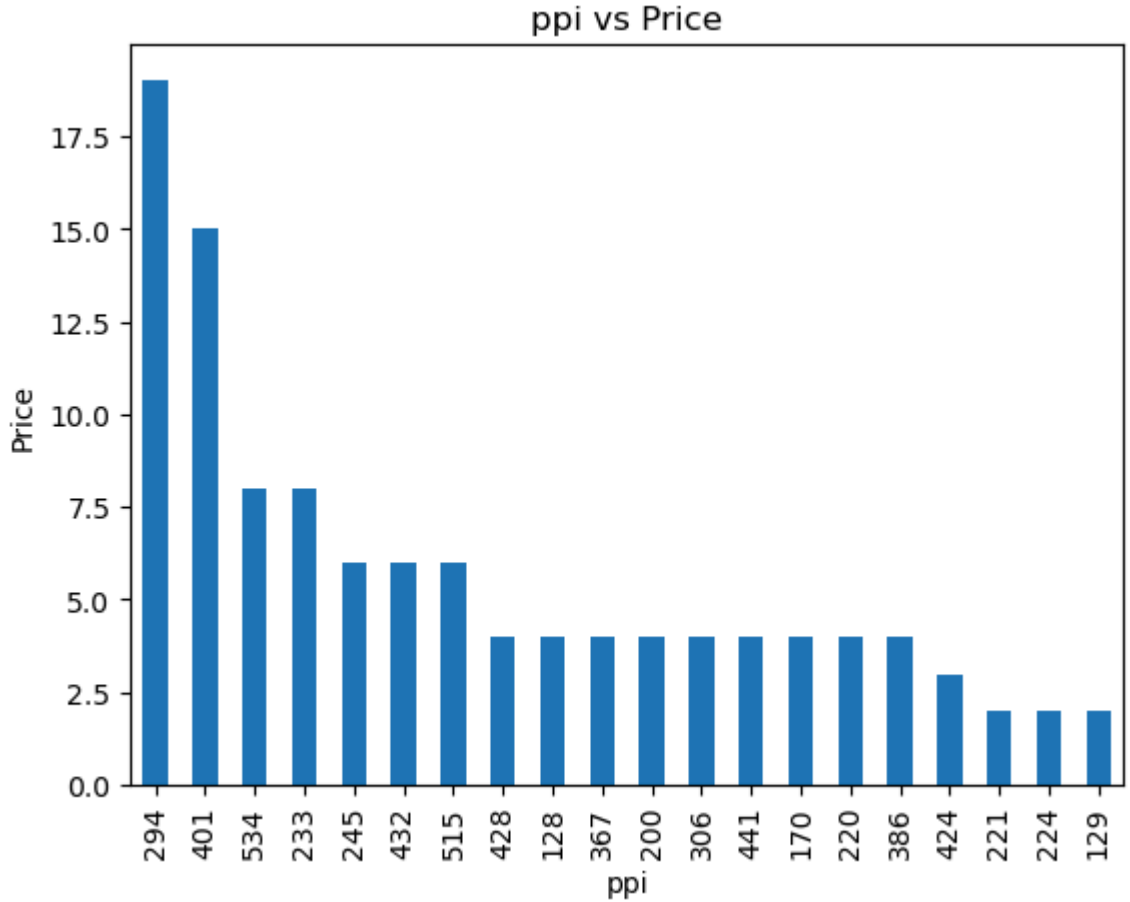
In [25]:
```
data['internal mem'].value_counts().plot(kind='pie')
#plt.show()
```

Out[25]:
```
<Axes: ylabel='count'>
```

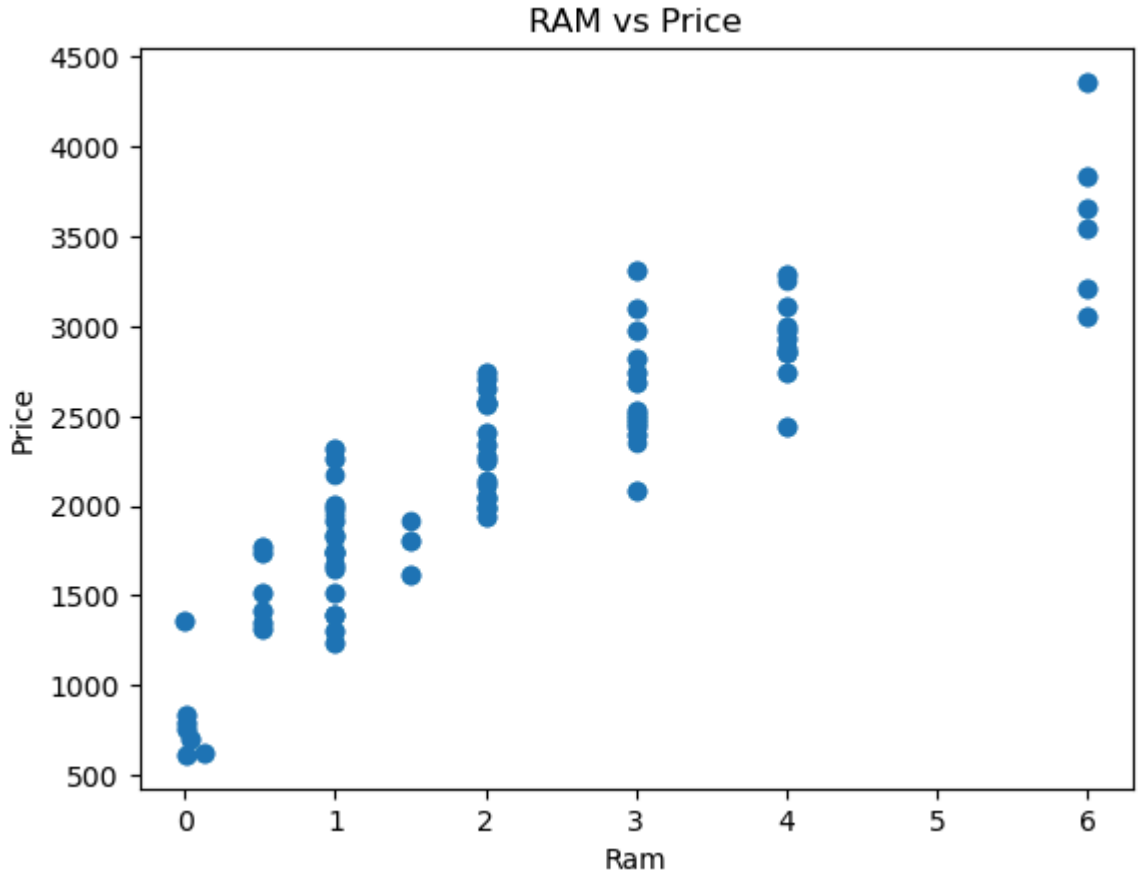


In [27]:
```
data['ppi'].value_counts(ascending=False).head(20).plot(kind='bar')
plt.title("ppi vs Price")
plt.xlabel("ppi")
plt.ylabel("Price")
```

