In [ ]:

```
A Explain the main assumptions of Linear Regression in detail.
Answer
1.      Linearity: The independent and dependent variables have a linear relationship with one another
2.      There should not be multicollinearity
3.      Homoscedasticity should be there-errors are in defined range (variance of the errors is constant.)
4. Heteroscedasticity should not be there-errors are increasing continuously (variance of the errors is not con
5.      Normality: Errors should follow normally distribution. This means that the should follow a bell-shaped
```

In [ ]:

```
B.      What is the difference between R-squared and Adjusted R-squared?
Answer
The difference between R-squared and Adjusted R-squared lies in how they handle the number of predictors (featu
1. R-squared (R²):
R-squared measures the proportion of the variance in the dependent variable (target) that is explained by the i
Mathematically, it is the ratio of the variance explained by the model to the total variance in the data.
The value of R-squared ranges from 0 to 1.
A higher R² indicates that a larger proportion of the variance in the target variable is Explained by the predi
Drawback: R-squared can increase even if the added predictors do not actually improve the model. This can be mi

3. Adjusted R-squared:
Definition: Adjusted R-squared modifies the R-squared value to account for the number of predictors in the mode
when comparing models with different numbers of features.It penalizes the addition of unnecessary predictors,
Adjusted R-squared provides a more accurate measure by adjusting for the number of predictors,helping you avoid
and making it better for comparing models with different features.
```

In [ ]:

```
C.      What are the different types of Regularization techniques in Regression. Explain in detail with cost fu
Answer
The two main types of regularization techniques used in regression are Lasso and Ridge regularization.
There is also a third method, Elastic Net, which combines both Lasso and Ridge.
1-Ridge regression(L2) adds a penalty term proportional to the sum of the squared coefficients to the cost func
2-Lasso regression(L1) adds a penalty term proportional to the sum of the absolute values of the coefficients t
3-Elastic Net (L1+L2) is a regularization technique that combines the penalties of both Ridge (L2) and Lasso (L
It balances the benefits of both methods and is useful when there are multiple correlated features.
```

In [ ]:

```
D.      How logistic regression works for multiclass classification. Explain in detail.
Answer
OVR Method (One vs Rest)
It is a logistic regression algorithm that is used when the target variable has two or more classes.
It trains model for each class, with that class as the positive class and all other classes as the negative cla
It predicts the probability of each class and selects the class with the highest probability as the predicted c
One-vs-Rest method will break down this problem into three or more binary classification problems:

Eg to classify various fruits into three types of fruits:
banana, orange or apple. Since there are three classes in the classification problem,
the One-vs-Rest method will break down this problem into three binary classification problems:

•       Problem 1 : Banana vs [Orange, Apple]
•       Problem 2 : Orange vs [Banana, Apple]
•       Problem 3 : Apple vs [Banana, Orange]

A major downside or disadvantage of this method is that many models have to be created. For a multi-class probl
'n' number of models have to be created, which may slow down the entire process.
Takes more time to train. However, it is very useful with datasets having a small number of classes

Softmax function
•       estimates the probability of an instance belonging to a given class by using the softmax function
•       Softmax function computes the exponential of every score, then normalizes them (dividing by the sum of
•       Higher score value-Higher probability

Softmax function helps us to achieve two functionalities:
1. Convert all scores to probabilities.
2. Sum of all probabilities is 1.
```

In [ ]:

```
E.      Explain the performance metrics of logistic regression.
Answer

1.      Confusion Matrix

•       A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, wh
•       gives us a holistic view of how well our classification model is performing and what kinds of errors it
•       The matrix compares the actual target values with those predicted by the machine learning model.

Binary classification confusion Matrix
It breaks down the predictions into four categories: correct predictions for both classes (true positives and t
incorrect predictions (false positives and false negatives). This helps you understand where the model is makin

•       The target variable has two values: Positive or Negative
•       The columns represent the actual values of the target variable
•       The rows represent the predicted values of the target variable

Important Terms in a Confusion Matrix
True Positive (TP)
```

```
•       The predicted value matches the actual value
•       The actual value was positive, and the model predicted a positive value.

True Negative (TN)
•       The predicted value matches the actual value
•       The actual value was negative, and the model predicted a negative value.

False Positive (FP) – Type I Error
•       The predicted value was falsely predicted.
•       The actual value was negative, but the model predicted a positive value.
•       Also known as the type I error.

False Negative (FN) – Type II Error
•       The predicted value was falsely predicted.
•       The actual value was positive, but the model predicted a negative value.
•       Also known as the type II error.

2-Accuracy
Accuracy is calculated by dividing the number of correct predictions by the total number of predictions across a

3-Precision
•       how many of the instances predicted as positive are actually positive

4-Recall (Sensitivity / True Positive Rate)
•       how many of the actual positive cases we were able to predict correctly with our model.

5- F1-score
•       F1-score gives a combined idea about these two metrics (precision and recall) It provides a better sense
•        maximum when Precision is equal to Recall.

6-Area Under the curve (AUC)-Receiver Operating Characteristic Curve (AUC-ROC):
•       The ROC curve plots the true positive rate (Sensitivity) against the false positive rate at various thre
•       AUC-ROC measures the area under this curve, providing an aggregate measure of a model's performance acro
```

F. Use the Mobile price prediction dataset from below Kaggle link and create an end to end project on Jupyter/Colab. https://www.kaggle.com/datasets/mohannapd/mobile-price-prediction/data i. Download the dataset from above link and load it into your Python environment. ii. Perform the EDA and do the visualizations. iii. Check the distributions/skewness in the variables and do the transformations if required. iv. Check/Treat the outliers and do the feature scaling if required. v. Create a ML model to predict the price of the phone based on the specifications given. vi. Check for overfitting and use the Regularization techniques if required vii. Compare the performance metrics of training dataset and testing dataset for all the different algorithms used (Linear/Ridge/Lasso/ElasticNet)

# Mobile Price Prediction Project-Linear Regression

About Dataset Mobile price depends on various factors such as resolution, brand, size, weight, imaging quality, RAM, battery and cpu power. In this dataset, we want to estimate the price of mobile phones using the above features

Problem Statement

The objective is to develop a predictive model that accurately estimates the price of mobile phones based on a variety of features including resolution, brand, size, weight, imaging quality, RAM, battery capacity, and CPU power. By leveraging machine learning techniques(Linear Regression), the goal is to create a pricing model that assists consumers, manufacturers, and retailers in making informed decisions regarding mobile phone pricing strategies, product positioning, and purchasing choices.

In [206…

```
import warnings
warnings.filterwarnings('ignore')

import lightgbm as lgb
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pickle
import seaborn as sns
import xgboost as xgb

from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
from sklearn.linear_model import ElasticNet, Lasso, LinearRegression, Ridge
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor

sns.set()
%matplotlib inline
```

Loading the Dataset

In [208…
```
df=pd.read_excel("Cellphone.xlsx")
```

EDA

In [210…
```
df.shape
```

Out[210…
```
(161, 14)
```

In [212…
```
df.head()
```

Out[212…

| | 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 [214…
```
df.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 [17]:
```
df.describe().T
# Avg Price 2215
#Avg wt-170
#Avg Ram 2 gb
```

Out[17]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **Product_id** | 161.0 | 675.559006 | 410.851583 | 10.0 | 237.0 | 774.00 | 1026.000 | 1339.0 |
| **Price** | 161.0 | 2215.596273 | 768.187171 | 614.0 | 1734.0 | 2258.00 | 2744.000 | 4361.0 |
| **Sale** | 161.0 | 621.465839 | 1546.618517 | 10.0 | 37.0 | 106.00 | 382.000 | 9807.0 |
| **weight** | 161.0 | 170.426087 | 92.888612 | 66.0 | 134.1 | 153.00 | 170.000 | 753.0 |
| **resoloution** | 161.0 | 5.209938 | 1.509953 | 1.4 | 4.8 | 5.15 | 5.500 | 12.2 |
| **ppi** | 161.0 | 335.055901 | 134.826659 | 121.0 | 233.0 | 294.00 | 428.000 | 806.0 |
| **cpu core** | 161.0 | 4.857143 | 2.444016 | 0.0 | 4.0 | 4.00 | 8.000 | 8.0 |
| **cpu freq** | 161.0 | 1.502832 | 0.599783 | 0.0 | 1.2 | 1.40 | 1.875 | 2.7 |
| **internal mem** | 161.0 | 24.501714 | 28.804773 | 0.0 | 8.0 | 16.00 | 32.000 | 128.0 |
| **ram** | 161.0 | 2.204994 | 1.609831 | 0.0 | 1.0 | 2.00 | 3.000 | 6.0 |
| **RearCam** | 161.0 | 10.378261 | 6.181585 | 0.0 | 5.0 | 12.00 | 16.000 | 23.0 |
| **Front_Cam** | 161.0 | 4.503106 | 4.342053 | 0.0 | 0.0 | 5.00 | 8.000 | 20.0 |
| **battery** | 161.0 | 2842.111801 | 1366.990838 | 800.0 | 2040.0 | 2800.00 | 3240.000 | 9500.0 |
| **thickness** | 161.0 | 8.921739 | 2.192564 | 5.1 | 7.6 | 8.40 | 9.800 | 18.5 |

In [19]:
```
df.isnull().sum()
```

Out[19]:
```
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 [21]:
```
df.duplicated().sum()
```

