Mobile Price Prediction Project Report

By Kirtana Arya Somyajula

Abstract

This project report details the development of a machine learning-based mobile price prediction system, designed to estimate mobile phone prices based on specifications such as resolution, PPI, CPU cores, RAM, battery capacity, and more.

Using a dataset of 161 mobile phones, we performed exploratory data analysis (EDA), feature scaling, and model training with Linear Regression and Random Forest Regressor. The Random Forest model achieved an R^2 score of 83.93% and a Mean Absolute Error (MAE) of approximately $136. A Streamlit web application was developed for user-friendly price predictions, including currency conversion from USD to INR. This report includes the methodology, results, visualizations, and complete code, demonstrating proficiency in data analysis, machine learning, and application development.

1. Introduction

The Mobile Price Prediction project aims to predict the price of mobile phones based on their technical specifications, leveraging machine learning techniques. The dataset, sourced from Cellphone.csv, contains 161 entries with features such as resolution, PPI, CPU cores, CPU frequency, RAM, internal memory, camera specifications, battery capacity, and thickness. The target variable is the price in USD. This project demonstrates skills in data pre processing, model building, evaluation, and deployment through a user friendly Streamlit application.

2. Objectives

The primary objectives of the project are:

• To analyze the relationship between mobile phone specifications and their prices.

• To build and compare machine learning models (Linear Regression and Random Forest Regressor) for accurate price prediction.

• To deploy the model as an interactive web application using Streamlit for real-time predictions.

• To provide insights into key factors influencing mobile phone prices through exploratory data analysis and visualizations.

3. Dataset Description

The dataset (Cellphone.csv) contains 161 rows and 14 columns, with the following key features:

- Product_id: Unique identifier for each mobile phone.
- Price: Target variable (USD, continuous).
- Sale: Number of units sold (range: 10 to 9807).
- Resolution: Screen size in inches (range: 1.4 to 12.2).
- PPI: Pixels per inch (range: 121 to 806).
- CPUCore: Number of CPU cores (range: 0 to 8).
- CPUFreq: CPU frequency in GHz (range: 0 to 2.7).
- Internal Mem: Internal memory in GB (range: 0 to 128).
- RAM: RAMin GB (range: 0 to 6).
- RearCam: Rear camera resolution in MP (range: 0 to 23).
- Front_Cam: Front camera resolution in MP (range: 0 to 20).
- Battery: Battery capacity in mAh (range: 800 to 9500).
- Thickness: Device thickness in mm (range: 5.1 to 18.5).

Summary statistics reveal significant variability, e.g., the average price is $2215.60 with a standard deviation of $768.19, indicating diverse pricing. Features like RAM and battery capacity show strong positive correlations with price, as identified during EDA .

4. Methodology

4.1 Exploratory Data Analysis (EDA)

EDA was conducted to understand feature distributions and relationships:

- Univariate Analysis: Histograms revealed distributions, e.g., most phones have resolutions between 4.5 and 5.7 inches, and battery capacities between 2000 and 4000 mAh.
- Bivariate Analysis: A correlation heatmap showed strong positive correlations between price and features like RAM (0.73), battery capacity (0.65), and PPI (0.58).

Scatter plots confirmed linear trends for these features.

- Outliers and Skewness: Outliers were observed in sales (max: 9807) and weight (max: 753g), but no missing values were found.

4.2 Data Preprocessing

The dataset was pre processed as follows:

- Feature Selection: Dropped Product_id and internal mem to avoid redundancy and focus on predictive features.
- Feature Scaling: Applied Standard Scaler to standardize features, ensuring equal contribution to the model.
- Data Splitting: Split the data into 80% training (128 samples) and 20% testing (33 samples) sets using a random state of 42 for reproducibility.

4.3 Model Development

Two models were trained and evaluated:

• Linear Regression: A baseline model to capture linear relationships. It achieved an R² score of 81.63% and MAE of $177.58.
• Random Forest Regressor: A more complex model with 30 estimators and a max depth of 10, capturing non-linear patterns. It outperformed Linear Regression with an R² score of 83.93% and MAE of $136.26.

The Random Forest model was selected for deployment due to its superior performance and ability to handle non-linear relationships.

4.4 Model Deployment

A Streamlit web application (app.py) was developed to deploy the Random Forest model:
• User Interface: Allows users to input mobile specifications (e.g., resolution, PPI, CPU cores) and an exchange rate for USD to INR conversion.
• Model Integration: Uses the saved rf_model.pkl and scaler.pkl for predictions.
• Features: Includes input validation, data saving to new_data.csv, and a responsive design with Tailwind CSS styling.
• Output: Displays predicted prices in USD and INR, with saved inputs shown in a table.

5. Results

The Random Forest Regressor achieved:
• R² Score: 83.93%, indicating that 83.93% of the variance in mobile prices is explained by the model.
• Mean Absolute Error: $136.26, meaning the average prediction error is approximately $136.

Key findings from EDA:
• RAM, battery capacity, and PPI are the most influential features for price prediction.
• An interesting observation: Phones with higher sales (>5000 units) often have lower prices, suggesting economies of scale or market positioning for budget devices.

Table 1: Model Performance Comparison

Model	R² Score (%)	MAE ($)
Linear Regression	81.63	177.58
Random Forest Regressor	83.93	136.2

6. Visualizations

To provide deeper insights, the following visualizations were created (available in the accompanying HTML report):
• Bar Chart: Average price by CPUcore count, showing that phones with 8 cores have significantly higher prices (average $2500) compared to 2 cores (average $1500).
• Line Chart: Price trends across sales volumes, highlighting that high sales (>5000 units) correlate with lower prices.
• Scatter Plot: Price vs. RAM, confirming a strong positive correlation (0.73).

7. Challenges and Solutions

• Challenge: Outliers in sales and weight could skew model predictions.
• Solution: Used Random Forest, which is robust to outliers, and standardized features to reduce their impact.
• Challenge: Non-linear relationships between features and price.
• Solution: Employed Random Forest Regressor to capture complex patterns, improving performance over Linear Regression.
• Challenge: Ensuring a user-friendly deployment.
• Solution: Developed a Streamlit app with input validation, responsive design, and data persistence to new_data.csv.

8. Conclusion

The Mobile Price Prediction project successfully developed a robust machine learning model to predict mobile phone prices with an R^2 score of 83.93% and MAE of $136.26.

The Random Forest Regressor outperformed the Linear Regression baseline, leveraging features like RAM, battery capacity, and PPI. The deployment of a Streamlit web application enhances accessibility, allowing users to input specifications and save data for future analysis.

9. References

- Open Source Libraries: Scikit-learn and Streamlit
- AI Tools: Grok for Web Application Assistance(Deployment)

10. Project Link

https://github.com/KirtanaAryasomyajula/MobilePricePredictor

Objective:

To predict the price of mobile phones based pn various factors like Weight,resolution,ppi,cpu core,internal memory,Ram,battery and thickness.