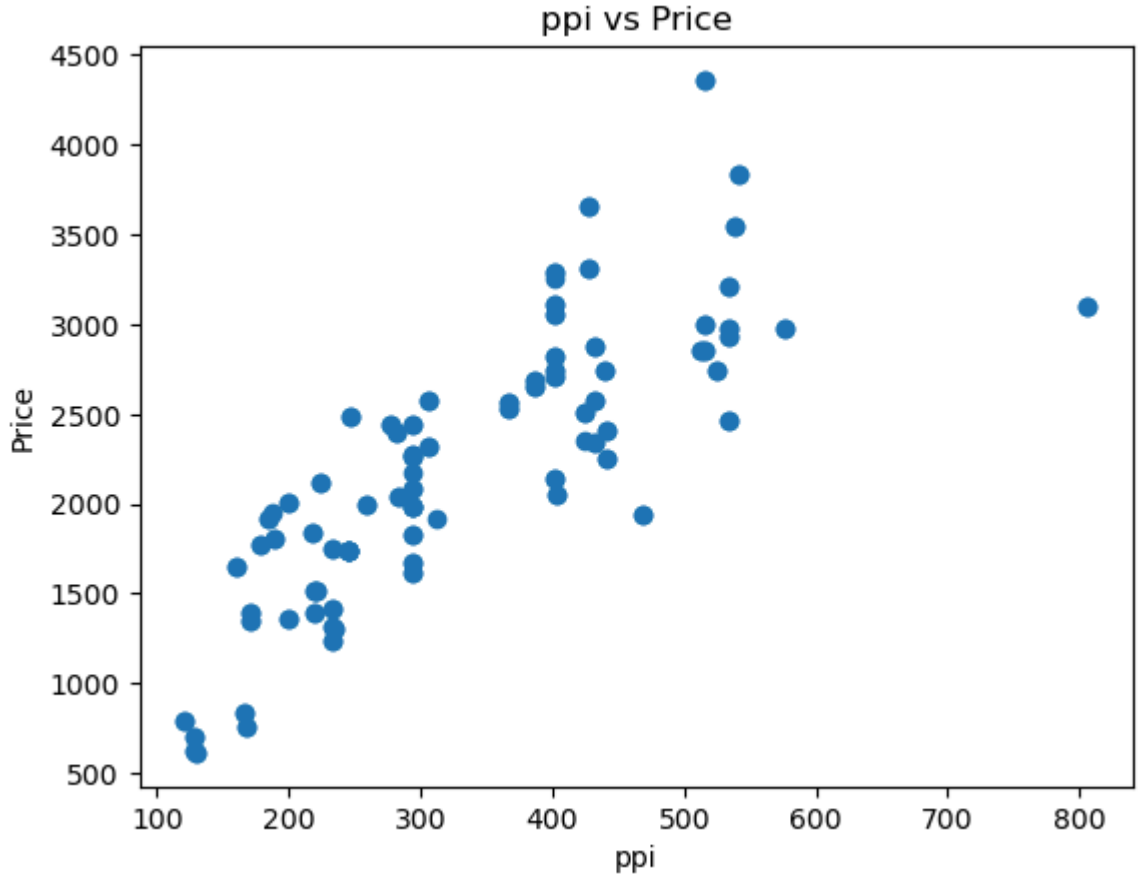
Out[27]:
```
Text(0, 0.5, 'Price')
```



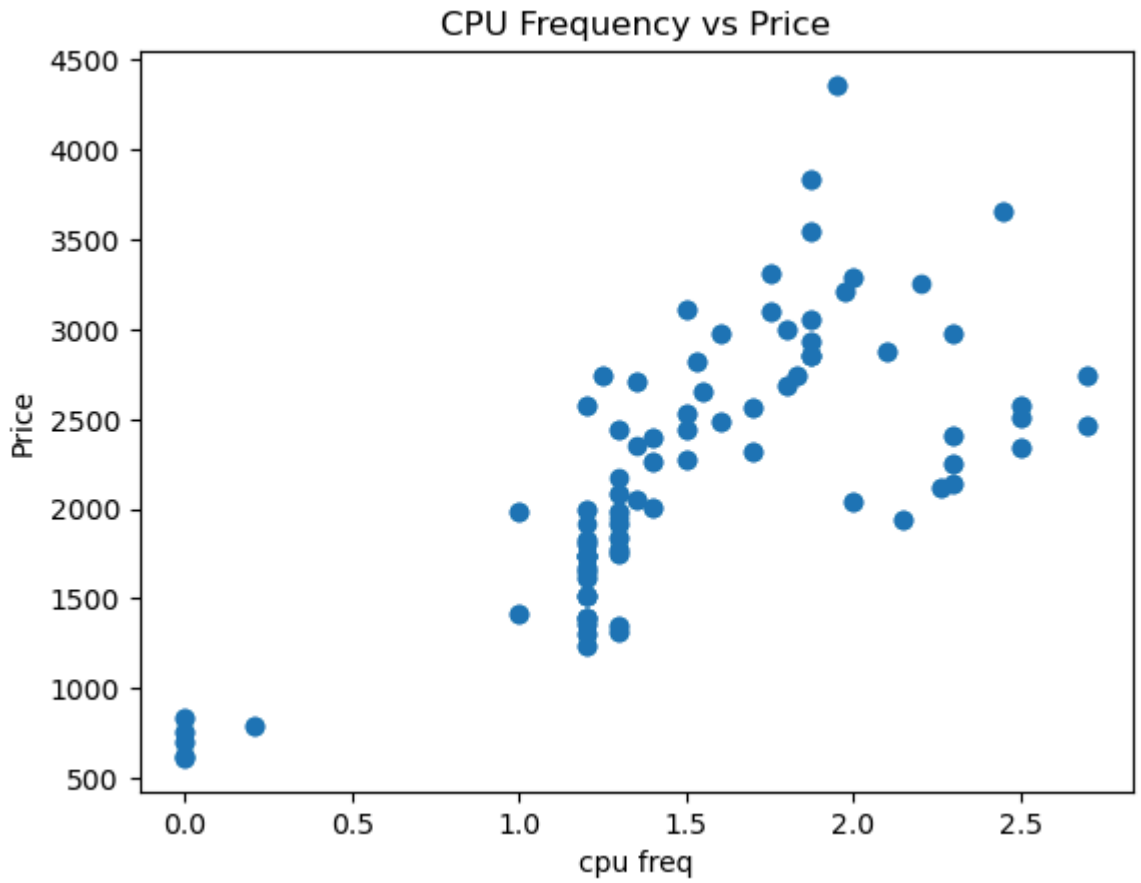
In [29]:
```
plt.scatter(data['ram'],data['Price'])
plt.title("RAM vs Price")
plt.xlabel("Ram")
plt.ylabel("Price")
plt.show()
```



In [31]:
```
plt.scatter(data['ppi'],data['Price'])
plt.title("ppi vs Price")
plt.xlabel("ppi")
plt.ylabel("Price")
plt.show()
```

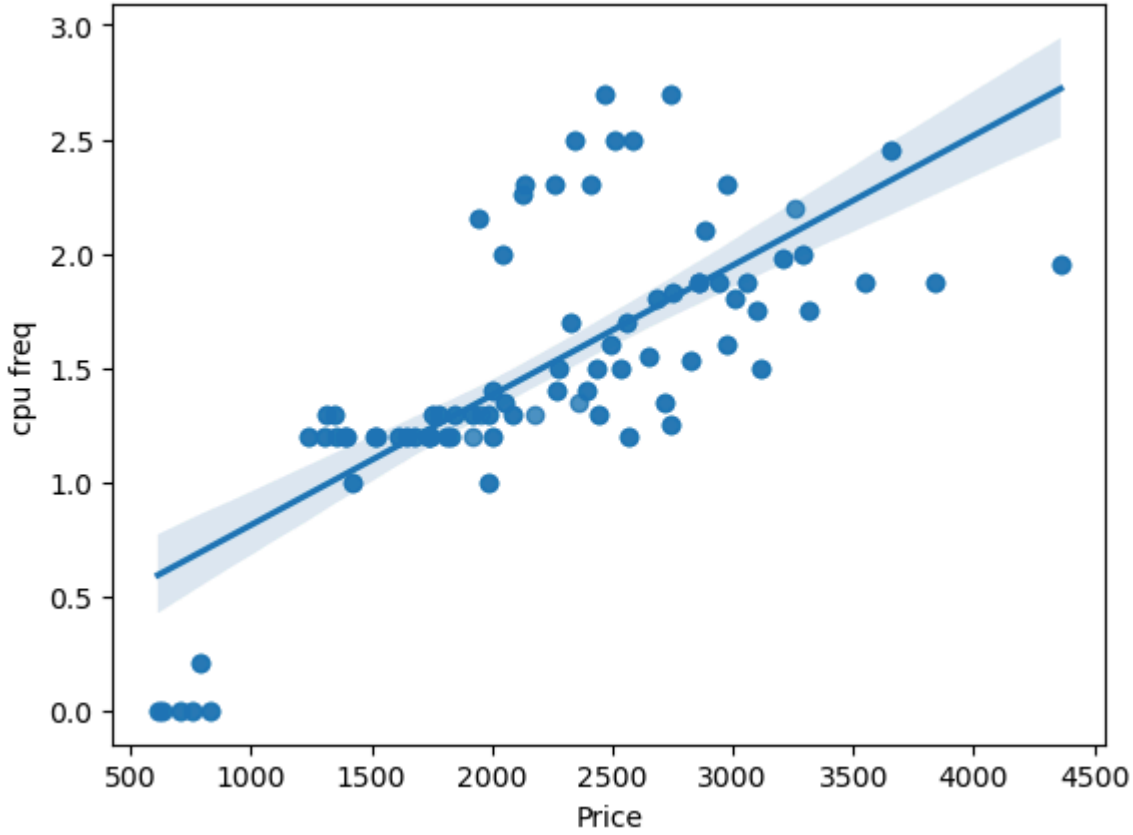


In [33]:
```
plt.scatter(data['cpu freq'],data['Price'])
plt.title("CPU Frequency vs Price")
plt.xlabel("cpu freq")
plt.ylabel("Price")
plt.show()
```

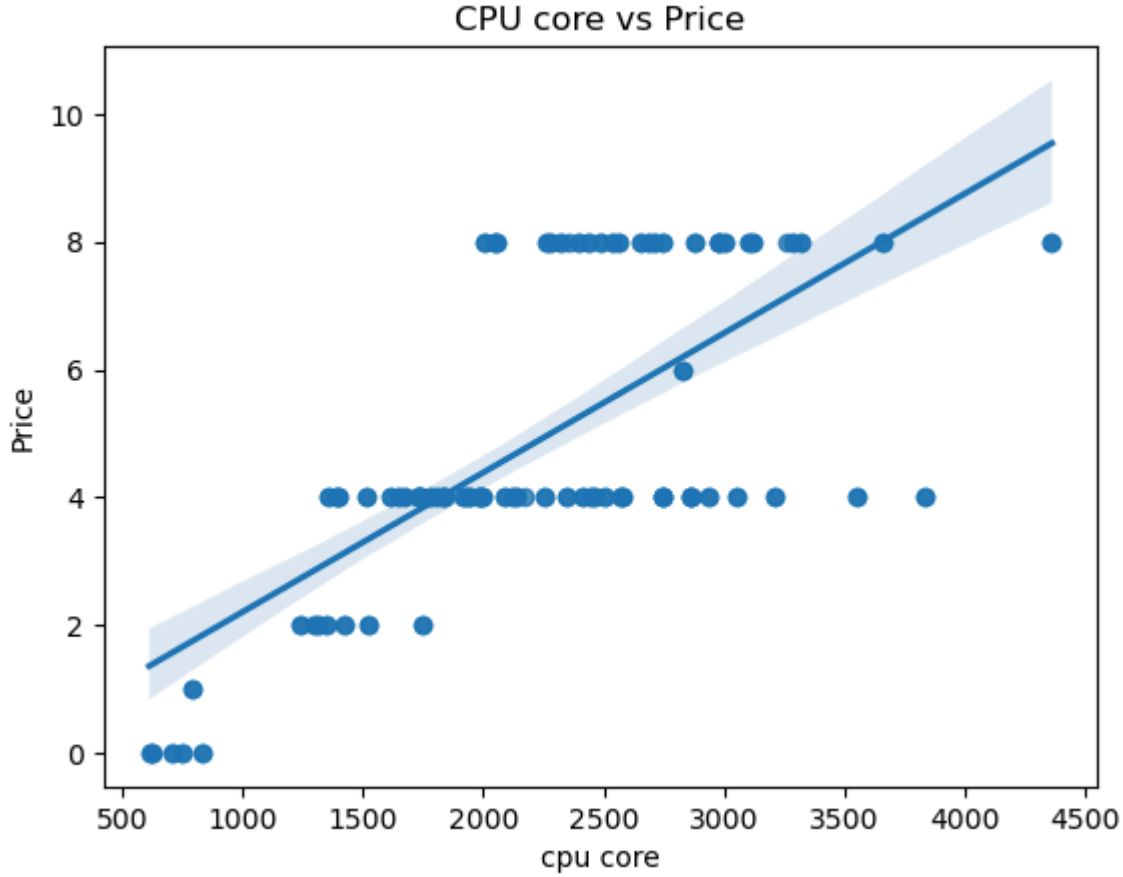


In [35]:
```
sns.regplot(x='Price',y='cpu freq',data=data)
```

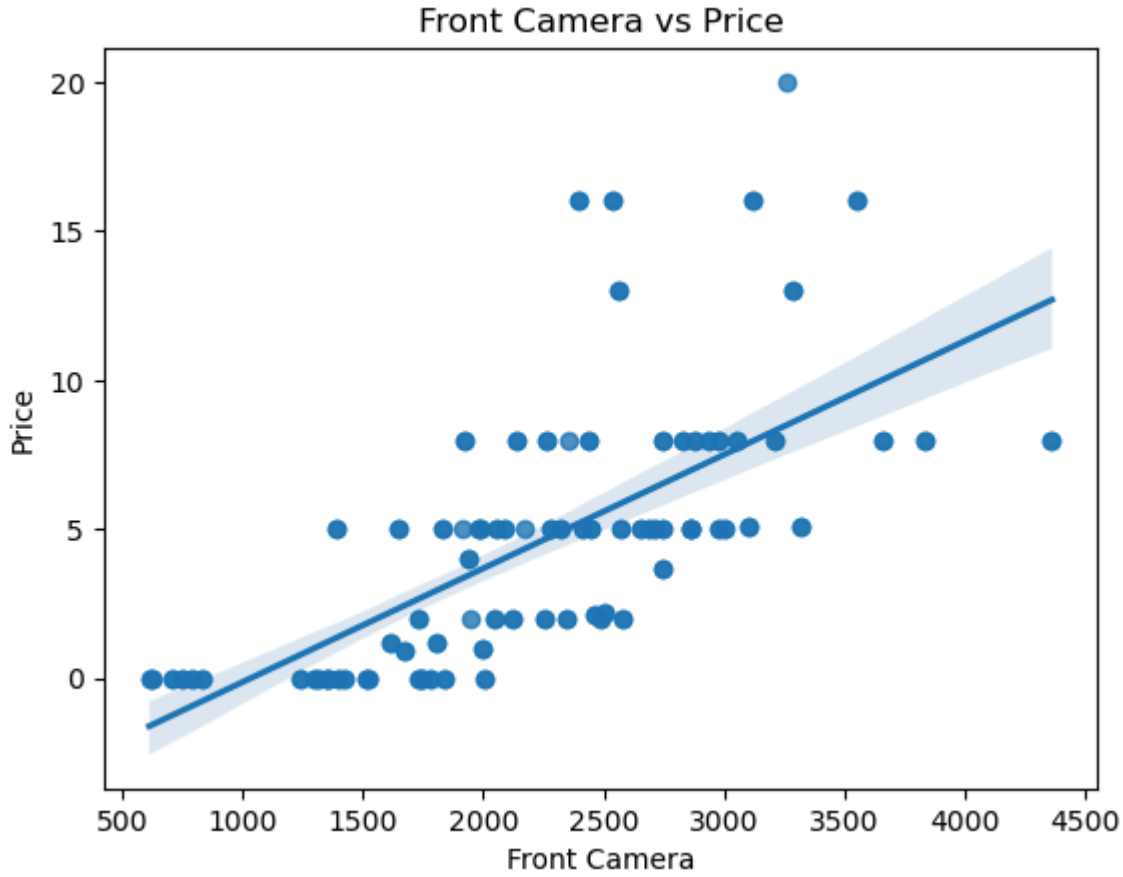
Out[35]:
```
<Axes: xlabel='Price', ylabel='cpu freq'>
```