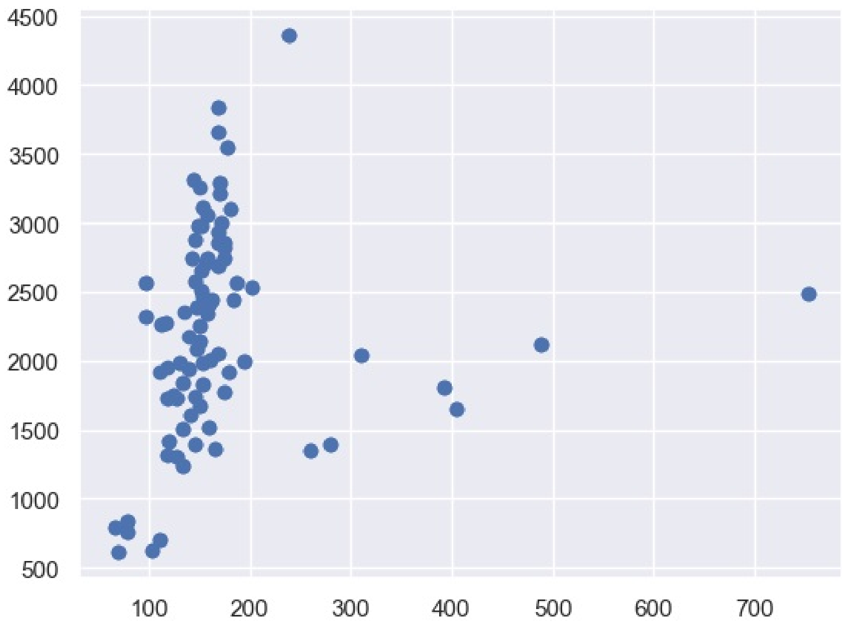
Out[21]:
```
0
```

In [216…
```
df.columns
```

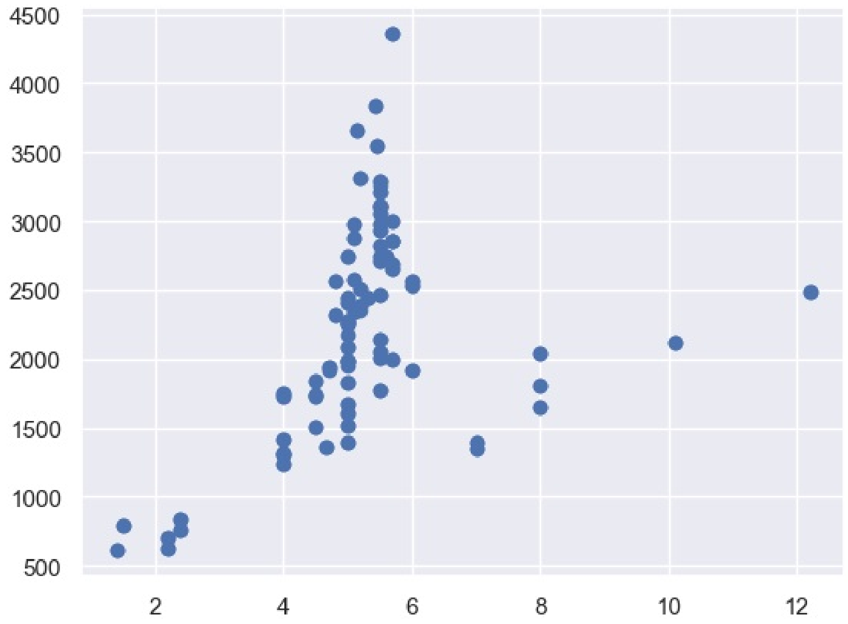
Out[216…
```
Index(['Product_id', 'Price', 'Sale', 'weight', 'resoloution', 'ppi',
       'cpu core', 'cpu freq', 'internal mem', 'ram', 'RearCam', 'Front_Cam',
       'battery', 'thickness'],
      dtype='object')
```

Data Visualization We will be using Scatter plots. They will observe the relationship between variables and uses dots to represent the connection between them.

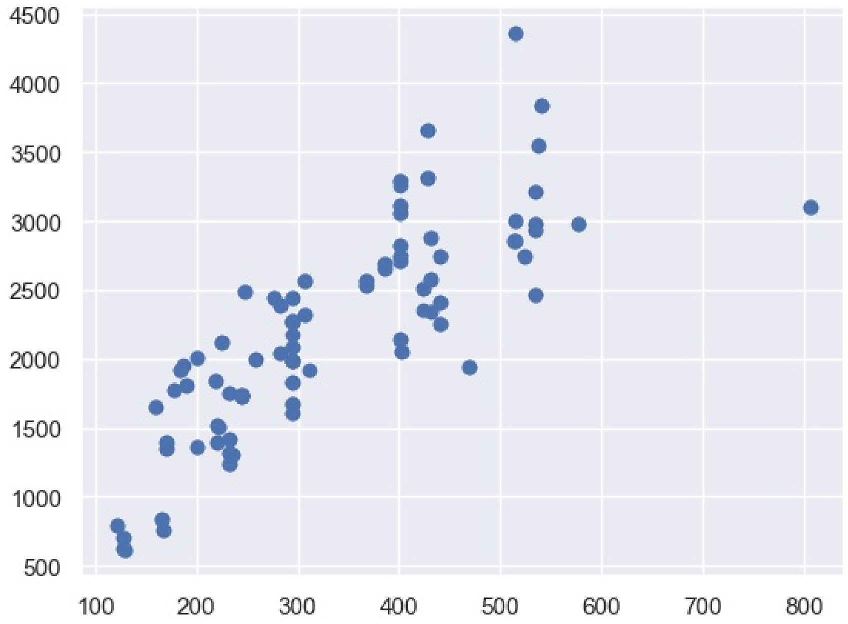
In [194…
```
plt.scatter(df["weight"], df["Price"])
plt.show()
```