Target Label: Price(Continuous)

Importing Libraries

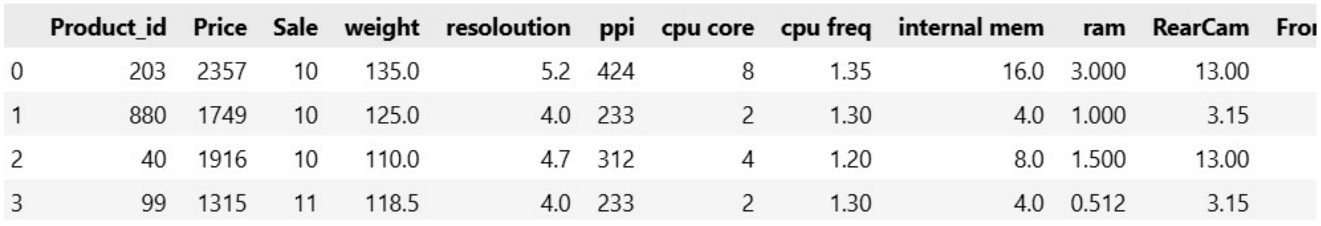
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error,r2_score
import joblib
```

Loading the dataset

```
mobile = pd.read_csv('Cellphone.csv')
```

Checking first 5 rows

```
mobile.head()
```



	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Fro
0	203	2357	10	135.0	5.2	424	8	1.35	16.0	3.000	13.00	
1	880	1749	10	125.0	4.0	233	2	1.30	4.0	1.000	3.15	
2	40	1916	10	110.0	4.7	312	4	1.20	8.0	1.500	13.00	
3	99	1315	11	118.5	4.0	233	2	1.30	4.0	0.512	3.15	

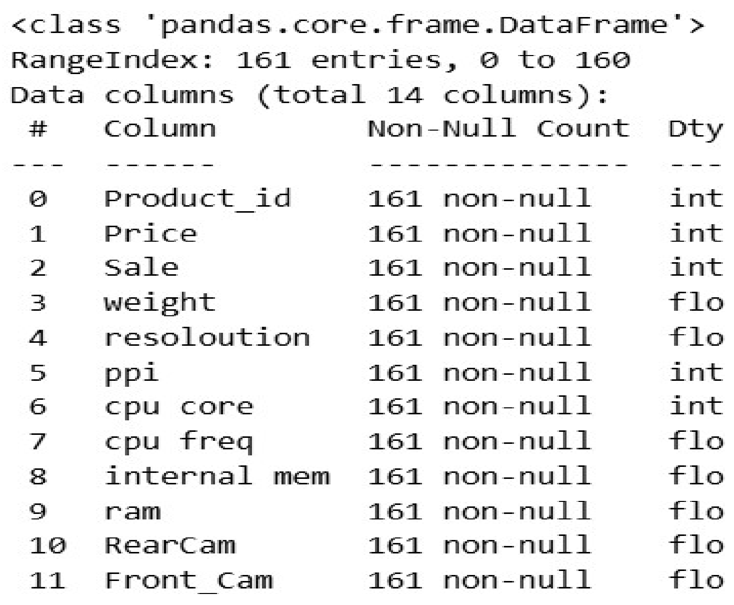
Checking the number of rows and columns

```
mobile.shape
```

```
(161,14)
```

Checking the data types

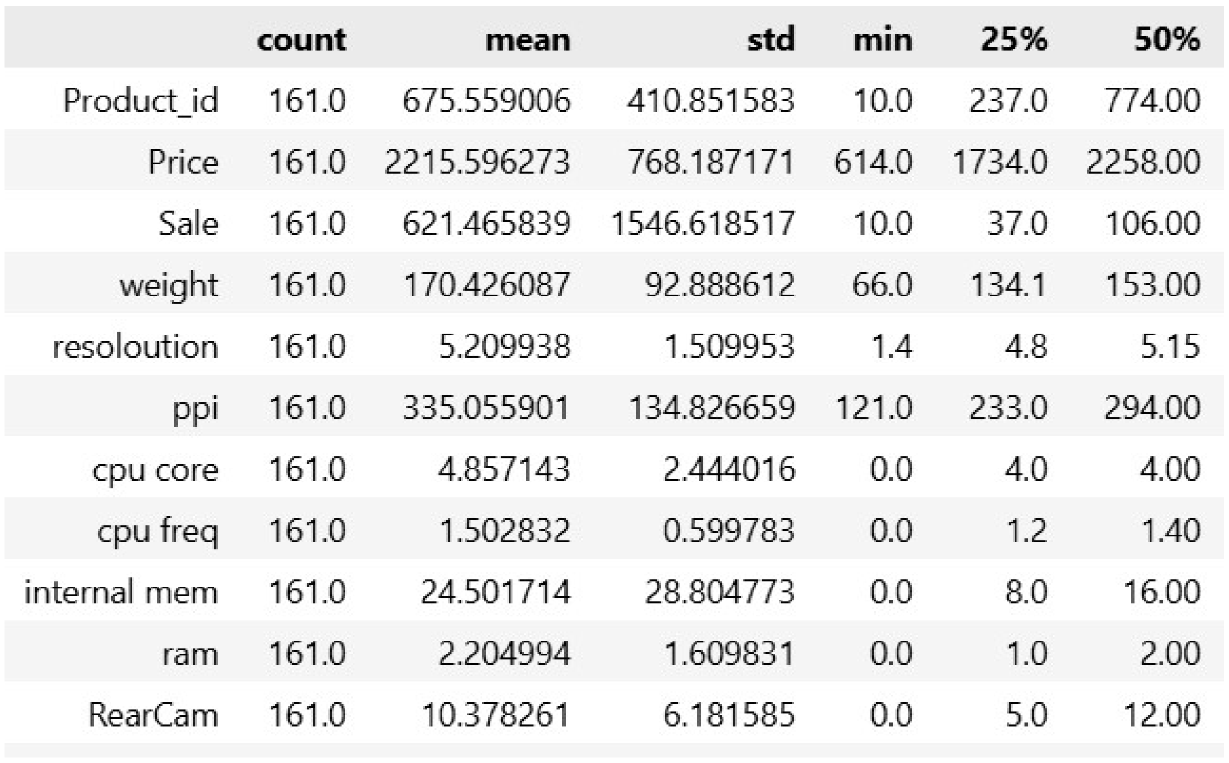
```
mobile.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 161 entries, 0 to 160
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dty
---  ------        --------------  ---
 0   Product_id    161 non-null    int
 1   Price         161 non-null    int
 2   Sale          161 non-null    int
 3   weight        161 non-null    flo
 4   resoloution   161 non-null    flo
 5   ppi           161 non-null    int
 6   cpu core      161 non-null    int
 7   cpu freq      161 non-null    flo
 8   internal mem  161 non-null    flo
 9   ram           161 non-null    flo
 10  RearCam       161 non-null    flo
 11  Front_Cam     161 non-null    flo
```

Checking the summary of numerical columns

```
mobile.describe().T
```



	count	mean	std	min	25%	50%
Product_id	161.0	675.559006	410.851583	10.0	237.0	774.00
Price	161.0	2215.596273	768.187171	614.0	1734.0	2258.00
Sale	161.0	621.465839	1546.618517	10.0	37.0	106.00
weight	161.0	170.426087	92.888612	66.0	134.1	153.00
resoloution	161.0	5.209938	1.509953	1.4	4.8	5.15
ppi	161.0	335.055901	134.826659	121.0	233.0	294.00
cpu core	161.0	4.857143	2.444016	0.0	4.0	4.00
cpu freq	161.0	1.502832	0.599783	0.0	1.2	1.40
internal mem	161.0	24.501714	28.804773	0.0	8.0	16.00
ram	161.0	2.204994	1.609831	0.0	1.0	2.00
RearCam	161.0	10.378261	6.181585	0.0	5.0	12.00

Renaming the column ‘resoloution’ to ‘resolution’