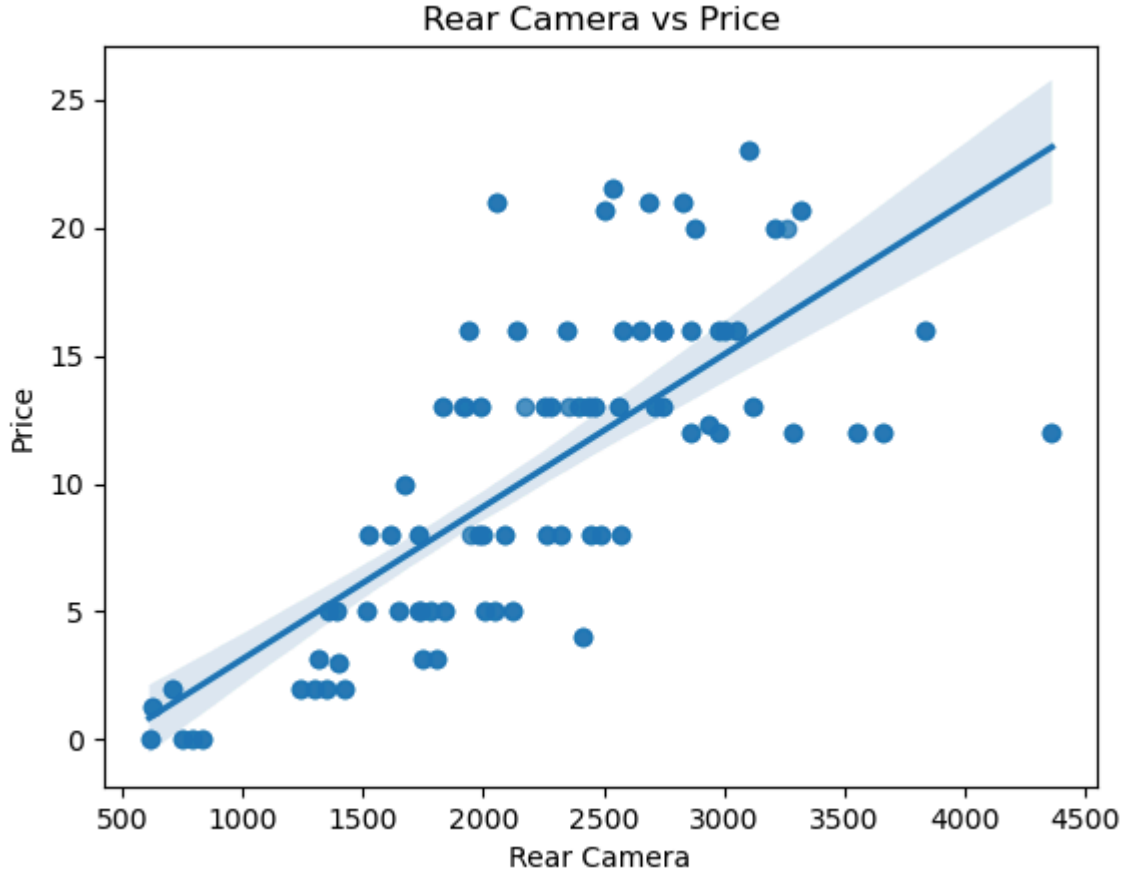
In [37]:
```
sns.regplot(x='Price',y='cpu core',data=data)
plt.title("CPU core vs Price")
plt.xlabel("cpu core")
plt.ylabel("Price")
plt.show()
```



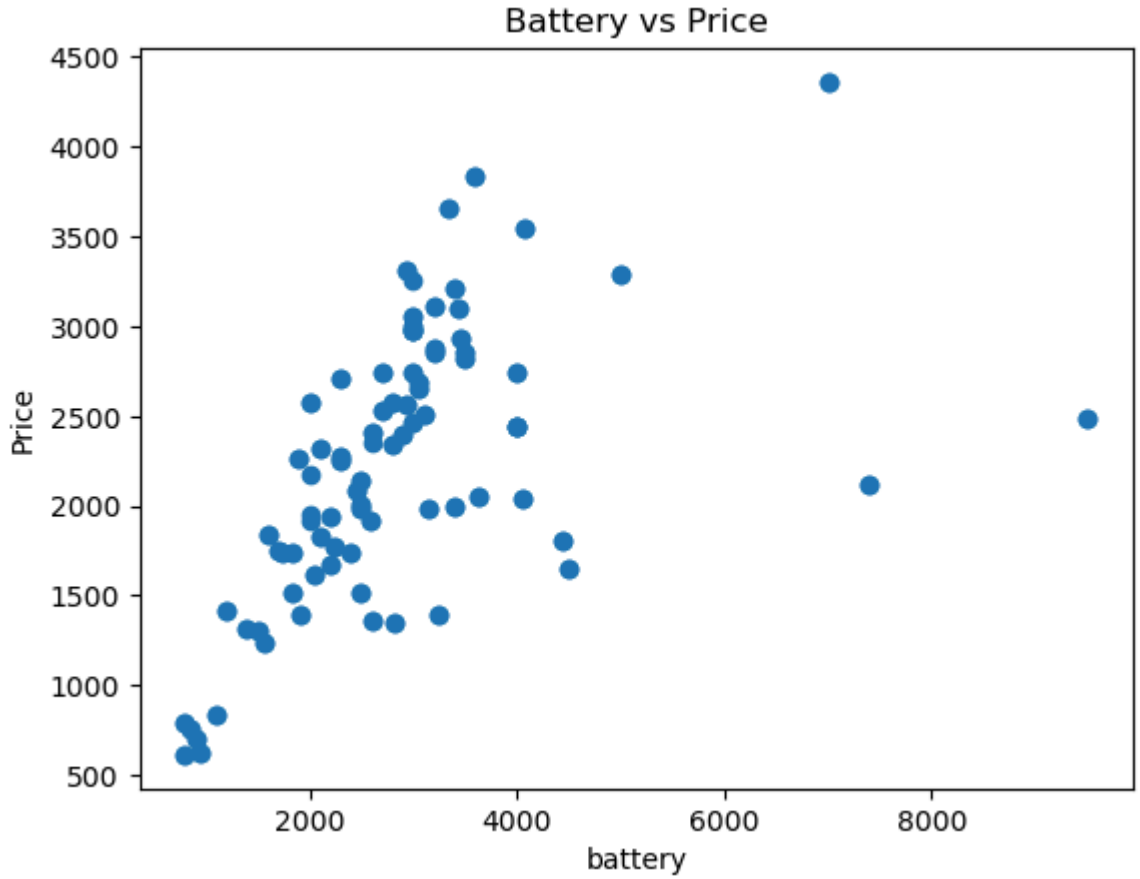
In [37]:
```
sns.regplot(x='Price',y='Front_Cam',data=data)
plt.title("Front Camera vs Price")
plt.xlabel("Front Camera")
plt.ylabel("Price")
plt.show()
```



In [39]:
```
sns.regplot(x='Price',y='RearCam',data=data)
plt.title("Rear Camera vs Price")
plt.xlabel("Rear Camera")
plt.ylabel("Price")
plt.show()
```

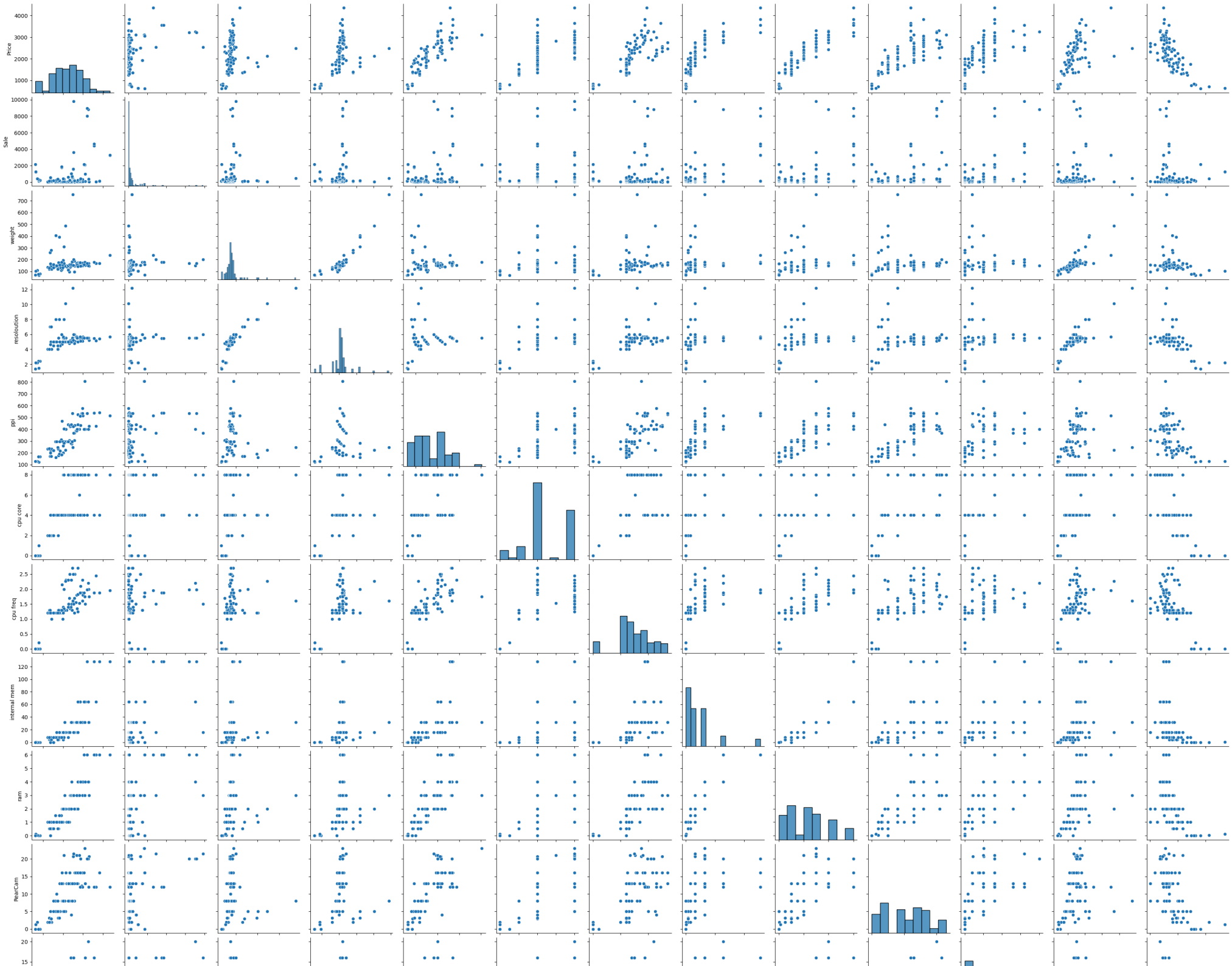


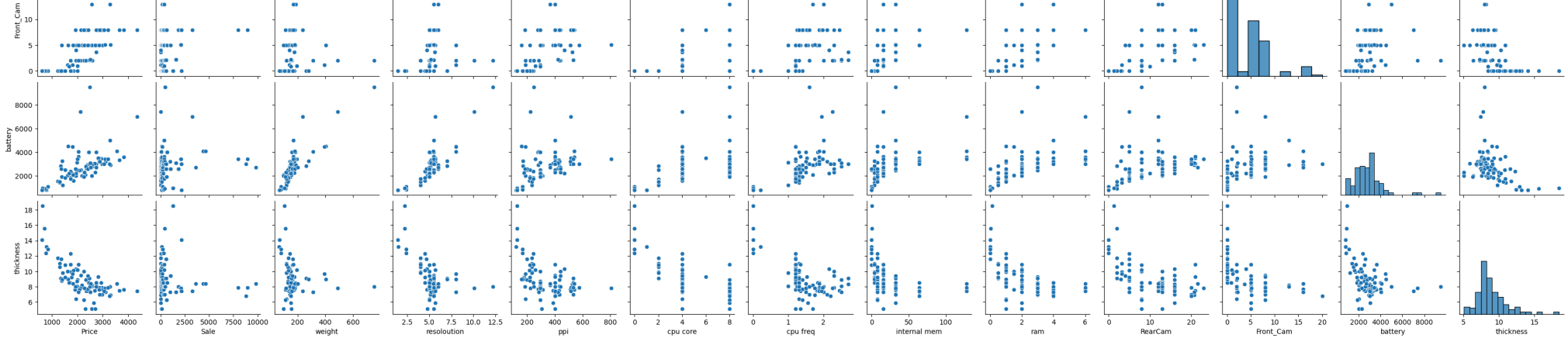
In [43]:
```
plt.scatter(data['battery'],data['Price'])
plt.title("Battery vs Price")
plt.xlabel("battery")
plt.ylabel("Price")
plt.show()
```