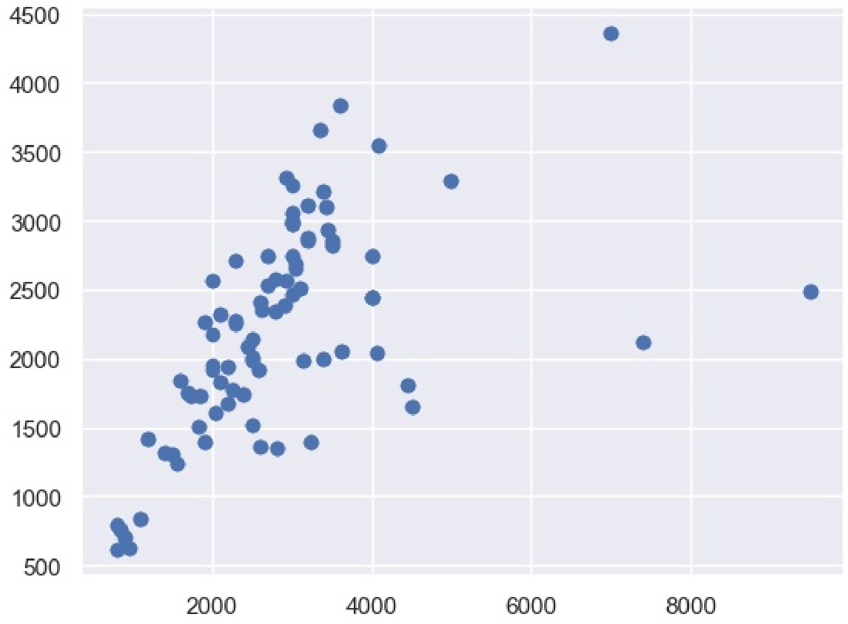

In [27]:
```
plt.scatter(df["resoloution"], df["Price"])
plt.show()
```




In [35]:
```
plt.scatter(df["ppi"], df["Price"])
plt.show()
```

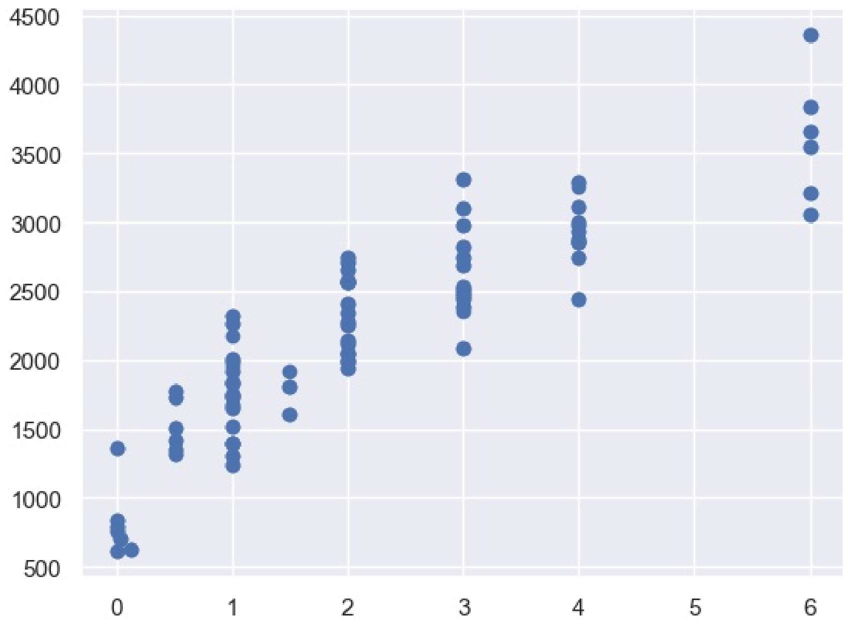



In [37]:
```
plt.scatter(df["battery"], df["Price"])
plt.show()
```



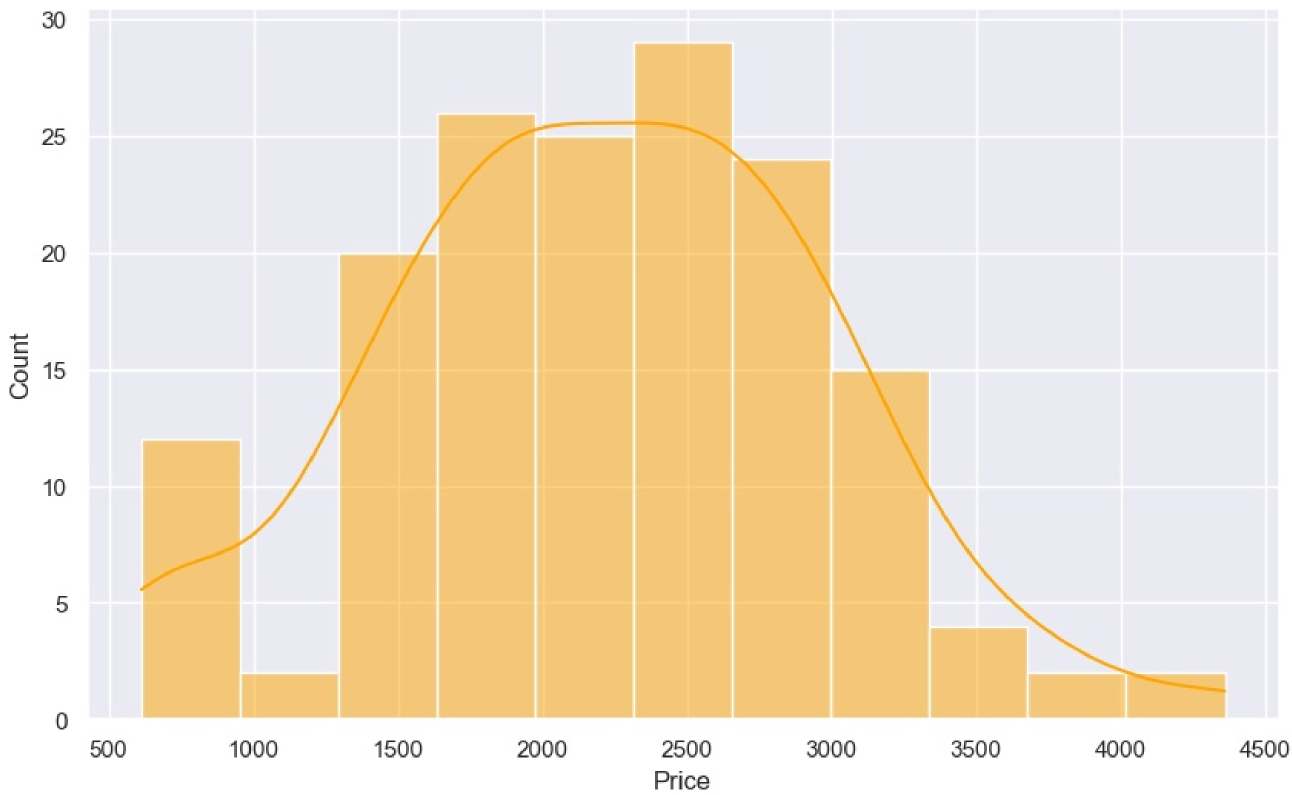

In [39]:
```
plt.scatter(df["ram"], df["Price"])
```

```
plt.show()
# price increases with ram
```