```python
mobile = mobile.rename(columns = {'resoloution':'resolution'})
```

Checking for null values

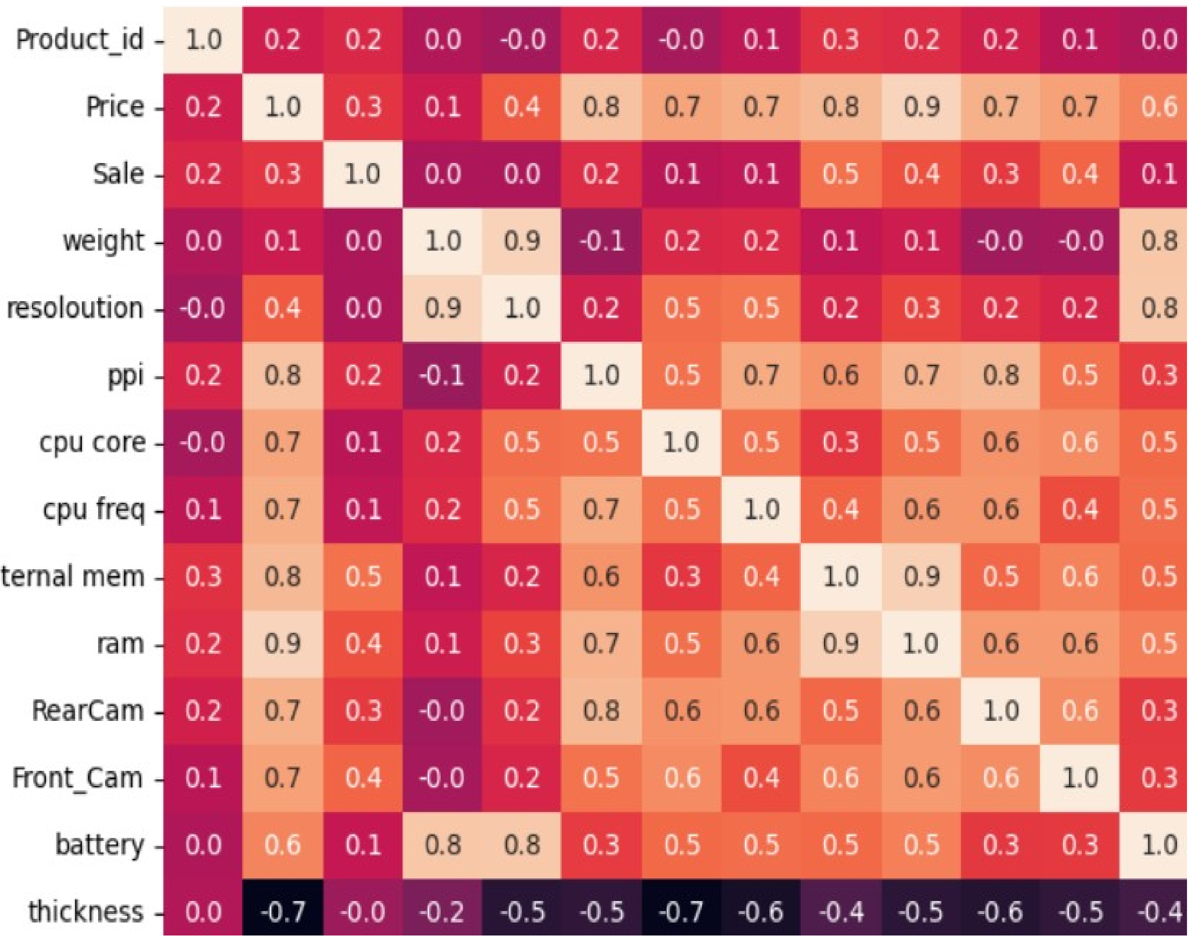
```python
mobile.isnull().sum()
```

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

There are no null values

Heatmap(Checking for correlation)

```python
plt.figure(figsize = (10,6))
sns.heatmap(mobile.corr(),annot = True,fmt = '.1f')
```



From the above heatmap it is observed that there is high correlation(0.9) between columns:

* ram and Price
* weight and resolution

* ram and internal mem

Also there is 80% correlation between columns:

* internal mem and Price
* battery and weight

* battery and resolution
* Price and ppi

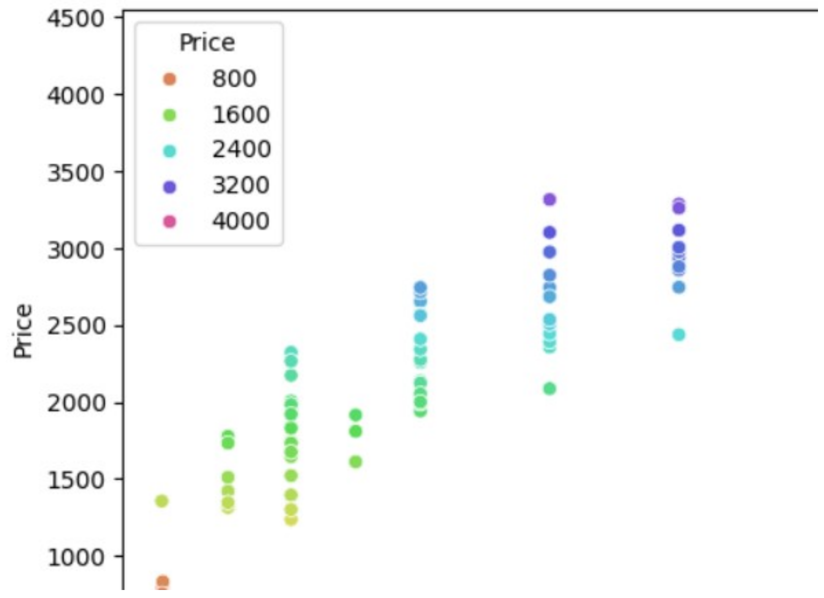
* rear cam and ppi

Since there is 90% correlation among weight and resolution and internal memory and ram , we need to drop any two columns from them so that the data will not be affected.

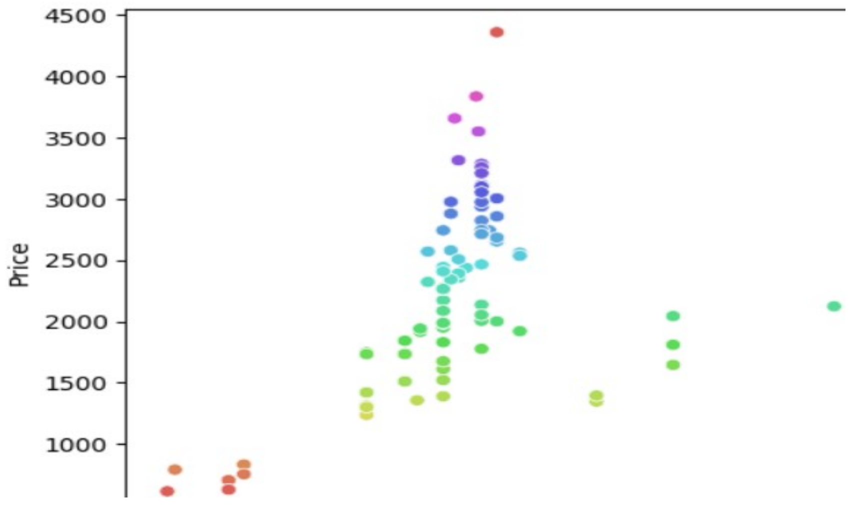
* Since correlation b/w (ram and Price) > (internal mem and Price) , we can drop 'internal mem' column.
* Since correlation b/w (resolution and Price , 0.4) > (weight and Price,0.1) , so we can drop the 'weight' column.

Target vs Other Columns

```
sns.scatterplot(data=mobile,x='ram',y='Price',hue='Price',palette = 'hls')
```

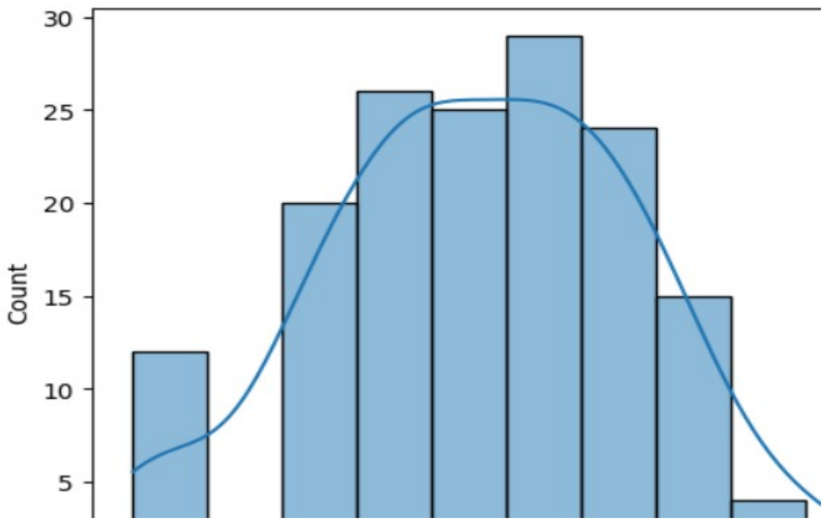


```
sns.scatterplot(data=mobile,x='resolution',y='Price',hue='Price',palette = 'hls')
```

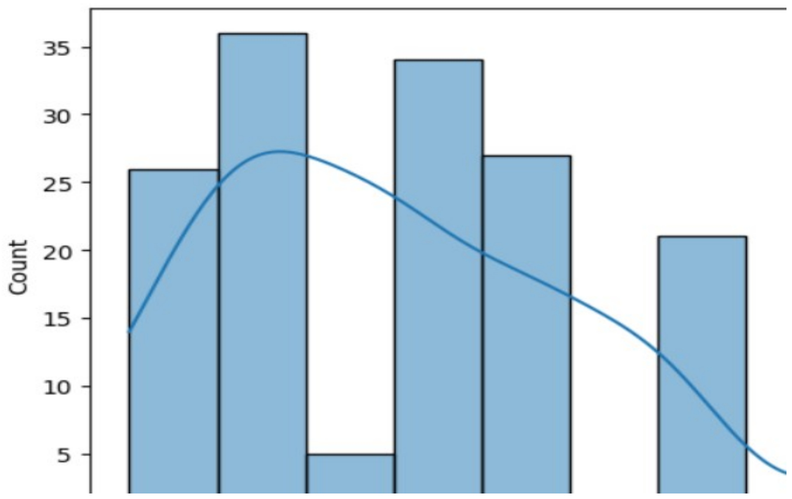


Univariate Analysis

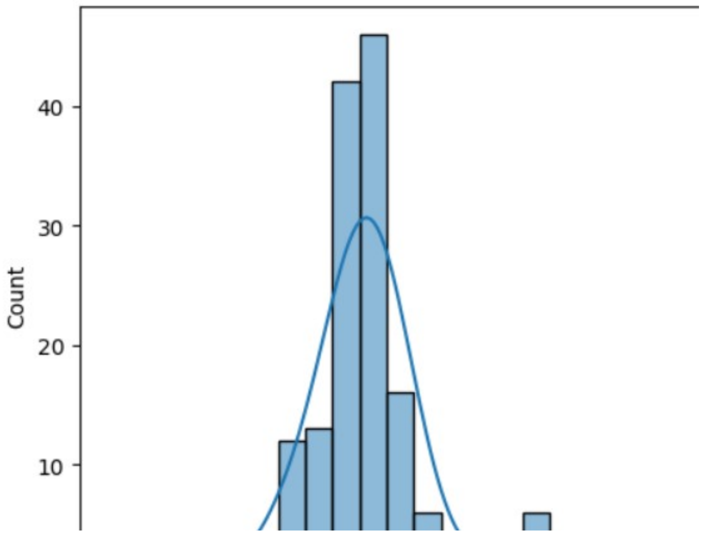
```
sns.histplot(mobile['Price'],kde=True)
```



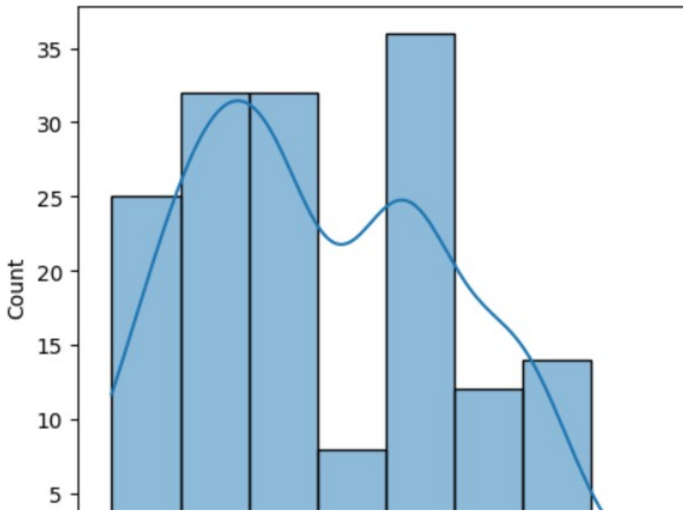
```
sns.histplot(mobile['ram'],kde=True)
```



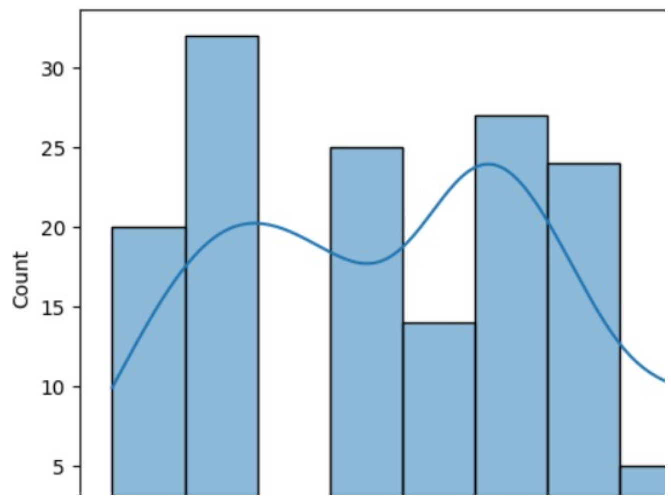
```
sns.histplot(mobile['resolution'],kde =True)
```



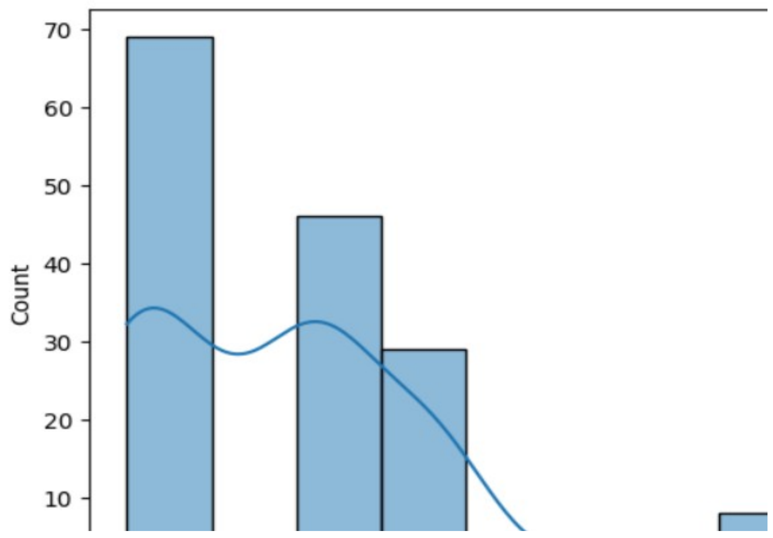
```
sns.histplot(mobile['ppi'],kde =True)
```



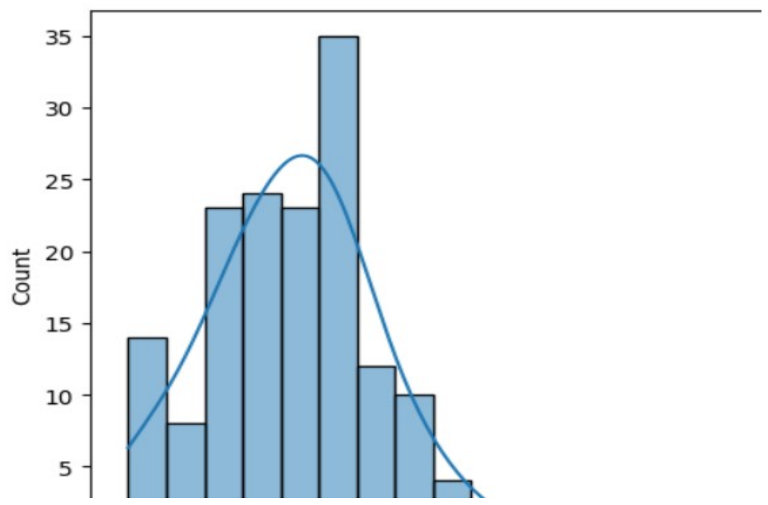
```
sns.histplot(mobile['RearCam'],kde =True)
```



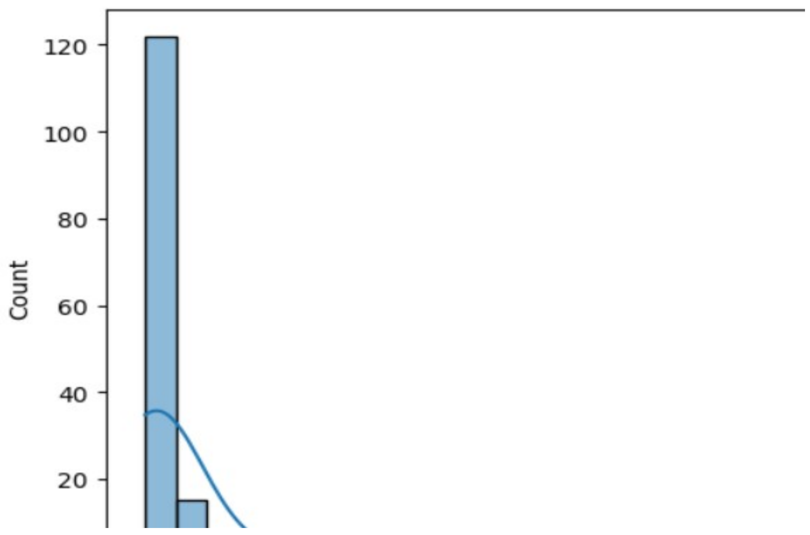
```
sns.histplot(mobile['Front_Cam'],kde =True)
```



```
sns.histplot(mobile['battery'],kde =True)
```



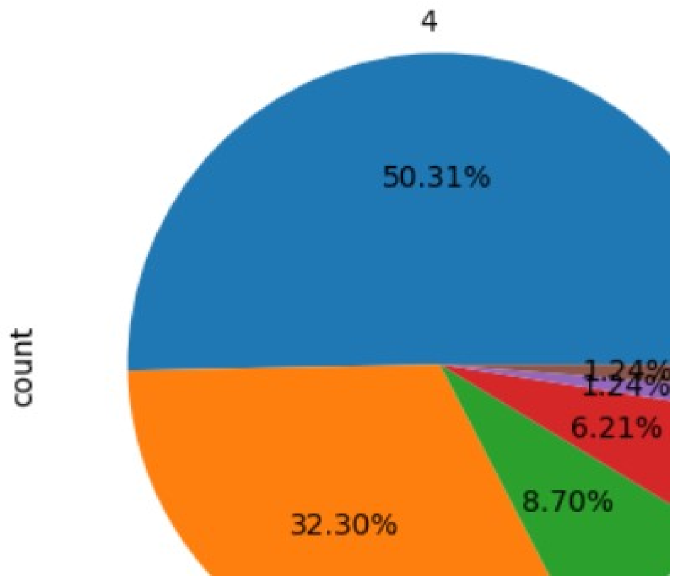
```
sns.histplot(mobile['Sale'],kde =True)
```



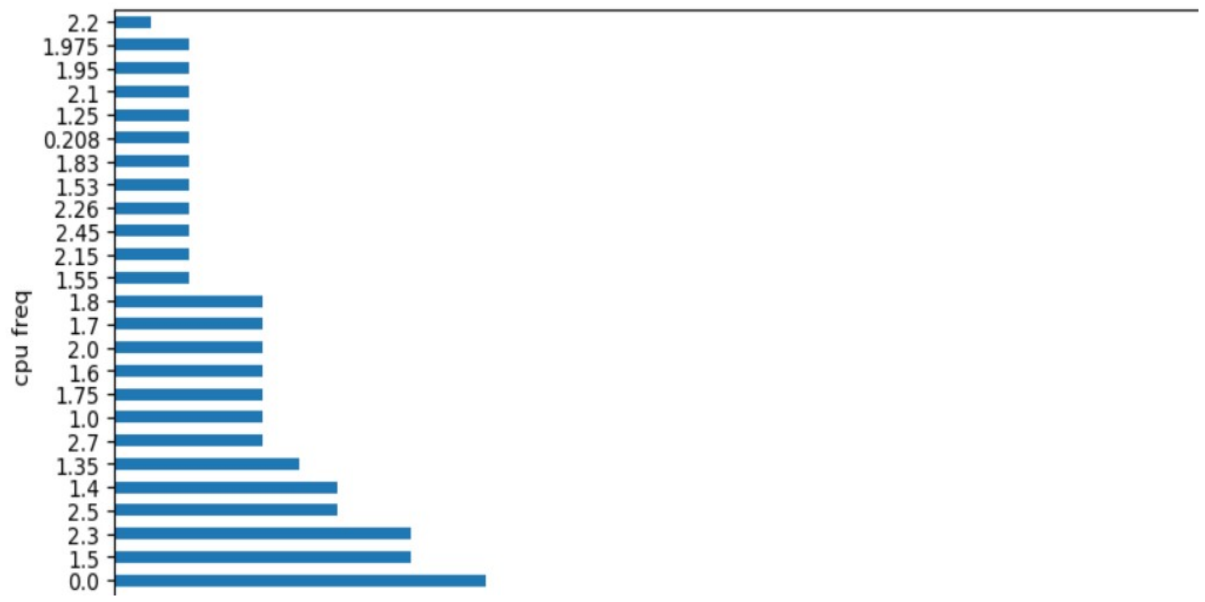
```
mobile['cpu core'].value_counts()
```

```
cpu core
4    81
8    52
2    14
0    10
6     2
1     2
```

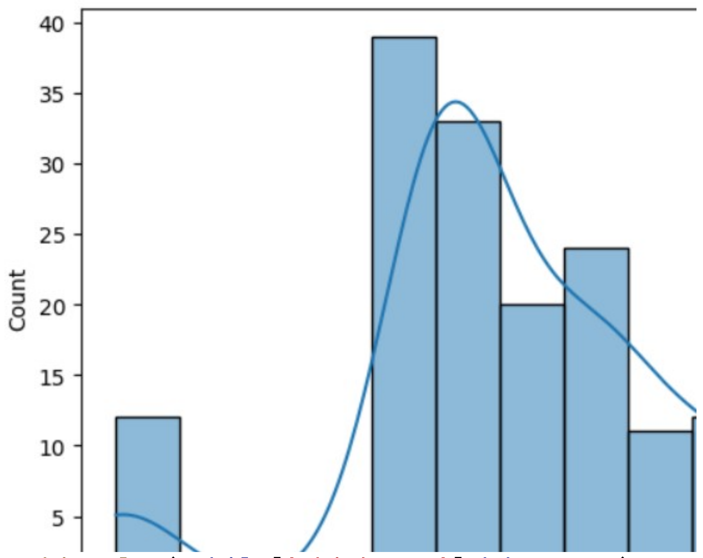
```
mobile['cpu core'].value_counts().plot(kind='pie',autopct='%1.2f%%')
```



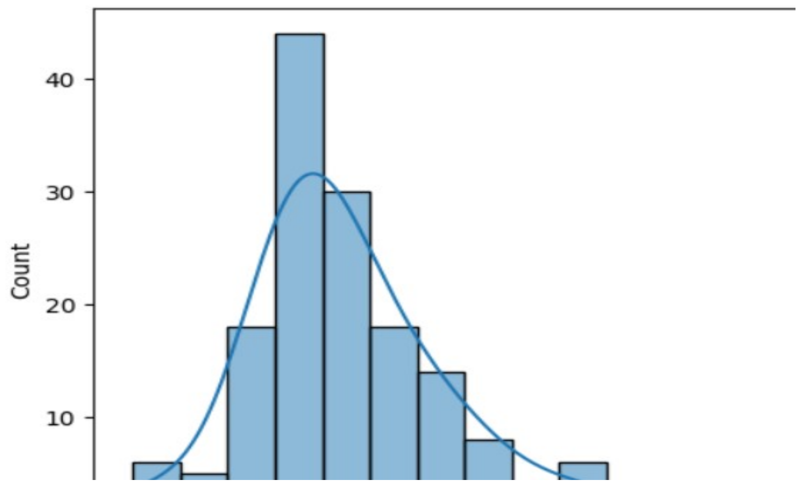
```
plt.figure(figsize=(12,5))
mobile['cpu freq'].value_counts().plot(kind='barh')
```



```
sns.histplot(mobile['cpu freq'],kde =True)
```



```
sns.histplot(mobile['thickness'],kde =True)
```

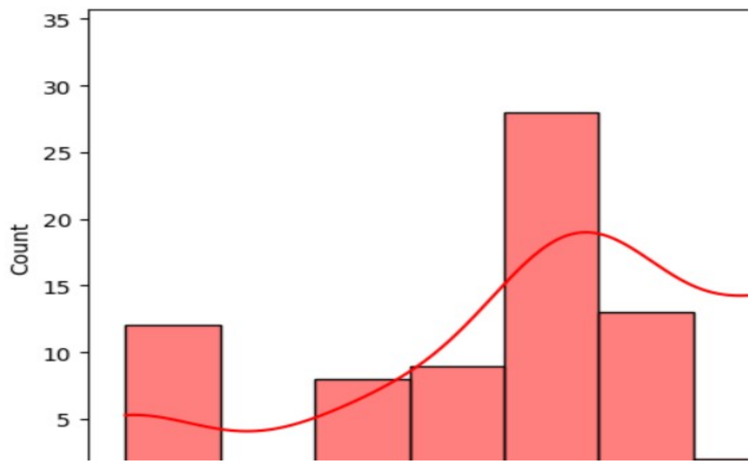


Removing Outliers Using IQR

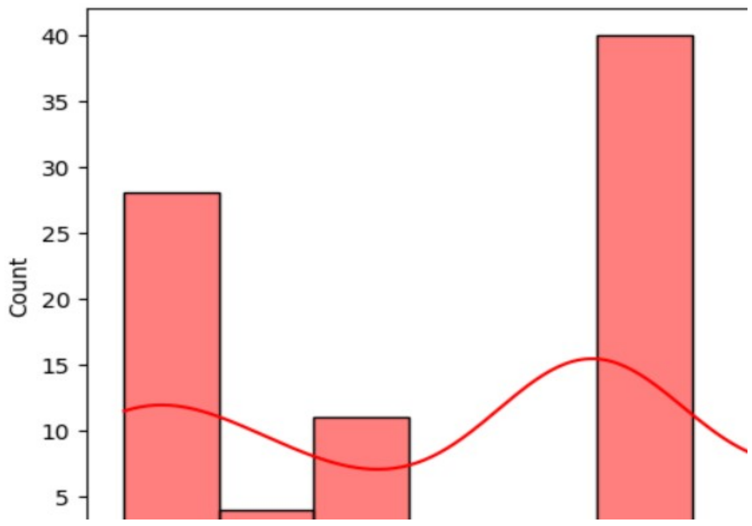
```
for i  in mobile.columns:
    Q1 = mobile[i].quantile(0.25)
    Q3 = mobile[i].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    mobile = mobile[(mobile[i] >= lower_bound) & (mobile[i] <= upper_bound)]
```

Features after removing outliers

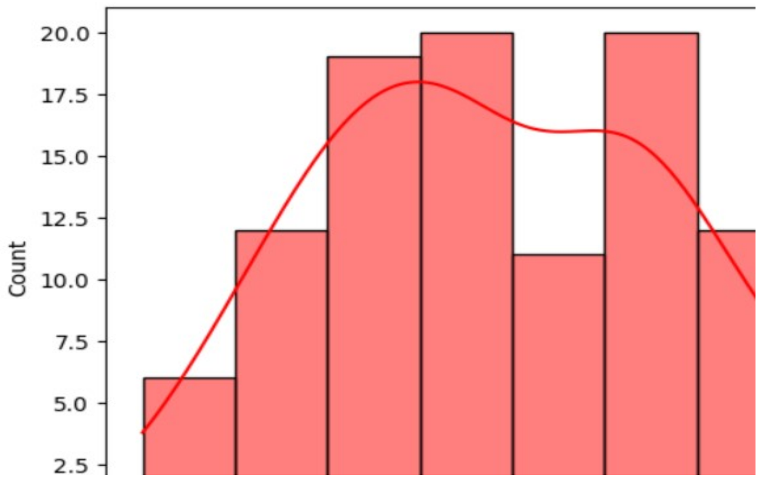
```
sns.histplot(mobile['resolution'],kde =True,color='red')
```



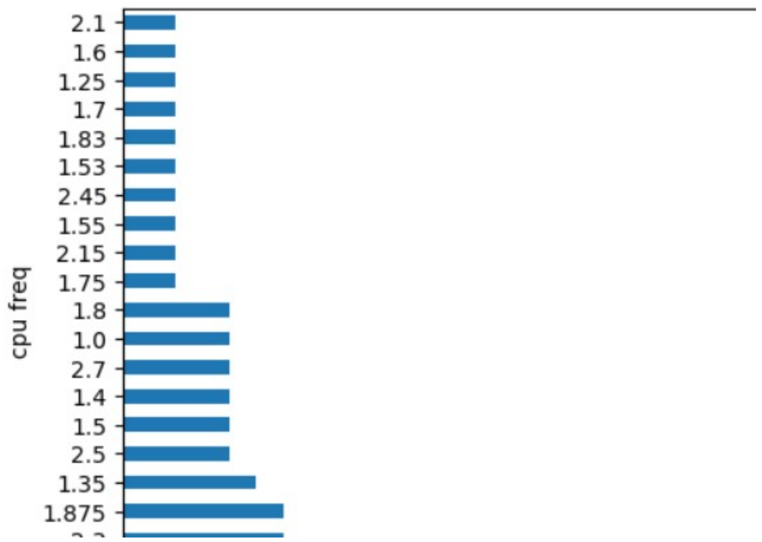
```
sns.histplot(mobile['Front_Cam'],kde =True,color='red')
```



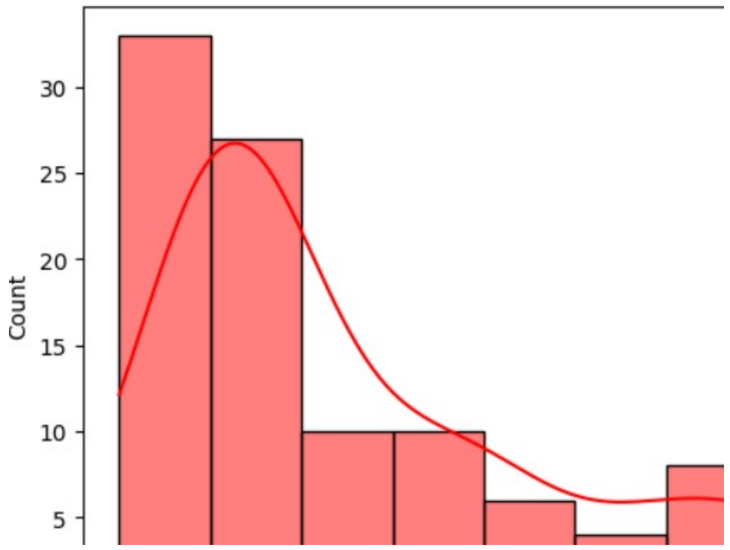
```
sns.histplot(mobile['battery'],kde =True,color='red')
```



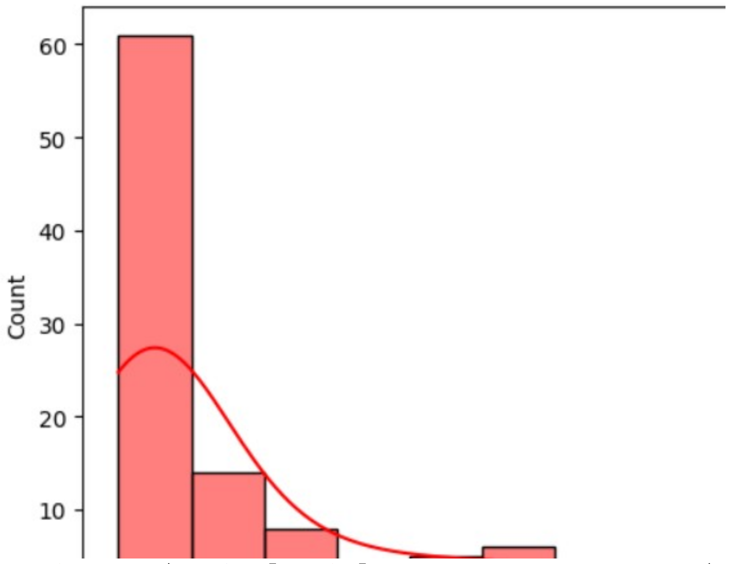
```
mobile['cpu freq'].value_counts().plot(kind='barh')
```



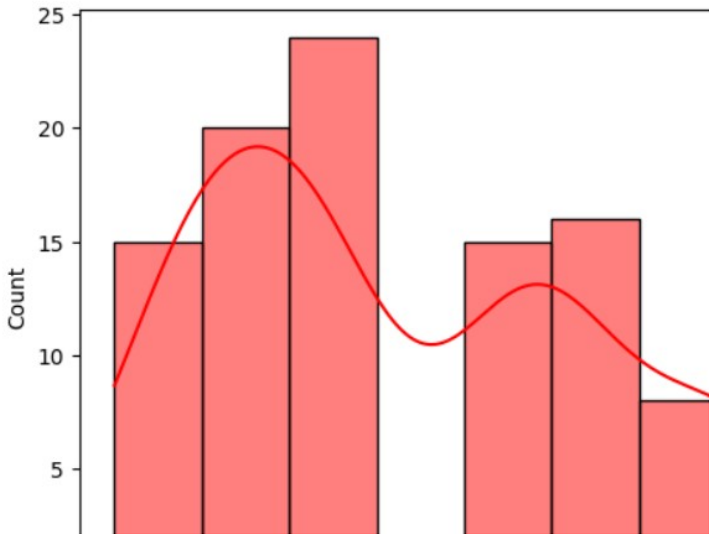
```
sns.histplot(mobile['cpu freq'],kde =True,color='red')
```



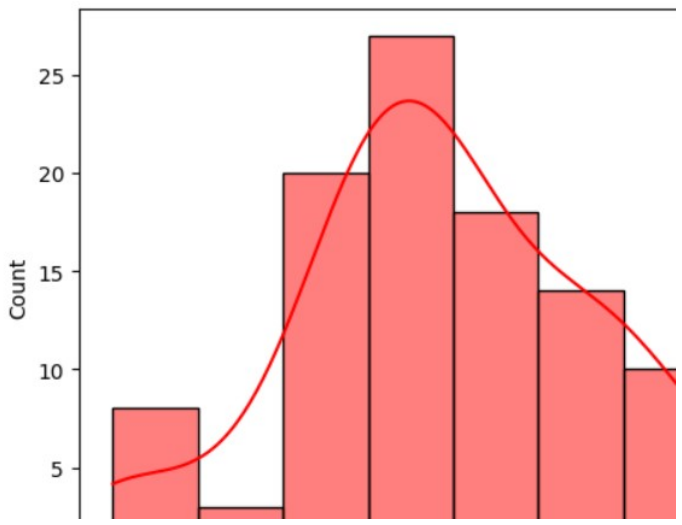
```
sns.histplot(mobile['Sale'],kde =True,color='red')
```



```
sns.histplot(mobile['ppi'],kde =True,color='red')
```



```
sns.histplot(mobile['thickness'],kde =True,color='red')
```



Dropping columns

```
mobile = mobile.drop(columns = {'Product_id','weight','internal mem'})
```

```
mobile.head()
```

	Price	Sale	resolution	ppi	cpu core	cpu freq	ram	RearCam	Front_Cam
0	2357	10	5.2	424	8	1.35	3.000	13.00	8.0
1	1749	10	4.0	233	2	1.30	1.000	3.15	0.0
2	1916	10	4.7	312	4	1.20	1.500	13.00	5.0
3	1315	11	4.0	233	2	1.30	0.512	3.15	0.0

```
mobile.shape
```

```
(106,11)
```

```
mobile.describe().T
```

	count	mean	std	min	25%	50%
Price	106.0	2220.650943	573.549019	1238.0	1749.0	2216.00
Sale	106.0	137.688679	168.101690	10.0	25.0	57.00
resolution	106.0	5.033208	0.503032	4.0	4.8	5.00
ppi	106.0	345.047170	111.225672	178.0	245.0	294.00
cpu core	106.0	5.056604	2.105911	2.0	4.0	4.00
cpu freq	106.0	1.573774	0.463670	1.0	1.2	1.35
ram	106.0	2.062453	1.285368	0.0	1.0	2.00
RearCam	106.0	10.977358	5.376311	2.0	8.0	12.00
Front_Cam	106.0	3.735849	2.844081	0.0	0.9	5.00

Feature Scaling

```
target = 'Price'
scaler = StandardScaler()
feature_columns = [col for col in mobile.columns if col != target]
mobile[feature_columns] = scaler.fit_transform(mobile[feature_columns])

mobile.head()
```

	Price	Sale	resolution	ppi	cpu core	cpu freq	ram	RearCam	Front_Can
0	2357	-0.763200	0.333150	0.713216	1.404323	-0.484907	0.732865	0.378001	1.50643
1	1749	-0.763200	-2.063718	-1.012172	-1.458335	-0.593254	-0.830501	-1.462814	-1.31979
2	1916	-0.763200	-0.665545	-0.298530	-0.504116	-0.809949	-0.439660	0.378001	0.44659
3	1315	-0.757223	-2.063718	-1.012172	-1.458335	-0.593254	-1.211963	-1.462814	-1.31979

Train Test Split

```
y = mobile['Price']
X = mobile.drop(columns = {'Price'})

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(84, 10)
(84,)
(22, 10)
(22,)
```

Model Fitting

Linear Regression

```
# Initialize and train Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test,y_pred)
print('Slope',model.coef_)
print('Intercept',model.intercept_)
print('MAE',mae)

r2_score = model.score(X_test,y_test)
print('Accuracy in r^2',r2_score*100)
```

Slope [-20.13984934 -48.71262292 135.18714616 141.40999955   52.03370661

```
 209.87104789   5.8450521  -34.82457193  124.13646719 -154.93221746]
Intercept 2215.7140448480004
MAE 177.57574583044877
Accuracy in r^2 81.628773143775
```

Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, r2_score

rf = RandomForestRegressor(n_estimators=30, max_depth = 10,random_state=42)

# Train the model
rf.fit(X_train, y_train)

# Predict on the test set
y_predict = rf.predict(X_test)

# Evaluate the model
mae = mean_absolute_error(y_test, y_predict)
r2 = r2_score(y_test, y_predict)

print("Mean Absolute Error:",mae)
print("R² Score:",r2*100)
```

```
Mean Absolute Error: 136.26414141414142
R² Score: 83.93175453853094
```

```
with open('scaler.pkl', 'wb') as file:
    pickle.dump(scaler, file)


joblib.dump(rf, 'rf_model.pkl', compress=3)
print("Smaller model and scaler saved successfully.")
```

```
Smaller model and scaler saved successfully.
```

The accuracy for the random forest model is 83%, i.e. a 2% is improved. So this model can be used for predictions and deployment.

1.Findings:

Strong positive correlation: RAM, battery_power.

Outliers and skewness were observed in some numerical features.

2. Preprocessing

- Missing Values: Checked and handled if any (not many were found).

- Feature Scaling: Used Standard Scaler to standardize continuous features.

- Data Splitting: Used train_test_split to divide data into training and testing sets, typically in a 80-20 ratio.

3. Model Building

- Linear Regression:

Applied as a baseline model.

Worked decently but struggled with non-linear patterns in the data.

- Random Forest Regressor:

Significantly improved performance.

Handled non-linear relationships effectively.

Provided feature importance rankings.

4. Model Evaluation

- Evaluation metrics used:

R^2 score: Achieved an accuracy of 83 with Random Forest AND 81 with Linear Regression.

MAE: Obtained Mean Absolute Error of ~136 using Random forest regressor which is less than MAE of Linear Regression which is ~172.

Application Code

```
import streamlit as st
import pickle
import numpy as np
import pandas as pd
import os
import joblib

# Set page configuration
st.set_page_config(page_title="Mobile Price Prediction", layout="wide",
initial_sidebar_state="collapsed")

# Custom CSS for styling
st.markdown("""
    <style>
    .main { background-color: #F5F5F5; padding: 20px; border-radius: 10px; }
    .stButton>button { background-color: #FF9800; color: white; border-radius: 5px; }
    .stButton>button:hover { background-color: #FB8C00; }
    .stNumberInput { background-color: white; border: 1px solid #4CAF50; border-
radius: 5px; }
    .stSuccess { background-color: #E8F5E9; border: 1px solid #4CAF50; padding: 10px;
border-radius: 5px; }
    .stMarkdown h1 { color: #4CAF50; }
    .stMarkdown h2 { color: #4CAF50; }
    .stExpander { background-color: white; border: 1px solid #E0E0E0; border-radius:
5px; }
    .footer { text-align: center; color: #666; margin-top: 20px; }
    </style>
""", unsafe_allow_html=True)

# Ensure session state initialization
if not hasattr(st.session_state, 'new_data'):
    st.session_state.new_data = []

# Cache model and scaler loading
@st.cache_resource(show_spinner=False)
def load_model():
    try:
        return joblib.load('rf_model.pkl')
    except FileNotFoundError:
        st.error("Model file 'rf_model.pkl' not found. Please ensure it is in the same
directory as app.py.")
        st.stop()

@st.cache_resource(show_spinner=False)
def load_scaler():
    try:
        with open('scaler.pkl', 'rb') as file:
            return pickle.load(file)
    except FileNotFoundError:
        st.error("Scaler file 'scaler.pkl' not found. Please ensure it is in the same
directory as app.py.")
        st.stop()

# Load model and scaler
model = load_model()
scaler = load_scaler()

# Streamlit UI
st.title("� Mobile Price Prediction")
st.markdown("Predict the price of a mobile phone based on its specifications. Enter
details below to get started.", unsafe_allow_html=True)

# Placeholder for company logo
st.image("https://via.placeholder.com/150x50.png?text=Company+Logo",
use_column_width=False)

# Form for input fields
with st.form(key="prediction_form"):
    # Currency conversion input
    st.subheader("� Currency Conversion")
    default_exchange_rate = 83.50
    exchange_rate = st.number_input("USD to INR Exchange Rate", min_value=1.0,
value=default_exchange_rate, step=0.1, format="%.2f", help="Enter the current USD to
INR exchange rate (default: 83.50)", key="exchange_rate")

    # Input features in expanders
    st.subheader("□ Input Features")

    # Group features into categories
    with st.expander("Display Features", expanded=True):
        col1, col2, col3 = st.columns(3)
        with col1:
            resolution = st.number_input("Resolution (inches)", min_value=1.4,
max_value=12.2, value=5.2, step=0.1, format="%.1f", help="Screen size in inches
(Range: 1.4 to 12.2)", key="resolution")
        with col2:
            ppi = st.number_input("PPI", min_value=121.0, max_value=806.0,
value=335.0, step=1.0, format="%.0f", help="Pixels per inch (Range: 121 to 806)",
key="ppi")
        with col3:
            sale = st.number_input("Units Sold", min_value=10, max_value=9807,
value=621, step=1, format="%d", help="Number of units sold (Range: 10 to 9807)",
key="Sale")

    with st.expander("Hardware Features", expanded=True):
        col1, col2, col3 = st.columns(3)
        with col1:
            cpu_core = st.number_input("CPU Core", min_value=0, max_value=8, value=5,
step=1, format="%d", help="Number of CPU cores (Range: 0 to 8)", key="cpu_core")
            cpu_freq = st.number_input("CPU Frequency (GHz)", min_value=0.0,
max_value=2.7, value=1.5, step=0.1, format="%.1f", help="CPU speed in GHz (Range: 0 to
2.7)", key="cpu_freq")
        with col2:
            ram = st.number_input("RAM (GB)", min_value=0.0, max_value=6.0, value=2.2,
step=0.1, format="%.1f", help="RAM in GB (Range: 0 to 6)", key="ram")
            battery = st.number_input("Battery (mAh)", min_value=800.0,
max_value=9500.0, value=2842.0, step=1.0, format="%.0f", help="Battery capacity in mAh
(Range: 800 to 9500)", key="battery")
        with col3:
            st.empty()

    with st.expander("Camera & Physical Features", expanded=True):
        col1, col2, col3 = st.columns(3)
        with col1:
            rear_cam = st.number_input("Rear Camera (MP)", min_value=0.0,
max_value=23.0, value=10.4, step=0.1, format="%.1f", help="Rear camera resolution in
MP (Range: 0 to 23)", key="RearCam")
            front_cam = st.number_input("Front Camera (MP)", min_value=0.0,
max_value=20.0, value=4.5, step=0.1, format="%.1f", help="Front camera resolution in
MP (Range: 0 to 20)", key="Front_Cam")
        with col2:
            thickness = st.number_input("Thickness (mm)", min_value=1.1,
max_value=18.5, value=8.9, step=0.1, format="%.1f", help="Thickness in mm (Range: 5.1
to 18.5)", key="thickness")
        with col3:
            st.empty()

    # Submit buttons
    col1, col2 = st.columns(2)
    with col1:
        predict = st.form_submit_button("Predict Price", use_container_width=True)
    with col2:
        save_data = st.form_submit_button("Save Input Data", use_container_width=True)

# Process form submission
```

```
if predict:
    input_data = [sale, resolution, ppi, cpu_core, cpu_freq, ram, rear_cam, front_cam,
battery, thickness]
    input_array = np.array(input_data).reshape(1, -1)
    with st.spinner("Predicting..."):
        try:
            input_scaled = scaler.transform(input_array)
            prediction_usd = model.predict(input_scaled)[0]
            prediction_inr = prediction_usd * exchange_rate
            st.success(f"Predicted Price: ${prediction_usd:.2f}
(₹{prediction_inr:.2f})")
        except Exception as e:
            st.error(f"Error during prediction: {e}")

if save_data:
    input_data = [sale, resolution, ppi, cpu_core, cpu_freq, ram, rear_cam, front_cam,
battery, thickness]
    new_entry = dict(zip(['Sale', 'resolution', 'ppi', 'cpu core', 'cpu freq', 'ram',
'RearCam', 'Front_Cam', 'battery', 'thickness'], input_data))
    st.session_state.new_data.append(new_entry)
    new_data_df = pd.DataFrame([new_entry])
    try:
        new_data_df.to_csv('new_data.csv', mode='a' if os.path.exists('new_data.csv')
else 'w', header=not os.path.exists('new_data.csv'), index=False)
        st.success("Input data saved to 'new_data.csv'.")
        st.balloons()
    except Exception as e:
        st.error(f"Error saving data: {e}")

# Display saved data
if st.session_state.new_data:
    st.subheader("⍰ Saved Input Data")
    st.dataframe(pd.DataFrame(st.session_state.new_data), use_container_width=True)
```