In [45]:
```
plt.figure(figsize=(10, 8))
#sns.pairplot(data.select_dtypes(['number']))
sns.pairplot(data= data.drop(["Product_id"], axis =1))
plt.show()
```

```
<Figure size 1000x800 with 0 Axes>
```


Price
Sale
weight
resolution
ppi
cpu core
cpu freq
internal mem
ram
RearCam



In [125…

```
#Removing 'Product_id' variable
data.drop('Product_id', inplace = True, axis = 1)
```

In [ ]:

```
##iii.  Check the distributions/skewness in the variables and do the transformations if required.
```

In [127…

```
data.skew()
```

Out[127…

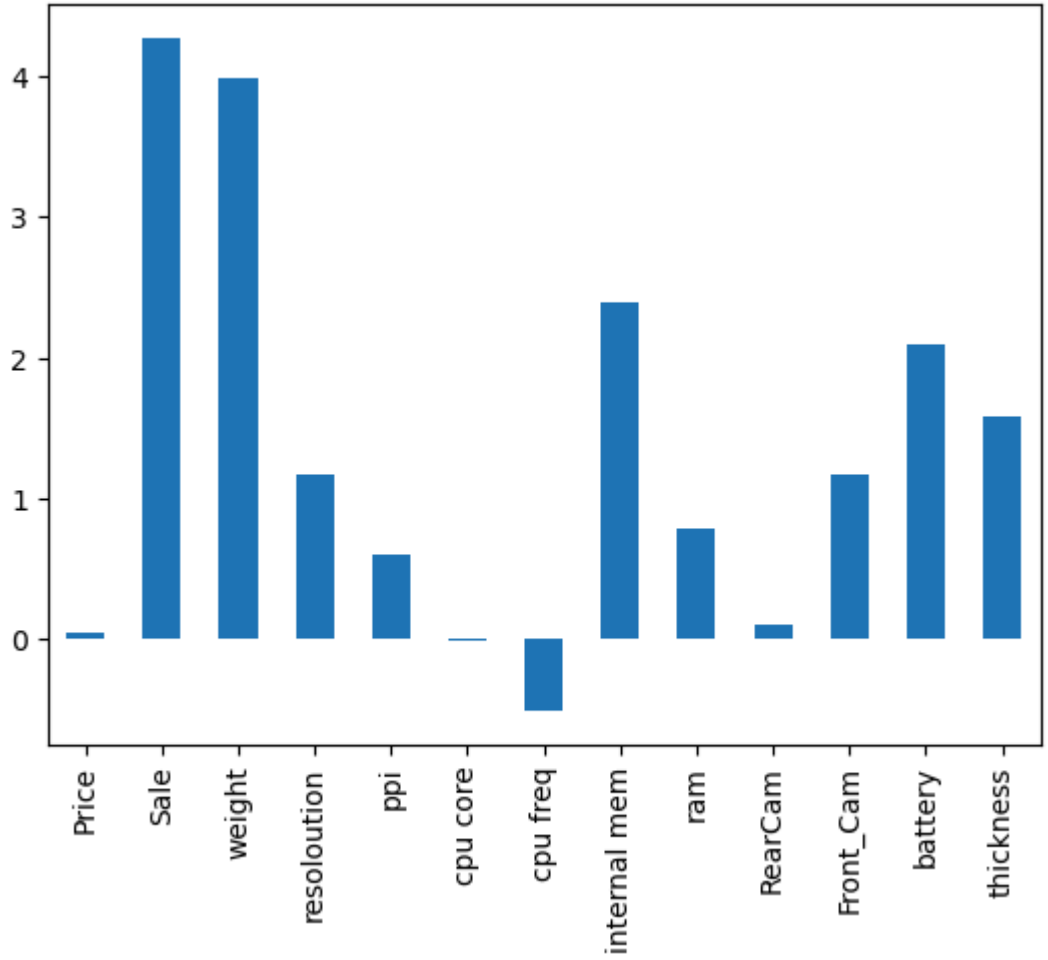
```
Price           0.05
Sale            4.27
weight          3.98
resoloution     1.18
ppi             0.60
cpu core       -0.01
cpu freq       -0.51
internal mem    2.39
ram             0.79
RearCam         0.11
Front_Cam       1.17
battery         2.09
thickness       1.59
dtype: float64
```

In [43]:

```
data.skew().plot(kind='bar')
```

Out[43]:

```
<Axes: >
```



In [45]:
```
skewness = data.skew()
skewness
def Level_of_skewness(s):
    if s > 1 or s < -1:
        return 'Highly Skewed'
    elif s > 0.5 or s<-0.5:
        return 'Moderately Skewed'
    else:
        return 'Lightly Skewed'

skewness_category = skewness.apply(Level_of_skewness)
print(skewness_category)
#print(f"Skewness of the data: {skewness}")
```

```
Price              Lightly Skewed
Sale                Highly Skewed
weight              Highly Skewed
resoloution         Highly Skewed
ppi             Moderately Skewed
cpu core           Lightly Skewed
cpu freq        Moderately Skewed
internal mem        Highly Skewed
ram             Moderately Skewed
RearCam            Lightly Skewed
Front_Cam           Highly Skewed
battery             Highly Skewed
thickness           Highly Skewed
dtype: object
```

Observation:

- The variables like Sale, Weight, resolution,internal mem, Front_Cam,battery and thickness are highly skewed data
- The variables like ppi,cpu freq and ram are moderately skewed with a value of 0.6,-0.5 and 0.79 respectevely
- The variables cpu core, RearCam Product_id and Price are lightly skewed where the Price is a target variable.

In [ ]:

In [129…

```
#Transformation for skewness
from scipy.stats import boxcox

# log transformation
data['thickness'] = np.log(data['thickness'] + 1)
#data['battery'] = np.log(data['battery'] + 1)
#data['Front_Cam'] = np.log(data['Front_Cam'] + 1)
#data['internal mem'] = np.log(data['internal mem']+1)
data['weight'] = np.log(data['weight']+1)
data['Sale'] = np.log(data['Sale']+1)
data['cpu freq'] = np.log(data['cpu freq']+1)
 # Square root transformation
#data['resoloution'] = np.sqrt(data['resoloution'])
#data['cpu freq'] = np.sqrt(data['cpu freq'])
data['battery'] = np.sqrt(data['battery'])
#data['internal mem'] = np.sqrt(data['internal mem'])
data['weight'] = np.sqrt(data['weight'])
data['Front_Cam'] = np.sqrt(data['Front_Cam'])
# Box-Cox Transformation

data['resoloution'], _ = boxcox(data['resoloution'] - data['resoloution'].min() + 1)
```

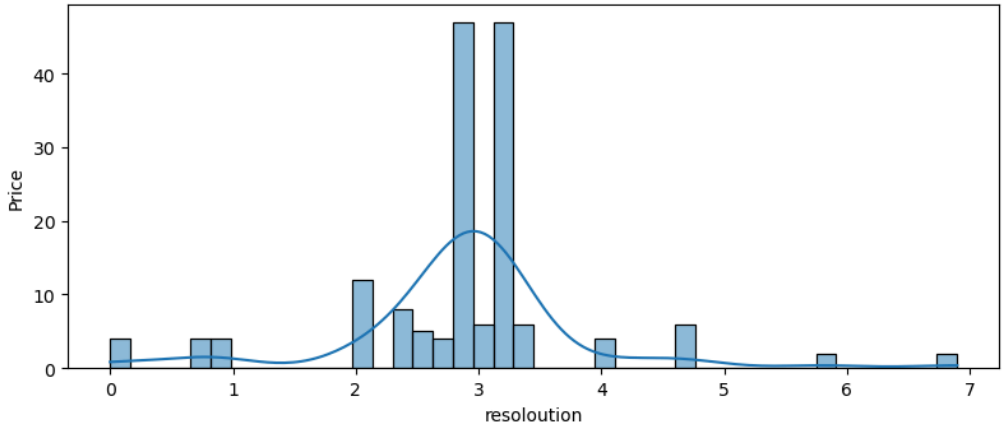
```
skewness_after_transformation = data.skew()

print("Skewness after transformations:")
print(skewness_after_transformation)
```

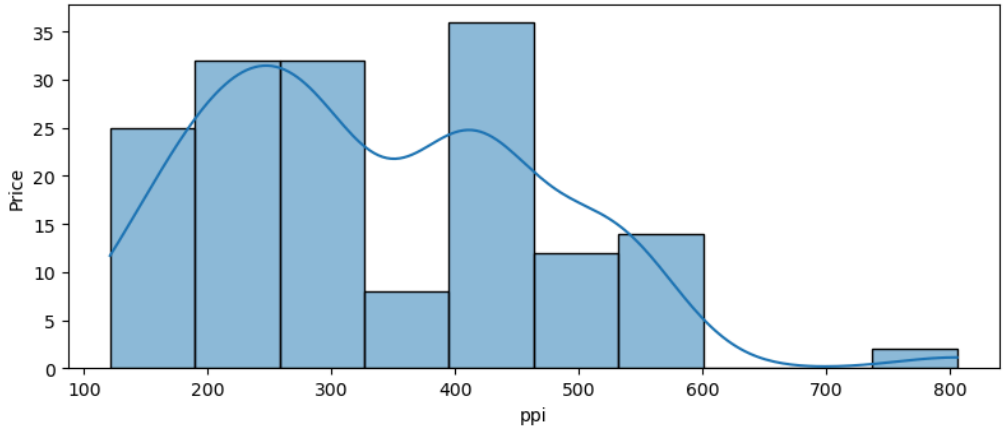
```
Skewness after transformations:
Price           0.05
Sale            0.61
weight          1.07
resoloution     0.32
ppi             0.60
cpu core       -0.01
cpu freq       -1.65
internal mem    2.39
ram             0.79
RearCam         0.11
Front_Cam      -0.13
battery         0.79
thickness       0.59
dtype: float64
```