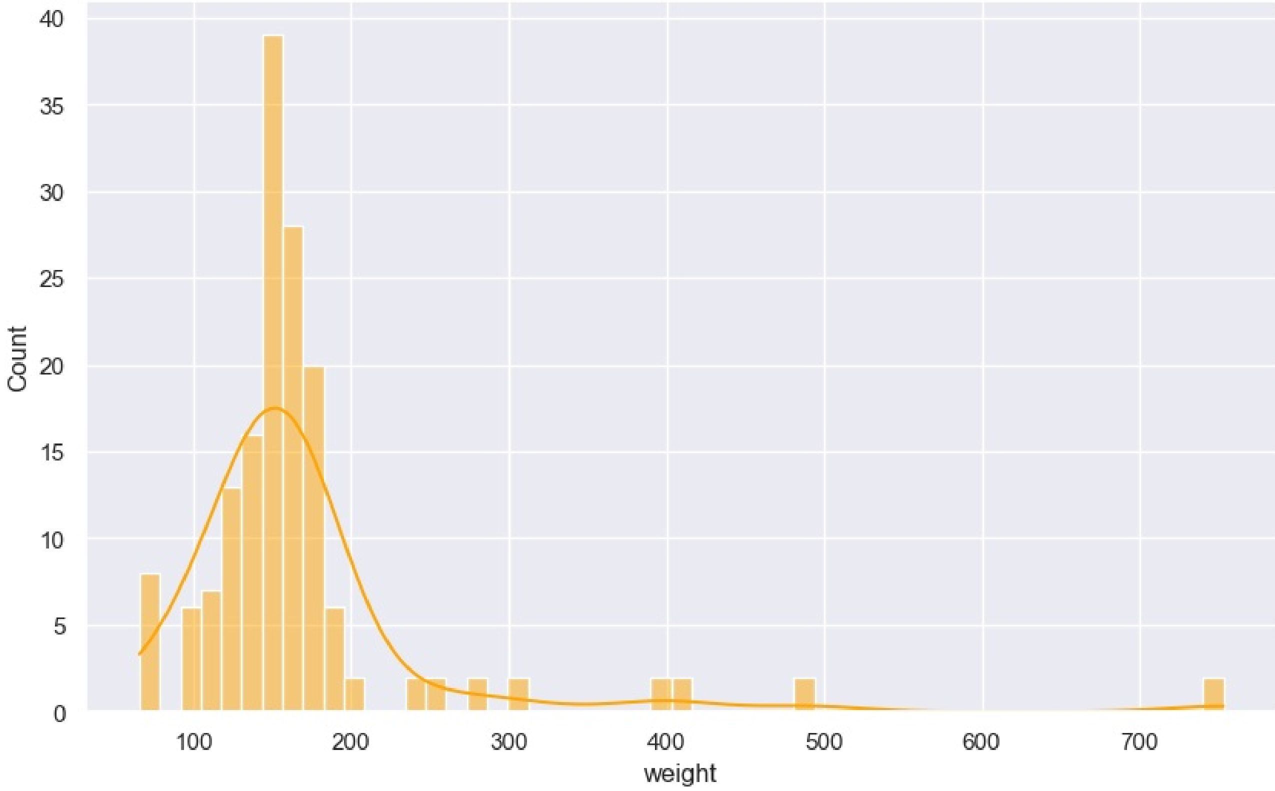

In [218…

```
plt.figure(figsize=(10,6))
sns.histplot(df['Price'],kde=True,color='orange')
plt.show()
```




In [43]:

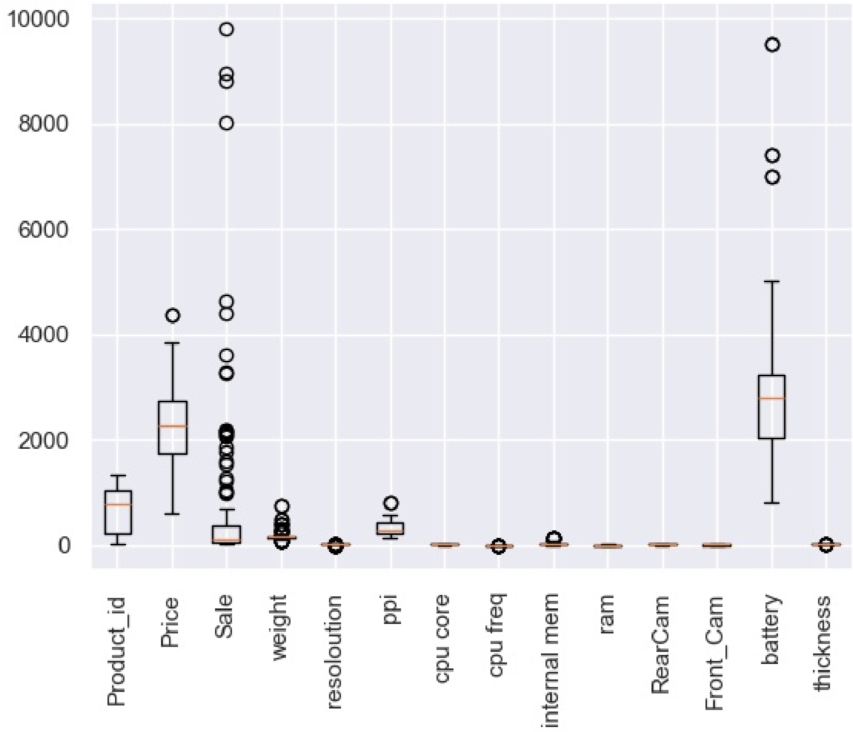
```
plt.figure(figsize=(10,6))
sns.histplot(df['weight'],kde=True,color='orange')
plt.show()
```




In [63]:
```
sns.pairplot(df)
plt.show()
```


In [ ]:
```
Outlier Detection
```

In [220…
```
plt.boxplot(df,labels=df.columns)
plt.xticks(rotation=90)
plt.show()
```




In [222…

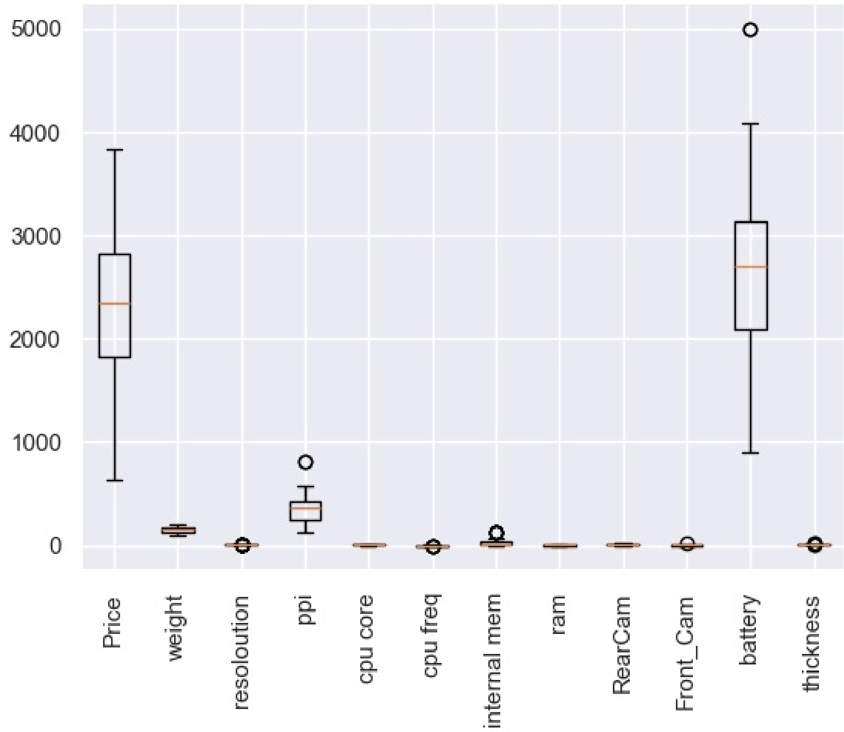
```
q1 = df['weight'].quantile(0.25)
q3 = df['weight'].quantile(0.75)

iqr = q3 - q1
lower_bound = q1 - (1.5 * iqr)
upper_bound = q3 + (1.5 * iqr)

outlier = (df['weight'] < lower_bound) | (df['weight'] > upper_bound)
df = df[~outlier]
```

In [224…

```
df.drop(columns=['Product_id','Sale'],inplace=True)
```

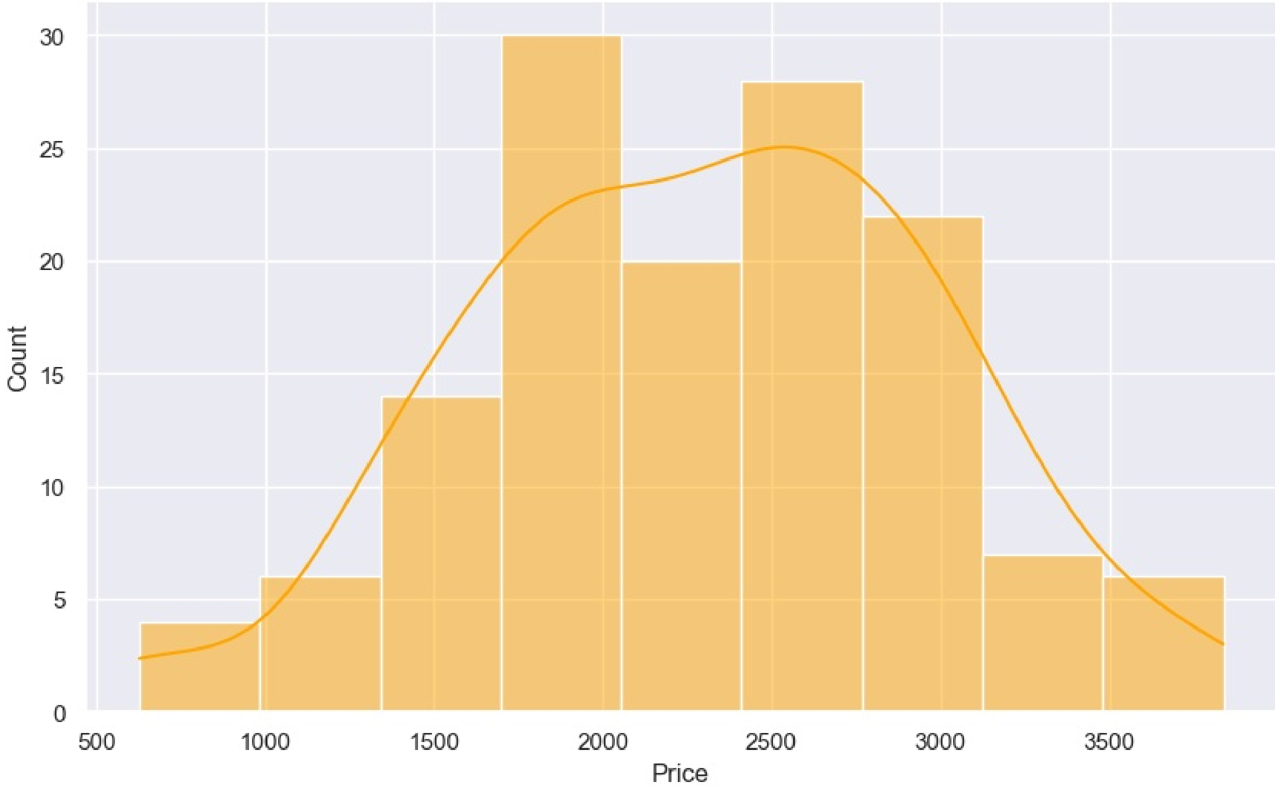
In [226…

```
plt.boxplot(df,labels=df.columns)
plt.xticks(rotation=90)
plt.show()
```