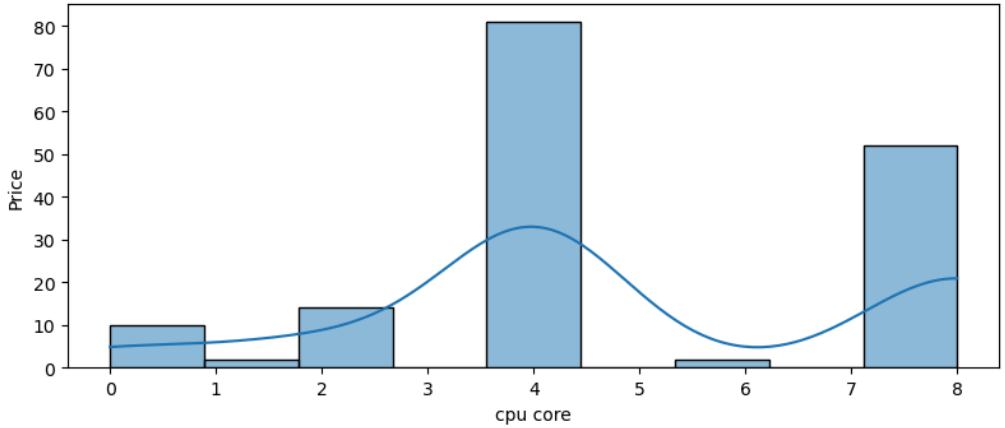
In [ ]:

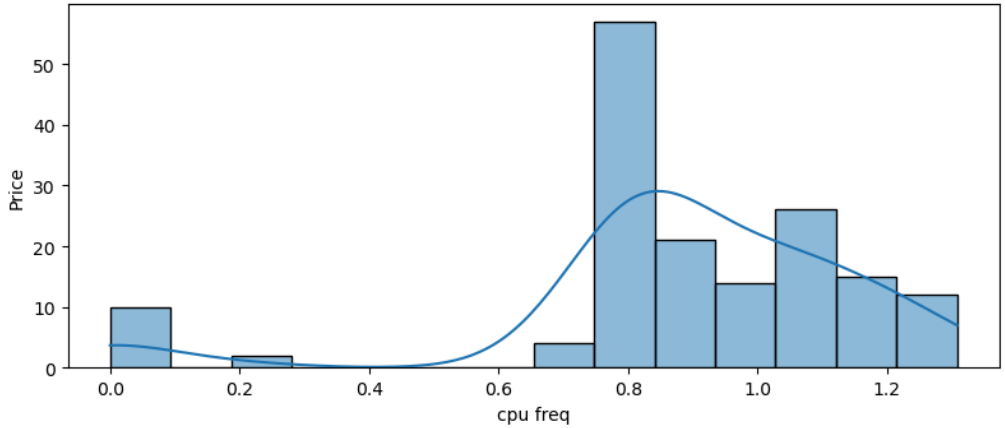
In [131…

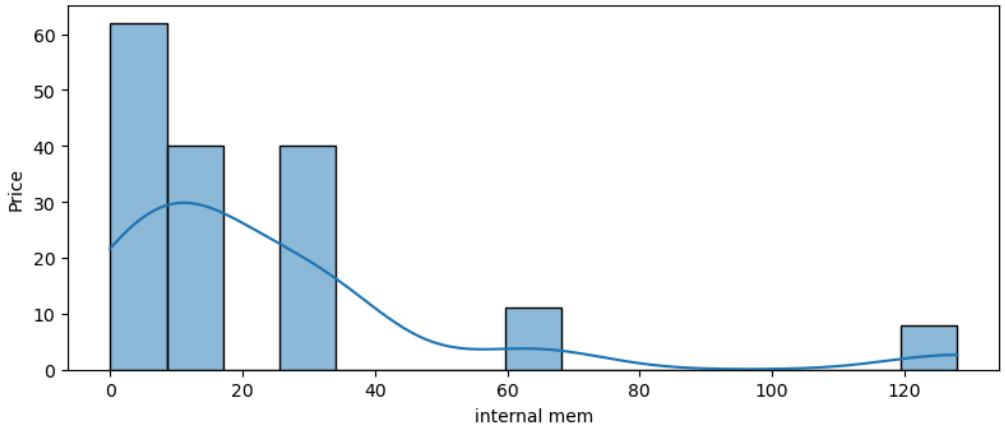
```
cols = ['resoloution','ppi',    'cpu core',     'cpu freq',     'internal mem'  ,'ram'  ,'RearCam',     'Front_Cam']
fig,axes=plt.subplots(figsize=(16,14),nrows=4,ncols=2)

axes = axes.flatten()

for i in range(8):
    sns.histplot(data,x=data[cols[i]],ax=axes[i],kde=True)
    axes[i].set_ylabel('Price')
    axes[i].set_xlabel(f"{cols[i]}")
plt.tight_layout()
plt.show()
```

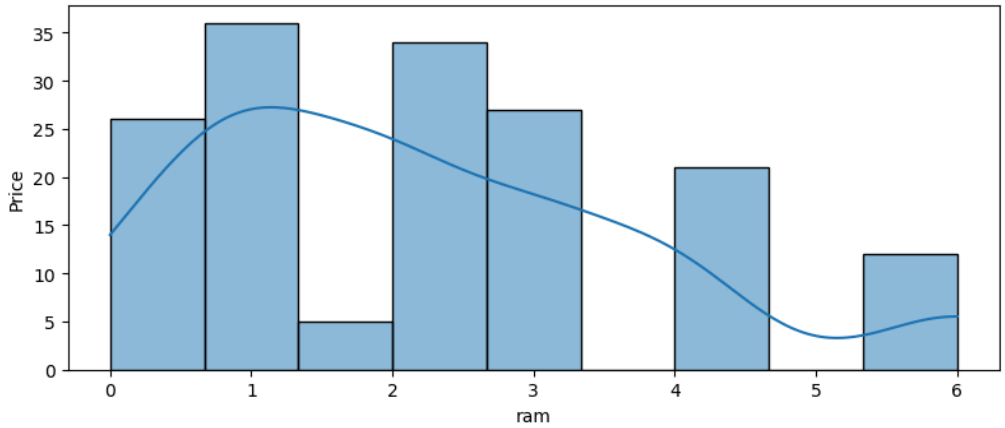

Price
resoloution

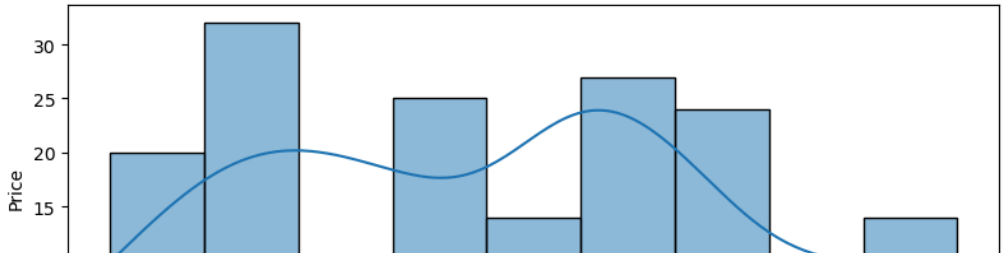

Price
ppi

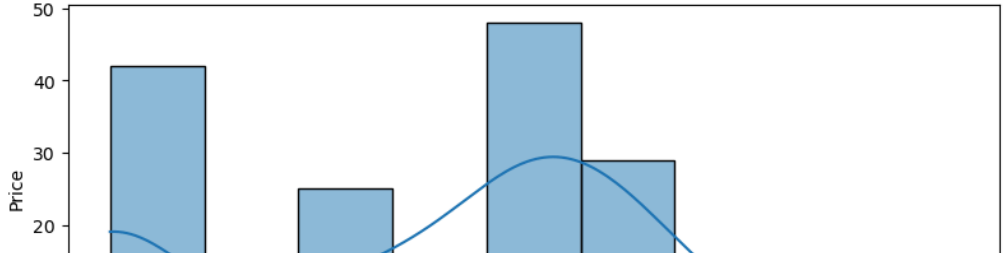

Price
cpu core

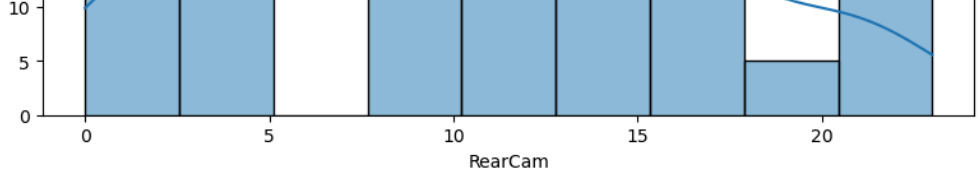

Price
cpu freq

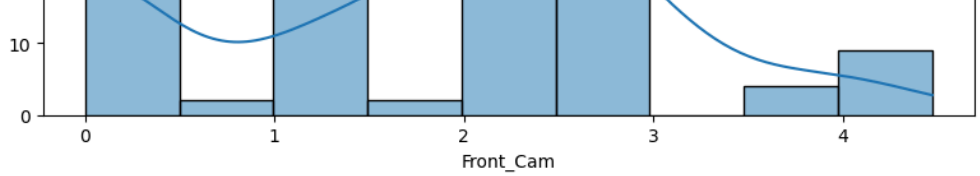

Price
internal mem


Price
ram


Price


Price





iv. Check/Treat the outliers and do the feature scaling if required.

In [133…

```
def check_outliers(data):
    outliers = {}
    for column in data.select_dtypes(include=['number']).columns:
        Q1 = data[column].quantile(0.25)
        Q3 = data[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        outliers[column] = data[(data[column] < lower_bound) | (data[column] > upper_bound)]
        #data['weight'] = np.where(data['weight'] < lower_bound, lower_bound, data['weight'])
        #data['internal mem'] = np.where(data['weight'] > upper_bound, upper_bound, data['weight'])
        #data['weight'] = np.where(data['internal mem'] < lower_bound, lower_bound, data['internal mem'])
       # data['internal mem'] = np.where(data['internal mem'] > upper_bound, upper_bound, data['internal mem'])
       # data = data[data['weight'] < upper_limit ]
    return outliers
outliers = check_outliers(data)
print(outliers)
```

```
{'Price':      Price  Sale  weight  resoloution  ppi  cpu core  cpu freq  internal mem  \
152   4361  8.09    2.34         3.25  515         8      1.08        128.00
153   4361  8.10    2.34         3.25  515         8      1.08        128.00

     ram  RearCam  Front_Cam  battery  thickness
152 6.00    12.00       2.83    83.67       2.13
153 6.00    12.00       2.83    83.67       2.13  , 'Sale': Empty DataFrame
Columns: [Price, Sale, weight, resoloution, ppi, cpu core, cpu freq, internal mem, ram, RearCam, Front_Cam, battery, thickness]
Index: [], 'weight':      Price  Sale  weight  resoloution  ppi  cpu core  cpu freq  internal mem  \
37    2124  3.58    2.49         5.80  224         4      1.18         16.00
40    2124  3.64    2.49         5.80  224         4      1.18         16.00
51    1347  3.78    2.36         4.05  170         2      0.83          4.00
62    1347  4.22    2.36         4.05  170         2      0.83          4.00
67    2044  4.42    2.40         4.63  283         8      1.10          8.00
72    2044  4.54    2.40         4.63  283         8      1.10          8.00
77    1396  4.62    2.37         4.05  170         4      0.79          4.00
81     791  4.67    2.05         0.10  121         1      0.19          0.00
84    1646  4.74    2.45         4.63  160         4      0.79          8.00
85    1396  4.74    2.37         4.05  170         4      0.79          4.00
86     791  4.74    2.05         0.10  121         1      0.19          0.00
90    1646  4.96    2.45         4.63  160         4      0.79          8.00
93    1810  5.11    2.44         4.63  189         4      0.79         16.00
95    1810  5.12    2.44         4.63  189         4      0.79         16.00
101    833  5.31    2.09         0.90  166         0      0.00          0.00
105    833  5.37    2.09         0.90  166         0      0.00          0.00
110    754  5.70    2.09         0.90  167         0      0.00          0.00
113    754  5.73    2.09         0.90  167         0      0.00          0.00
127   2491  6.12    2.57         6.90  247         8      0.96         32.00
128   2491  6.12    2.57         6.90  247         8      0.96         32.00
149    614  7.68    2.06         0.00  129         0      0.00          0.00
150    614  7.68    2.06         0.00  129         0      0.00          0.00

     ram  RearCam  Front_Cam  battery  thickness
37  2.00     5.00       1.41    86.02       2.17
40  2.00     5.00       1.41    86.02       2.17
51  0.51     2.00       0.00    53.10       2.31
62  0.51     2.00       0.00    53.10       2.31
67  2.00     5.00       1.41    63.72       2.12
72  2.00     5.00       1.41    63.72       2.12
77  1.00     3.00       0.00    56.92       2.30
81  0.00     0.00       0.00    28.28       2.65
84  1.00     5.00       2.24    67.08       2.30
85  1.00     3.00       0.00    56.92       2.30
86  0.00     0.00       0.00    28.28       2.65
90  1.00     5.00       2.24    67.08       2.30
93  1.50     3.15       1.10    66.71       2.37
```

```
95  1.50     3.15       1.10    66.71       2.37
101 0.01     0.00       0.00    33.17       2.63
105 0.01     0.00       0.00    33.17       2.63
110 0.00     0.00       0.00    29.15       2.60
113 0.00     0.00       0.00    29.15       2.60
127 3.00     8.00       1.41    97.47       2.20
128 3.00     8.00       1.41    97.47       2.20
149 0.00     0.00       0.00    28.28       2.71
150 0.00     0.00       0.00    28.28       2.71  , 'resoloution':      Price  Sale  weight  resoloution  ppi  cpu core  cpu freq  internal
mem  \
37    2124  3.58    2.49         5.80  224         4      1.18         16.00
40    2124  3.64    2.49         5.80  224         4      1.18         16.00
51    1347  3.78    2.36         4.05  170         2      0.83          4.00
62    1347  4.22    2.36         4.05  170         2      0.83          4.00
67    2044  4.42    2.40         4.63  283         8      1.10          8.00
72    2044  4.54    2.40         4.63  283         8      1.10          8.00
77    1396  4.62    2.37         4.05  170         4      0.79          4.00
81     791  4.67    2.05         0.10  121         1      0.19          0.00
84    1646  4.74    2.45         4.63  160         4      0.79          8.00
85    1396  4.74    2.37         4.05  170         4      0.79          4.00
86     791  4.74    2.05         0.10  121         1      0.19          0.00
90    1646  4.96    2.45         4.63  160         4      0.79          8.00
93    1810  5.11    2.44         4.63  189         4      0.79         16.00
95    1810  5.12    2.44         4.63  189         4      0.79         16.00
101    833  5.31    2.09         0.90  166         0      0.00          0.00
105    833  5.37    2.09         0.90  166         0      0.00          0.00
110    754  5.70    2.09         0.90  167         0      0.00          0.00
113    754  5.73    2.09         0.90  167         0      0.00          0.00
125    705  6.05    2.17         0.73  128         0      0.00          0.13
126    705  6.06    2.17         0.73  128         0      0.00          0.13
127   2491  6.12    2.57         6.90  247         8      0.96         32.00
128   2491  6.12    2.57         6.90  247         8      0.96         32.00
140    628  7.11    2.15         0.73  128         0      0.00          0.26
141    628  7.15    2.15         0.73  128         0      0.00          0.26
149    614  7.68    2.06         0.00  129         0      0.00          0.00
150    614  7.68    2.06         0.00  129         0      0.00          0.00

     ram  RearCam  Front_Cam  battery  thickness
37  2.00     5.00       1.41    86.02       2.17
40  2.00     5.00       1.41    86.02       2.17
51  0.51     2.00       0.00    53.10       2.31
62  0.51     2.00       0.00    53.10       2.31
67  2.00     5.00       1.41    63.72       2.12
72  2.00     5.00       1.41    63.72       2.12
77  1.00     3.00       0.00    56.92       2.30
81  0.00     0.00       0.00    28.28       2.65
```

```
84  1.00     5.00       2.24    67.08       2.30
85  1.00     3.00       0.00    56.92       2.30
86  0.00     0.00       0.00    28.28       2.65
90  1.00     5.00       2.24    67.08       2.30
93  1.50     3.15       1.10    66.71       2.37
95  1.50     3.15       1.10    66.71       2.37
101 0.01     0.00       0.00    33.17       2.63
105 0.01     0.00       0.00    33.17       2.63
110 0.00     0.00       0.00    29.15       2.60
113 0.00     0.00       0.00    29.15       2.60
125 0.03     2.00       0.00    30.00       2.81
126 0.03     2.00       0.00    30.00       2.81
127 3.00     8.00       1.41    97.47       2.20
128 3.00     8.00       1.41    97.47       2.20
140 0.13     1.30       0.00    30.82       2.97
141 0.13     1.30       0.00    30.82       2.97
149 0.00     0.00       0.00    28.28       2.71
150 0.00     0.00       0.00    28.28       2.71  , 'ppi':      Price  Sale  weight  resoloution  ppi  cpu core  cpu freq  internal mem  \
146   3102  7.64    2.28         3.12  806         8      1.01         32.00
147   3102  7.64    2.28         3.12  806         8      1.01         32.00

     ram  RearCam  Front_Cam  battery  thickness
146 3.00    23.00       2.26    58.57       2.17
147 3.00    23.00       2.26    58.57       2.17  , 'cpu core': Empty DataFrame
Columns: [Price, Sale, weight, resoloution, ppi, cpu core, cpu freq, internal mem, ram, RearCam, Front_Cam, battery, thickness]
Index: [], 'cpu freq':      Price  Sale  weight  resoloution  ppi  cpu core  cpu freq  internal mem  \
81     791  4.67    2.05         0.10  121         1      0.19          0.00
86     791  4.74    2.05         0.10  121         1      0.19          0.00
101    833  5.31    2.09         0.90  166         0      0.00          0.00
105    833  5.37    2.09         0.90  166         0      0.00          0.00
110    754  5.70    2.09         0.90  167         0      0.00          0.00
113    754  5.73    2.09         0.90  167         0      0.00          0.00
125    705  6.05    2.17         0.73  128         0      0.00          0.13
126    705  6.06    2.17         0.73  128         0      0.00          0.13
140    628  7.11    2.15         0.73  128         0      0.00          0.26
141    628  7.15    2.15         0.73  128         0      0.00          0.26
149    614  7.68    2.06         0.00  129         0      0.00          0.00
150    614  7.68    2.06         0.00  129         0      0.00          0.00

     ram  RearCam  Front_Cam  battery  thickness
81  0.00     0.00       0.00    28.28       2.65
86  0.00     0.00       0.00    28.28       2.65
101 0.01     0.00       0.00    33.17       2.63
105 0.01     0.00       0.00    33.17       2.63
110 0.00     0.00       0.00    29.15       2.60
113 0.00     0.00       0.00    29.15       2.60
```

```
125 0.03     2.00       0.00    30.00       2.81
126 0.03     2.00       0.00    30.00       2.81
140 0.13     1.30       0.00    30.82       2.97
141 0.13     1.30       0.00    30.82       2.97
149 0.00     0.00       0.00    28.28       2.71
150 0.00     0.00       0.00    28.28       2.71  , 'internal mem':      Price  Sale  weight  resoloution  ppi  cpu core  cpu freq  internal
mem  \
83    3837  4.72    2.26         3.08  541         4      1.06        128.00
87    3837  4.79    2.26         3.08  541         4      1.06        128.00
152   4361  8.09    2.34         3.25  515         8      1.08        128.00
153   4361  8.10    2.34         3.25  515         8      1.08        128.00
155   3551  8.39    2.28         3.09  538         4      1.06        128.00
156   3551  8.44    2.28         3.09  538         4      1.06        128.00
157   3211  8.99    2.27         3.12  534         4      1.09        128.00
159   3211  9.10    2.27         3.12  534         4      1.09        128.00

     ram  RearCam  Front_Cam  battery  thickness
83  6.00    16.00       2.83    60.00       2.15
87  6.00    16.00       2.83    60.00       2.15
152 6.00    12.00       2.83    83.67       2.13
153 6.00    12.00       2.83    83.67       2.13
155 6.00    12.00       4.00    63.87       2.24
156 6.00    12.00       4.00    63.87       2.24
157 6.00    20.00       2.83    58.31       2.19
159 6.00    20.00       2.83    58.31       2.19  , 'ram': Empty DataFrame
Columns: [Price, Sale, weight, resoloution, ppi, cpu core, cpu freq, internal mem, ram, RearCam, Front_Cam, battery, thickness]
Index: [], 'RearCam': Empty DataFrame
Columns: [Price, Sale, weight, resoloution, ppi, cpu core, cpu freq, internal mem, ram, RearCam, Front_Cam, battery, thickness]
Index: [], 'Front_Cam': Empty DataFrame
Columns: [Price, Sale, weight, resoloution, ppi, cpu core, cpu freq, internal mem, ram, RearCam, Front_Cam, battery, thickness]
Index: [], 'battery':      Price  Sale  weight  resoloution  ppi  cpu core  cpu freq  internal mem  \
37    2124  3.58    2.49         5.80  224         4      1.18         16.00
40    2124  3.64    2.49         5.80  224         4      1.18         16.00
127   2491  6.12    2.57         6.90  247         8      0.96         32.00
128   2491  6.12    2.57         6.90  247         8      0.96         32.00
152   4361  8.09    2.34         3.25  515         8      1.08        128.00
153   4361  8.10    2.34         3.25  515         8      1.08        128.00

     ram  RearCam  Front_Cam  battery  thickness
37  2.00     5.00       1.41    86.02       2.17
40  2.00     5.00       1.41    86.02       2.17
127 3.00     8.00       1.41    97.47       2.20
128 3.00     8.00       1.41    97.47       2.20
152 6.00    12.00       2.83    83.67       2.13
153 6.00    12.00       2.83    83.67       2.13  , 'thickness':      Price  Sale  weight  resoloution  ppi  cpu core  cpu freq  internal me
m  \
```