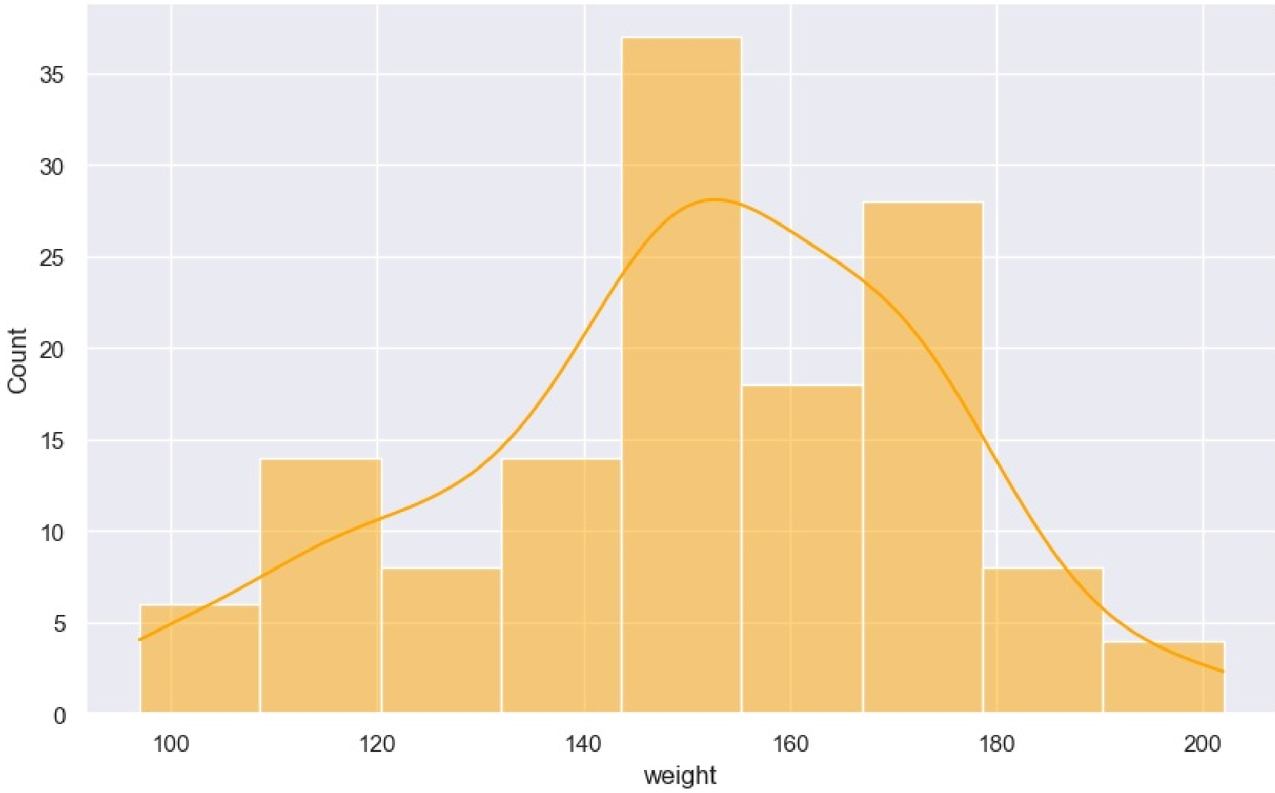



In [58]:

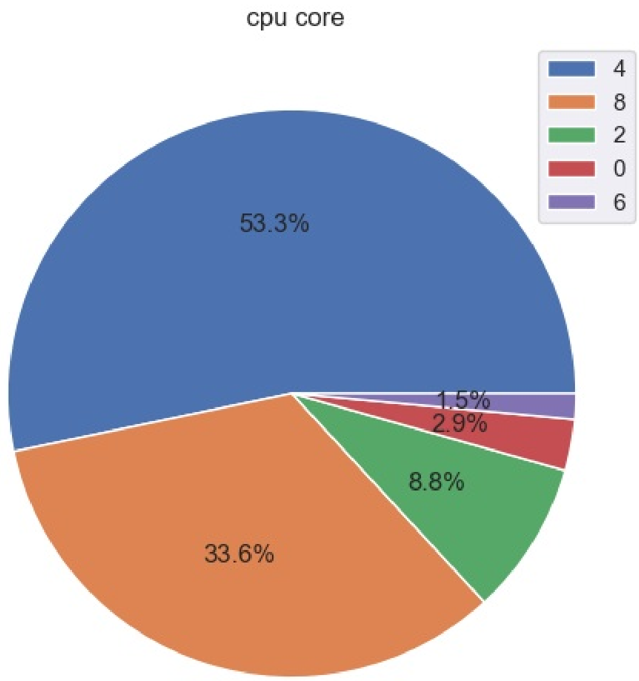
```
plt.figure(figsize=(10,6))
sns.histplot(df['Price'],kde=True,color='orange')
plt.show()
# Price column is normally distributed
```




In [17]:
```
plt.figure(figsize=(10,6))
sns.histplot(df['weight'],kde=True,color='orange')
plt.show()
# The weight column seems to be normally distributed in the dataset
```

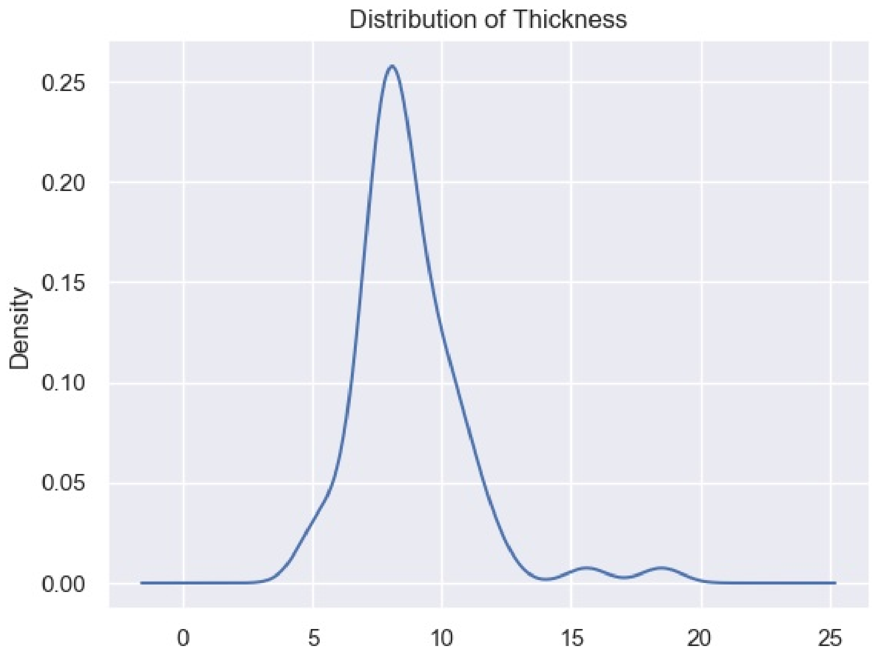



In [71]:
```
plt.figure(figsize=(10,6))
plt.pie(df['cpu core'].value_counts(),autopct = ("%1.1f%%"))
plt.title(" cpu core")
plt.legend(df['cpu core'].value_counts().index)
plt.show()
# Most of the smartphones have 4 CPU cores followed by 8 and 2 respectively
```



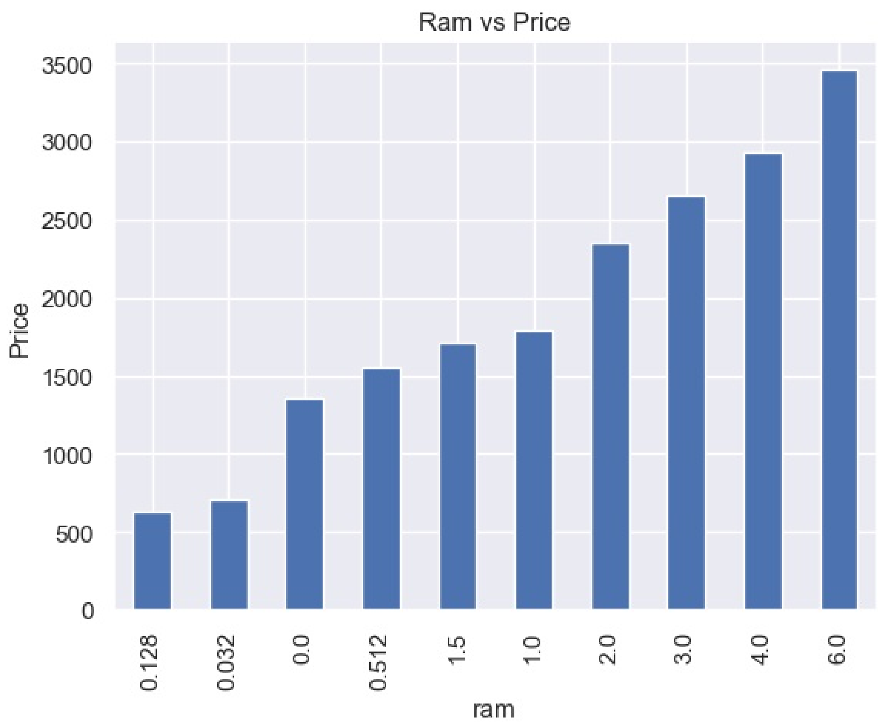

In [228…

```
df['thickness'].plot(kind= 'kde')
plt.title("Distribution of Thickness")
plt.show()
## Most common density is around 9mm
```

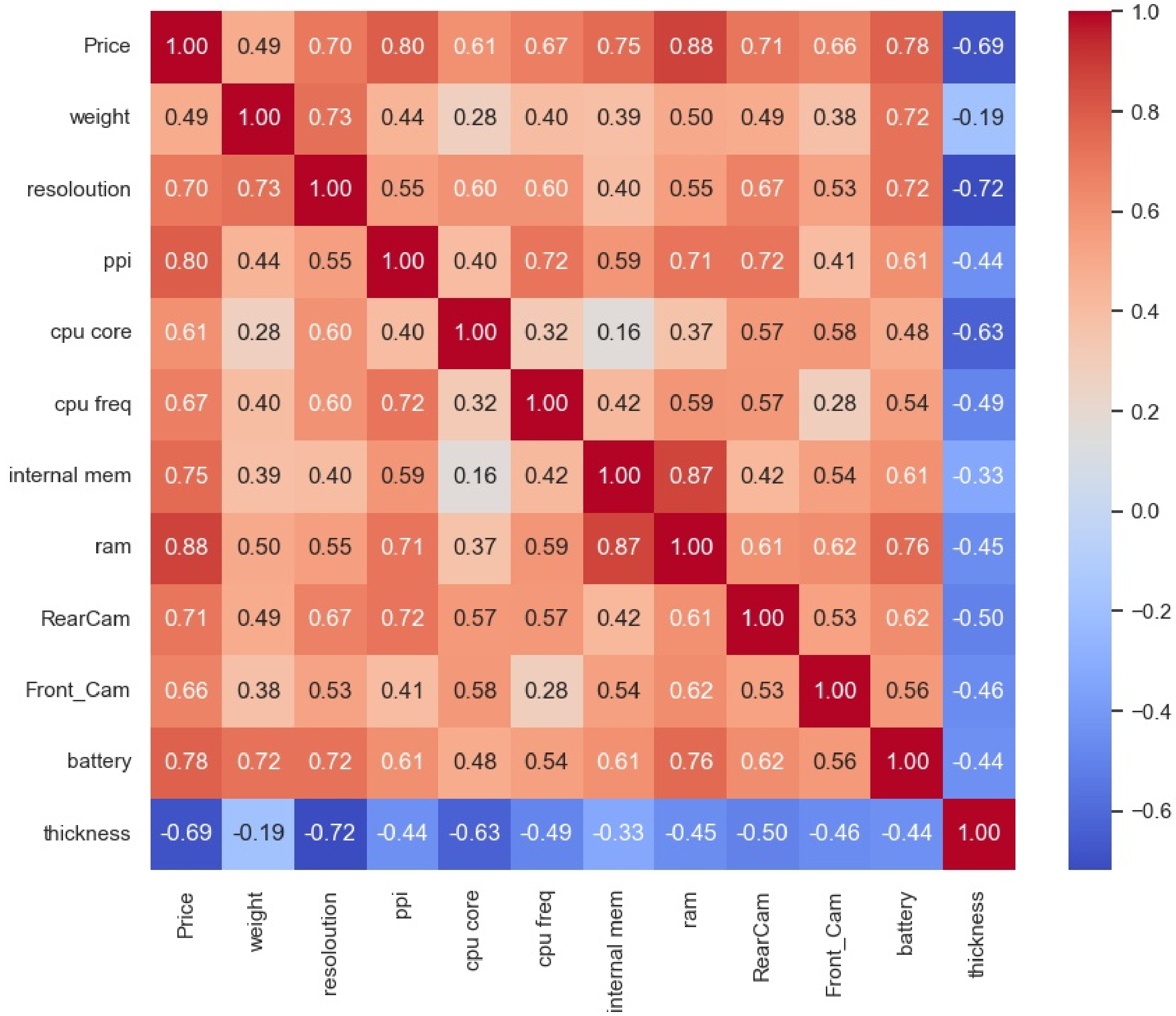



In [230…

```
df.groupby('ram').Price.mean().sort_values().head(20).plot(kind = 'bar')
plt.ylabel("Price")
plt.title("Ram vs Price")
plt.show()
# it shows that when ram increases then the price also increases
```

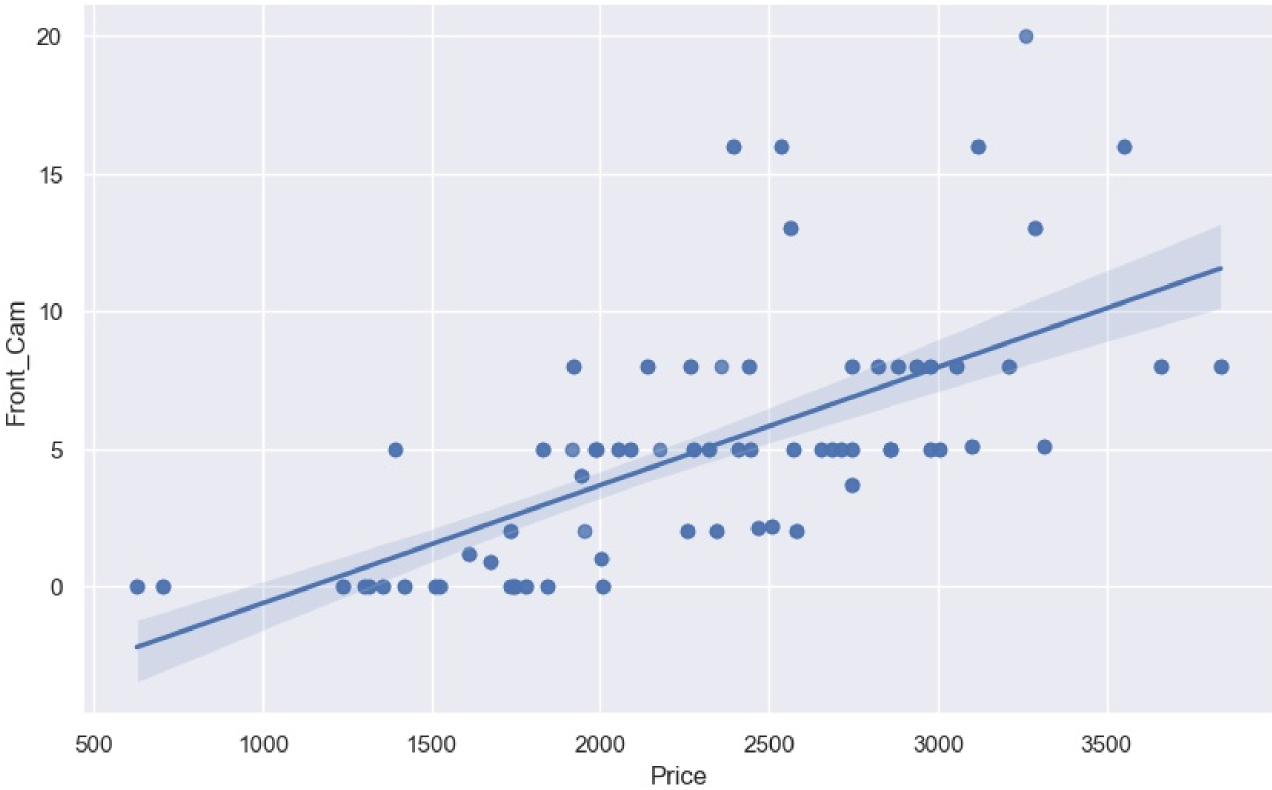



In [76]:
```
# checking correlation between features
plt.figure(figsize=(10,8))
corr = df.corr()
sns.heatmap(corr,cmap='coolwarm',annot=True,fmt='.2f')
plt.show()
```



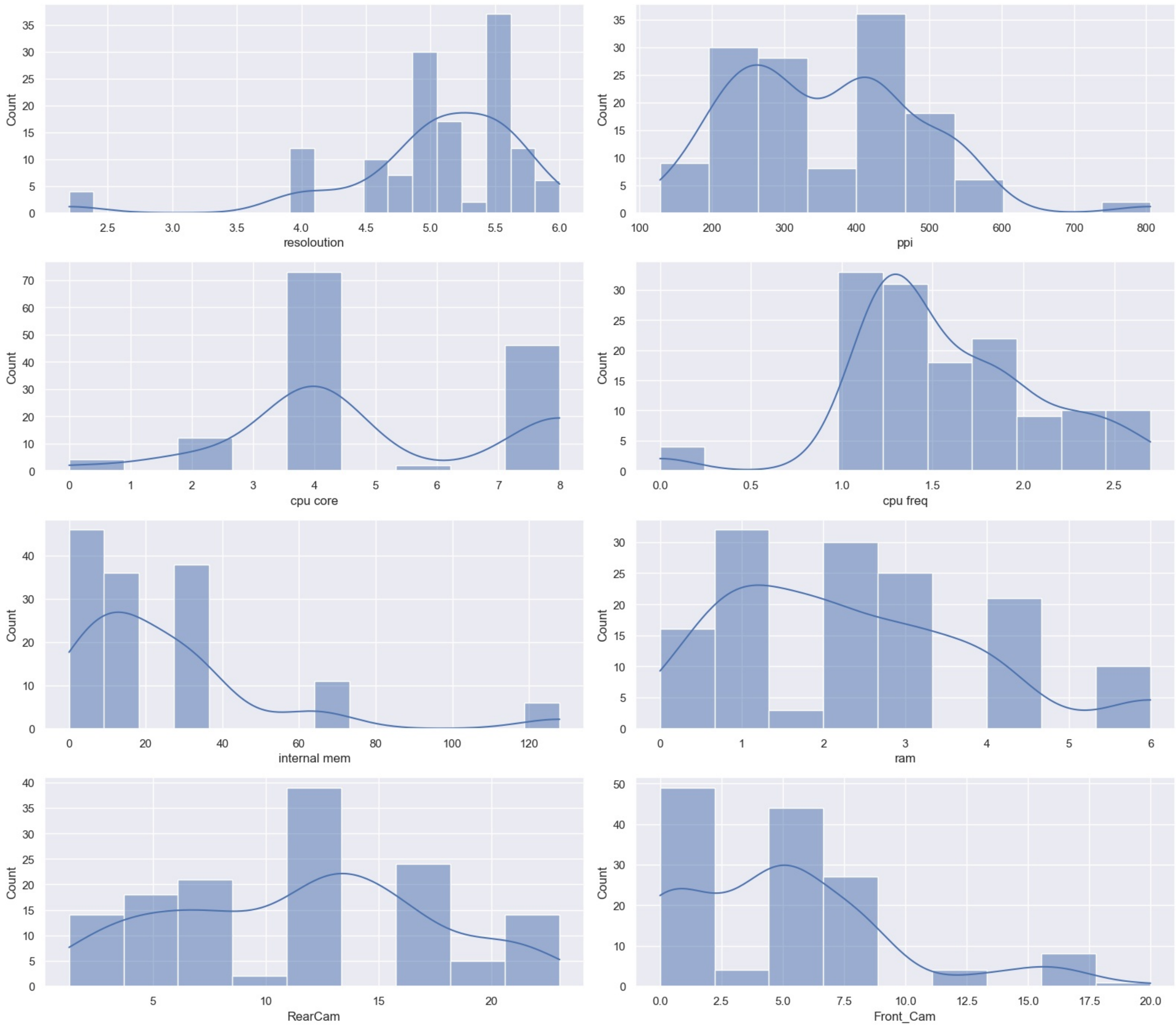

In [288…
```
plt.figure(figsize=(10,6))
sns.regplot(x = df['Price'], y = df['Front_Cam'])
plt.show()
```

```
# there is a linear positive relationship between price and front cam
```




In [290…

```
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(df,x=df[cols[i]],ax=axes[i],kde=True)
    #axes[i].set_ylabel('Price')
    axes[i].set_xlabel(f"{cols[i]}")
plt.tight_layout()
plt.show()
```




In [292…

```
cols=['weight', 'resoloution', 'ppi',
       'cpu core', 'cpu freq', 'internal mem', 'ram', 'RearCam', 'Front_Cam',
       'battery', 'thickness']
x=df[cols]
y=df["Price"]
```

In [238…

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=1)
```

In [310…

```
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

# Building the Model

Linear Regression Model

In [312…

```
Linear_model=LinearRegression()
Linear_model.fit(x_train_scaled,y_train)
y_train_predict=Linear_model.predict(x_train_scaled)
y_test_predict=Linear_model.predict(x_test_scaled)
```

In [314…

```
print("Accuracy Scores for Linear Regression model on raw data")
mse=mean_squared_error(y_test,y_test_predict)
rmse = np.sqrt(mse)
r2_Train=r2_score(y_train,y_train_predict)
r2_Test=r2_score(y_test,y_test_predict)
print("Mean Squared Error",mse)
print("Root Mean Squared Error",rmse)
print("R-Squared Score(Train)", r2_Train)
print("R-Squared Score(Test)", r2_Test)
print("***************" * 7)
```

```
Accuracy Scores for Linear Regression model on raw data
Mean Squared Error 28548.04569994511
Root Mean Squared Error 168.96166932161006
R-Squared Score(Train) 0.9302076685923554
R-Squared Score(Test) 0.9507837401560588
**************************************************************************************************
```

In [ ]:
```
Lasso Regression(L1 Regularisation)
```

In [250…
```
Lasso_model=Lasso()
Lasso_model.fit(x_train_scaled,y_train)
y_train_predict=Lasso_model.predict(x_train_scaled)
y_test_predict=Lasso_model.predict(x_test_scaled)
```

In [302…
```
print("Accuracy Scores for Lasso Regression (L1 Regularization) model on raw data")
mse=mean_squared_error(y_test,y_test_predict)
rmse = np.sqrt(mse)
r2_Train=r2_score(y_train,y_train_predict)
r2_Test=r2_score(y_test,y_test_predict)
print("Mean Squared Error",mse)
print("Root Mean Squared Error",rmse)
print("R-Squared Score(Train)", r2_Train)
print("R-Squared Score(Test)", r2_Test)
print("**************" * 7)
```

```
Accuracy Scores for Lasso Regression (L1 Regularization) model on raw data
Mean Squared Error 28548.04569994511
Root Mean Squared Error 168.96166932161006
R-Squared Score(Train) 0.9302076685923554
R-Squared Score(Test) 0.9507837401560588
**************************************************************************************************
```

In [ ]:
```
ridge_Regression(L2 Regularisation)
```

In [304…
```
Ridge_model=Ridge()
Ridge_model.fit(x_train_scaled,y_train)
y_train_predict=Ridge_model.predict(x_train_scaled)
y_test_predict=Ridge_model.predict(x_test_scaled)
```

In [306…
```
print("Accuracy Scores for Ridge Regression (L2 Regularization) model on raw data")
mse=mean_squared_error(y_test,y_test_predict)
rmse = np.sqrt(mse)
r2_Train=r2_score(y_train,y_train_predict)
r2_Test=r2_score(y_test,y_test_predict)
print("Mean Squared Error",mse)
print("Root Mean Squared Error",rmse)
print("R-Squared Score(Train)", r2_Train)
print("R-Squared Score(Test)", r2_Test)
print("**************" * 7)
```

```
Accuracy Scores for Ridge Regression (L2 Regularization) model on raw data
Mean Squared Error 27703.649855463536
Root Mean Squared Error 166.4441343378118
R-Squared Score(Train) 0.9301210922887598
R-Squared Score(Test) 0.9522394617045651
**************************************************************************************************
```

In [ ]:
```
Elastic Net_Regression(L1+L2 Regularisation)
```

In [258…
```
enet_model=ElasticNet()
enet_model.fit(x_train_scaled,y_train)
y_train_predict=enet_model.predict(x_train_scaled)
y_test_predict=enet_model.predict(x_test_scaled)
```

In [308…
```
print("Accuracy Scores for Elastic Net Regression (L1+L2 Regularization) model on raw data")
mse=mean_squared_error(y_test,y_test_predict)
r2_Train=r2_score(y_train,y_train_predict)
r2_Test=r2_score(y_test,y_test_predict)
print("Mean Squared Error",mse)
print("R-Squared Score(Train)", r2_Train)
print("R-Squared Score(Test)", r2_Test)
print("**************" * 7)
```

```
Accuracy Scores for Elastic Net Regression (L1+L2 Regularization) model on raw data
Mean Squared Error 27703.649855463536
R-Squared Score(Train) 0.9301210922887598
R-Squared Score(Test) 0.9522394617045651
**************************************************************************************************
```

In [38]:
```
# Decision Tree regression
```

In [262…
```
dtree_model=DecisionTreeRegressor (max_depth=6)
dtree_model.fit(x_train_scaled,y_train)
```

```
y_train_predict=dtree_model.predict(x_train_scaled)
y_test_predict=dtree_model.predict(x_test_scaled)
```

In [264…

```
print("Accuracy Scores for Decision Tree model on raw data")
mse=mean_squared_error(y_test,y_test_predict)
r2_Train=r2_score(y_train,y_train_predict)
r2_Test=r2_score(y_test,y_test_predict)
print("Mean Squared Error",mse)
print("R-Squared Score(Train)", r2_Train)
print("R-Squared Score(Test)", r2_Test)
print("***************" * 7)
```

```
Accuracy Scores for Decision Tree model on raw data
Mean Squared Error 9254.11889239823
R-Squared Score(Train) 0.9922541190051953
R-Squared Score(Test) 0.9840460841059999
*****************************************************************************************************
```

In [ ]:

```
Random Forest regression
```

In [266…

```
rf_model=RandomForestRegressor (n_estimators=500,random_state=1,max_depth=6)
rf_model.fit(x_train_scaled,y_train)
y_train_predict=rf_model.predict(x_train_scaled)
y_test_predict=rf_model.predict(x_test_scaled)
```

In [268…

```
print("Accuracy Scores for Random Forest model on raw data")
mse=mean_squared_error(y_test,y_test_predict)
r2_Train=r2_score(y_train,y_train_predict)
r2_Test=r2_score(y_test,y_test_predict)
print("Mean Squared Error",mse)
print("R-Squared Score(Train)", r2_Train)
print("R-Squared Score(Test)", r2_Test)
print("***************" * 7)
```

```
Accuracy Scores for Random Forest model on raw data
Mean Squared Error 11183.095289741332
R-Squared Score(Train) 0.9887946476384629
R-Squared Score(Test) 0.9807205673752819
*****************************************************************************************************
```

In [ ]:

```
# Gradient Boosting Regression model
```

In [270…

```
gb_model=GradientBoostingRegressor (n_estimators=100,random_state=123,max_depth=3)
gb_model.fit(x_train_scaled,y_train)
y_train_predict=gb_model.predict(x_train_scaled)
y_test_predict=gb_model.predict(x_test_scaled)
```

In [272…

```
print("Accuracy Scores for Gradient Boosting model on raw data")
mse=mean_squared_error(y_test,y_test_predict)
r2_Train=r2_score(y_train,y_train_predict)
r2_Test=r2_score(y_test,y_test_predict)
print("Mean Squared Error",mse)
print("R-Squared Score(Train)", r2_Train)
print("R-Squared Score(Test)", r2_Test)
print("***************" * 7)
```

```
Accuracy Scores for Gradient Boosting model on raw data
Mean Squared Error 5247.283128217644
R-Squared Score(Train) 0.9990250076507815
R-Squared Score(Test) 0.9909537888292793
*****************************************************************************************************
```

In [ ]:

```
XGBoost Regression model
```

In [274…

```
xgb_model=xgb.XGBRegressor(random_state = 111, max_depth = 2)
xgb_model.fit(x_train_scaled,y_train)
y_train_predict=xgb_model.predict(x_train_scaled)
y_test_predict=xgb_model.predict(x_test_scaled)
```

In [276…

```
print("Accuracy Scores for XGBoost Regression model on raw data")
mse=mean_squared_error(y_test,y_test_predict)
r2_Train=r2_score(y_train,y_train_predict)
r2_Test=r2_score(y_test,y_test_predict)
print("Mean Squared Error",mse)
print("R-Squared Score(Train)", r2_Train)
print("R-Squared Score(Test)", r2_Test)
print("***************" * 7)
```

```
Accuracy Scores for XGBoost Regression model on raw data
Mean Squared Error 4020.9772702032433
R-Squared Score(Train) 0.9986622437283539
R-Squared Score(Test) 0.9930679156031587
*****************************************************************************************************
```

In [278…
```
# Support Vector Regression model - Linear kernel
svr_model=SVR(kernel = 'linear')
svr_model.fit(x_train_scaled,y_train)
y_train_predict=svr_model.predict(x_train_scaled)
y_test_predict=svr_model.predict(x_test_scaled)
```

In [282…
```
print("Accuracy Scores for Support Vector Regression  model on raw data")
mse=mean_squared_error(y_test,y_test_predict)
r2_Train=r2_score(y_train,y_train_predict)
r2_Test=r2_score(y_test,y_test_predict)
print("Mean Squared Error",mse)
print("R-Squared Score(Train)", r2_Train)
print("R-Squared Score(Test)", r2_Test)
print("**************" * 7)
```

```
Accuracy Scores for Support Vector Regression  model on raw data
Mean Squared Error 109565.38598021647
R-Squared Score(Train) 0.7526777981480327
R-Squared Score(Test) 0.811111465808169
**************************************************************************************************
```

In [286…
```
import matplotlib.pyplot as plt
# Plot Actual vs Predicted for Test Data
plt.scatter(y_test, y_test_predict, color='blue', label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', label='Ideal Prediction (y = x)')
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual vs Predicted Prices (Test Set)")
plt.legend()
plt.show()
```


# Gradient Boosting Model appears to be best Model

In [ ]:
```
Saving model for deployment
```

In [64]:
```
final_model = rf_model
filename = 'Mobile price prediction.sav'
pickle.dump(final_model, open(filename, 'wb'))
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js