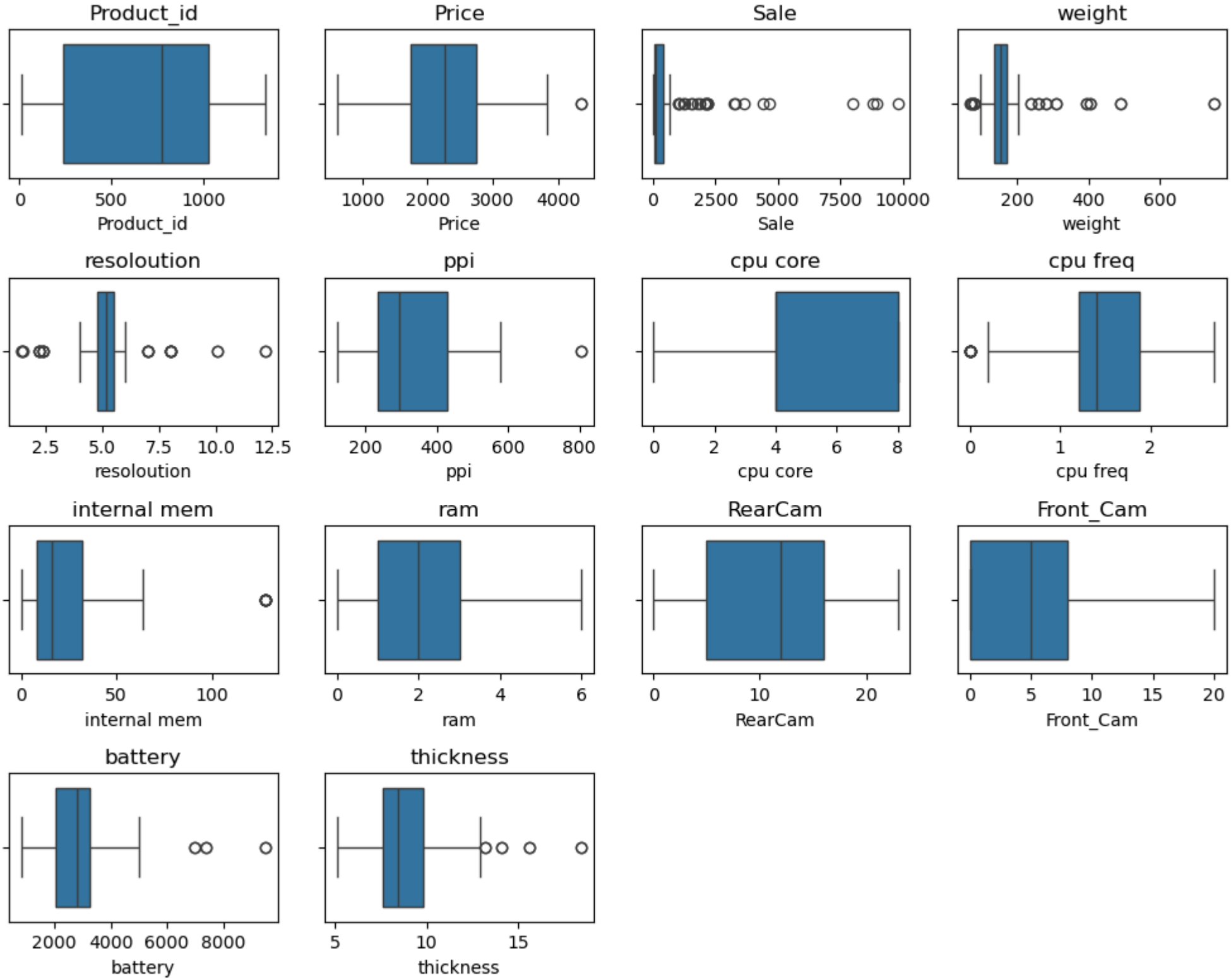
```
63    2323  4.28    2.14         2.66  306         8      0.99         16.00
69    2323  4.48    2.14         2.66  306         8      0.99         16.00
74    2571  4.57    2.14         2.66  306         4      0.79         16.00
76    2571  4.62    2.14         2.66  306         4      0.79         16.00
78    2714  4.62    2.25         3.12  401         8      0.85         16.00
80    2714  4.67    2.25         3.12  401         8      0.85         16.00
125    705  6.05    2.17         0.73  128         0      0.00          0.13
126    705  6.06    2.17         0.73  128         0      0.00          0.13
140    628  7.11    2.15         0.73  128         0      0.00          0.26
141    628  7.15    2.15         0.73  128         0      0.00          0.26

     ram  RearCam  Front_Cam  battery  thickness
63  1.00     8.00       2.24    45.83       1.81
69  1.00     8.00       2.24    45.83       1.81
74  2.00     8.00       2.24    44.72       1.81
76  2.00     8.00       2.24    44.72       1.81
78  2.00    13.00       2.24    47.96       1.81
80  2.00    13.00       2.24    47.96       1.81
125 0.03     2.00       0.00    30.00       2.81
126 0.03     2.00       0.00    30.00       2.81
140 0.13     1.30       0.00    30.82       2.97
141 0.13     1.30       0.00    30.82       2.97  }
```

In [ ]:

```
#Before Removing skewness
```

In [129…

```
#sns.boxplot(data)
plt.figure(figsize=(10, 8))

for col in data.columns:
    plt.subplot(4, 4, data.columns.get_loc(col) + 1)
    sns.boxplot(x=data[col])
    plt.title(col)

plt.tight_layout()
plt.show()
```

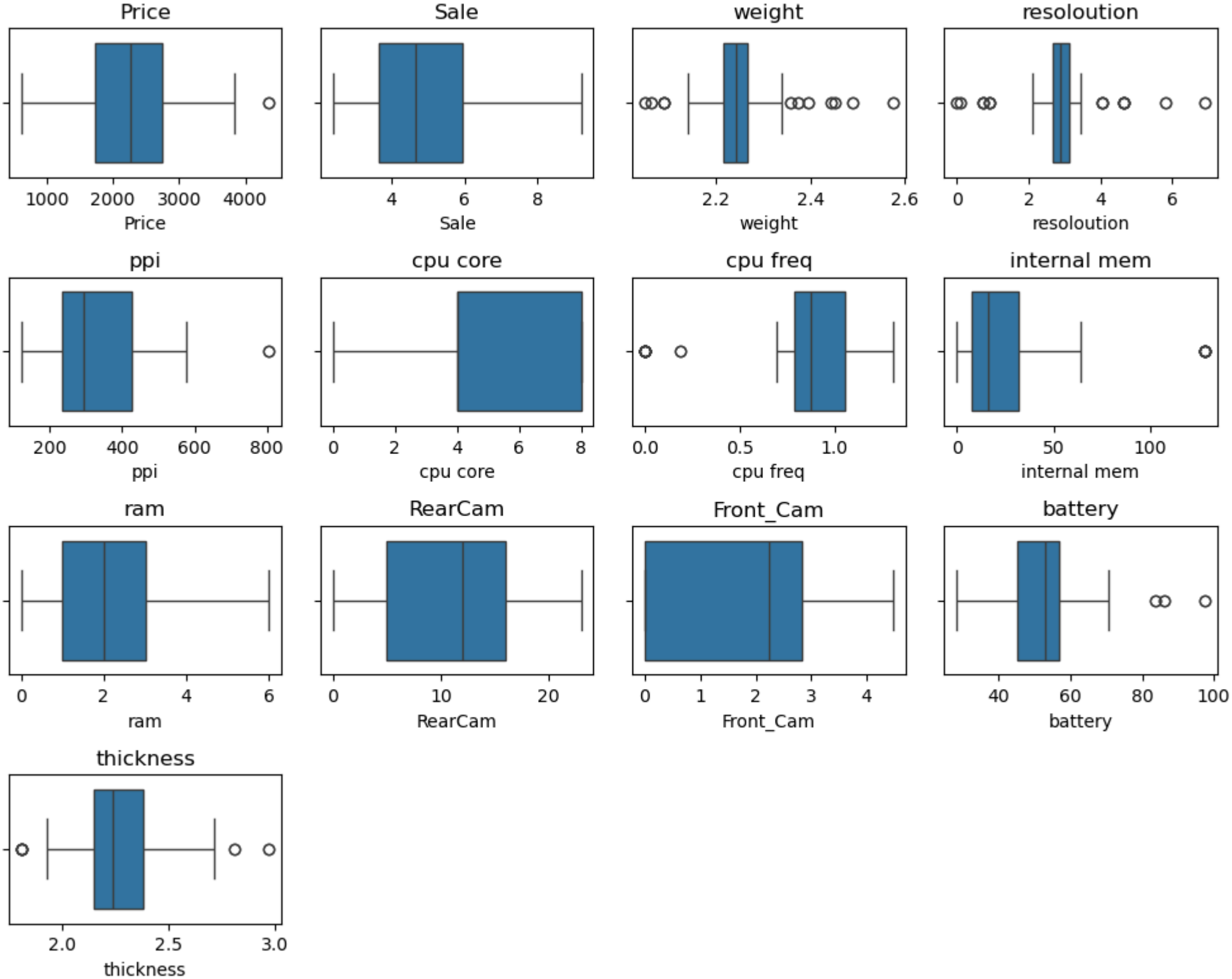

Product_id
0
500
1000
Product_id
Price
1000
2000
3000
4000
Price
Sale
0
2500
5000
7500
10000
Sale
weight
200
400
600
weight
resoloution
2.5
5.0
7.5
10.0
12.5
resoloution
ppi
200
400
600
800
ppi
cpu core
0
2
4
6
8
cpu core
cpu freq
0
1
2
cpu freq
internal mem
0
50
100
internal mem
ram
0
2
4
6
ram
RearCam
0
10
20
RearCam
Front_Cam
0
5
10
15
20
Front_Cam
battery
2000
4000
6000
8000
battery
thickness
5
10
15
thickness

Observations: -The variables 'Sale','Weight' and 'resolution' have higher amount of outliers.

- Whereas the variables 'battery' and 'Thickness' have few outliers.
- 'Sale','Weight' and 'resolution' do not have strong correlation with the target variable 'Price' and data is highly skewed whereas 'battery and 'thickness' have moderate correlation with target variable but highly skewed data.

In [ ]:
```
After removing skewness:
```

In [135…
```
plt.figure(figsize=(10, 8))

for col in data.columns:
    plt.subplot(4, 4, data.columns.get_loc(col) + 1)
    sns.boxplot(x=data[col])
    plt.title(col)

plt.tight_layout()
plt.show()
```


Price
1000 2000 3000 4000
Price
Sale
4 6 8
Sale
weight
2.2 2.4 2.6
weight
resoloution
0 2 4 6
resoloution
ppi
200 400 600 800
ppi
cpu core
0 2 4 6 8
cpu core
cpu freq
0.0 0.5 1.0
cpu freq
internal mem
0 50 100
internal mem
ram
0 2 4 6
ram
RearCam
0 10 20
RearCam
Front_Cam
0 1 2 3 4
Front_Cam
battery
40 60 80 100
battery
thickness
2.0 2.5 3.0
thickness

Observation : A lot of outliers have reduced for 'Sale' variable outliers are completely removed after dealing with skewness,

In [137…

```
#Feature scaling
from sklearn.preprocessing import StandardScaler, MinMaxScaler

scaler_standard = StandardScaler()
##scaler_minmax = MinMaxScaler()

standard_scaled = pd.DataFrame(scaler_standard.fit_transform(data), columns=data.columns)
#minmax_scaled = pd.DataFrame(scaler_minmax.fit_transform(data), columns=data.columns)
print(standard_scaled.head())
#print(minmax_scaled.head())
```

```
   Price  Sale  weight  resoloution   ppi  cpu core  cpu freq  internal mem  \
0   0.18 -1.53   -0.40         0.04  0.66      1.29     -0.09         -0.30
1  -0.61 -1.53   -0.61        -0.79 -0.76     -1.17     -0.17         -0.71
2  -0.39 -1.53   -0.97        -0.30 -0.17     -0.35     -0.32         -0.57
3  -1.18 -1.47   -0.76        -0.79 -0.76     -1.17     -0.17         -0.71
4  -0.61 -1.47   -0.61        -0.79 -0.76     -1.17     -0.17         -0.71

    ram  RearCam  Front_Cam  battery  thickness
0  0.50     0.43       0.89    -0.08      -0.72
1 -0.75    -1.17      -1.42    -0.91       0.57
2 -0.44     0.43       0.41    -0.62      -0.60
3 -1.05    -1.17      -1.42    -1.24       1.04
4 -0.75    -1.17      -1.42    -0.91       0.57
```

In [91]:

v. Create a ML model to predict the price of the phone based on the specifications given.

In [139…

```
X = data.drop('Price', axis=1)
y = data['Price']
```

In [141…

```
std_scale = StandardScaler()
std_scale.fit(X)

X_scaled=pd.DataFrame(data=std_scale.transform(X),columns=X.columns)
X_scaled
```

Out[141…

	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
0	-1.53	-0.40	0.04	0.66	1.29	-0.09	-0.30	0.50	0.43	0.89	-0.08	-0.72
1	-1.53	-0.61	-0.79	-0.76	-1.17	-0.17	-0.71	-0.75	-1.17	-1.42	-0.91	0.57
2	-1.53	-0.97	-0.30	-0.17	-0.35	-0.32	-0.57	-0.44	0.43	0.41	-0.62	-0.60
3	-1.47	-0.76	-0.79	-0.76	-1.17	-0.17	-0.71	-1.05	-1.17	-1.42	-1.24	1.04
4	-1.47	-0.61	-0.79	-0.76	-1.17	-0.17	-0.71	-0.75	-1.17	-1.42	-0.91	0.57
...	...	...	...	...	...	...	...	...	...	...	...	...
156	2.15	0.36	0.21	1.51	-0.35	0.61	3.60	2.36	0.26	1.85	1.01	-0.16
157	2.48	0.24	0.24	1.48	-0.35	0.72	3.60	2.36	1.56	0.89	0.54	-0.43
158	2.54	-0.10	0.24	0.49	1.29	0.98	1.38	1.12	1.56	2.24	0.24	-1.08
159	2.55	0.24	0.24	1.48	-0.35	0.72	3.60	2.36	1.56	0.89	0.54	-0.43
160	2.61	0.71	0.56	0.24	1.29	0.12	-0.30	0.50	1.80	1.85	-0.00	-0.16

161 rows × 12 columns

In [143…

```
from sklearn.model_selection import train_test_split
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

In [145…

```
# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

In [149…

```
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(X_train, y_train)
reg
```

Out[149…

```
▾ LinearRegression
LinearRegression()
```

In [151…

```
y_pred=reg.predict( X_test)
```

In [153…

```
reg.score(X_train,y_train)
```

Out[153…

```
0.9528779474724716
```

In [155…

```
reg.score(X_test, y_test)
```

Out[155…

```
0.9585020569247696
```

In [157…

```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 126.08398480556045
Mean Squared Error: 23524.728709487677
Root Mean Squared Error: 153.37773211743507
```

In [159…

```
tem_df = pd.DataFrame({'Actual Values': y_test, 'Predicted Values': y_pred})
tem_df.head()
```

Out[159…

	Actual Values	Predicted Values
105	833	773.48
108	1676	1874.96
142	2508	2691.71
55	1777	1528.25
94	1511	1617.42

Since their is overfitting issue we will apply the lasso/Ridge/Elastic net regularization techniques.

vi. Check for overfitting and use the Regularization techniques if required

In [161…

```
from sklearn import linear_model
lasso_reg = linear_model.Lasso(alpha=50, max_iter=100, tol=0.1)
lasso_reg.fit(X_train, y_train)
```

Out[161…

```
Lasso
Lasso(alpha=50, max_iter=100, tol=0.1)
```

In [163…

```
lasso_reg.score(X_train, y_train)
```

Out[163…

```
0.939550120995186
```

In [165…

```
lasso_reg.score(X_test,y_test)
```

Out[165…

```
0.9377902353931526
```

In [167…

```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 126.08398480556045
Mean Squared Error: 23524.728709487677
Root Mean Squared Error: 153.37773211743507
```

In [169…

```
from sklearn.linear_model import Ridge
ridge_reg= Ridge(alpha=50, max_iter=100, tol=0.1)
ridge_reg.fit(X_train,y_train)
```

Out[169…

```
Ridge
Ridge(alpha=50, max_iter=100, tol=0.1)
```

In [171…

```
ridge_reg.score(X_train, y_train)
```

Out[171…

```
0.9378605759518488
```

In [173…

```
ridge_reg.score(X_test, y_test)
```

Out[173…

```
0.9305843435918163
```

In [175…

```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 126.08398480556045
Mean Squared Error: 23524.728709487677
Root Mean Squared Error: 153.37773211743507
```

In [177…

```
from sklearn.linear_model import ElasticNet
elasticnet_model = ElasticNet(alpha=0.1, l1_ratio=0.5)
elasticnet_model.fit(X_train, y_train)
#y_train = elasticnet_model.predict(X_train)
#y_pred = elasticnet_model.predict(X_test)
```

Out[177…

```
ElasticNet
ElasticNet(alpha=0.1)
```

In [179…

```
elasticnet_model.score(X_train, y_train)
```

Out[179…

```
0.9508955446221606
```

In [181…

```
elasticnet_model.score(X_test, y_test)
```

Out[181…

```
0.9535242518477336
```

In [183…

```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 126.08398480556045
Mean Squared Error: 23524.728709487677
Root Mean Squared Error: 153.37773211743507
```

In [185…

```
from sklearn.tree import DecisionTreeRegressor
DT_regressor_object = DecisionTreeRegressor()
DT_regressor_object.fit(X_train, y_train)
```

Out[185…

```
DecisionTreeRegressor
DecisionTreeRegressor()
```

In [187…

```
y_test_prediction = DT_regressor_object.predict(X_test)
```

In [189…

```
temp_df = pd.DataFrame({'Actual values':y_test, 'predicted values':y_test_prediction})
temp_df.head(7)
```

Out[189…

	Actual values	predicted values
105	833	754.00
108	1676	1676.00
142	2508	2580.00
55	1777	1777.00
94	1511	1831.00
29	1950	1916.00
101	833	754.00

In [191…

```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 126.08398480556045
Mean Squared Error: 23524.728709487677
Root Mean Squared Error: 153.37773211743507
```

In [193…

```
from sklearn.ensemble import RandomForestRegressor
RF_regressor_object = RandomForestRegressor()
RF_regressor_object.fit(X_train, y_train)
```

Out[193…

```
▾ RandomForestRegressor
RandomForestRegressor()
```

In [195…

```
y_test_prediction = RF_regressor_object.predict(X_test)
```

In [197…

```
temp_df = pd.DataFrame({'Actual values':y_test, 'predicted values':y_test_prediction})
temp_df.head()
```

Out[197…

	Actual values	predicted values
105	833	831.08
108	1676	1760.18
142	2508	2658.55
55	1777	1746.02
94	1511	1590.01

In [199…

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_test_prediction))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_test_prediction))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_test_prediction)))
```

```
Mean Absolute Error: 96.23393939393941
Mean Squared Error: 14986.166018181824
Root Mean Squared Error: 122.41799711717972
```

vii. #Compare the performance metrics of training dataset and testing dataset for all the different algorithms used (Linear/Ridge/Lasso/ElasticNet)

In [201…

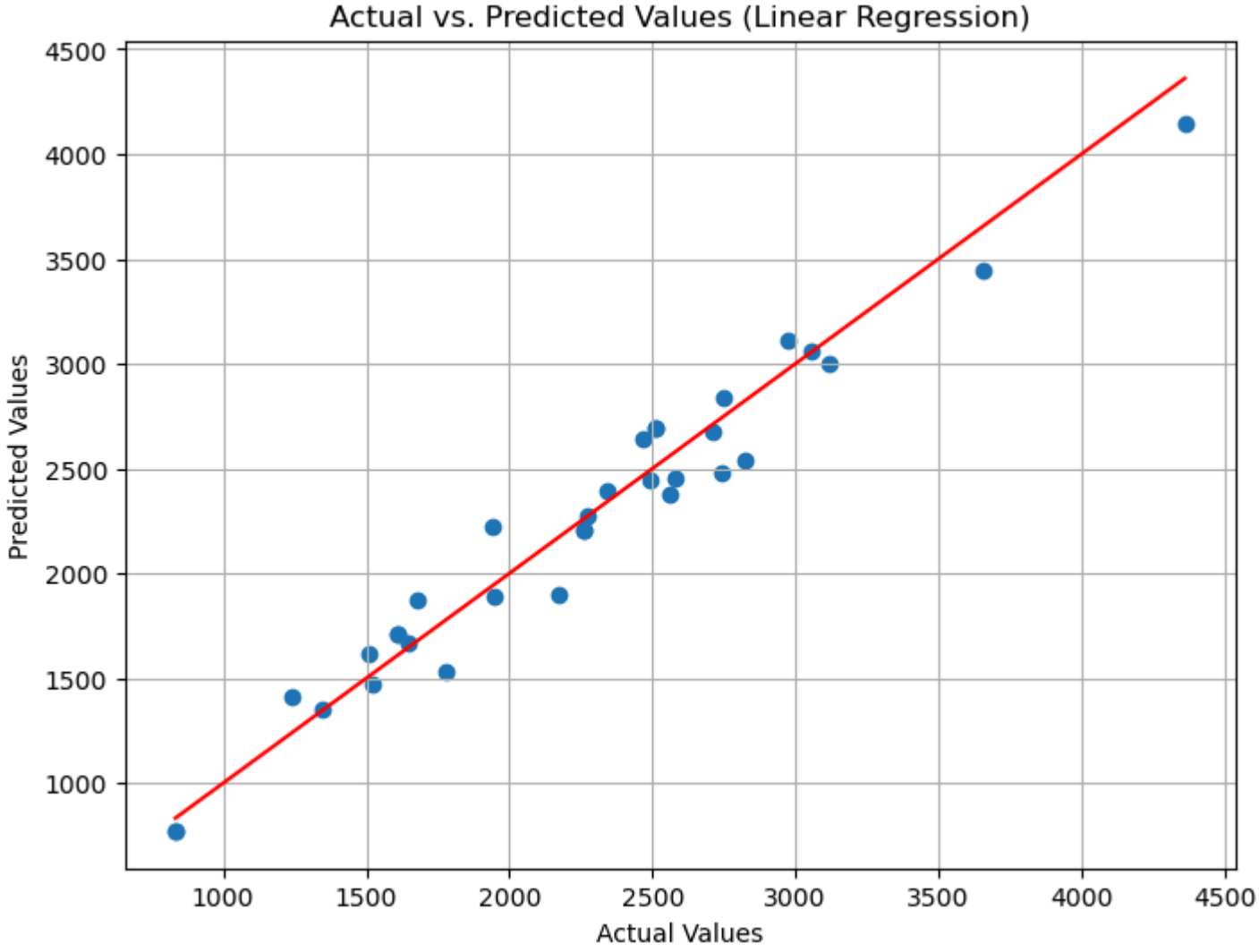
```
models = {
    "Linear Regression": LinearRegression(),
    "Ridge": Ridge(),
    "Lasso": Lasso(),
    "ElasticNet": ElasticNet(alpha=0.1, l1_ratio=0.5, random_state=42)  # Using some default hyperparameters
}

# Initialize dictionary to store results
results = {
    "Model": [],
    "Training RMSE": [],
    "Testing RMSE": [],
    "Training R^2": [],
    "Testing R^2": []
}

# Train and evaluate each model
for model_name, model in models.items():
    # Train the model
    model.fit(X_train, y_train)

    # Predict on the training and testing data
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)
```

```
    # Calculate performance metrics
    train_rmse = mean_squared_error(y_train, y_train_pred, squared=False)
    test_rmse = mean_squared_error(y_test, y_test_pred, squared=False)

    train_r2 = r2_score(y_train, y_train_pred)
    test_r2 = r2_score(y_test, y_test_pred)

    # Store the results in the dictionary
    results["Model"].append(model_name)
    results["Training RMSE"].append(train_rmse)
    results["Testing RMSE"].append(test_rmse)
    results["Training R^2"].append(train_r2)
    results["Testing R^2"].append(test_r2)


results_df = pd.DataFrame(results)
print(results_df)
```

```
               Model  Training RMSE  Testing RMSE  Training R^2  Testing R^2
0  Linear Regression         166.91        153.38          0.95         0.96
1              Ridge         167.30        155.33          0.95         0.96
2              Lasso         167.36        156.18          0.95         0.96
3         ElasticNet         170.38        162.32          0.95         0.95
```

In [203…

```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs. Predicted Values (Linear Regression)")
plt.grid(True)
plt.show()
```


Actual vs. Predicted Values (Linear Regression)
Predicted Values
Actual Values
4500
4000
3500
3000
2500
2000
1500
1000
1000
1500
2000
2500
3000
3500
4000
4500

In [ ]:

In [ ]: