Documentation: Smartprix Smartphone Data Analysis – Web Scraping, Cleaning, Code, and Insights

A. WEB SCRAPING Project: Extracting Smartphone Data from Smartprix for Data Analysis

Introduction

In this project, I gathered, cleaned, and analyzed data from the smartphone retail website Smartprix using web scraping techniques. Then created a detailed dataset with information such as model, price, operating system, and other specifications of smartphones. Then cleaned data to use to understand market dynamics, customer preferences, and trends in the smartphone market.

i)Understanding the Website

The target website was Smartprix, which lists various smartphone models along with their prices, features, and ratings. I focused on extracting detailed information about each smartphone model listed on the site.

ii)Tools and Libraries Used for Web Scraping

Programming Language: Python

Web Scraping Libraries: Selenium, BeautifulSoup, time.

Data Processing: Pandas, NumPy

Data Storage: html file, CSV files

Web Scraping Process

Inspecting the Website: Used browser developer tools to understand the HTML structure.

Extracting Data:

Step 1: Automating the Web Browser with Selenium(smartprix step-1 selenium scrapping.ipynb**)**

Imported necessary libraries (webdriver and By from Selenium and time).

Set up the Firefox web browser using webdriver.Firefox().

Opened the Smartprix mobiles page.

Applied filter options by clicking specific elements on the webpage.

Continuously clicked the "load more" button until no more new content was loaded.

Handling Dynamic Content: Selenium was used to handle dynamic content and load more products by clicking buttons to ensure all data was captured.

Saved the HTML content of the entire page into a file named smartprix.html.

In [4]:
```
from selenium import webdriver
import time
from selenium.webdriver.common.by import By
```

In [11]:
```
driver = webdriver.Firefox()

driver.get('https://www.smartprix.com/mobiles')
time.sleep(1)

driver.find_element(by = By.XPATH,value = '/html/body/div[1]/main/aside/div/div[5]/div[2]/label[1]/input').click()
time.sleep(1)
driver.find_element(by = By.XPATH,value = '/html/body/div[1]/main/aside/div/div[5]/div[2]/label[2]/input').click()
time.sleep(2)

old_height = driver.execute_script('return document.body.scrollHeight')
while True:
    driver.find_element(by = By.XPATH,value = '/html/body/div[1]/main/div[1]/div[2]/div[3]').click()
    time.sleep(1)

    new_height = driver.execute_script('return document.body.scrollHeight')

    if new_height == old_height:
        break
    old_height = new_height

html = driver.page_source
with open('smartprix.html','w',encoding = 'utf-8') as f:
    f.write(html)
```

Step 2: Parsing HTML with BeautifulSoup(smartprix step-2 BeautifulSoup parser.ipynb**)to extract product details and store them in lists.**

1. Read the content of the smartprix.html file.
2. Used BeautifulSoup to parse the HTML content.

3. Located all the sections containing product information.

```
In [1]: # smartprix step-2 BeautifulSoup parser

        with open('smartprix.html','r',encoding = 'utf-8') as f:
            html = f.read()
```

```
In [18]: from bs4 import BeautifulSoup
         import numpy as np
         import pandas as pd
```

```
In [9]: soup = BeautifulSoup(html,'lxml')
```

```
In [10]: containers = soup.find_all('div',{'class':"sm-product has-tag has-features has-actions"})
```

4. Created empty lists to store details like names, prices, ratings, SIM card information, processor details, RAM, battery capacity, display details, camera specifications, memory card information, and operating system.
5. For each product, extracted the relevant details and appended them to the corresponding lists.
6. Added placeholder values (np.nan) for any missing details.

In [1]:
```
names = []
prices = []
ratings = []
sim = []
processor = []
ram = []
battery = []
display = []
camera = []
card = []
os = []
for i in containers:
    names.append(i.find('h2').text)

    prices.append(i.find('span',{'class':'price'}).text)

    ratings.append(i.find('b').text)


    x = i.find('ul',{'class':"sm-feat specs"}).find_all('li')

    try:
        sim.append(x[0].text)
    except:
        os.append(np.nan)

    try:
        processor.append(x[1].text)
    except:
        processor.append(np.nan)

    try:
        ram.append(x[2].text)
    except:
        ram.append(np.nan)

    try:
        battery.append(x[3].text)
    except:
        battery.append(np.nan)

    try:
        display.append(x[4].text)
    except:
        display.append(np.nan)

    try:
        camera.append(x[5].text)
    except:
        camera.append(np.nan)

    try:
        card.append(x[6].text)
    except:
        card.append(np.nan)



    try:
        os.append(x[7].text)
    except:
        os.append(np.nan)
```

Stored the parsed data in a CSV file using Pandas for easy access and further analysis.

In [76]:
```
df = pd.DataFrame({
    'model':names,
    'price':prices,
    'rating':ratings,
    'sim':sim,
    'processor':processor,
    'ram':ram,
    'battery':battery,
    'display':display,
    'camera':camera,
    'card':card,
    'os':os
})
```

In [78]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1020 entries, 0 to 1019
Data columns (total 11 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   model      1020 non-null   object
 1   price      1020 non-null   object
 2   rating     1020 non-null   object
 3   sim        1020 non-null   object
 4   processor  1020 non-null   object
 5   ram        1020 non-null   object
 6   battery    1020 non-null   object
 7   display    1020 non-null   object
 8   camera     1014 non-null   object
 9   card       965 non-null    object
 10  os         853 non-null    object
dtypes: object(11)
memory usage: 87.8+ KB
```

In [80]:
```
df
```

Out[80]:

	model	price	rating	sim	processor	ram	battery	display	camera	card	os
0	Motorola Moto G34 5G	₹10,999	75	Dual Sim, 3G, 4G, 5G, VoLTE, Vo5G, Wi-Fi	Snapdragon 695, Octa Core, 2.2 GHz Processor	4 GB RAM, 128 GB inbuilt	5000 mAh Battery with 18W Fast Charging	6.5 inches, 1080 x 2400 px, 120 Hz Display wit...	50 MP + 2 MP Dual Rear & 16 MP Front Camera	Memory Card (Hybrid), upto 1 TB	Android v14
1	Samsung Galaxy S24 Ultra	₹1,29,999	95	Dual Sim, 3G, 4G, 5G, VoLTE, Vo5G, Wi-Fi, NFC	Snapdragon 8 Gen3, Octa Core, 3.3 GHz Processor	12 GB RAM, 256 GB inbuilt	5000 mAh Battery with 45W Fast Charging	6.8 inches, 1440 x 3120 px, 120 Hz Display wit...	200 MP Quad Rear & 12 MP Front Camera	Memory Card Not Supported	Android v14
2	OPPO Reno 11	₹29,999	88	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC, IR Bl...	Dimensity 7050, Octa Core, 2.6 GHz Processor	8 GB RAM, 128 GB inbuilt	5000 mAh Battery with 67W Fast Charging	6.7 inches, 1080 x 2412 px, 120 Hz Display wit...	50 MP + 32 MP + 8 MP Triple Rear & 32 MP Front...	Memory Card (Hybrid)	Android v14
3	Xiaomi Redmi Note 13 Pro 5G	₹25,990	84	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, IR Blaster	Snapdragon 7s Gen 2, Octa Core, 2.4 GHz Processor	8 GB RAM, 128 GB inbuilt	5100 mAh Battery with 67W Fast Charging	6.67 inches, 1220 x 2712 px, 120 Hz Display wi...	200 MP + 8 MP + 2 MP Triple Rear & 16 MP Front...	Android v13	No FM Radio
4	Motorola Moto G34 5G (8GB RAM + 128GB)	₹11,999	79	Dual Sim, 3G, 4G, 5G, VoLTE, Vo5G, Wi-Fi	Snapdragon 695, Octa Core, 2.2 GHz Processor	8 GB RAM, 128 GB inbuilt	5000 mAh Battery with 18W Fast Charging	6.5 inches, 1080 x 2400 px, 120 Hz Display wit...	50 MP + 2 MP Dual Rear & 16 MP Front Camera	Memory Card (Hybrid), upto 1 TB	Android v14
...	...	...	...	...	...	...	...	...	...	...	...
1015	Realme C25	₹8,890	65	Dual Sim, 3G, 4G, VoLTE, Wi-Fi	Helio G70, Octa Core, 2 GHz Processor	4 GB RAM, 64 GB inbuilt	6000 mAh Battery with 18W Fast Charging	6.5 inches, 1600 x 720 px Display with Water D...	13 MP + 2 MP + 2 MP Triple Rear & 8 MP Front C...	Memory Card Supported, upto 256 GB	Android v11
1016	Giva G5	₹850	7	Dual Sim	1 MHz Processor	32 MB RAM, 32 MB inbuilt	998 mAh Battery	1.8 inches, 240 x 320 px Display	0.3 MP Rear Camera	No FM Radio	NaN
1017	Realme Narzo 30A (4GB RAM + 64GB)	₹9,999	67	Dual Sim, 3G, 4G, VoLTE, Wi-Fi	Helio G85, Octa Core, 2 GHz Processor	4 GB RAM, 64 GB inbuilt	6000 mAh Battery	6.51 inches, 720 x 1600 px Display with Water ...	13 MP + 2 MP Dual Rear & 8 MP Front Camera	Memory Card Supported, upto 256 GB	Android v10
1018	Micromax X743	₹1,275	7	Dual Sim	1.2 MHz Processor	32 MB RAM, 32 MB inbuilt	1000 mAh Battery	2.4 inches, 240 x 320 px Display	0.03 MP Rear Camera	No FM Radio	NaN
1019	Giva G3	₹750	7	Dual Sim	1 MHz Processor	32 MB RAM, 32 MB inbuilt	1000 mAh Battery	1.8 inches, 128 x 160 px Display	0.3 MP Rear Camera	No FM Radio	NaN

1020 rows × 11 columns

```
|: df.to_csv('smartphones.csv')
```

B. DATA CLEANING: smartprix step-3 data cleaning.ipynb & smartprix step-4 EDA.ipynb

Introduction

I performed data cleaning in both 'smartprix step-3 data cleaning.ipynb' and 'smartprix step-4 EDA.ipynb,' both before and while conducting the exploratory data analysis (EDA).

a) Understanding the Dataset

- **Data Description**:

  This dataset was created by web scraping data from Smartprix website using Selenium, time and BeautifulSoup libraries. It contains 1020 rows and 11 columns of information about smartphones. The dataset includes various attributes related to smartphones, such as model, price, and operating system. It has 1 column of float64 data type and 10 columns of object data type. The goal is to perform a detailed analysis and gain insights into the smartphone market.

```
In [5]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1020 entries, 0 to 1019
Data columns (total 11 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   model      1020 non-null   object 
 1   price      1020 non-null   object 
 2   rating     879 non-null    float64
 3   sim        1020 non-null   object 
 4   processor  1020 non-null   object 
 5   ram        1020 non-null   object 
 6   battery    1020 non-null   object 
 7   display    1020 non-null   object 
 8   camera     1019 non-null   object 
 9   card       1013 non-null   object 
 10  os         1003 non-null   object 
dtypes: float64(1), object(10)
memory usage: 87.8+ KB
```

- ## Initial Observations:

  In the initial observations (this is a human observation done by visually inspecting the data without using any code), I have found issues related to both Quality and Tidiness.

Quality Issues:

1. **Model column:** In the model column, brand names are written inconsistently in terms of case sensitivity, such as “OPPO” and “Oppo”.
2. **Price column:**
   a. Includes '₹' symbol, making it an object datatype, which prevents mathematical calculations.
   b. Contains commas, making it an object datatype, which prevents mathematical calculations.
   c. Phone "Namotel" has an unrealistically low price of 99.
3. **OS column:**
   a. Missing values(NaN).

    b. Incorrect values (sometimes include information about Bluetooth and FM radio, as well as OS version names like Lollipop).

4. **Display column:**
    a. Missing frequency values in some rows.
    b. Incorrect values in specific rows.
5. **Camera column:**
    a. Uses terms like Dual, Triple, and Quad to represent the number of cameras.
    b. Front and rear camera values are specified in a single column, with 'rear' and 'front' keywords separated by '&'.
    c. Incorrect values in specific rows.
6. **Card column:**
    a. In the memory card column, some values contain information about the OS and camera instead of memory.
7. **Processor column:** Incorrect values for some Samsung phones in specific rows.
8. **Data Entry:** An iPod is incorrectly included in row index 756.
9. **RAM column:** Incorrect values in specific rows.
10. **Battery column:** Incorrect values in specific rows.
11. **Miscellaneous:** Information for foldable phones is scattered.
1. **Missing Values:** Present in camera, card, and OS fields.
12. **Data Types:** Incorrect data types for price and rating columns.

Tidiness Issues:

1. **SIM column:** Can be split into has_5G, has_NFC, and has_IR_Blaster.
2. **RAM column:** Can be split into ram_capacity and internal_memory.
3. **Processor column:** Can be split into processor name, num_cores, and processor_speed.
4. **Battery column:** Can be split into battery_capacity and fast_charging_available and fast_charging.
5. **Display column:** Can be split into screen_size, resolution, and refresh_rate.
6. **Camera column:** Can be split into num_rear_cameras, num_front_cameras, primary_camera_rear, primary_camera_front.
7. **Card column:** Can be split into extended_memory_available and extended_upto.

b) Data Cleaning Steps:

- Handling Missing Values:

Smartprix step-4

```
In [634]: adf = df.copy()
```

```
In [635]: adf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 890 entries, 0 to 979
Data columns (total 25 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   brand_name                 890 non-null    object
 1   model                      890 non-null    object
 2   price                      890 non-null    int64
 3   rating                     814 non-null    float64
 4   has_5g                     890 non-null    bool
 5   has_nfc                    890 non-null    bool
 6   has_ir_blaster             890 non-null    bool
 7   processor_brand            877 non-null    object
 8   num_cores                  884 non-null    float64
 9   processor_speed            850 non-null    float64
 10  ram_capacity               890 non-null    int64
 11  internal_memory            890 non-null    int64
 12  fast_charging_available    890 non-null    int64
 13  fast_charging              720 non-null    float64
 14  battery_capacity           879 non-null    float64
 15  refresh_rate               890 non-null    int64
 16  screen_size                890 non-null    float64
 17  resolution                 890 non-null    object
 18  num_rear_cameras           890 non-null    int64
 19  num_front_cameras          887 non-null    float64
 20  primary_camera_rear        890 non-null    float64
 21  primary_camera_front       887 non-null    float64
 22  extended_memory_available  890 non-null    int64
 23  extended_upto              468 non-null    float64
 24  os                         883 non-null    object
dtypes: bool(3), float64(10), int64(7), object(5)
memory usage: 194.8+ KB
```

Extended_upto column:

There are 322 models that do not support memory extension, so I filled these values with 0. Since there is no memory available, there is also no 'extended_upto' quantity; therefore, I used 0, as I planned to run KNNImputer.

```
In [638]: adf[adf['extended_memory_available'] == 0].shape
Out[638]: (322, 25)
```

```
In [639]: # Missing value imputation for extended_upto
          adf.loc[adf['extended_memory_available'] == 0, 'extended_upto'] = 0
```

Fast_charging column:
For models that do not support 'fast_charging_available', I filled their 'fast_charging' column values with 0, as I planned to run KNNImputer.

```
In [641]: # Missing value imputation for fast_charging_available
          adf.loc[adf['fast_charging_available'] == 0, 'fast_charging'] = 0
```

OS column:
"To fill in missing OS values, I created lists of brand names where the OS is categorized as Android, iOS, or other operating systems. Using these lists, I identified empty OS cells and filled them accordingly.

```
In [640]: Android_brands = adf[adf['os'] == 'Android']['brand_name'].unique().tolist()
          Apple_brands = adf[adf['os'] == 'iOS']['brand_name'].unique().tolist()

          nan_os = adf[adf['os'].isna()]['brand_name']

          # Missing value imputation for os
          if (nan_os.isin(Android_brands)).any():
              adf.loc[df['os'].isna(),'os'] = 'Android'

          elif (nan_os.isin(Apple_brands)).any():
              adf.loc[df['os'].isna(),'os'] = 'iOS'

          else:
              adf.loc[df['os'].isna(),'os'] = 'Other'
```

Num_core column:
Apple processors are exclusively hexa-core, so I filled the missing Apple processor values with 6.

In [642]:
```
nan_core = adf[adf['num_cores'].isna()]['brand_name']

# Missing value imputation for num_cores
if (nan_core == 'apple').any():
    adf.loc[df['num_cores'].isna(),'num_cores'] = 6
```

In [643]:
```
adf.info()
# rating, processor_brand, processor_speed, fast_charging, battery_capacity,
# num_front_cameras, priamary_camera_front, extended_upto.
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 890 entries, 0 to 979
Data columns (total 25 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   brand_name                 890 non-null    object
 1   model                      890 non-null    object
 2   price                      890 non-null    int64
 3   rating                     814 non-null    float64
 4   has_5g                     890 non-null    bool
 5   has_nfc                    890 non-null    bool
 6   has_ir_blaster             890 non-null    bool
 7   processor_brand            877 non-null    object
 8   num_cores                  890 non-null    float64
 9   processor_speed            850 non-null    float64
 10  ram_capacity               890 non-null    int64
 11  internal_memory            890 non-null    int64
 12  fast_charging_available    890 non-null    int64
 13  fast_charging              829 non-null    float64
 14  battery_capacity           879 non-null    float64
 15  refresh_rate               890 non-null    int64
 16  screen_size                890 non-null    float64
 17  resolution                 890 non-null    object
 18  num_rear_cameras           890 non-null    int64
 19  num_front_cameras          887 non-null    float64
 20  primary_camera_rear        890 non-null    float64
 21  primary_camera_front       887 non-null    float64
 22  extended_memory_available  890 non-null    int64
 23  extended_upto              790 non-null    float64
 24  os                         890 non-null    object
dtypes: bool(3), float64(10), int64(7), object(5)
```

Imputing Missing Values:

I used the **"KNNImputer"** from the sklearn library to fill missing values in our dataset. The imputer was set to consider 5 neighbors (n_neighbors=5) and was applied to numerical columns. After fitting the imputer on the entire DataFrame, the following columns were imputed

and converted to integers: extended_upto, rating, fast_charging, processor_speed, battery_capacity, primary_camera_front, and num_front_cameras.

For the processor_brand column, I used **"SimpleImputer"** with the 'most_frequent' strategy to fill missing values, then converted the values back to strings.

In [646]:
```
from sklearn.impute import KNNImputer

# Initialize KNNImputer
imputer = KNNImputer(n_neighbors=5)

features_for_imputation = []
for variable in adf.columns.tolist():
    if adf[variable].dtype in ['int64', 'float64', 'int32']:
        features_for_imputation.append(variable)

# Fit the imputer on the entire DataFrame
imputer.fit(adf[features_for_imputation])

# 1. extended_upto
adf['extended_upto'] = imputer.transform(adf[features_for_imputation])[:, features_for_imputation.index('extended_upto')]
adf['extended_upto'] = adf['extended_upto'].astype(int)

# 2. rating
adf['rating'] = imputer.transform(adf[features_for_imputation])[:, features_for_imputation.index('rating')]
adf['rating'] = adf['rating'].astype(int)

# 3. fast_charging
adf['fast_charging'] = imputer.transform(adf[features_for_imputation])[:, features_for_imputation.index('fast_charging')]
adf['fast_charging'] = adf['fast_charging'].astype(int)



# 4. processor_speed
# perform imputation on 'processor_speed' column
adf['processor_speed'] = imputer.fit_transform(adf[features_for_imputation])[:, features_for_imputation.index('processor_speed')]
adf['processor_speed'] = adf['processor_speed'].astype(int)
```

```
# 6. battery_capacity
# perform imputation on 'battery_capacity' column
adf['battery_capacity'] = imputer.fit_transform(adf[features_for_imputation])[:, features_for_imputation.index('battery_capacity'
adf['battery_capacity'] = adf['battery_capacity'].astype(int)



# 7. primary_camera_front
# perform imputation on 'priamary_camera_front' column
adf['primary_camera_front'] = imputer.fit_transform(adf[features_for_imputation])[:, features_for_imputation.index('primary_camer
adf['primary_camera_front'] = adf['primary_camera_front'].astype(int)



# 8. num_front_cameras
# perform imputation on 'num_front_cameras' column
adf['num_front_cameras'] = imputer.fit_transform(adf[features_for_imputation])[:, features_for_imputation.index('num_front_camera
adf['num_front_cameras'] = adf['num_front_cameras'].astype(int)
```

In [648]:

```
from sklearn.impute import SimpleImputer

# # 5. processor_brand
# Initialize SimpleImputer with 'most_frequent' strategy
imputer = SimpleImputer(strategy='most_frequent')

# Extract 'processor_brand' column as a Series
processor_brand_series = adf['processor_brand']

# Impute missing values in the 'processor_brand' column
processed_processor_brand = imputer.fit_transform(processor_brand_series.values.reshape(-1, 1))

# Convert the imputed values back to string type
adf['processor_brand'] = processed_processor_brand.flatten()
adf['processor_brand'] = adf['processor_brand'].astype(str)
```

```
In [666]: adf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 890 entries, 0 to 979
Data columns (total 25 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   brand_name                 890 non-null    object
 1   model                      890 non-null    object
 2   price                      890 non-null    int64
 3   rating                     890 non-null    int32
 4   has_5g                     890 non-null    bool
 5   has_nfc                    890 non-null    bool
 6   has_ir_blaster             890 non-null    bool
 7   processor_brand            890 non-null    object
 8   num_cores                  890 non-null    float64
 9   processor_speed            890 non-null    int32
 10  ram_capacity               890 non-null    int64
 11  internal_memory            890 non-null    int64
 12  fast_charging_available    890 non-null    bool
 13  fast_charging              890 non-null    int32
 14  battery_capacity           890 non-null    int32
 15  refresh_rate               890 non-null    int64
 16  screen_size                890 non-null    float64
 17  resolution                 890 non-null    object
 18  num_rear_cameras           890 non-null    int64
 19  num_front_cameras          890 non-null    int32
 20  primary_camera_rear        890 non-null    float64
 21  primary_camera_front       890 non-null    int32
 22  extended_memory_available  890 non-null    bool
 23  extended_upto              890 non-null    int32
 24  os                         890 non-null    object
dtypes: bool(5), float64(3), int32(7), int64(5), object(5)
memory usage: 158.3+ KB
```

- **Removing Duplicates**:

```
In [7]: df.duplicated().sum()
Out[7]: 0
```

There are no duplicate rows in the dataset.

- **Correcting Data Types**:

```
In [228]: df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 980 entries, 0 to 1019
Data columns (total 26 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   brand_name                 980 non-null    object
 1   model                      980 non-null    object
 2   price                      980 non-null    int32
 3   rating                     879 non-null    float64
 4   has_5g                     980 non-null    bool
 5   has_nfc                    980 non-null    bool
 6   has_ir_blaster             980 non-null    bool
 7   processor_name             960 non-null    object
 8   processor_brand            960 non-null    object
 9   num_cores                  974 non-null    float64
 10  processor_speed            938 non-null    float64
 11  ram_capacity               980 non-null    float64
 12  internal_memory            980 non-null    float64
 13  fast_charging_available    980 non-null    int64
 14  fast_charging              769 non-null    float64
 15  battery_capacity           969 non-null    float64
 16  screen_size                980 non-null    float64
 17  refresh_rate               980 non-null    int32
 18  resolution                 980 non-null    object
 19  num_rear_cameras           980 non-null    int64
 20  num_front_cameras          980 non-null    object
 21  primary_camera_rare        980 non-null    object
 22  primary_camera_front       976 non-null    object
 23  extended_memory_available  980 non-null    int64
 24  extended_upto              507 non-null    float64
 25  os                         966 non-null    object
```

➢ I converted the **'num_front_cameras'** column to the float data type. Although the values in the column are numeric and suitable for int, we have NaN values in it; therefore, the only numeric format we can use is float.

```
In [234]: df1['num_front_cameras'] = df1['num_front_cameras'].astype(float)
```

➢ converted the **"primary_camera_rear"** column to numeric using pd.to_numeric with errors = 'coerce' as a parameter.

```
In [238]: df1['primary_camera_rear'] = pd.to_numeric(df1['primary_camera_rear'],errors = 'coerce')
```

➢ converted the **"primary_camera_front"** column to numeric using pd.to_numeric with errors = 'coerce' as a parameter.

```
In [243]: df1['primary_camera_front'] = pd.to_numeric(df1['primary_camera_front'], errors = 'coerce')
```

➢ I converted the **'ram_capacity'** column to the int data type, as the data consists of integer values.

```
In [248]: df1['ram_capacity'] = df1['ram_capacity'].astype(int)
```

➢ I converted the **'internal_memory'** column to the int data type, as the data consists of integer values.

```
In [249]: df1['internal_memory'] = df1['internal_memory'].astype(int)
```

- **num_cores:**
  Converted the 'num_cores' column from object to float by converting the values into numerical format.

```
In [112]: df1['num_cores'].value_counts()

Out[112]: num_cores
          Octa Core              866
          Hexa Core               31
          Quad Core               31
          Octa Core Processor     19
          Octa Core Processor      9
          Hexa Core Processor      8
          Quad Core                6
          Octa Core                5
          Name: count, dtype: int64
```

In [113]:
```python
import math

def num_cores_converter(text):
    if isinstance(text, float) and not math.isnan(text):
        # Handle float values
        return int(text)  # Convert float to integer
    elif isinstance(text, str) and 'Octa' in text:
        return 8
    elif isinstance(text, str) and 'Hexa' in text:
        return 6
    elif isinstance(text, str) and 'Quad' in text:
        return 4
    # Add more conditions as needed
    else:
        return None  # or any default value you prefer
```

In [114]:
```python
df1['num_cores'].apply(num_cores_converter).value_counts()
```

Out[114]:
```
num_cores
8.0    899
6.0     39
4.0     37
Name: count, dtype: int64
```

In [115]:
```python
df1['num_cores'] = df1['num_cores'].apply(num_cores_converter)
```

```
In [116]: df1['num_cores'].info()

<class 'pandas.core.series.Series'>
Index: 982 entries, 0 to 1019
Series name: num_cores
Non-Null Count  Dtype
--------------  -----
975 non-null    float64
dtypes: float64(1)
memory usage: 47.6 KB
```

- **processor_speed:**
  Converted the **'processor_speed'** column from object to float by converting the values into numerical format.

In [117]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 982 entries, 0 to 1019
Data columns (total 20 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   index            982 non-null    int64
 1   brand_name       982 non-null    object
 2   model            982 non-null    object
 3   price            982 non-null    int32
 4   rating           879 non-null    float64
 5   sim              982 non-null    object
 6   has_5g           982 non-null    bool
 7   has_nfc          982 non-null    bool
 8   has_ir_blaster   982 non-null    bool
 9   processor        982 non-null    object
 10  processor_name   962 non-null    object
 11  processor_brand  962 non-null    object
 12  num_cores        975 non-null    float64
 13  processor_speed  939 non-null    object
 14  ram              982 non-null    object
 15  battery          971 non-null    object
 16  display          982 non-null    object
 17  camera           982 non-null    object
 18  card             982 non null    object
 19  os               967 non-null    object
dtypes: bool(3), float64(2), int32(1), int64(1), object(13)
memory usage: 169.4+ KB
```

In [118]:
```
df1['processor_speed'].value_counts()
```

Out[118]:

processor_speed	
2 GHz Processor	146
2.2 GHz Processor	135
2.4 GHz Processor	128
3.2 GHz Processor	94
2.3 GHz Processor	86
3 GHz Processor	53
2.84 GHz Processor	36
2.05 GHz Processor	28
2.5 GHz Processor	23
1.8 GHz Processor	23
1.6 GHz Processor	20
2.85 GHz Processor	19
3.22 GHz Processor	18
3.1 GHz Processor	15
2.6 GHz Processor	14
2.9 GHz Processor	13
1.82 GHz Processor	10
1.3 GHz Processor	10
2.73 GHz Processor	9
2.8 GHz Processor	9
3.05 GHz Processor	8
2.36 GHz Processor	8
2.7 GHz Processor	5
1.4 GHz Processor	5
2.65 GHz Processor	5
1.5 GHz Processor	4
2.96 GHz Processor	3
2.86 GHz Processor	3
3.13 GHz Processor	2
1.1 GHz Processor	1
2.35 GHz Processor	1

```
In [119]: # let us only take number
          df1['processor_speed'].str.strip().str.split(' ').str.get(0)
```

```
Out[119]: 0        3.2 GHz
          1        2.2 GHz
          2        2.4 GHz
          3        2.2 GHz
          4        2.6 GHz
                    ...
          1015       3 GHz
          1016     2.2 GHz
          1017    2.85 GHz
          1018     2.2 GHz
          1019         NaN
          Name: processor_speed, Length: 982, dtype: object
```

```
In [120]: df1['processor_speed'].str.strip().str.split(' ').str.get(0)[0]
```

```
Out[120]: '3.2\u2009GHz'
```

```
In [121]: df1['processor_speed'] = df1['processor_speed'].str.strip().str.split(' ').str.get(0).
          str.replace('\u2009',' ').str.split(' ').str.get(0).astype(float)
```

- **ram_capacity:** The 'ram' column was in object data type. I extracted the numerical RAM data from it and created a new column called 'ram_capacity' and typecasted it into float data type.

```
In [123]: # in ram column we have ram and internal memory together
          df1['ram'].str.strip().str.split(',').str.get(0).str.split(' ')
```

```
Out[123]: 0       [12 GB, RAM]
          1        [6 GB, RAM]
          2        [4 GB, RAM]
          3        [6 GB, RAM]
          4        [6 GB, RAM]
                      ...
          1015     [8 GB, RAM]
          1016     [6 GB, RAM]
          1017     [8 GB, RAM]
          1018     [6 GB, RAM]
          1019     [8 GB, RAM]
          Name: ram, Length: 982, dtype: object
```

```
In [124]: df1['ram'].str.strip().str.split(',').str.get(0).str.split(' ').get(0)[0]
```

```
Out[124]: '12\u2009GB'
```

```
In [125]: ram_capacity = df1['ram'].str.strip().str.split(',').str.get(0).str.findall(r'\b(\d+)\b').str.get(0)
```

```
In [127]: df1.insert(15,'ram_capacity',ram_capacity)
```

```
In [133]: df1['ram_capacity'] = df1['ram_capacity'].astype(float)
```

- **Internal_memory:** The 'ram' column was in object data type. extracted the numerical internal memory data from it and created a new column called internal_memory and type casted to float datatype.

```
In [129]: internal_memory = df1['ram'].str.strip().str.split(',').str.get(1).str.strip().str.findall(r'\b(\d+)\b').str.get(0)
```

```
In [130]: df1.insert(16,'internal_memory',internal_memory)
```

- ## Data Relevance And Validity:

The following sets contain the index numbers of the rows where there are problems with the values in specific columns of the dataset. Each set is assigned to a variable that corresponds to the column name, allowing for easy identification and manipulation of the problematic rows.

Each variable can be used to track and address the specific issues related to the corresponding data column.

```
In [17]: processor_rows = set((642,647,649,659,667,701,750,759,819,859,883,884,919,927,929,932,990,1002))
         ram_rows = set((441,485,534,584,610,613,642,647,649,659,667,701,750,759,819,859,884,919,927,929,932,990,1002))
         battery_rows = set((113,151,309,365,378,441,450,553,584,610,613,630,642,647,649,659,667,701,750,756,759,764,819,855,859,884,915,
                             916,927,929,932,990,1002))
         display_rows = set((378,441,450,553,584,610,613,630,642,647,649,659,667,701,750,759,764,819,859,884,915,916,927,929,932,990,
                             1002))
         camera_rows = set((100,113,151,157,161,238,273,308,309,323,324,365,367,378,394,441,450,484,506,534,553,571,572,575,584,610,613,
                            615,630,642,647,649,659,667,684,687,705,711,723,728,750,756,759,764,792,819,846,854,855,858,883,884,896,915,
                            916,927,929,932,945,956,990,995,1002,1016))
```

I found a few feature phones in the dataset, so I decided to drop them, as the dataset is meant for smartphones only.

In [24]:
```
# below index number phones are feature phones(not smartphones) so let's drop them
df1.drop([645,857,882,925],inplace = True)
```

In [27]:
```
# it is not a smartphone
df1.drop(582,inplace = True)
```

In [262]:
```
# they are not smartphones
df1.drop([376,754],inplace = True)
```

In [42]:
```
# found few feature phones lets drop them
df1.drop([155,271],inplace = True)
```

➢ **Battery column:**

Shifting data from incorrect columns to their correct places is an action taken to ensure **data validity** and **accuracy**. This process can also be considered a part of **data consistency** because it ensures that the data adheres to the expected format and structure.

To correct the misplaced data in the **"battery"** column (which is currently in the **"ram"** column to its left), I need to shift the data in the affected columns one position to the right.

In [31]:
```
# as you can see from battery to till last column value need a one right shift
df1[df1['index'].isin(battery_rows)]
```

Out[31]:

	index	model	price	rating	sim	processor	ram	battery	display	camera	card	os
111	113	Apple iPhone 12	51999	74.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A14, Hexa Core, 3.1 GHz Processor	4 GB RAM, 64 GB inbuilt	6.1 inches, 1170 x 2532 px Display with Large ...	12 MP + 12 MP Dual Rear & 12 MP Front Camera	Memory Card Not Supported	iOS v14	No FM Radio
149	151	Apple iPhone 12 Mini	40999	74.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A14, Hexa Core, 3.1 GHz Processor	4 GB RAM, 64 GB inbuilt	5.4 inches, 1080 x 2340 px Display	12 MP + 12 MP Dual Rear & 12 MP Front Camera	Memory Card Not Supported	iOS v14	No FM Radio
307	309	Apple iPhone 12 (128GB)	55999	75.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A14, Hexa Core, 3.1 GHz Processor	4 GB RAM, 128 GB inbuilt	6.1 inches, 1170 x 2532 px Display with Large ...	12 MP + 12 MP Dual Rear & 12 MP Front Camera	Memory Card Not Supported	iOS v14	No FM Radio
363	365	Apple iPhone 12 Mini (128GB)	45999	75.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A14, Hexa Core, 3.1 GHz Processor	4 GB RAM, 128 GB inbuilt	5.4 inches, 1080 x 2340 px Display	12 MP + 12 MP Dual Rear & 12 MP Front Camera	Memory Card Not Supported	iOS v14	No FM Radio
439	441	Apple iPhone SE 3 2022	43900	NaN	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A15, Hexa Core, 3.22 GHz Processor	64 GB inbuilt	4.7 inches, 750 x 1334 px Display	12 MP Rear & 7 MP Front Camera	Memory Card Not Supported	iOS v15	No FM Radio
448	450	Apple iPhone 15 Pro	130990	75.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A16	8 GB RAM, 128 GB inbuilt	6.06 inches, 1170 x 2532 px, 120 Hz Display wi...	50 MP + 12 MP + 12 MP Triple Rear & 12 MP Fron...	Memory Card Not Supported	iOS v15	No FM Radio
628	630	Apple iPhone 12 Pro (512GB)	139900	80.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A14, Hexa Core, 3.1 GHz Processor	6 GB RAM, 512 GB inbuilt	6.1 inches, 1170 x 2532 px Display with Large ...	12 MP + 12 MP + 12 MP Triple Rear & 12 MP Fron...	Memory Card Not Supported	iOS v14.0	No FM Radio
762	764	Apple iPhone SE 4	49990	60.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A15, Hexa Core, 3.22 GHz Processor	64 GB inbuilt	6.1 inches, 750 x 1580 px Display	12 MP Rear & 10.8 MP Front Camera	Memory Card Not Supported	iOS v16	No FM Radio

In [32]:
```
# as we don't any problem in original dataframe even by accident
temp_df = df1[df1['index'].isin(battery_rows)]
```

In [33]:
```
# all rows and columns from 7 to till end and a single right shift,axis = 1 means row wise shifting
temp_df.iloc[:,7:].shift(1,axis = 1)
```

Out[33]:

	battery	display	camera	card	os
111	None	6.1 inches, 1170 x 2532 px Display with Large ...	12 MP + 12 MP Dual Rear & 12 MP Front Camera	Memory Card Not Supported	iOS v14
149	None	5.4 inches, 1080 x 2340 px Display	12 MP + 12 MP Dual Rear & 12 MP Front Camera	Memory Card Not Supported	iOS v14
307	None	6.1 inches, 1170 x 2532 px Display with Large ...	12 MP + 12 MP Dual Rear & 12 MP Front Camera	Memory Card Not Supported	iOS v14
363	None	5.4 inches, 1080 x 2340 px Display	12 MP + 12 MP Dual Rear & 12 MP Front Camera	Memory Card Not Supported	iOS v14
439	None	4.7 inches, 750 x 1334 px Display	12 MP Rear & 7 MP Front Camera	Memory Card Not Supported	iOS v15
448	None	6.06 inches, 1170 x 2532 px, 120 Hz Display wi...	50 MP + 12 MP + 12 MP Triple Rear & 12 MP Fron...	Memory Card Not Supported	iOS v15
628	None	6.1 inches, 1170 x 2532 px Display with Large ...	12 MP + 12 MP + 12 MP Triple Rear & 12 MP Fron...	Memory Card Not Supported	iOS v14.0
762	None	6.1 inches, 750 x 1580 px Display	12 MP Rear & 10.8 MP Front Camera	Memory Card Not Supported	iOS v16
853	None	6.1 inches, 1170 x 2532 px Display with Large ...	12 MP + 12 MP + 12 MP Triple Rear & 12 MP Fron...	Memory Card Not Supported	iOS v14.0
913	None	5.4 inches, 1080 x 2340 px Display	12 MP + 12 MP Dual Rear & 12 MP Front Camera	Memory Card Not Supported	iOS v14
914	None	6.1 inches, 1170 x 2532 px Display with Large ...	12 MP + 12 MP Dual Rear & 12 MP Front Camera	Memory Card Not Supported	iOS v14

I created a dummy DataFrame with only the rows that needed to be shifted and performed a single right shift to verify the solution. It worked as expected. Now, I will apply these changes to the original data.

In [35]:
```
x = temp_df.iloc[:,7:].shift(1,axis = 1).values
```

In [36]:
```
temp_df.index
```

Out[36]:
```
Index([111, 149, 307, 363, 439, 448, 628, 762, 853, 913, 914], dtype='int64')
```

In [37]:
```
x_index = temp_df.index
```

In [38]:
```
# in rows(is temp_df.index) and columns(is temp_df.columns[7:]) of original data we are storing x vlaues
df1.loc[temp_df.index,temp_df.columns[7:]] = x
```

In [39]:
```
# check
df1[df1['index'].isin(battery_rows)]
```

Out[39]:

	index	model	price	rating	sim	processor	ram	battery	display	camera	card	os
111	113	Apple iPhone 12	51999	74.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A14, Hexa Core, 3.1 GHz Processor	4 GB RAM, 64 GB inbuilt	None	6.1 inches, 1170 x 2532 px Display with Large ...	12 MP + 12 MP Dual Rear & 12 MP Front Camera	Memory Card Not Supported	iOS v14
149	151	Apple iPhone 12 Mini	40999	74.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A14, Hexa Core, 3.1 GHz Processor	4 GB RAM, 64 GB inbuilt	None	5.4 inches, 1080 x 2340 px Display	12 MP + 12 MP Dual Rear & 12 MP Front Camera	Memory Card Not Supported	iOS v14
307	309	Apple iPhone 12 (128GB)	55999	75.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A14, Hexa Core, 3.1 GHz Processor	4 GB RAM, 128 GB inbuilt	None	6.1 inches, 1170 x 2532 px Display with Large ...	12 MP + 12 MP Dual Rear & 12 MP Front Camera	Memory Card Not Supported	iOS v14
363	365	Apple iPhone 12 Mini (128GB)	45999	75.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A14, Hexa Core, 3.1 GHz Processor	4 GB RAM, 128 GB inbuilt	None	5.4 inches, 1080 x 2340 px Display	12 MP + 12 MP Dual Rear & 12 MP Front Camera	Memory Card Not Supported	iOS v14
439	441	Apple iPhone SE 3 2022	43900	NaN	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A15, Hexa Core, 3.22 GHz Processor	64 GB inbuilt	None	4.7 inches, 750 x 1334 px Display	12 MP Rear & 7 MP Front Camera	Memory Card Not Supported	iOS v15
448	450	Apple iPhone 15 Pro	130990	75.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A16	8 GB RAM, 128 GB inbuilt	None	6.06 inches, 1170 x 2532 px, 120 Hz Display wi...	50 MP + 12 MP + 12 MP Triple Rear & 12 MP Fron...	Memory Card Not Supported	iOS v15
628	630	Apple iPhone 12 Pro (512GB)	139900	80.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A14, Hexa Core, 3.1 GHz Processor	6 GB RAM, 512 GB inbuilt	None	6.1 inches, 1170 x 2532 px Display with Large ...	12 MP + 12 MP + 12 MP Triple Rear & 12 MP Fron...	Memory Card Not Supported	iOS v14.0
762	764	Apple iPhone SE 4	49990	60.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A15, Hexa Core, 3.22 GHz Processor	64 GB inbuilt	None	6.1 inches, 750 x 1580 px Display	12 MP Rear & 10.8 MP Front Camera	Memory Card Not Supported	iOS v16

➢ **Camera column:**

As you can see in the output, the only correction I need is to shift the values from the card column into the camera column.

```
In [42]: df1[df1['index'].isin(camera_rows)]
```

Out[42]:

	index	model	price	rating	sim	processor	ram	battery	display	camera	card	os
98	100	Vivo X Fold 5G	106990	NaN	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Snapdragon 8 Gen1, Octa Core, 3 GHz Processor	12 GB RAM, 256 GB inbuilt	4600 mAh Battery with 66W Fast Charging	8.03 inches, 1916 x 2160 px, 120 Hz Display	Foldable Display	50 MP Quad Rear & 16 MP Front Camera	Android v12
111	113	Apple iPhone 12	51999	74.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A14, Hexa Core, 3.1 GHz Processor	4 GB RAM, 64 GB inbuilt	None	6.1 inches, 1170 x 2532 px Display with Large ...	12 MP + 12 MP Dual Rear & 12 MP Front Camera	Memory Card Not Supported	iOS v14
149	151	Apple iPhone 12 Mini	40999	74.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A14, Hexa Core, 3.1 GHz Processor	4 GB RAM, 64 GB inbuilt	None	5.4 inches, 1080 x 2340 px Display	12 MP + 12 MP Dual Rear & 12 MP Front Camera	Memory Card Not Supported	iOS v14
159	161	Oppo Find N2 5G	94990	NaN	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Snapdragon 8+ Gen1, Octa Core, 3.2 GHz Processor	12 GB RAM, 256 GB inbuilt	4520 mAh Battery with 67W Fast Charging	7.1 inches, 1792 x 1920 px, 120 Hz Display wit...	Foldable Display, Dual Display	50 MP + 48 MP + 32 MP Triple Rear & 32 MP + 32...	Memory Card Not Supported

As you can see, only a few rows have data in the wrong column, not all. So, let us select only those rows where there is no data related to the camera in the camera column.

```
In [43]: temp_df = df1[df1['index'].isin(camera_rows)]
```

```
In [44]: temp_df = temp_df[~temp_df['camera'].str.contains('MP')]
```

Since the camera column data is currently in the card column, I need to select this data and insert it into the camera column.After making this correction, the camera column will contain only data related to cameras.

```
In [46]: # rows temp_df.index,column camera
         df1.loc[temp_df.index,'camera'] = temp_df['card'].values
```

```
In [47]: df1[df1['index'].isin(camera_rows)]
```

Out[47]:

	index	model	price	rating	sim	processor	ram	battery	display	camera	card	os
98	100	Vivo X Fold 5G	106990	NaN	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Snapdragon 8 Gen1, Octa Core, 3 GHz Processor	12 GB RAM, 256 GB inbuilt	4600 mAh Battery with 66W Fast Charging	8.03 inches, 1916 x 2160 px, 120 Hz Display	50 MP Quad Rear & 16 MP Front Camera	50 MP Quad Rear & 16 MP Front Camera	Android v12
111	113	Apple iPhone 12	51999	74.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A14, Hexa Core, 3.1 GHz Processor	4 GB RAM, 64 GB inbuilt	None	6.1 inches, 1170 x 2532 px Display with Large ...	12 MP + 12 MP Dual Rear & 12 MP Front Camera	Memory Card Not Supported	iOS v14
149	151	Apple iPhone 12 Mini	40999	74.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A14, Hexa Core, 3.1 GHz Processor	4 GB RAM, 64 GB inbuilt	None	5.4 inches, 1080 x 2340 px Display	12 MP + 12 MP Dual Rear & 12 MP Front Camera	Memory Card Not Supported	iOS v14
159	161	Oppo Find N2 5G	94990	NaN	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Snapdragon 8+ Gen1, Octa Core, 3.2 GHz Processor	12 GB RAM, 256 GB inbuilt	4520 mAh Battery with 67W Fast Charging	7.1 inches, 1792 x 1920 px, 120 Hz Display wit...	50 MP + 48 MP + 32 MP Triple Rear & 32 MP + 32...	50 MP + 48 MP + 32 MP Triple Rear & 32 MP + 32...	Memory Card Not Supported

➢ **Card column:**

```
In [47]: df1[df1['index'].isin(camera_rows)]
```

Out[47]:

	index	model	price	rating	sim	processor	ram	battery	display	camera	card	os
98	100	Vivo X Fold 5G	106990	NaN	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Snapdragon 8 Gen1, Octa Core, 3 GHz Processor	12 GB RAM, 256 GB inbuilt	4600 mAh Battery with 66W Fast Charging	8.03 inches, 1916 x 2160 px, 120 Hz Display	50 MP Quad Rear & 16 MP Front Camera	50 MP Quad Rear & 16 MP Front Camera	Android v12
111	113	Apple iPhone 12	51999	74.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A14, Hexa Core, 3.1 GHz Processor	4 GB RAM, 64 GB inbuilt	None	6.1 inches, 1170 x 2532 px Display with Large ...	12 MP + 12 MP Dual Rear & 12 MP Front Camera	Memory Card Not Supported	iOS v14
149	151	Apple iPhone 12 Mini	40999	74.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Bionic A14, Hexa Core, 3.1 GHz Processor	4 GB RAM, 64 GB inbuilt	None	5.4 inches, 1080 x 2340 px Display	12 MP + 12 MP Dual Rear & 12 MP Front Camera	Memory Card Not Supported	iOS v14
159	161	Oppo Find N2 5G	94990	NaN	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Snapdragon 8+ Gen1, Octa Core, 3.2 GHz Processor	12 GB RAM, 256 GB inbuilt	4520 mAh Battery with 67W Fast Charging	7.1 inches, 1792 x 1920 px, 120 Hz Display wit...	50 MP + 48 MP + 32 MP Triple Rear & 32 MP + 32...	50 MP + 48 MP + 32 MP Triple Rear & 32 MP + 32...	Memory Card Not Supported

As you can see in the output, some rows in the card column contain irrelevant data. This occurred because, during web scraping, there was no memory card data for these models, and camera data mistakenly got into this column.

```
In [50]: df1['card'].value_counts()
```

```
Out[50]: card
Memory Card Supported, upto 1 TB                                  171
Memory Card Not Supported                                         123
Android v12                                                       107
Memory Card Supported, upto 512 GB                                105
Memory Card (Hybrid), upto 1 TB                                    91
Memory Card Supported                                              89
Memory Card Supported, upto 256 GB                                 87
Android v13                                                        46
Android v11                                                        41
Memory Card (Hybrid)                                               30
Memory Card (Hybrid), upto 256 GB                                  13
Android v10                                                        11
Memory Card (Hybrid), upto 512 GB                                  11
Memory Card Supported, upto 128 GB                                  6
Memory Card Supported, upto 2 TB                                    5
Memory Card Supported, upto 32 GB                                   4
Memory Card (Hybrid), upto 64 GB                                    3
Memory Card (Hybrid), upto 128 GB                                   3
Android v9.0 (Pie)                                                  2
50 MP + 12 MP + 10 MP Triple Rear & 10 MP + 4 MP Dual Front Camera  2
64 MP + 16 MP + 8 MP Triple Rear & 32 MP Front Camera               2
50 MP + 8 MP Dual Rear & 32 MP Front Camera                         2
12 MP + 12 MP Dual Rear & 10 MP Front Camera                        2
50 MP Quad Rear & 16 MP Front Camera                                2
64 MP + 13 MP + 0.3 MP Triple Rear & 10 MP Front Camera             1
48 MP Quad Rear Camera                                              1
HarmonyOS                                                           1
Memory Card (Hybrid), upto 2 TB                                     1
5 MP Rear & 2 MP Front Camera                                       1
108 MP + 13 MP + 8 MP Triple Rear & 20 MP Front Camera              1
HarmonyOS v2.0                                                      1
50 MP + 13 MP + 8 MP Triple Rear & 10.7 MP Front Camera             1
```

I selected only the rows with camera data in the card column, as I had moved placed this data into the camera column. I then created a new dataframe with this data from the card column, identified the index numbers, and replaced this data with the correct value, 'Memory Card Not Supported.' As a result, the number of rows with the value 'Memory Card Not Supported' increased from 123 to 149.

```
In [51]: # let us replace camera values with Memory Card Not Supported in card column
         temp_df = df1[df1['card'].str.contains('MP')]

In [52]: # in loc gave temp_df as rows and card as column
         df1.loc[temp_df.index,'card'] = 'Memory Card Not Supported'

In [53]: df1['card'].value_counts()

Out[53]: card
         Memory Card Supported, upto 1 TB       171
         Memory Card Not Supported              149
         Android v12                            107
         Memory Card Supported, upto 512 GB     105
         Memory Card (Hybrid), upto 1 TB         91
         Memory Card Supported                   89
         Memory Card Supported, upto 256 GB      87
         Android v13                             46
         Android v11                             41
         Memory Card (Hybrid)                    30
         Memory Card (Hybrid), upto 256 GB       13
         Memory Card (Hybrid), upto 512 GB       11
         Android v10                             11
         Memory Card Supported, upto 128 GB       6
         Memory Card Supported, upto 2 TB         5
         Memory Card Supported, upto 32 GB        4
         Memory Card (Hybrid), upto 64 GB         3
         Memory Card (Hybrid), upto 128 GB        3
         Android v9.0 (Pie)                       2
         Android v12.1                            1
         Memory Card Supported, upto 1000 GB      1
         iOS v10                                  1
```

There are only two types of values in the card column: memory-related and OS-related. I created a new dataframe with rows that don't contain memory card information since they contain OS data. I moved these values to the respective rows in the OS column.

```
In [54]: # as you can see in above output ther are only two types of values in card one is memory related and another one is os
         temp_df = df1[~df1['card'].str.contains('Memory Card')]

In [55]: # in loc temp_df.index is rows and card is column
         df1.loc[temp_df.index,'os'] = temp_df['card'].values
```

Using the same index, I inserted 'Memory Card Not Supported' into the card column.

```
In [56]: df1.loc[temp_df.index,'card'] = 'Memory Card Not Supported'
```

```
In [57]: df1['card'].value_counts()
```

```
Out[57]: card
Memory Card Not Supported               362
Memory Card Supported, upto 1 TB        171
Memory Card Supported, upto 512 GB      105
Memory Card (Hybrid), upto 1 TB          91
Memory Card Supported                    89
Memory Card Supported, upto 256 GB       87
Memory Card (Hybrid)                     30
Memory Card (Hybrid), upto 256 GB        13
Memory Card (Hybrid), upto 512 GB        11
Memory Card Supported, upto 128 GB        6
Memory Card Supported, upto 2 TB          5
Memory Card Supported, upto 32 GB         4
Memory Card (Hybrid), upto 128 GB         3
Memory Card (Hybrid), upto 64 GB          3
Memory Card Supported, upto 1000 GB       1
Memory Card (Hybrid), upto 2 TB           1
Name: count, dtype: int64
```

- **OS Column:**
  Os column contains data about os and also about memory card

```
In [58]: # os column contains data of memory card in it
         df1['os'].value_counts()
```

```
Out[58]: os
Android v12                                394
Android v11                                274
Android v13                                 91
Android v10                                 69
Android v9.0 (Pie)                          29
Android v10.0                               23
iOS v16                                     15
iOS v15                                     12
Android v8.1 (Oreo)                         10
iOS v14                                      6
Memory Card Not Supported                    6
Android v11.0                                4
Android v8.0 (Oreo)                          4
iOS v13                                      4
iOS v15.0                                    3
Android v6.0 (Marshmallow)                   2
Memory Card (Hybrid), upto 256 GB            2
Memory Card (Hybrid), upto 2 TB              2
HarmonyOS v2.0                               2
Android v5.1.1 (Lollipop)                    2
Harmony v2.0                                 2
EMUI v12                                     2
Memory Card Supported, upto 256 GB           2
Android v12.1                                2
iOS v14.0                                    2
Memory Card Supported, upto 128 GB           1
Android v7.1 (Nougat)                        1
Android                                      1
Hongmeng OS v4.0                             1
HarmonyOS                                    1
```

I then extracted the memory card value from the rows where the "Os" column contains memory card information and inserted it into the "card" column in the main DataFrame.

```
In [59]: temp_df = df1[df1['os'].str.contains('Memory Card')]

In [60]: df1.loc[temp_df.index,'card'] = temp_df['os'].values

In [61]: df1['card'].value_counts()

Out[61]: card
         Memory Card Not Supported               354
         Memory Card Supported, upto 1 TB        171
         Memory Card Supported, upto 512 GB      105
         Memory Card (Hybrid), upto 1 TB          91
         Memory Card Supported                    89
         Memory Card Supported, upto 256 GB       89
         Memory Card (Hybrid)                     31
         Memory Card (Hybrid), upto 256 GB        15
         Memory Card (Hybrid), upto 512 GB        11
         Memory Card Supported, upto 128 GB        7
         Memory Card Supported, upto 2 TB          5
         Memory Card Supported, upto 32 GB         4
         Memory Card (Hybrid), upto 128 GB         3
         Memory Card (Hybrid), upto 64 GB          3
         Memory Card (Hybrid), upto 2 TB           3
         Memory Card Supported, upto 1000 GB       1
         Name: count, dtype: int64
```

- I created a temporary data frame containing rows where the 'os' column has data about memory cards. Using the index numbers from this temporary dataframe, I replaced memory card values in the 'os' column with NaN.

```
In [62]: temp_df = df1[df1['os'].str.contains('Memory Card')]

In [63]: # in loc we gave temp_df.index as rows and os column as column
         df1.loc[temp_df.index,'os'] = np.nan
```

Additionally, I replaced the value 'Bluetooth' with NaN in the OS column.

```
In [65]: df1[df1['os'] == 'Bluetooth']
```

Out[65]:

	index	model	price	rating	sim	processor	ram	battery	display	camera	card	os
486	488	Nokia 8000 4G	6899	NaN	Dual Sim, 3G, 4G, Wi-Fi	Snapdragon 210, Quad Core, 1.1 GHz Processor	512 MB RAM, 4 GB inbuilt	1500 mAh Battery	2.8 inches, 240 x 320 px Display	2 MP Rear Camera	Memory Card Supported	Bluetooth

```
In [66]: temp_df = df1[df1['os'] == 'Bluetooth']
```

```
In [67]: # in loc we gave temp_df.index as rows and os column as column
         df1.loc[temp_df.index,'os'] = np.nan
```

- **Processor_name Column :**
  I observed that the values in the **'processor_name'** column contain some unrelated entries, including the number of cores. I will research the remaining ones, as I currently have no information about them.:
  - Octa Core Processor: 9
  - Quad Core: 6
  - Octa Core: 5
  - SC9863A: 2
  - (28 nm): 1

```
In [97]: df1['processor_name'].value_counts()
         # if we observe the values their are some unrelated values they are
         # Octa Core Processor         9
         # Quad Core                   6
         # Octa Core                   5
         # SC9863A                     2
         # (28 nm)                     1
```

```
Out[97]: processor_name
         Dimensity 700 5G      29
         Helio P35             28
         Snapdragon 8+ Gen1    26
         Snapdragon 8 Gen2     23
         Snapdragon 695        23
                               ..
         Tiger T606             1
         Snapdragon 615         1
         Helio MT6580           1
         Helio  G95             1
         Google  Tensor 3       1
         Name: count, Length: 246, dtype: int64
```

Let's filter only those rows where there is 'core' in the 'processor_name' column.

```
In [98]: df1['processor_name'] = df1['processor_name'].str.strip()
```

```
In [99]: df1[df1['processor_name'].str.contains('Core')]
```

Out[99]:

	index	brand_name	model	price	rating	sim	has_5g	has_nfc	has_ir_blaster	processor	processor_name	num_cores	processor_speed	ram
118	120	tesla	Tesla Pi Phone	69999	83.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	True	True	False	Octa Core Processor	Octa Core Processor	NaN	NaN	16 GB RAM, 512 GB inbuilt
143	145	jio	Jio Phone 3	4499	NaN	Dual Sim, 3G, 4G, VoLTE, Wi-Fi	False	False	False	Quad Core, 1.4 GHz Processor	Quad Core	1.4 GHz Processor	NaN	2 GB RAM, 64 GB inbuilt
						Dual Sim,				Quad				4 GB

In the output, it seems that due to missing processor names, the data in the 'processor_name' column has shifted 'num_cores' data to 'processor_name' column, and 'processor_speed' data into 'num_cores' column, leaving 'processor_speed' with NaN values. I need to perform a right shift of data from the 'processor_name' and 'num_cores' columns. Let's do that.

```
In [100]: df1[df1['processor_name'].str.contains('Core')][['processor_name','num_cores','processor_speed']]
```

Out[100]:

	processor_name	num_cores	processor_speed
118	Octa Core Processor	NaN	NaN
143	Quad Core	1.4 GHz Processor	NaN
188	Quad Core	1.6 GHz Processor	NaN
201	Octa Core	2 GHz Processor	NaN
309	Octa Core	2 GHz Processor	NaN
315	Quad Core	1.3 GHz Processor	NaN
496	Octa Core Processor	NaN	NaN
529	Octa Core Processor	NaN	NaN
587	Octa Core	2 GHz Processor	NaN
758	Quad Core	1.6 GHz Processor	NaN
778	Octa Core	2.2 GHz Processor	NaN
794	Quad Core	1.3 GHz Processor	NaN
825	Octa Core Processor	NaN	NaN
826	Octa Core Processor	NaN	NaN
875	Octa Core Processor	NaN	NaN
948	Octa Core Processor	NaN	NaN
949	Octa Core Processor	NaN	NaN
991	Octa Core	1.8 GHz Processor	NaN
1005	Quad Core	1.3 GHz Processor	NaN

```
In [101]: temp_df = df1[df1['processor_name'].str.contains('Core')][['processor_name','num_cores','processor_speed']].shift(1,axis = 1)
```

```
In [102]: temp_df.index
```

```
Out[102]: Index([ 118,  143,  188,  201,  309,  315,  496,  529,  587,  758,  778,  794,
                  825,  826,  875,  948,  949,  991, 1005, 1019],
                dtype='int64')
```

```
In [103]: # loc[rows,columns] = values (to be inserted)
          df1.loc[temp_df.index,['processor_name','num_cores','processor_speed']] = temp_df.values
```

- Upon searching, I found out that the processor name "(28 nm)" is Mediatek MT6739, and I have replaced it.

```
In [104]: df1[df1['processor_name'] == '(28 nm)']
```

Out[104]:

	index	brand_name	model	price	rating	sim	has_5g	has_nfc	has_ir_blaster	processor	proces
856	858	samsung	Samsung Galaxy A01 Core	4999	NaN	Dual Sim, 3G, 4G, VoLTE, Wi-Fi	False	False	False	(28 nm), Quad Core, 1.5 GHz Processor	

```
In [105]: df1['processor_name'] = df1['processor_name'].replace('(28 nm)','Mediatek MT6739')
```

- Upon searching, I found out that the processor name "SC9863A" is 'Unisoc SC9863A', and I have replaced it.

```
In [106]: df1[df1['processor_name'] == 'SC9863A']
```

Out[106]:

	index	brand_name	model	price	rating	sim	has_5g	has_nfc	has_ir_blaster	processor	processor_name	r
294	296	realme	Realme C11 2021 (4GB RAM + 64GB)	7749	61.0	Dual Sim, 3G, 4G, VoLTE, Wi-Fi	False	False	False	SC9863A, Octa Core, 1.6 GHz Processor	SC9863A	
972	974	realme	Realme C11 2021	6499	61.0	Dual Sim, 3G, 4G, VoLTE, Wi-Fi	False	False	False	SC9863A, Octa Core, 1.6 GHz Processor	SC9863A	

```
In [107]: df1['processor_name'] = df1['processor_name'].replace('SC9863A','Unisoc SC9863A')
```

- **Internal_memory:** Checked the types of internal memory capacities in our smartphones dataset.

```
In [141]: df1['internal_memory'].value_counts()

Out[141]: internal_memory
          128    523
          64     193
          256    157
          32      67
          512     22
          16      12
          1        5
          8        1
          Name: count, dtype: int64
```

Drew only those rows from the dataset where the internal memory of smartphones is less than 32GB and checked their genuineness. Found that a smartphone models with an internal capacity of 1GB is actually 1TB. Since the remaining capacities are in GB, I initially mistook them all to be in GB and removed the 'GB' suffix. However, the 1TB capacity was an exception. Since 1TB equals 1024GB, let us replace 1 with 1024 to correct this.

```
In [143]: df1[df1['internal_memory'] < 32]
```

Out[143]:

	index	brand_name	model	price	rating	sim	has_5g	has_nfc	has_ir_blaster	processor	...	num_cores	processor_speed	ram	ram_capacity
290	292	apple	Apple iPhone 14 Pro Max (1TB)	182999	78.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	True	True	False	Bionic A16, Hexa Core Processor	...	6.0	NaN	6 GB RAM, 1 TB inbuilt	6.0
399	401	itel	itel P36	6490	NaN	Dual Sim, 3G, 4G, VoLTE, Wi-Fi	False	False	False	Spreadtrum SC7731E, Quad Core, 1.3 GHz Processor	...	4.0	1.30	1 GB RAM, 16 GB inbuilt	1.0
						Dual Sim.				Helio MT6580.				1 GB RAM.	

In [144]:
```
df1[df1['internal_memory'] == 1]
# as you can see where ever internal_memory is 1 it is not GB, it is TB
```

Out[144]:

	index	brand_name	model	price	rating	sim	has_5g	has_nfc	has_ir_blaster	processor	...	num_cores	processor_speed	ram	ram_capacity	int
290	292	apple	Apple iPhone 14 Pro Max (1TB)	182999	78.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	True	True	False	Bionic A16, Hexa Core Processor	...	6.0	NaN	6 GB RAM, 1 TB inbuilt	6.0	
781	783	apple	Apple iPhone 13 Pro Max (1TB)	179900	86.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	True	True	False	Bionic A15, Hexa Core, 3.22 GHz Processor	...	6.0	3.22	6 GB RAM, 1 TB inbuilt	6.0	
814	816	apple	Apple iPhone 14 Pro (1TB)	172999	77.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	True	True	False	Bionic A16, Hexa Core Processor	...	6.0	NaN	6 GB RAM, 1 TB inbuilt	6.0	
943	945	samsung	Samsung Galaxy Z Fold 4	163980	NaN	Dual Sim, 3G, 4G,	True	True	False	Snapdragon 8+ Gen1, Octa Core		8.0	3.20	12 GB RAM	12.0	

```
In [145]: temp_df = df1[df1['internal_memory'] == 1]

In [146]: df1.loc[temp_df.index,'internal_memory'] = 1024

In [147]: df1[df1['internal_memory'] == 1]
```

Out[147]:

index	brand_name	model	price	rating	sim	has_5g	ha

0 rows × 22 columns

```
In [148]: df1['internal_memory'].value_counts()
```

Out[148]:

```
internal_memory
128.0     523
64.0      193
256.0     157
32.0       67
512.0      22
16.0       12
1024.0      5
8.0         1
Name: count, dtype: int64
```

- Checked for smartphones with ram_capacity more than 18 GB to verify their genuineness. Found that at indexes 486,627,439, 762, and 483, they contain internal memory data in the ram column due to the absence of RAM data while splitting the 'Ram' column.

```
In [134]: df1[df1['ram_capacity'] > 18]
```

Out[134]:

	index	brand_name	model	price	rating	sim	has_5g	has_nfc	has_ir_blaster	processor	...	num_cores	processor_speed	ram	ram_capacity	inte
439	441	apple	Apple iPhone SE 3 2022	43900	NaN	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	True	True	False	Bionic A15, Hexa Core, 3.22 GHz Processor	...	6.0	3.22	64 GB inbuilt	64.0	
483	485	huawei	Huawei Mate 50 RS Porsche Design	239999	81.0	Dual Sim, 3G, 4G, VoLTE, Wi-Fi, NFC, IR Blaster	False	True	True	Snapdragon 8+ Gen1, Octa Core, 3.2 GHz Processor	...	8.0	3.20	512 GB inbuilt	512.0	
486	488	nokia	Nokia 8000 4G	6899	NaN	Dual Sim, 3G, 4G, Wi-Fi	False	False	False	Snapdragon 210, Quad Core, 1.1 GHz Processor	...	4.0	1.10	512 MB RAM, 4 GB inbuilt	512.0	
627	629	nokia	Nokia 225 4G	3589	NaN	Dual Sim, 3G, 4G, VoLTE	False	False	False	Unisoc T117	...	NaN	NaN	64 MB RAM, 128 MB inbuilt	64.0	
762	764	apple	Apple iPhone	49900	60.0	Dual Sim, 3G, 4G,	True	True	False	Bionic A15, Hexa Core,	...	6.0	3.22	64 GB	64.0	

searched for the correct values of ‘ ram_capacity’ and ‘internal_memory’ for indexes 439, 762, and 483, and inserted them.

```
In [135]: df1.loc[[439,762],['ram_capacity','internal_memory']] = [[4.0,'64'],[4.0,'64']]
```

```
In [137]: df1.loc[[483],['ram_capacity','internal_memory']] = [12.0,'512']
```

Indexes 486 and 627 are not smartphones; they are feature phones. Since our dataset is supposed to contain only smartphones, I dropped these rows.

```
In [139]: # they are not smartphones they are feature phones
          df1.drop([486,627],inplace = True)
```

- ## Data tidying:
- **brand_name:**
  Let's convert the 'brand_name' column to lowercase to address case inconsistencies, where the same brand might be represented differently, such as 'SAMSUNG' and 'Samsung.' By converting everything to lowercase, it ensures that all instances of the same brand are treated as the same value.

In [83]:
```
df1.head()
```

Out[83]:

	index	brand_name	model	price	rating	sim	processor	ram	battery	display	camera	card	os
0	2	OnePlus	OnePlus 11 5G	54999	89.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Snapdragon 8 Gen2, Octa Core, 3.2 GHz Processor	12 GB RAM, 256 GB inbuilt	5000 mAh Battery with 100W Fast Charging	6.7 inches, 1440 x 3216 px, 120 Hz Display wit...	50 MP + 48 MP + 32 MP Triple Rear & 16 MP Fron...	Memory Card Not Supported	Android v13
1	3	OnePlus	OnePlus Nord CE 2 Lite 5G	19989	81.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	Snapdragon 695, Octa Core, 2.2 GHz Processor	6 GB RAM, 128 GB inbuilt	5000 mAh Battery with 33W Fast Charging	6.59 inches, 1080 x 2412 px, 120 Hz Display wi...	64 MP + 2 MP + 2 MP Triple Rear & 16 MP Front ...	Memory Card (Hybrid), upto 1 TB	Android v12
2	4	Samsung	Samsung Galaxy A14 5G	16499	75.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	Exynos 1330, Octa Core, 2.4 GHz Processor	4 GB RAM, 64 GB inbuilt	5000 mAh Battery with 15W Fast Charging	6.6 inches, 1080 x 2408 px, 90 Hz Display with...	50 MP + 2 MP + 2 MP Triple Rear & 13 MP Front ...	Memory Card Supported, upto 1 TB	Android v13
3	5	Motorola	Motorola Moto G62 5G	14999	81.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	Snapdragon 695, Octa Core, 2.2 GHz Processor	6 GB RAM, 128 GB inbuilt	5000 mAh Battery with Fast Charging	6.55 inches, 1080 x 2400 px, 120 Hz Display wi...	50 MP + 8 MP + 2 MP Triple Rear & 16 MP Front ...	Memory Card (Hybrid), upto 1 TB	Android v12
4	6	Realme	Realme 10 Pro Plus	24999	82.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	Dimensity 1080, Octa Core, 2.6 GHz Processor	6 GB RAM, 128 GB inbuilt	5000 mAh Battery with 67W Fast Charging	6.7 inches, 1080 x 2412 px, 120 Hz Display wit...	108 MP + 8 MP + 2 MP Triple Rear & 16 MP Front...	Memory Card Not Supported	Android v13

In [84]:
```
# lets convert brand_name into small letters beacause we
# places as 'Samsung' this would be considered as two dif

df1['brand_name'] = df1['brand_name'].str.lower()
```

In [85]:
```
df1.head()
```

Out[85]:

	index	brand_name	model	price	rating	sim	processor	ram	battery	display	camera	card	os
0	2	oneplus	OnePlus 11 5G	54999	89.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Snapdragon 8 Gen2, Octa Core, 3.2 GHz Processor	12 GB RAM, 256 GB inbuilt	5000 mAh Battery with 100W Fast Charging	6.7 inches, 1440 x 3216 px, 120 Hz Display wit...	50 MP + 48 MP + 32 MP Triple Rear & 16 MP Fron...	Memory Card Not Supported	Android v13
1	3	oneplus	OnePlus Nord CE 2 Lite 5G	19989	81.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	Snapdragon 695, Octa Core, 2.2 GHz Processor	6 GB RAM, 128 GB inbuilt	5000 mAh Battery with 33W Fast Charging	6.59 inches, 1080 x 2412 px, 120 Hz Display wi...	64 MP + 2 MP + 2 MP Triple Rear & 16 MP Front ...	Memory Card (Hybrid), upto 1 TB	Android v12
2	4	samsung	Samsung Galaxy A14 5G	16499	75.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	Exynos 1330, Octa Core, 2.4 GHz Processor	4 GB RAM, 64 GB inbuilt	5000 mAh Battery with 15W Fast Charging	6.6 inches, 1080 x 2408 px, 90 Hz Display with...	50 MP + 2 MP + 2 MP Triple Rear & 13 MP Front ...	Memory Card Supported, upto 1 TB	Android v13
3	5	motorola	Motorola Moto G62 5G	14999	81.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	Snapdragon 695, Octa Core, 2.2 GHz Processor	6 GB RAM, 128 GB inbuilt	5000 mAh Battery with Fast Charging	6.55 inches, 1080 x 2400 px, 120 Hz Display wi...	50 MP + 8 MP + 2 MP Triple Rear & 16 MP Front ...	Memory Card (Hybrid), upto 1 TB	Android v12
4	6	realme	Realme 10 Pro Plus	24999	82.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	Dimensity 1080, Octa Core, 2.6 GHz Processor	6 GB RAM, 128 GB inbuilt	5000 mAh Battery with 67W Fast Charging	6.7 inches, 1080 x 2412 px, 120 Hz Display wit...	108 MP + 8 MP + 2 MP Triple Rear & 16 MP Front...	Memory Card Not Supported	Android v13

- **processor_name:**
  Converted all letters in the ‘processor_name’ column to lowercase to avoid issues caused by case sensitivity.

In [109]:
```
processor_brand = df1['processor_name'].str.split(' ').str.get(0).str.lower()
```

In [110]:
```
df1.insert(11,'processor_brand',processor_brand)
```

- ## Splitting Columns:
- **Brand_name:** This is our current dataframe, but it needs more columns.

```
In [80]: df1.head()
```

Out[80]:

	index	model	price	rating	sim	processor	ram	battery	display	camera	card	os
0	2	OnePlus 11 5G	54999	89.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Snapdragon 8 Gen2, Octa Core, 3.2 GHz Processor	12 GB RAM, 256 GB inbuilt	5000 mAh Battery with 100W Fast Charging	6.7 inches, 1440 x 3216 px, 120 Hz Display wit...	50 MP + 48 MP + 32 MP Triple Rear & 16 MP Fron...	Memory Card Not Supported	Android v13
1	3	OnePlus Nord CE 2 Lite 5G	19989	81.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	Snapdragon 695, Octa Core, 2.2 GHz Processor	6 GB RAM, 128 GB inbuilt	5000 mAh Battery with 33W Fast Charging	6.59 inches, 1080 x 2412 px, 120 Hz Display wi...	64 MP + 2 MP + 2 MP Triple Rear & 16 MP Front ...	Memory Card (Hybrid), upto 1 TB	Android v12
2	4	Samsung Galaxy A14 5G	16499	75.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	Exynos 1330, Octa Core, 2.4 GHz Processor	4 GB RAM, 64 GB inbuilt	5000 mAh Battery with 15W Fast Charging	6.6 inches, 1080 x 2408 px, 90 Hz Display with...	50 MP + 2 MP + 2 MP Triple Rear & 13 MP Front ...	Memory Card Supported, upto 1 TB	Android v13
3	5	Motorola Moto G62 5G	14999	81.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	Snapdragon 695, Octa Core, 2.2 GHz Processor	6 GB RAM, 128 GB inbuilt	5000 mAh Battery with Fast Charging	6.55 inches, 1080 x 2400 px, 120 Hz Display wi...	50 MP + 8 MP + 2 MP Triple Rear & 16 MP Front ...	Memory Card (Hybrid), upto 1 TB	Android v12

I need a column for brand names, so let us create one using the 'model' column."

```
In [81]: # lets make a new column named brand from model as you can see there is brand name infront of every column
         brand_names = df['model'].str.split(' ').str.get(0)
```

```
In [82]: # dataFrame.insert(index_position,'column_name',series_name)
         df1.insert(1,'brand_name',brand_names)
```

```
In [83]: df1.head()
```

Out[83]:

	index	brand_name	model	price	rating	sim	processor	ram	battery	display	camera	card	os
0	2	OnePlus	OnePlus 11 5G	54999	89.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Snapdragon 8 Gen2, Octa Core, 3.2 GHz Processor	12 GB RAM, 256 GB inbuilt	5000 mAh Battery with 100W Fast Charging	6.7 inches, 1440 x 3216 px, 120 Hz Display wit...	50 MP + 48 MP + 32 MP Triple Rear & 16 MP Fron...	Memory Card Not Supported	Android v13
1	3	OnePlus	OnePlus Nord CE 2 Lite 5G	19989	81.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	Snapdragon 695, Octa Core, 2.2 GHz Processor	6 GB RAM, 128 GB inbuilt	5000 mAh Battery with 33W Fast Charging	6.59 inches, 1080 x 2412 px, 120 Hz Display wi...	64 MP + 2 MP + 2 MP Triple Rear & 16 MP Front ...	Memory Card (Hybrid), upto 1 TB	Android v12
2	4	Samsung	Samsung Galaxy A14 5G	16499	75.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	Exynos 1330, Octa Core, 2.4 GHz Processor	4 GB RAM, 64 GB inbuilt	5000 mAh Battery with 15W Fast Charging	6.6 inches, 1080 x 2408 px, 90 Hz Display with...	50 MP + 2 MP + 2 MP Triple Rear & 13 MP Front ...	Memory Card Supported, upto 1 TB	Android v13
3	5	Motorola	Motorola Moto G62 5G	14999	81.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	Snapdragon 695, Octa Core, 2.2 GHz Processor	6 GB RAM, 128 GB inbuilt	5000 mAh Battery with Fast Charging	6.55 inches, 1080 x 2400 px, 120 Hz Display wi...	50 MP + 8 MP + 2 MP Triple Rear & 16 MP Front ...	Memory Card (Hybrid), upto 1 TB	Android v12
4	6	Realme	Realme 10 Pro Plus	24999	82.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	Dimensity 1080, Octa Core, 2.6 GHz Processor	6 GB RAM, 128 GB inbuilt	5000 mAh Battery with 67W Fast Charging	6.7 inches, 1080 x 2412 px, 120 Hz Display wit...	108 MP + 8 MP + 2 MP Triple Rear & 16 MP Front...	Memory Card Not Supported	Android v13

- **has_5G, has_NFC, has_IR_Blaster:**
  Let's examine the 'sim' column, which contains various information like dual sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC, and IR Blaster. Most phones today are dual SIM and support 3G and 4G, so the key differentiating factors are 5G, NFC, and IR Blaster, which is not available on all phones.

  I will create three new columns from the 'sim' column:

'has_5G' (True/False)
'has_NFC' (True/False)
'has_IR_Blaster' (True/False)

```
In [88]: has_5g = df1['sim'].str.contains('5G')
         has_nfc = df1['sim'].str.contains('NFC')
         has_ir_blaster = df1['sim'].str.contains('IR Blaster')
```

```
In [89]: df1.insert(6,'has_5g',has_5g)
         df1.insert(7,'has_nfc',has_nfc)
         df1.insert(8,'has_ir_blaster',has_ir_blaster)
```

```
In [90]: df1.head()
```

Out[90]:

	index	brand_name	model	price	rating	sim	has_5g	has_nfc	has_ir_blaster	processor	ram	battery	display	camera	card	os
0	2	oneplus	OnePlus 11 5G	54999	89.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	True	True	False	Snapdragon 8 Gen2, Octa Core, 3.2 GHz Processor	12 GB RAM, 256 GB inbuilt	5000 mAh Battery with 100W Fast Charging	6.7 inches, 1440 x 3216 px, 120 Hz Display wit...	50 MP + 48 MP + 32 MP Triple Rear & 16 MP Fron...	Memory Card Not Supported	Android v13
1	3	oneplus	OnePlus Nord CE 2 Lite 5G	19989	81.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	True	False	False	Snapdragon 695, Octa Core, 2.2 GHz Processor	6 GB RAM, 128 GB inbuilt	5000 mAh Battery with 33W Fast Charging	6.59 inches, 1080 x 2412 px, 120 Hz Display wi...	64 MP + 2 MP + 2 MP Triple Rear & 16 MP Front ...	Memory Card (Hybrid), upto 1 TB	Android v12
2	4	samsung	Samsung Galaxy A14 5G	16499	75.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	True	False	False	Exynos 1330, Octa Core, 2.4 GHz Processor	4 GB RAM, 64 GB inbuilt	5000 mAh Battery with 15W Fast Charging	6.6 inches, 1080 x 2408 px, 90 Hz Display with...	50 MP + 2 MP + 2 MP Triple Rear & 13 MP Front ...	Memory Card Supported, upto 1 TB	Android v13

- **processor_name , num_cores,processor_speed:**
  In the 'processor' column, there is information about the processor name, number of cores, and processor speed, separated by commas. Let's split this information into three separate columns.

```
In [92]: processor_name = df1['processor'].str.split(',').str.get(0)
```

```
In [93]: num_cores = df1['processor'].str.split(',').str.get(1)
```

```
In [94]: processor_speed = df1['processor'].str.split(',').str.get(2)
```

```
In [95]: df1.insert(10,'processor_name',processor_name)
         df1.insert(11,'num_cores',num_cores)
         df1.insert(12,'processor_speed',processor_speed)
```

In [96]:
```
df1.head()
```

Out[96]:

	index	brand_name	model	price	rating	sim	has_5g	has_nfc	has_ir_blaster	processor	processor_name	num_cores	processor_speed	ram	b
0	2	oneplus	OnePlus 11 5G	54999	89.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	True	True	False	Snapdragon 8 Gen2, Octa Core, 3.2 GHz Processor	Snapdragon 8 Gen2	Octa Core	3.2 GHz Processor	12 GB RAM, 256 GB inbuilt	E Ch
1	3	oneplus	OnePlus Nord CE 2 Lite 5G	19989	81.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	True	False	False	Snapdragon 695, Octa Core, 2.2 GHz Processor	Snapdragon 695	Octa Core	2.2 GHz Processor	6 GB RAM, 128 GB inbuilt	E Ch

- **Battery_capacity, fast_charging, fast_charging_availabile:**

  In the 'battery' column, I have two pieces of information: one is the battery capacity and the second is whether it has fast charging or not. If there is fast charging, it includes the wattage. If there is no fast charging, I will consider it as '0' watts.

  Let's create three columns from this:

  1. 'battery_capacity'
  2. 'fast_charging' (if there is no fast charging, then 0 watts).
  3. ‘fast_charging_available’

In [149]:
```
df1.head()
```

	sim	has_5g	has_nfc	has_ir_blaster	processor	...	num_cores	processor_speed	ram	ram_capacity	internal_memory	battery
	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	True	True	False	Snapdragon 8 Gen2, Octa Core, 3.2 GHz Processor	...	8.0	3.2	12 GB RAM, 256 GB inbuilt	12.0	256.0	5000 mAh Battery with 100W Fast Charging
	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	True	False	False	Snapdragon 695, Octa Core, 2.2 GHz Processor	...	8.0	2.2	6 GB RAM, 128 GB inbuilt	6.0	128.0	5000 mAh Battery with 33W Fast Charging
	Dual											5000

```
In [150]: # In battery column we have two information one is b
          # is fast charging then what are the Watts if there
          # lets make 2 columns from this 1st battery capacity
          # as you can see values in column are breaking at wi

          df1['battery'].str.strip().str.split('with')

          # as you can see in the output there are 2 values in
```

```
Out[150]: 0         [5000 mAh Battery ,  100W Fast Charging]
          1          [5000 mAh Battery ,  33W Fast Charging]
          2          [5000 mAh Battery ,  15W Fast Charging]
          3              [5000 mAh Battery ,  Fast Charging]
          4          [5000 mAh Battery ,  67W Fast Charging]
                                    ...
          1015     [5000 mAh Battery ,  68.2W Fast Charging]
          1016     [5000 mAh Battery ,  22.5W Fast Charging]
          1017       [5080 mAh Battery ,  67W Fast Charging]
          1018           [5000 mAh Battery ,  Fast Charging]
          1019           [5000 mAh Battery ,  Fast Charging]
          Name: battery, Length: 980, dtype: object
```

As you can see in the output, there are two values in most places but only one value in a few places. Not all devices have fast charging.

```
In [151]: df1['battery'].str.strip().str.split('with').str.get(0).str.strip().str.split(' ').get(0)
```

```
Out[151]: ['5000\u2009mAh', 'Battery']
```

```
In [152]: df1['battery'].str.strip().str.split('with').str.get(0).str.strip().str.findall(r'\b(\d+)\b').str.get(0)
```

```
Out[152]: 0       5000
          1       5000
          2       5000
          3       5000
          4       5000
                  ...
          1015    5000
          1016    5000
          1017    5080
          1018    5000
          1019    5000
          Name: battery, Length: 980, dtype: object
```

Extracted only the battery capacity from the 'battery' column and created a new column called 'battery_capacity' in the dataset.

```
In [151]: df1['battery'].str.strip().str.split('with').str.get(0).str.strip().str.split(' ').get(0)
```

```
Out[151]: ['5000\u2009mAh', 'Battery']
```

```
In [153]: battery_capacity = df1['battery'].str.strip().str.split('with').str.get(0).str.strip().
          str.findall(r'\b(\d+)\b').str.get(0).astype(float)
```

```
In [155]: df1.insert(18,'battery_capacity',battery_capacity)
```

The second piece of information in the 'battery' column is about fast charging. Let's extract this information and create a new column called 'fast_charging' in the dataset.

```
In [156]: df1['battery'].str.strip().str.split('with').str.get(1)
```

```
Out[156]: 0          100W Fast Charging
          1           33W Fast Charging
          2           15W Fast Charging
          3               Fast Charging
          4           67W Fast Charging
                        ...
          1015      68.2W Fast Charging
          1016      22.5W Fast Charging
          1017        67W Fast Charging
          1018            Fast Charging
          1019            Fast Charging
          Name: battery, Length: 980, dtype: object
```

```
In [157]: fast_charging = df1['battery'].str.strip().str.split('with').str.get(1).str.strip().str.findall(r'\d{2,3}')
```

```
In [158]: df1.insert(18,'fast_charging',fast_carging)
```

Values in the 'fast_charging' column are in list form.

```
In [160]: df1['fast_charging'].value_counts()

Out[160]: fast_carging
          [33]     152
          [18]     128
          []        68
          [67]      65
          [25]      53
          [120]     46
          [15]      43
          [80]      42
          [66]      37
          [10]      33
          [30]      32
          [65]      30
          [44]      23
          [45]      17
          [20]      10
          [68]       8
          [100]      7
          [150]      7
          [125]      6
          [40]       5
          [22]       5
          [60]       4
          [55]       3
          [50]       2
          [210]      2
          [21]       2
          [180]      1
          [200]      1
          [240]      1
          [19]       1
          [135]      1
          [165]      1
```

I wrote a function to extract only the values from the list and return them. If there is no value or more than one value in the list, It returned 0. If there is no list in the row, I returned -1. I applied this function to the 'fast_charging' column.

In [161]:
```
def fast_charging_extractor(item):
    if type(item) == list:
        if len(item) == 1:
            return item[0]
        else:
            return 0
    else:
        return -1
```

Converted 'fast_charging' to integer datatype.

In [162]:
```
df1['fast_charging'] = df1['fast_charging'].apply(fast_charging_extractor).astype(int)
```

```
In [164]: df1['fast_charging'].value_counts()

Out[164]: fast_charging
           33     152
          -1      143
           18     128
           0       68
           67      65
           25      53
           120     46
           15      43
           80      42
           66      37
           10      33
           30      32
           65      30
           44      23
           45      17
           20      10
           68       8
           100      7
           150      7
           125      6
           40       5
           22       5
           60       4
           55       3
           50       2
           210      2
           21       2
           180      1
           200      1
           240      1
           19       1
```

Replaced 0 and -1 values in the 'fast_charging' column with NaN.

In [165]:
```
df1['fast_charging'] = df1['fast_charging'].replace(0,np.nan).replace(-1,np.nan)
```

In [166]:
```
df1['fast_charging'].value_counts()
```

Out[166]:

fast_charging	
33.0	152
18.0	128
67.0	65
25.0	53
120.0	46
15.0	43
80.0	42
66.0	37
10.0	33
30.0	32
65.0	30
44.0	23
45.0	17
20.0	10
68.0	8
100.0	7
150.0	7
125.0	6
22.0	5
40.0	5
60.0	4
55.0	3
21.0	2
210.0	2
50.0	2
180.0	1
200.0	1
240.0	1
19.0	1
135.0	1
165.0	1
27.0	1

```
In [167]: df1.head(2)
```

Out[167]:

	index	brand_name	model	price	rating	sim	has_5g	has_nfc	has_ir_blaster	processor	...	ram	ram_capacity	internal_memory	battery
0	2	oneplus	OnePlus 11 5G	54999	89.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	True	True	False	Snapdragon 8 Gen2, Octa Core, 3.2 GHz Processor	...	12 GB RAM, 256 GB inbuilt	12.0	256.0	5000 mAh Battery with 100W Fast Charging
1	3	oneplus	OnePlus Nord CE 2 Lite 5G	19989	81.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	True	False	False	Snapdragon 695, Octa Core, 2.2 GHz Processor	...	6 GB RAM, 128 GB inbuilt	6.0	128.0	5000 mAh Battery with 33W Fast Charging

Since not all models have fast charging capability, I created a new column 'fast_charging_available' and filled it with the values 0 for no fast charging feature and 1 for fast charging feature.

```
In [168]: def fast_charging_extractor(text):
              if text is None:
                  return 0
              if 'Fast Charging' in text:
                  return 1
              else:
                  return 0
```

```
In [169]: fast_charging_available = df1['battery']
```

```
In [170]: fast_charging_available = fast_charging_available.apply(fast_charging_extractor)
```

```
In [171]: fast_charging_available.value_counts()
```

```
Out[171]: battery
          1    837
          0    143
          Name: count, dtype: int64
```

```
In [172]: df1.insert(18,'fast_charging_available',fast_charging_available)
```

- **Screen_size, resolution, refresh_rate:**
  3 new columns were created from display column they are screen_size, resolution and refresh_rate columns.

```
In [173]: df1.head(2)
```

Out[173]:

has_ir_blaster	processor	...	ram_capacity	internal_memory	battery	fast_charging_available	fast_charging	battery_capacity	display	camera	card	os
False	Snapdragon 8 Gen2, Octa Core, 3.2 GHz Processor	...	12.0	256.0	5000 mAh Battery with 100W Fast Charging	1	100.0	5000.0	6.7 inches, 1440 x 3216 px, 120 Hz Display wit...	50 MP + 48 MP + 32 MP Triple Rear & 16 MP Fron...	Memory Card Not Supported	Android v13
False	Snapdragon 695, Octa Core, 2.2 GHz Processor	...	6.0	128.0	5000 mAh Battery with 33W Fast Charging	1	33.0	5000.0	6.59 inches, 1080 x 2412 px, 120 Hz Display wi...	64 MP + 2 MP + 2 MP Triple Rear & 16 MP Front ...	Memory Card (Hybrid), upto 1 TB	Android v12

Screen size

First, I extracted the screen size from the 'display' column and created a new column called 'screen_size'.

```
In [174]: screen_size = df1['display'].str.strip().str.split(',').str.get(0).str.split(' ').str.get(0).astype(float)
```

```
In [175]: df1.insert(21,'screen_size',screen_size)
```

Resolution

Second, I extracted the resolution from the 'display' column and created a new column called 'resolution'.'

```
In [176]: resolution = df1['display'].str.strip().str.split(',').str.get(1).str.split('px').str.get(0)
```

```
In [177]: df1.insert(22,'resolution',resolution)
```

Refresh rate

Third, I extracted the refresh rate from the 'display' column and created a new column called 'refresh_rate'.

```
In [178]: df1['display'].str.strip().str.split(',').str.get(2).str.strip().str.findall(r'\d{2,3}').str.get(0)
```

```
Out[178]: 0       120
          1       120
          2        90
          3       120
          4       120
                 ...
          1015    120
          1016    NaN
          1017    144
          1018    NaN
          1019    NaN
          Name: display, Length: 980, dtype: object
```

In [179]:
```
# where ever there isn't a refresh rate it will become 60 according to industry standards
refresh_rate = df1['display'].str.strip().str.split(',').str.get(2).str.strip().str.
findall(r'\d{2,3}').str.get(0).apply(lambda x:60 if pd.isna(x) else x).astype(int)
```

In [180]:
```
df1.insert(22,'refresh_rate',refresh_rate)
```

- **`Num_front_cameras`, `Priamry_camera_front`, `Num_rear_cameras`, `priamry_camera_rear`:**

  The 'Camera' column contains data for both front and rear cameras. From this, I split it into four new columns in the dataset:

  - `num_front_cameras`: Number of front cameras
  - `priamry_camera_front`: Primary front camera
  - `num_rear_cameras`: Number of rear cameras
  - `priamry_camera_rear`: Primary rear camera

In [181]:
```
df1['camera'].value_counts()
```

Out[181]:
```
camera
50 MP + 8 MP + 2 MP Triple Rear & 16 MP Front Camera                 40
64 MP + 8 MP + 2 MP Triple Rear & 16 MP Front Camera                 38
50 MP + 2 MP + 2 MP Triple Rear & 16 MP Front Camera                 34
13 MP + 2 MP Dual Rear & 5 MP Front Camera                           23
12 MP + 12 MP Dual Rear & 12 MP Front Camera                         20
                                                                     ..
12 MP + 12 MP + 12 MP Triple Rear & 10 MP + 4 MP Dual Front Camera     1
64 MP + 13 MP + 12 MP Triple Rear & 32 MP Front Camera                 1
50 MP + 13 MP + 2 MP Triple Rear & 16 MP Front Camera                  1
48 MP + 2 MP + 2 MP Triple Rear & 32 MP Front Camera                   1
48 MP + 2 MP + Depth Sensor Triple Rear & 8 MP Front Camera            1
Name: count, Length: 284, dtype: int64
```

`num_rear_cameras`:

The front camera and rear camera data are split using '&'. Since the rear camera data is in the first position, I extracted it and changed 'Quad' to 4, 'Triple' to 3, 'Dual' to 2, and not mentioned to 1 using a function `camera_extractor`, and entered this data as a new column `num_rear_cameras`.

```
In [182]: df1['camera'].str.strip().str.split('&').str.get(0)
```

```
Out[182]: 0              50 MP + 48 MP + 32 MP Triple Rear
          1                64 MP + 2 MP + 2 MP Triple Rear
          2                50 MP + 2 MP + 2 MP Triple Rear
          3                50 MP + 8 MP + 2 MP Triple Rear
          4               108 MP + 8 MP + 2 MP Triple Rear
                                    ...
          1015             64 MP + 8 MP + 2 MP Triple Rear
          1016    48 MP + 2 MP + Depth Sensor Triple Rear
          1017             64 MP + 8 MP + 2 MP Triple Rear
          1018            108 MP + 8 MP + 2 MP Triple Rear
          1019             64 MP + 8 MP + 5 MP Triple Rear
          Name: camera, Length: 980, dtype: object
```

```
In [183]: def camera_extractor(text):
              if 'Quad' in text:
                  return 4
              elif 'Triple' in text:
                  return 3
              elif 'Dual' in text:
                  return 2
              else:
                  return 1
```

```
In [184]: num_rear_cameras = df1['camera'].str.strip().str.split('&').str.get(0).apply(camera_extractor)
```

```
In [185]: df1.insert(25,'num_rear_cameras',num_rear_cameras)
```

num_front_cameras:

created a new column `num_front_cameras` by extracting second position data from the camera column, changed ('Quad' to 4, 'Triple' to 3, 'Dual' to 2, and not mentioned to 1) using a function `camera_exteactor_f`, and entered this data.

In [186]:
```
def camera_extractor_f(text):
    if 'Quad' in text:
        return 4
    elif 'Triple' in text:
        return 3
    elif 'Dual' in text:
        return 2
    elif 'missing' in text:
        return 'missing'
    else:
        return 1
```

In [187]:
```
num_front_cameras = df1['camera'].str.strip().str.split('&').str.get(1).str.strip().fillna('missing').apply(camera_extractor_f)
```

In [188]:
```
df1.head()
```

Out[188]:

	index	brand_name	model	price	rating	sim	has_5g	has_nfc	has_ir_blaster	processor	...	fast_charging	battery_capacity	screen_size	refresh_rate
0	2	oneplus	OnePlus 11 5G	54999	89.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	True	True	False	Snapdragon 8 Gen2, Octa Core, 3.2 GHz Processor	...	100.0	5000.0	6.70	120
1	3	oneplus	OnePlus Nord CE 2 Lite 5G	19989	81.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	True	False	False	Snapdragon 695, Octa Core, 2.2 GHz Processor	...	33.0	5000.0	6.59	120

In [189]:
```
df1.insert(26,'num_front_cameras',num_front_cameras)
```

In [191]:
```
df1[df1['num_front_cameras'] == 'missing']
```

Out[191]:

has_ir_blaster	processor	...	battery_capacity	screen_size	refresh_rate	resolution	display	num_rear_cameras	num_front_cameras	camera	card	os
False	Snapdragon 8+ Gen1, Octa Core, 3.2 GHz Processor	...	5000.0	7.10	120	1792 x 1920	7.1 inches, 1792 x 1920 px, 120 Hz Display wit...	2	missing	Foldable Display, Dual Display	Memory Card Not Supported	Android v12
False	Snapdragon 855+, Octa Core, 2.96 GHz Processor	...	4050.0	7.92	60	2088 x 2250	7.92 inches, 2088 x 2250 px Display	3	missing	108 MP + 20 MP + 12 MP Triple Rear Camera	Memory Card Not Supported	Android v10
True	Kirin 990, Octa Core, 2.86 GHz Processor	...	4500.0	8.00	60	2200 x 2480	8 inches, 2200 x 2480 px Display	4	missing	48 MP Quad Rear Camera	Memory Card (Hybrid), upto 256 GB	NaN

In the 'Camera' column, there is unrelated data for an Oppo model, specifically 'foldable display'. I performed a search and replaced it with the correct data.

```
In [193]: df1[df1['camera'] == 'Foldable Display, Dual Display']
```

Out[193]:

	index	brand_name	model	price	rating	sim	has_5g	has_nfc	has_ir_blaster	processor	...
69	71	oppo	Oppo Find N Fold	99990	NaN	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	True	True	False	Snapdragon 8+ Gen1, Octa Core, 3.2 GHz Processor	...

1 rows × 30 columns

```
In [194]: temp_df = df1[df1['camera'] == 'Foldable Display, Dual Display']

In [195]: df1.loc[temp_df.index,'camera'] = '50'
```

primary_camera_rear:

created new column named `primary_camera_rear` this column only stores the megapixel of the rear camera's main camera

```
In [196]: df1['camera'].str.split( ).str.get(0)

Out[196]: 0        50
          1        64
          2        50
          3        50
          4       108
                 ...
          1015     64
          1016     48
          1017     64
          1018    108
          1019     64
          Name: camera, Length: 980, dtype: object

In [197]: primary_camera_rare = df1['camera'].str.split( ).str.get(0)

In [199]: df1.insert(27,'primary_camera_rare',primary_camera_rare)
```

Primary_camera_front:

Created a new column named `primary_camera_front` to store only the megapixels of the front camera's main camera."

```
In [200]: df1['camera'].str.split('&').str.get(1).str.strip().str.split(' ').str.get(0).str.
          replace('\u2009',' ').str.split(' ').str.get(0)
```

```
Out[200]: 0       16
          1       16
          2       13
          3       16
          4       16
                  ..
          1015    16
          1016     8
          1017    16
          1018    32
          1019    32
          Name: camera, Length: 980, dtype: object
```

```
In [201]: primary_camera_front = df1['camera'].str.split('&').str.get(1).str.strip().str.
          split(' ').str.get(0).str.replace('\u2009',' ').str.split(' ').str.get(0)
```

```
In [202]: df1.insert(28,'primary_camera_front',primary_camera_front)
```

- **extended_upto and extended_memory_available:**
  In the 'card' column, there are two pieces of information: extended_upto and extended_memory_available. I split this information into two new columns.

```
In [204]: df1['card'].value_counts()

Out[204]: card
          Memory Card Not Supported                354
          Memory Card Supported, upto 1 TB         171
          Memory Card Supported, upto 512 GB       105
          Memory Card (Hybrid), upto 1 TB           91
          Memory Card Supported, upto 256 GB        89
          Memory Card Supported                     88
          Memory Card (Hybrid)                      31
          Memory Card (Hybrid), upto 256 GB         15
          Memory Card (Hybrid), upto 512 GB         11
          Memory Card Supported, upto 128 GB         7
          Memory Card Supported, upto 2 TB           5
          Memory Card (Hybrid), upto 128 GB          3
          Memory Card Supported, upto 32 GB          3
          Memory Card (Hybrid), upto 64 GB           3
          Memory Card (Hybrid), upto 2 TB            3
          Memory Card Supported, upto 1000 GB        1
          Name: count, dtype: int64
```

First, I extracted the extended_upto information and created a new column called extended_upto. To do this, I created a card_extention_extractor function. This function works as follows:

- ➢ If the input data is in GB, it returns only the numerical part.
- ➢ If the data is in TB, it multiplies the numerical part by 1024 and returns the result.
- ➢ If the input data does not contain GB or TB, it returns NaN.
- ➢ If the input data is not a string, it returns NaN.

```
In [206]: df1['card'].str.split('upto').str.get(1).str.strip()
```

```
Out[206]: 0        NaN
          1       1 TB
          2       1 TB
          3       1 TB
          4        NaN
                  ...
          1015     NaN
          1016    1 TB
          1017     NaN
          1018    1 TB
          1019    1 TB
          Name: card, Length: 980, dtype: object
```

```
In [207]: def card_extention_extractor(text):
              if isinstance(text, str):
                  if 'GB' in text:
                      return text.replace('GB','')
                  elif 'TB' in text:
                      return str(float(text.replace('TB','')) * 1024)
                  else:
                      return np.nan
              else:
                  return np.nan
```

```
In [208]: extended_upto = df1['card'].str.split('upto').str.get(1).str.strip().apply(card_extention_extractor)
```

converted the extended_upto column to float and inserted it into the dataset.

```
In [213]: extended_upto = extended_upto.astype(float)
```

```
In [216]: extended_upto.value_counts()
```

```
Out[216]: card
          1024.0    262
          512.0     116
          256.0     104
          128.0      10
          2048.0      8
          32.0        3
          64.0        3
          1000.0      1
          Name: count, dtype: int64
```

Inserted the new column extended_upto into the dataset.

```
In [217]: df1.insert(30,'extended_upto',extended_upto)
```

Next, I created a new column called extended_memory_available from the 'card' column. For this, I used the extended_memory_avilable_extractor function, which returns 0 if the input text contains 'Memory Card Not Supported', and 1 otherwise. This new column was then added to the dataset.

```
In [219]: df1['card'].str.split(',').str.get(0).str.strip().value_counts()
```

```
Out[219]: card
          Memory Card Supported        469
          Memory Card Not Supported    354
          Memory Card (Hybrid)         157
          Name: count, dtype: int64
```

```
In [220]: def extended_memory_available_extractor(text):
              if text == 'Memory Card Not Supported':
                  return 0
              else:
                  return 1
```

```
In [221]: extended_memory_available = df1['card'].str.split(',').str.get(0).str.strip().apply(extended_memory_available_extractor)
```

```
In [222]: extended_memory_available.value_counts()

Out[222]: card
          1    626
          0    354
          Name: count, dtype: int64
```

```
In [224]: df1.insert(30,'extended_memory_available',extended_memory_available)
```

Splitting the columns is completed, so I dropped the original columns from where the columns are splitted.

```
In [226]: cols_to_drop = ['index','sim','processor','ram','battery','display','camera','card']
```

```
In [227]: df1.drop(columns = cols_to_drop,inplace = True)
```

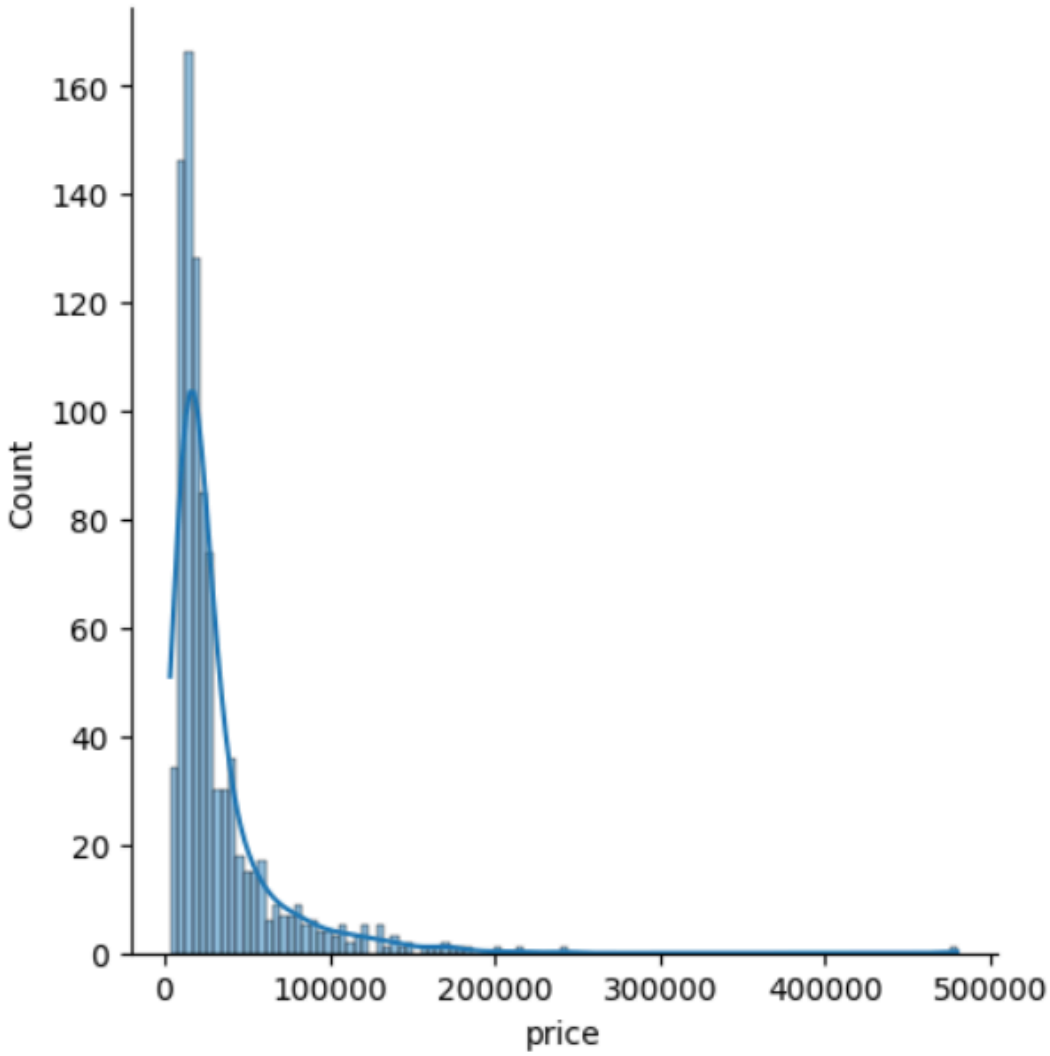
```
In [246]: df1.drop(columns = 'processor_name',inplace = True)
```

- **Handling Outliers**:

➢ **Price:**

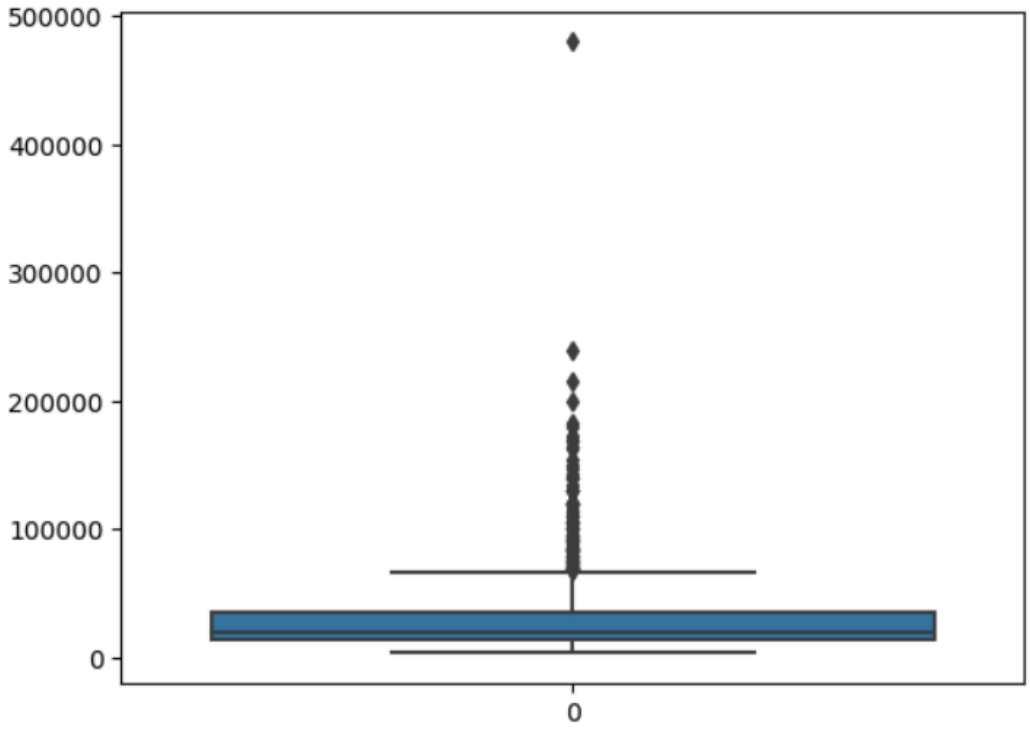
Outlier treatment for the “price” column in the dataset.

```
In [33]: # let us plot histogram and kde plot together for price column
         sns.displot(kind = 'hist',data = df,x = 'price',kde = True)

         # as you can see in the output this is highly right skewed
         # by looking at data we can understand that most of the phone ar

         C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\Lib\site
         a option is deprecated and will be removed in a future version.
           with pd.option_context('mode.use_inf_as_na', True):

Out[33]: <seaborn.axisgrid.FacetGrid at 0x1f02d091f10>
```


Count
160
140
120
100
80
60
40
20
0
0
100000
200000
300000
400000
500000
price

```
In [35]: # let us check the outliers
         sns.boxplot(df['price'])
```

```
Out[35]: <AxesSubplot: >
```



there are only outliers in the upper region These phones are highly priced because they use gold and diamonds in their making. These kinds of materials are not common in the making of most smartphones. They are not priced high because of their phone features, but because of elements unrelated to the phone itself. To keep these smartphones, I would need to add columns for gold and diamond content, but that's not our intention. Therefore, I decided to drop these rows as they will act as outliers. The models affected are:

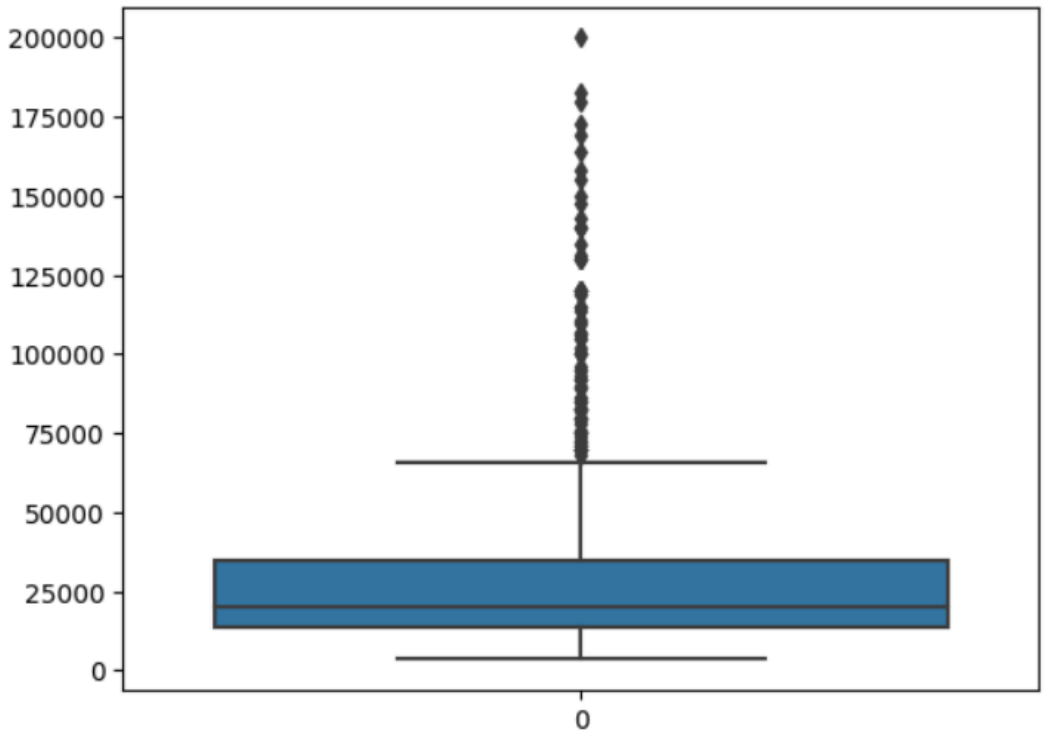
- ➢ Huawei Mate 50 RS Porsche Design
- ➢ Xiaomi Redmi K20 Pro Signature Edition
- ➢ Huawei Mate 30 RS Porsche Design

Additionally, I corrected the prices for two models in our records:

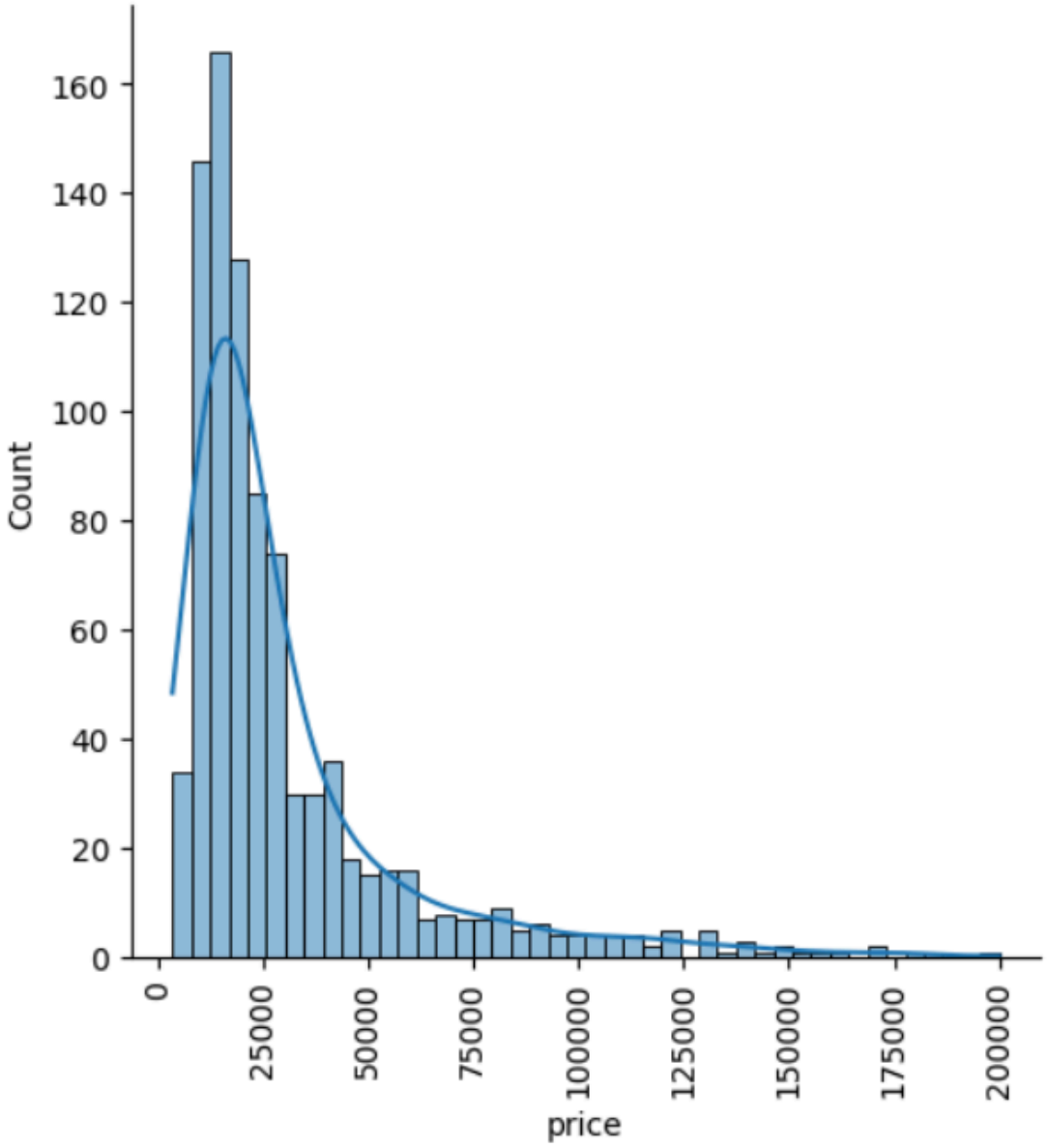
- ➢ Updated the "Huawei Mate Xs 2" to 115,000.
- ➢ Updated the "Apple iPhone 14 Pro Max (512GB)" to 157,999.

In [44]:
```
sns.boxplot(df['price'])
```

Out[44]: <AxesSubplot: >



In [45]:
```
sns.displot(kind = 'hist',data = df,x = 'price',kde = True)
plt.xticks(rotation = 'vertical');
```



➢ **battery_capacity:**

In the “**battery_capacity”** column, I corrected the incorrect values. Other than these corrections, all other outliers are genuine values. These corrections did not show any noticeable changes in the graph. The only visible change was in the skewness, which decreased from 0.67 to 0.65, but this change is insignificant.

```
In [117]: df['battery_capacity'].skew()
          # distribution is negatively skewed with -0.67.

Out[117]: -0.6747102646162833
```

```
In [129]: df['battery_capacity'].skew()

Out[129]: -0.6534659735600558
```

- ## Dealing with Inconsistent Data:
- **price:**
  The **“price”** column contains **'₹'** and **','** symbols, making it unsuitable for mathematical calculations. Let’s remove these symbols.

```
In [10]: # make a copy
         df1 = df.copy()

In [11]: df1
```

Out[11]:

	model	price	rating	sim	processor	ram	battery	display	camera	card	os
0	OnePlus 11 5G	₹54,999	89.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi, NFC	Snapdragon 8 Gen2, Octa Core, 3.2 GHz Processor	12 GB RAM, 256 GB inbuilt	5000 mAh Battery with 100W Fast Charging	6.7 inches, 1440 x 3216 px, 120 Hz Display wit...	50 MP + 48 MP + 32 MP Triple Rear & 16 MP Fron...	Memory Card Not Supported	Android v13
1	OnePlus Nord CE 2 Lite 5G	₹19,989	81.0	Dual Sim, 3G, 4G, 5G, VoLTE, Wi-Fi	Snapdragon 695, Octa Core, 2.2 GHz Processor	6 GB RAM, 128 GB inbuilt	5000 mAh Battery with 33W Fast Charging	6.59 inches, 1080 x 2412 px, 120 Hz Display wi...	64 MP + 2 MP + 2 MP Triple Rear & 16 MP Front ...	Memory Card (Hybrid), upto 1 TB	Android v12


```
In [10]: # make a copy
         df1 = df.copy()

In [11]: df1
```


Out[11]:

	model	price
0	OnePlus 11 5G	₹54,999
1	OnePlus Nord CE 2 Lite 5G	₹19,989

```
In [12]: df1['price'] = df1['price'].str.replace('₹','').str.replace(',','').astype('int')
```

```
In [13]: df1
```

Out[13]:

	model	price
0	OnePlus 11 5G	54999
1	OnePlus Nord CE 2 Lite 5G	19989

- **num_front_cameras:**

  In the **'num_front_cameras'** column, there is only numerical data except for the value 'missing', which is a string. To correct this inconsistency, I replaced **'missing'** with **np.nan**. This step is part of dealing with inconsistencies in the data.

```
In [232]: df1['num_front_cameras'] = df1['num_front_cameras'].replace('missing',np.nan)
```

```
In [231]: df1['num_front_cameras'].value_counts()

Out[231]: num_front_cameras
          1          947
          2           29
          missing      4
          Name: count, dtype: int64
```

- **primary_camera_front:**
  In the **'primary_camera_front'** column, there is only numerical data except for the value 'main', which is a string. To correct this inconsistency, I replaced **'main'** with **np.nan**. This step is part of dealing with inconsistencies in the data.

```
In [240]: df1['primary_camera_front'].value_counts()

Out[240]: primary_camera_front
          16      307
          8       178
          32      155
          5       119
          12       50
          13       41
          20       37
          10       24
          50       12
          60       10
          44        8
          40        6
          2         5
          7         5
          24        3
          25        3
          10.8      3
          48        2
          11.1      2
          0.3       1
          2.1       1
          Main      1
          10.7      1
          10.1      1
          12.6      1
          Name: count, dtype: int64
```

```
In [241]: df1['primary_camera_front'].replace('Main', np.nan, inplace=True)
```

- **OS:**
  In the **"OS"** column, passed the entire os column through the os_extractor function, which changes 'Android' to 'Android', 'iOS' to 'iOS', '0' to '0', remaining text to 'other', and any non-text values to 'unknown'. Finally, I replaced '0' with np.nan.

```
In [251]: df[df['os'].isnull()]
```

Out[251]:

	model	price	rating	sim	processor	ram	battery	display	camera	card	os
473	Nokia 110 4G	₹1,762	NaN	Dual Sim, 3G, 4G, VoLTE	No Wifi	128 MB RAM, 48 MB inbuilt	1020 mAh Battery	1.8 inches, 120 x 160 px Display	0.3 MP Rear Camera	Memory Card Supported, upto 32 GB	NaN
532	Samsung Guru Music 2 Dual Sim	₹1,949	NaN	Dual Sim	No Wifi	Single Core, 208 MHz Processor	800 mAh Battery	2 inches, 128 x 160 px Display	No Rear Camera	Memory Card Supported, upto 16 GB	NaN
573	Nokia 105 (2019)	₹1,299	NaN	Single Sim	No Wifi	4 MB RAM, 4 MB inbuilt	800 mAh Battery	1.77 inches, 120 x 160 px Display	No Rear Camera	NaN	NaN
608	Namotel Achhe Din	₹99	NaN	Dual Sim, 3G, Wi-Fi	1 GB RAM, 4 GB inbuilt	1325 mAh Battery	4 inches, 720 x 1280 px Display	2 MP Rear & 0.3 MP Front Camera	Android v5.0 (Lollipop)	Bluetooth	NaN
640	Nokia 105 Plus	₹1,299	NaN	Dual Sim	4 MB RAM, 4 MB inbuilt	800 mAh Battery	1.77 inches, 128 x 160 px Display	No Rear Camera	Memory Card Supported, upto 32 GB	Bluetooth	NaN
645	Nokia 2760 Flip	₹5,490	NaN	Dual Sim, 3G, 4G, Wi-Fi	1450 mAh Battery	3.6 inches, 240 x 320 px Display	5 MP Rear & 5 MP Front Camera	Memory Card Supported, upto 32 GB	Kaios v3.0	Bluetooth	NaN

In [253]:
```
def os_extracter(text):
    if isinstance(text, str):
        if 'Android' in text:
            return 'Android'
        elif 'iOS' in text:
            return 'iOS'
        elif text == '0':
            return '0'
        else:
            return 'Other'
    else:
        return 'Unknown'
```

In [254]:
```
# Convert NaN values to empty strings
df1['os'] = df1['os'].fillna('0')
```

In [255]:
```
# Apply the function to the 'os' column
df1['os'] = df1['os'].apply(os_extracter)
```

```
In [256]: df1['os'].value_counts()

Out[256]: os
          Android    910
          iOS         46
          0           14
          Other       10
          Name: count, dtype: int64
```

```
In [257]: df1['os'].replace('0',np.nan).value_counts()

Out[257]: os
          Android    910
          iOS         46
          Other       10
          Name: count, dtype: int64
```

- Dropped two rows of feature phones data which are not belong to smartphones.

```
In [10]: # screen size
         # not a smartphone
         df.drop(df[df['model'] == 'Duoqin F22 Pro'].index, inplace=True)
```

```
In [11]: # screen size
         # not a smartphone
         df.drop(df[df['model'] == 'CAT S22 Flip'].index,inplace = True)
```

- Replaced the misspelled ‘sanpdragon’ with correct spelling ‘snapdragon’.

```
In [13]: df['processor_brand'] = df['processor_brand'].replace('sanpdragon','snapdragon')
```

c) Libraries Used

Following libraries are used in the python for data cleaning

- Numpy
- Pandas

- math
- Matplotlib.pyplot
- Seaborn
- Scipy.stats
- from sklearn.impute imported KNNImputer and SimpleImputer.

d) Results of Data Cleaning

- **Data Summary**: Summarize the state of the dataset after cleaning (e.g., reduced number of rows, improved data quality).

➢ **Rows Removed:** Removed non-smartphone rows.
➢ **Missing Values:** Filled missing values in columns like Extended_upto, Fast_charging, OS, and Num_core.
➢ **Data Types:** Corrected data types for columns like num_front_cameras, primary_camera_front, ram_capacity, and internal_memory.
➢ **Consistency and Validity:** Fixed misplaced data and removed irrelevant data.
➢ **Split Information:** Split multi-value columns (e.g., battery, camera, display, card) into separate columns for further analysis.
➢ **Outliers:** Removed extreme outliers not relevant to smartphone features (e.g., phones made with gold and diamond).
➢ **Inconsistencies:** Standardized text data by removing special characters from price, and corrected misspellings.
➢ **Data Tidying:** Handled case sensitivity by converting all values in the brand_name and processor_name columns to lowercase for consistency.

- **Impact on Analysis**: After data cleaning, improved the quality and reliability of subsequent data analysis.

➢ **Accuracy:** improved accuracy.
➢ **Completeness:** attained completeness by filling missing values.
➢ **Consistency:** improved consistency by correcting data and data types.
➢ **Relevance:** improved relevance by removing outliers.

e) Challenges and Solutions

- **Challenges Faced**:

➢ **Missing Values:** Finding the best method to fill them.
➢ **Data Types:** Converting mixed data types correctly.
➢ **Inconsistencies:** Identifying and fixing inconsistent data.
➢ **Outliers:** Identifying and handling extreme values.

- **Solutions Implemented**:

➢ **Imputation:** Used KNNImputer and SimpleImputer for filling missing values.
➢ **Shifting Data:** Moved misplaced data to the correct columns.
➢ **Standardization:** Converted text to lowercase and corrected spellings.
➢ **Verification:** Manually checked and corrected values.

f) Conclusion

- **Summary**: Recap the key points of data cleaning process.

  The data cleaning process was improved the dataset by filling missing values, correcting data types, ensuring consistency, splitting multi-value columns, handling outliers, and fixing inconsistencies. This resulted in a structured and reliable dataset ready for analysis.

C. EDA : smartprix step-4 EDA.ipynb

Scraped the site smartprix and did data preprocessing for data analysis and drew insights

Libraries:

- numpy
- Pandas
- Matplotlib.pyplot
- Seaborn
- Scipy.stats
- from scikit_posthocs imported posthoc_dunn

Note: Analyzing the smartphone brands with at least 10 models.

0 brand_name:

Univariate Analysis:-

Univariate Analysis Summary: Smartphone Brand Comparison

- **Analysis Objective:** Analyzing the smartphone brands with at least 10 models.

- **Top Brand:** Xiaomi emerges as the smartphone brand with the highest number of models representing 14.9% of the total models with brands above 10 models. **Runner-Up:** Samsung follows closely behind Xiaomi, with a total of models accounting for about 14.8% of the share.
- In total, the top 4 brands, namely Xiaomi, Samsung, Vivo, and Realme, occupy 53.14% of the market models.

Bivariate Analysis:-

Smartphone Brand And Price:

- **Apple**: Premium pricing, wide price range, consistent strategy.
- **Nokia**: Affordable, lower price range.
- **Google**: Lower to mid-range pricing, below Apple's median price.
- **Motorola**: Concentrated in lower price range, exceeds Apple's first quartile by 2250 rupees.
- **Samsung**: Genuine outliers, varying pricing strategies.
- **Chinese brands**: Lower end of price scale, outliers present.

  All the remaining brands are Chinese, and they are predominantly positioned at the lower end of the price scale. A few of these brands do not contain outliers.

- **Oppo, Vivo, Xiaomi**: Extreme outliers, pricing variability.

Smartphone Brands And Operating Systems:

- **Operating Systems**: Except for Apple, all brands use Android. Apple stands out with its exclusive iOS.
- **Market Share**: 67.06% of Android models are from the top 5 brands: Xiaomi (16.04%), Samsung (15.68%), Vivo (13.9%), Realme (11.70%), and Oppo (10.25%).

1 model:

The "model" column is considered a myth column as it does not provide categorical or numeric data for analysis. With 890 unique values, matching the number of observations, it lacks patterns or repetition. This column serves as a reference for outlier detection or identifying specific issues with individual phones. Though limited in direct analysis, it remains valuable for supplementary insights in outlier examination.

2 price:

Univariate Analysis:-

Summary of Price Distribution:

- The mean price is 30690.91, close to the third quartile, indicating a skewed distribution likely influenced by outliers.
- High standard deviation of 29176.6 further suggests the presence of outliers.
- The minimum price is 3499.00 for a basic smartphone.
- 25% of phones are priced below 13499.00.
- The median price is 19999.00, reflecting a diverse range of phone prices in the dataset.
- A notable difference between the mean and median indicates a skewed distribution.
- 75% of phones are priced under 34990.00.
- The maximum price is 199990.00, indicating the presence of outliers in the data.

Bivariate Analysis:-

- Performed bi-variate analysis of price column with brand name and noted it above at brand name column's bivariate analysis.
- Performed bi-variate analysis of price column with os and noted it below at os column's bivariate analysis.

3 Rating:

Univariate Analysis:-

- 40.54% of smartphones fall in between 80 to 86 rating out of 100.
- 75.0% smartphones belong to chinese brands or associated with china in between the rating range of 80 to 86.
- slightly left skewed distribution with -0.68.

Bivariate Analysis:-

Rating And Price (Unfiltered Data):

- **Spearman's correlation coefficient** (0.762) suggests a strong positive relationship. Higher ratings tend to correspond with higher prices.
- The p-value (highly significant) confirms a statistically relevant association.

Rating And Price (Filtered Out Apple Data):

- **Spearman's correlation coefficient:** Removing Apple data strengthens the correlation (0.859) and lowers the p-value (even more significant). This implies a stronger positive relationship after filtering.

Rating And Smartphone Brands:

- Median ratings: OnePlus has the highest (84.0), followed by Huawei/Iqoo (82.5), and Tecno (74.0).
- Distribution:
    - Huawei: Highly left-skewed distribution and the only brand with outliers on both sides.
    - Samsung: Highest rating range with no outliers (slightly left-skewed -0.41).
    - Xiaomi, Realme and Infinix: fall within the same range as Samsung, all showing left-side outliers. However, while Xiaomi and Realme are moderately left-skewed, Infinix is closer to a normal distribution.
    - Vivo, Oppo and Techno: Similar rating ranges with no outliers (slightly left-skewed except Tecno which is normal).
    - Google, Nokia, Motorola, Honor and Iqoo: No outliers, mostly symmetric distributions or slightly skewed.
    - Apple, OnePlus and Poco: All exhibit outliers, with Apple showing a strong left skew, OnePlus a moderately strong left skew, and Poco a moderate left skew.

- **Shapiro-Wilk Test for Normality:** Ratings are not normally distributed based on the Shapiro-Wilk test (p-value < 0.05).
- **Kruskal-Wallis Test:** No significant differences in ratings between brands.

Rating And Operating Systems:

- **Kruskal-Wallis Test:**
    - there is no significant difference in ratings among the different operating systems.

Overall, the analysis reveals:

- A positive correlation between ratings and prices (stronger after filtering Apple).

- Variations in rating distribution and presence of outliers across brands.
- No statistically significant differences in ratings based on brand or OS.

4 5G Feature Column:

Univariate Analysis:-

- 56.85% of smartphones have 5G.

Bivariate Analysis:-

Analysis of 5G Feature and Prices:

- **Price Distribution:** The 'price' variable exhibits a non-normal distribution as confirmed by the Shapiro-Wilk test.
- **5G and Price:** Smartphones with 5G have a median price that is 130.67% higher than those without it.
- **Correlation between 5G and Price:** A Point-Biserial Correlation analysis reveals a moderate positive correlation between the presence of 5G and price. This indicates that 5G smartphones generally cost more, and this relationship is statistically significant.

5G Feature and Smartphone Brands

- **Brand Distribution:** 75.0% of smartphone brands offer more 5G models than non-5G models.
- **Brand Association with 5G:** Cramer's V coefficient (approximately 0.331) suggests a moderate to moderately strong association between a brand having a 5G model and the brand name itself.

5G Feature and Operating Systems

- **OS Distribution:** Both Android and iOS have a higher proportion of models with 5G compared to those without it.
- **OS Association with 5G: Cramer's V coefficient** (approximately 0.127) indicates a weak association between the presence of 5G and the operating system (Android/iOS). This suggests minimal influence of the OS on whether a smartphone has 5G capability in your data.

Key Takeaways:

- Smartphones with 5G tend to be priced higher.
- There's a moderate positive correlation between 5G and price.

- A moderate association exists between a brand having a 5G model and the brand name.
- The presence of 5G shows a weak association with the operating system.

5 NFC Feature:

Univariate Analysis:-

- 61.1% of smartphones do not have nfc.

Bivariate Analysis:-

NFC Feature And Price:

- The curve of the category with nfc models is less peaked and right skewed than the one without.
- The median price of the category of smartphone models with NFC feature are 166.67% more pricer than the ones without.
- **Shapiro-Wilk Test for Normality:** The has_nfc variable is not normally distributed (null hypothesis rejected).
- The **Point-Biserial Correlation Coefficient** between the presence of NFC (Near Field Communication) and the price of smartphones is approximately 0.568, indicating a moderate positive correlation. The p-value associated with this correlation coefficient is very small (approximately 4.49e-77), confirming that the correlation is statistically significant.

NFC Feature And Smartphone Brand:

- Apple, which is a premium brand, Its 97.83% models have NFC feature.
- Whereas Google only has models with NFC feature.
- percentage of smartphone brands with more models having NFC feature: 43.75%.
- The **Cramer's V** coefficient of approximately 0.506 indicates a moderate to strong association between the presence of the NFC feature and smartphone brand. The p-value from the **chi-square test of independence** is approximately 3.705e-40, providing strong evidence against the null hypothesis of independence. This confirms a statistically significant association between the presence of the NFC feature and smartphone brand.
- Overall, this analysis indicates a notable relationship between the presence of the NFC feature and smartphone brand.

NFC Feature And Operating System:

- Percentage of smartphone models with NFC in Android OS: 34.86%.
- Percentage of smartphone models with NFC in iOS: 97.83%.

6 IR blaster Feature:

Univariate Analysis:-

- 17.6% of smartphone models are with ir_blaster feature.

Bivariate Analysis:-

IR blaster Feature And Price:

- Both categories have a similar median price of 19,990.0 rupees, indicating that half of the items in each category fall below this price.
- In both categories (with and without IR blaster), the distribution is right-skewed with outliers.
- After thorough examination, it can be concluded that these higher prices are genuine values and not outliers. A violin plot shows that the distribution is similar for both categories.
- **Shapiro-Wilk test for normality results:**
    - The variable 'has_ir_blaster' is not normally distributed.
    - The variable 'price' is not normally distributed.
- **Spearman's rho test:** It fail to reject the null hypothesis, which suggests that there isn't enough evidence to say there's a clear, consistent relationship (either increasing or decreasing) between whether a smartphone has an IR blaster and its price.

IR blaster Feature And Smartphone Brand:

- Percentage of smartphone brands with ir_blaster feature: 43.75%.
- Except for one smartphone model from Samsung, every smartphone model with IR blaster feature in the dataset belongs to Chinese brands.
- Among smartphones with IR blaster feature, Xiaomi is in first place with 70.32%, followed by Poco with 19.35%. Together, they account for 89.67% of all smartphones with IR blaster feature.

IR blaster Feature And Operating System:

- Total percentage of Android models with IR blaster feature: 18.21%.
- Total percentage of iOS models with IR blaster feature: 0.0%.
- iOS smartphones do not have even a single model with an IR blaster feature.

7 Processor Brand:

Univariate Analysis:-

- 82.5% of smartphone models are equipped with processors from one of these three brands: Snapdragon processors are the most common, present in 41.6% of models, followed by Helio processors in 21.4% of models, and then Dimensity processors in 19.5% of models.

Bivariate Analysis:-

Processor Brand And Price:

Statistical summary:

Snapdragon Processors:

- **Broad Market Coverage:** Snapdragon processors are used in smartphone models that have the widest price range (from 8,990.0 to 199,990.0 rupees), indicating they are used in a wide variety of smartphones, from budget to premium models.
- **High Skewness:** The high right-skewness (skewness = 2.29) suggests that most Snapdragon-equipped smartphones are priced lower, with a few very high-priced outliers.

Helio Processors:

- **Budget Segment:** Helio processors are found in the budget segment of smartphone models, with the lowest starting price (3,499.0 rupees) and a maximum price of 26,990.0 rupees.
- **Right Skewness with Few Outliers:** The right-skewed distribution (skewness = 0.76) with few outliers suggests a relatively uniform distribution, concentrated towards the lower end of the price range.

Dimensity Processors:

- **Mid to High-End Focus:** Dimensity processors have a smartphone model price range of 9,999.0 to 89,990.0 rupees, indicating they are typically used in mid-range to high-end smartphones.

- **Right Skewness with Outliers:** The right-skewed distribution (skewness = 1.85) and presence of outliers indicate a concentration of lower to mid-range prices with some higher-priced exceptions.

Bionic Processors:

- **Premium Pricing:** Bionic processors exhibit a bimodal distribution without outliers. The minimum price of smartphone models with Bionic processors is higher than the maximum price of Helio processors, the Q3 of Snapdragon and Dimensity processor-equipped smartphone models, and the median price of Exynos processor-equipped smartphone models.
- **Exclusive to Apple:** Apple exclusively uses Bionic processors, with minimal use of other processor types, reinforcing the premium positioning of Apple devices.
- **Higher Minimum Price:** The minimum price of any Apple smartphone model processor is greater than the median price of most other processor brands' smartphone models, highlighting Apple's premium market positioning.
- **Kruskal-Wallis test:** The test statistic is 445.24 and the p-value is approximately 2.07e-85. This indicates that there is a significant difference in smartphone model prices across different processor brands.

Processor Brand And Smartphone Brand:

- 81.25% of smartphone brands use ‘snapdragon’ processors
- 62.5% of smartphone brands use ‘helio’ processors.
- 75.0% of smartphone brands use ‘dimensity’ processors.
- **Chi2-square:** There is a statistically significant association between brand_name and processor_brand columns with P-value: 0.0000.

Processor Brand And Operating System:

iOS:

- 84.78% of iOS devices have Bionic processors.
- The A13 processor is used in 10.87% of iOS smartphones.
- The remaining models are equally split between ’Apple’ and ‘Fusion’ brand processors.

These four processor brands are exclusively used for iOS devices and are not used in any other brands.

Android:

- 87.27% of Android smartphone models are from any of these three brands: snapdragon(43.57%), helio(23.01%) and dimensity(20.69%).

8 Number Of Cores:

Univariate Analysis:-

- 92.3% of smartphones feature octa-core technology.
- Hexa-core processors are only present in iOS models. 95.12% of Apple models are equipped with hexa-core processors. The price range of Apple brands is premium, which makes the number of cores important for price prediction, thus being useful for the model.
- Hexa-core processors are 325.25% higher than octa-core.

Bivariate Analysis:-

Number Of Cores And Price:

- The average price of smartphone models with hexa-core processors is significantly higher than those with octa-core and quad-core processors. This is because Apple brand smartphones, which are the only ones with hexa-core processors, have a significantly higher average price. The hexa-core price distribution is the only one without any outliers, while values are more dense at octa-core.
- **Kendall's Tau Correlation:** suggests that there is evidence to indicate that as the number of cores increases, the price tends to decrease.

Number Of Cores And Smartphone Brand:

- **Cramér's V:** A value of 0.696 indicates a strong association between the 'num_cores' and 'brand_name' variables in the dataset. This suggests that the number of cores in a device (e.g., CPU cores in a smartphone) and the brand name are closely related.

Number Of Cores And Operating System:

- 95.12% of iOS models have a hexa-core (6-core) processor.
- 98.19% of Android models have an octa-core (8-core) processor.

9 Processor Speed:

Univariate Analysis:-

- The data is nearly normally distributed with a skewness of 0.36.
- 70.35% of models are from the top 6 categories. The category of models with a
    - 2.0 processor speed is the highest at 15.76%
    - followed by the 2.2 processor speed which adds up to 15.41%.
    - The 2.4 processor speed follows with 14.47%.
    - In fourth place is the 2.3 processor speed with 9.65%
    - followed by the 3.2 processor speed with 9.53%
    - The sixth place is for the 3.0 processor speed with 5.53%.

Bivariate Analysis:-

Processor Speed And Price:

- **Spearman's correlation coefficient** between processor speed and price is approximately 0.778, indicating a strong positive monotonic relationship between the two variables. Additionally, the p-value associated with the correlation coefficient is extremely low (close to zero), indicating that the observed correlation is statistically significant. This suggests that as processor speed increases, the price tends to increase as well.

Processor Speed And Smartphone Brand:

- Both the minimum and maximum processor speeds of the 'Apple' brand are the highest . The second-highest maximum processor speed and the third-highest minimum speed belong to the brand "Xiaomi".
- ‘Vivo’ has a wide range of processor speeds, followed by ‘Tecno’.
- Among non-Chinese brands, 'Samsung' has a wide range of processor speeds, followed by 'Nokia'.
- The **Kruskal-Wallis test** statistic is 137.60, indicating a substantial difference in processor_speed across the various brands included in the analysis. This statistic suggests a significant variation in processor_speed between brands ($p < 0.001$), highlighting that processor_speed varies significantly depending on the brand of the device.

Processor Speed And Operating System:

iOS:

- There are only 4 categories in iOS model smartphones.
    - The highest percentage of models, 56.25%, has a processor speed of 3.22.
    - This is followed by 25.00% for 3.1
    - and 15.62% for 2.65.
    - The remaining 3.12% belongs to the 2.37 processor speed.

Android:

- 57.07% of models fall within the 2.0 to 2.4 processor speed range in the Android category.
- 73.52% of models in the Android category fall within the top 6 processor speeds. The distribution is as follows:
    - the highest is 16.67% for 2.0
    - followed by 16.29% for 2.2
    - 14.93% for 2.4
    - 10.20% for 2.3
    - 9.58% for 3.2
    - 5.85% for 3.0.
- The **chi-square statistic** of 991.47 with a very low p-value (approximately 3.65 * 10^-165) provides strong evidence against the null hypothesis of independence, indicating a substantial association between the processor_speed and os.

10 Ram Capacity:

Univariate Analysis:-

RAM Capacity Distribution:

- The top RAM capacities, including 8GB(35.4%), 6GB(25.2%), and 4GB(22.6%), contribute to 83.2% of the smartphone models.
- This suggests that the majority of smartphones fall into these three categories in terms of RAM capacity.

Bivariate Analysis:-

Ram Capacity And Price:

- A clear trend is evident in the bar plot, indicating that smartphones with higher RAM capacities generally command higher prices. However, an exception is observed with the 16GB RAM variant, which, despite having the highest capacity, is priced second highest. Interestingly, smartphones with 12GB RAM, the second highest RAM capacity, are priced the highest. The distributions without outliers are 1GB and 16GB.

- **Spearman correlation:** The correlation coefficient value of approximately 0.736 suggests a strong positive association between RAM capacity and price. Additionally, the very small p-value (< 0.05) indicates that this correlation is statistically significant, suggesting it is unlikely to have occurred by random chance alone.

Ram Capacity And Smartphone Brand:

- A **Chi-square** statistic of 227.67 with a p-value of 4.656067360215653e-11 indicates a significant association between RAM capacity and brand names. This suggests the observed distribution of RAM capacities across brands is unlikely due to chance. With the p-value less than 0.05, I reject the null hypothesis, confirming a significant relationship between RAM capacity and brand name.

The analysis of **adjusted residuals** reveals significant patterns in the distribution of smartphone brands across different RAM categories:

- **1GB Category**: Tecno is overrepresented (adjusted residual = 2.66), indicating a strong market presence.
- **2GB Category**: Realme is overrepresented (adjusted residual = 2.71), suggesting strategic positioning in this segment.
- **3GB Category**: Vivo is dominant (adjusted residual = 3.54), showing a strong foothold in this category.
- **4GB Category**: No significant deviations, indicating balanced brand representation.
- **6GB Category**: Apple, Poco, and Xiaomi are overrepresented (adjusted residuals = 3.36, 2.08, 2.68 respectively), while Vivo is underrepresented (adjusted residual = -3.77).
- **8GB Category**: Huawei, OnePlus, and Oppo are overrepresented (adjusted residuals = 3.16, 2.11, 2.12 respectively), while Apple is underrepresented (adjusted residual = -3.04).
- **12GB Category**: Google, OnePlus, and Vivo are overrepresented (adjusted residuals = 2.66, 2.45, 2.29 respectively), while Realme is underrepresented (adjusted residual = -2.11).
- **16GB Category**: Oppo is dominant (adjusted residual = 3.56), indicating a focus on high-RAM devices.

Ram Capacity And Operating System:

iOS: models use only 4 types of RAM categories.

- The highest category is 6GB, comprising exactly 50.00% of models.
- The second highest is 4GB with 34.78%
- followed by 8GB with 8.70%.

- The fourth category is 3GB, accounting for 6.52% of models.

Android: 91.09% of models fall into the top 4 types of RAM categories.

- The most common category is 8GB RAM, with 36.07% of models
- followed by 6GB with 24.13%
- 4GB with 22.20%
- and 12GB with 8.69%.

11 Internal Memory:

Univariate Analysis:-

- The top three internal memory capacities (128GB, 64GB, and 256GB) account for 91.4% of the dataset. Most smartphones offer one of these three capacities, with
  - 128GB being the most common (55.6%)
  - followed by 64GB (20.0%)
  - and 256GB (15.8%).

Bivariate Analysis:-

Internal Memory And Price:

- The bar plot shows a positive correlation between internal memory capacity (measured in GB) and the average price of smartphones. As internal memory capacity increases, the smartphone price also increases.
- **Spearman correlation coefficient:** between internal memory and price is approximately 0.743, indicating a strong positive correlation. The p-value is very close to zero (8.46e-157), confirming statistical significance. **Bootstrapping analysis** gives a similar correlation coefficient of 0.7426, with a 95% confidence interval of (0.7079, 0.7774), reinforcing the strong positive correlation between internal memory and price.

Internal Memory And Smartphone Brand:

- **Cramér's V:** value of approximately 0.211 indicates a moderate association between internal memory and brand name in the dataset. This suggests that there is a relationship between the two variables, but it is not very strong.

Internal Memory And Operating System:

- **iOS:** The distribution of internal memory capacities is as follows:
    - 128GB (34.78%)
    - 256GB (23.91%)
    - 512GB (19.57%)
    - 64GB (10.87%)
    - 512GB (8.70%)
    - and 32GB (2.17%), totaling 100%.
- **Android:** 98.08% of models fall into the top 4 categories:
    - 128GB (56.82%)
    - 64GB (20.27%)
    - 256GB (15.20%)
    - 32GB (5.79%).

12 Fast Charging Feature:

Univariate Analysis:-

- 87.8% smartphone models have fast charging available.

Bivariate Analysis:-

Fast Charging Feature And Price:

- The average price of smartphones with fast charging capability is approximately twice that of smartphones without this feature.
- Both category distributions are heavily right-skewed with numerous outliers.
- The majority of data points are clustered around the median in both category distributions.
- The Q3 price of smartphones without fast charging availability is less than the Q1 price of smartphones with fast charging availability.
- The median price of smartphones with fast charging availability (21,994.0) is 121.05% higher than the median price of smartphones without availability (9,950.0).
- Even the standard deviation and the mean of the category with fast charging availability are higher than those without.
- Both category means are not equal to their medians.
- Smartphones with fast charging available tend to have higher prices on average.
- **Spearman correlation:** There is a statistically significant moderate positive monotonic relationship between the availability of fast charging and the price of

smartphones in the dataset. This suggests that smartphones with fast charging available tend to have higher prices on average.

Fast Charging Feature And smartphone brand:

- In every smartphone brand, models with the fast charging feature out number those without it.
- Every model of 'iQOO' and 'Google' is equipped with fast charging.
- 15.62% of smartphones with fast charging belong to Samsung brand.Another 15.62% of smartphones with fast charging belong to Xiaomi brand.Followed by 'Vivo' with 11.91%, 'Realme' with 10.24%, and 'Oppo' with 9.60%. All these five brands together account for 62.99%.

Fast Charging Feature And Operating System:

- Android: 88.54% of smartphone models have fast charging available.
- iOS: 69.56% of smartphone models have fast charging available.

13 Fast Charging Capacity:

Univariate Analysis:-

- Fast charging is positively skewed, with outliers on the right side of the distribution, suggesting some models have significantly higher capacities compared to the majority.
- The median fast charging capacity is 33W, with approximately 50% of smartphone models having capacities below this value.
- The most common fast charging capacities are 33W (19.86% of models) and 18W (17.22% of models), totaling 37.08% of all models.

Bivariate Analysis:-

Fast Charging Capacity And Price:

- The highest median price is in the 55W category, followed by the 45W category. Both 55W and 45W have wide distributions without outliers.
- The Kruskal-Wallis statistic is 435.533 with a p-value of approximately 2.77e-75, indicating a strong association between different levels of fast_charging and smartphone prices. This suggests significant price differences exist across various levels of fast_charging.

Fast Charging Capacity And Smartphone Brand:

- **Cramér's V:** The value is approximately 0.581, indicating a moderate to strong association between 'fast_charging' and 'brand_name'. The p-value is extremely small (close to zero), confirming that this association is statistically significant. Therefore, I reject the null hypothesis and conclude that there is an association between the two variables.

Fast Charging Capacity And Operating System:

iOS:

- All models in the fast charging category for iOS smartphones are split between two types: 18W (66.67%) and 25W (33.33%).

Android:

- 60% of Android models fall into the top 5 fast charging categories:
    - 33W (20.23%)
    - 18W (17.38%)
    - 67W (8.83%)
    - 25W (7.12%)
    - 120W (6.13%)

14 Battery Capacity:

Univariate Analysis:-

- distribution is negatively skewed with -0.65.
- Although it contains many values that appear to be outliers on both ends, upon closer inspection, they are genuine.
- Due to the influence of outliers on the mean, I opt for the median as the measure of average, which is 5000mAh for battery capacity.
- exactly 51.76% of smartphone models have 5000mAh battery capacity.

Bivariate Analysis:-

Battery Capacity And Price:

- **The Kruskal-Wallis statistic** is 1329.99 with a p-value of 3.44×10^-29, indicating a substantial difference between groups. Higher values of the Kruskal-Wallis statistic suggest stronger evidence against the null hypothesis, indicating greater differences between groups.

- This result provides strong evidence against the null hypothesis, confirming a statistically significant relationship between battery capacity and price.

Battery Capacity And Smartphone Brand:

- **The Chi-square statistic** is 3024.28 with a very low p-value of approximately 1.415e-173 and 1125 degrees of freedom, indicating a strong association between battery capacity and smartphone brand.

Battery Capacity And Operating System:

iOS:

- 51.43% of models fall into the top 5 categories. The highest category is 4352mAh with 14.29%, followed by 4323mAh with 11.43%, 3279mAh with 8.57%, 4325mAh with 8.57%, and 2200mAh with 8.57%.

Android:

- 71.53% of models fall into the top 3 categories. The highest category is 5000mAh with 54.64%, followed by 4500mAh with 10.74%, and 6000mAh with 6.15%.

15 Refresh Rate:

Univariate Analysis:-

- The top three refresh rates, 60Hz, 120Hz, and 90Hz, account for 95.9% of the dataset, indicating that most smartphones offer one of these three options.
- Specifically, 120Hz is found in 36.3% of smartphones, followed by 60Hz at 35.6%.
- In the third position, around 24.0% of smartphones have a 90Hz refresh rate.

Bivariate Analysis:-

Refresh Rate And Price:

- Smartphones with higher refresh rates have higher average prices. The highest density of models are around 60Hz, 90Hz, and 120Hz.
- The Kruskal-Wallis statistic is approximately 1352.34 with a very low p-value of 4.77e-296, indicating a significant relationship between refresh rate and price.

This suggests strong evidence against the null hypothesis, meaning there is an association between the refresh rate feature and the price of smartphone models.

Refresh Rate And Smartphone Brand:

- The Kruskal-Wallis statistic for "refresh_rate" and "brand_name" is 98.79, with a p-value of approximately 2.22e-14. This very low p-value indicates a significant relationship between refresh rate and brand name, suggesting notable differences in refresh rates among different brands.

Refresh Rate And Operating System:

- iOS: There are only 2 categories in iOS. The highest percentage is for the 60Hz refresh rate with 63.04%. The second and last category is 120Hz with 36.96%.
- Android: There are 4 categories in Android. The highest percentage is for the 120Hz refresh rate with 35.71%. The second category is 60Hz with 34.50%. The third category is 90Hz with 25.57%. The last one is 165Hz with 0.24%

16 Screen Size:

Univariate Analysis:-

- The peak range of screen sizes is 6.40 to 6.80 inches, covering 86.74% of smartphones in the dataset. The most prevalent screen size is 6.5 inches (11.5%), followed by 6.67 inches (10.0%), 6.7 inches (9.5%), and 6.6 inches (9.3%), collectively covering 41% of the dataset. The distribution is negatively skewed (skewness = -0.83), indicating more smartphones have larger screen sizes.

Bivariate Analysis:-

Screen Size And Price:

- Highest density is around 6.5 screen size.
- A Kruskal-Wallis statistic of 450.50 and a p-value of 3.57e-57 indicate a significant difference in prices across different screen sizes.

Screen Size And Smartphone Brand:

- The highest cluster is at 6.5 inches, followed by 6.7, 6.67, 6.6, 6.43, and 6.4 inches. Except for 'Apple' and 'Google,' all brands have similar screen size distributions.
- 64.04% of smartphone models have screen sizes between 6.5 and 6.7 inches.

- **Cramér's V:** for the association between screen size and brand name is approximately 0.523, indicating a moderate to strong relationship between these variables.

Screen Size And Operating System:

iOS:

- 76.09% of iOS models fall into the top 3 screen sizes. The highest percentage of models have a 6.1-inch screen (45.65%), followed by 6.7 inches (21.74%) and 5.4 inches (8.70%).

Android:

- 53.44% of Android models fall into the top 6 screen sizes. The highest percentage of models have a 6.5-inch screen (12.18%), followed by 6.67 inches (10.98%), 6.6 inches (10.37%), 6.7 inches (9.65%), 6.43 inches (5.19%), and 6.4 inches (5.07%).

17 Resolution:

Univariate Analysis:-

- 70.3% of models fall into the top 5 resolution categories. The most common resolution is 1080x2400 pixels (36.5%), followed by 720x1600 pixels (15.5%) and 1080x2408 pixels (7.2%). The fourth and fifth most common resolutions are 1080x2412 (6.52%) and 1080x2340 (4.61%).

Bivariate Analysis:-

Resolution And Price:

- A large cluster is seen at 1080x2400, followed by slightly less dense clusters at 1080x2412, 720x1600, 1080x2408, 1080x2460, and 720x1612. Notably, three of these clusters have a width of 1080 pixels.
- The analysis suggests that the relationship between smartphone resolution and price may not be significant. However, distinct clusters are prominent at resolutions like 1080x2400 and 1080x2412. While no direct correlation between

resolution and price is evident, certain resolution values are prevalent among smartphones in the dataset.

- The Kruskal-Wallis statistic is 570.92 with a p-value of 1.26×10^{-79}.
    - This extremely low p-value indicates significant price differences across resolution categories.
    - The high statistical value suggests substantial variance among the groups.

Resolution And smartphone brand:

- **Chi-square test:** shows a strong association between resolution and brand name.
    - Chi-square statistic: 3111.52, indicating a substantial departure from expected frequencies under the null hypothesis.
    - p-value: ~9.53e-203, significantly smaller than 0.05, providing strong evidence against the null hypothesis and supporting a significant association between resolution and brand name.
- **Cramer's V** value of ~0.483 suggests a moderate to strong association between "resolution" and "brand name," indicating a non-negligible relationship.

Resolution And Operating System:

iOS:

- 82.61% of iOS models fall within the top 5 resolution categories. The highest resolution is 1170 x 2532 (34.78%), followed by 1284 x 2778 (17.39%), 1290 x 2796 (10.87%), 1080 x 2340 (10.87%), and 750 x 1334 (8.70%).

Android:

- 70.21% of Android models fall within the top 4 resolution categories. The highest resolution is 1080 x 2400 (38.96%), followed by 720 x 1600 (16.53%), 1080 x 2408 (7.72%), and 1080 x 2412 (7.00%).

18 Number Of Rear Cameras:

Univariate Analysis:-

- The majority of smartphones (57.4%) have 3 rear cameras, followed by 2 rear cameras (20.6%).

- Smaller proportions include smartphones with 4 rear cameras (16.7%) and those with 1 rear camera (5.3%).
- **Smartphones with 4 Rear Cameras:**
  59.73% of this market is captured by two brands:
    - Samsung: 36.91%
    - Xiaomi: 22.82%

Remaining brands have very low percentages.

- **Smartphones with 3 Rear Cameras:**
  49.9% is captured by four brands:
- Realme: 13.31%
- Xiaomi: 12.72%
- Vivo: 12.52%
- Samsung: 11.35%

Remaining brands have very low percentages.

- **Smartphones with 2 Rear Cameras:**
  72.13% is captured by six brands:
    - Vivo: 15.30%
    - Oppo: 14.21%
    - Xiaomi: 13.11%
    - Apple: 12.02%
    - Samsung: 9.29%
    - Tecno: 8.20%

  Remaining brands have very low percentages.

- **Smartphones with 1 Rear Camera:**
  82.99% is captured by five brands:
    - Xiaomi: 21.28%
    - Realme: 21.28%
    - Vivo: 19.15%
    - Oppo: 10.64%
    - Apple: 10.64%

  Remaining brands have very low percentages.

Bivariate Analysis:-

Number Of Rear Cameras And Price:

- The analysis shows a consistent increase in smartphone prices as the number of rear cameras increases up to three. There is a decline for models with four rear cameras, but they still remain higher than those with two cameras. All categories show a right-skewed distribution with numerous outliers.
- The highest median price is for models with 3 rear cameras (23,990.0), followed by 4 cameras (14,490.0), 2 cameras (12,990.0), and 1 rear camera (8,999.0).
- The density is highest for models with 3 rear cameras, followed by 2 cameras, 4 cameras, and lowest for models with 1 rear camera.
- **Kruskal-Wallis** H-test indicates significant differences in median prices among the categories of number of rear cameras:
    - H-statistic: 118.52
    - p-value: approximately 1.61e-25

  This suggests strong evidence against the null hypothesis of no difference in median prices, supporting the conclusion that there are significant differences in median prices across the different categories of 'num_rear_cameras'.

- **Dunn test**
    - There are significant differences in prices between models with 1 rear camera and those with 2, 3, and 4 rear cameras.
    - There are significant differences in prices between models with 2 rear cameras and those with 3 and 4 rear cameras.
    - There is no significant difference in prices between models with 3 rear cameras and those with 4 rear cameras.

Number Of Rear Cameras And Smartphone Brand:

- Except for Apple and Google, all brands have the highest number of models with 3 rear cameras. Apple and Google have their highest number of models in the 2 rear cameras category.
- Brands with models having 2 cameras in the first place, and 3 cameras in the second place: Apple and Google.
- Brands with models having 3 cameras in the first place, and 2 cameras in the second place: Infinix, iQOO, Motorola, Oppo, Poco, Realme, Tecno, Vivo.
- Brands with models having 3 cameras in the first place, and 4 cameras in the second place: Honor, Nokia, OnePlus, Samsung, Xiaomi.

- The **Chi-Square Test** of Independence revealed a statistically significant association between the 'num_rear_cameras' and 'brand_name' variables.
    - Chi-Square Statistic: 236.102
    - P-value: 9.767e-28
    - Degrees of Freedom: 45

  The small p-value (9.767e-28) indicates strong evidence against the null hypothesis of independence, leading us to reject it. Therefore, I conclude that there is a significant association between the number of rear cameras and the brand name of smartphones.

  The expected frequencies table shows the expected counts assuming no association between the variables.

- **Cramer's V** coefficient, measuring the strength of association, is approximately 0.297 with a p-value of approximately 9.77e-28.
    - Cramer's V coefficient of 0.297 indicates a moderate association between the number of rear cameras and the brand name.
    - The small p-value (much smaller than 0.05) confirms that this association is statistically significant.

  In conclusion, there is a moderate, statistically significant association between the number of rear cameras and the brand name of smartphones in your dataset.

Number Of Rear Cameras And Operating System:

iOS:

- No iOS models have 4 rear cameras.
- The majority of iOS models have 2 rear cameras (78.3%), followed by 3 cameras (41.3%), and then 1 camera (10.87%).

Android:

- The highest number of Android models have 3 rear cameras (58.62%), followed by 2 cameras (18.94%), and then 4 cameras (17.37%). A small percentage (5.07%) of models have other configurations.

19 Number Of Front Cameras:

Univariate Analysis:-

- 96.8% models are equipped with 1 front camera and remaining models with 2 front cameras.

Bivariate Analysis:-

Number Of Front Cameras And Price:

- The analysis indicates that, on average, smartphones equipped with two front cameras command a 55.06% higher price compared to those with just one front camera. The distribution of models with 1 front camera has numerous outliers, while the distribution of models with 2 front cameras has few outliers. Both distributions are right-skewed, with the skewness for models with 2 front cameras at 1.5 and the skewness for models with 3 front cameras at 2.5.
- The **Kruskal-Wallis H-test** shows a statistically significant difference in median prices among categories of rear cameras.
    - H-statistic: 19.39
    - P-value: 1.065e-05

  The small p-value indicates significant differences in median prices across rear camera categories, suggesting the number of rear cameras significantly impacts smartphone prices.

Number Of Front Cameras And Smartphone Brand:

- Out of the two categories, it is evident that the majority of brands have smartphone models with 1 front camera. In this category, the leading brands are:
    1. Xiaomi: 15.13%
    2. Samsung: 14.90%
    3. Vivo: 11.87%
    4. Realme: 10.94%
    5. Oppo: 9.90%

  These five brands account for 62.74% of this category.

- The **Chi-Square Test** of Independence shows a statistically significant relationship between the variables.
    - Chi-Square Statistic: 236.102
    - P-value: 9.768e-28
    - Degrees of Freedom: 45

The small p-value indicates strong evidence against the null hypothesis, suggesting a significant association between the variables. This means the distribution of one variable depends on the other. The expected frequencies table shows the counts assuming no association between the variables.

- **Cramer's V** Test results indicate a moderate association between the variables.
    - Cramer's V: 0.297
    - P-value: 9.768e-28

A Cramer's V value of 0.297 suggests a moderate association. The small p-value indicates this association is statistically significant, confirming a meaningful relationship between the variables.

Number Of Front Cameras And Operating System:

iOS:

- All iOS models have only 1 rear camera.
- 100% of iOS models have 1 front camera.

Android:

- 96.86% of Android models have 1 front camera, while the remaining models have 2 front cameras.

20 Primary Rear Camera Megapixels:

Univariate Analysis:-

Analysis of Primary Rear Camera Megapixels

- The highest peak at 48MP, 50MP, and 64MP accounts for 66.18% of smartphones, the highest concentration within a specific megapixel range.
- The second highest peak at 12MP and 13MP covers 16.74% of the dataset.
- The third highest peak at 108MP accounts for 8.54% of smartphones, forming the flattest segment.
- The distribution is positively skewed (skewness: 1.85), indicating a tendency towards higher megapixel values.
- Outliers are genuine, mainly 200MP camera models, with two-thirds from Chinese brands and the rest split between Samsung and Nokia.

Bivariate Analysis:-

Primary Rear Camera Megapixels And Price:

- The highest median price is 65449.5 for smartphones with a 12MP primary rear camera, followed by 58990.0 for 50.3MP, and so on.
- The highest density is around 50MP, followed by around 12MP, around 108MP, and around 200MP with the least density.
- The 12MP category is the only distribution that looks like a normal distribution in the whole dataset. It has the highest standard deviation and very few outliers.
- Scatters are only visible around 5MP, 8MP, 12MP, 13MP, 48MP, 50MP, 64MP, 108MP, and 200MP.
- **Kruskal-Wallis Test:**
    - Test statistic: 356.00
    - P-value: approximately 2.26e-67

  The small p-value indicates strong evidence to reject the null hypothesis, suggesting significant differences in median prices across different categories of the primary_camera_rear variable.

Primary Rear Camera Megapixels And Smartphone Brand:

- The majority of smartphone models have a 50.0MP primary rear camera, followed by 64.0MP, 48.0MP, 13.0MP, and 108.0MP.
- Every brand has models with a 50.0MP primary rear camera. 87.5% of brands have models with 13.0MP cameras, 87.5% with 48.0MP cameras, 75.0% with 64.0MP cameras, 75.0% with 108.0MP cameras, and 50.0% with 8.0MP cameras.
- 68.00% of smartphones with a 12MP primary rear camera are from Apple, which contributes to the highest median price for this category.

- **chi-square test:** The very high Chi-Square statistic (1171.38) and the extremely low p-value (essentially 0) suggest that the observed distribution of camera resolutions across different smartphone brands is not due to chance. There is a meaningful relationship between these two variables.
- **Cramer's V:** The Cramer's V value of 0.307 indicates a moderate association between Primary Rear Camera Megapixels and Smartphone Brand. While not a strong association, it is certainly more than a weak one

Primary Rear Camera Megapixels And Operating System:

iOS:

- 73.91% of iOS smartphone models have a 12.0MP primary rear camera, followed by 48.0MP at 17.39%, 50.0MP at 6.52%, and the least percentage is 2.17% for 13.0MP.

Android:

- Android smartphones have models in all categories of primary rear camera resolutions. The highest percentage is 36.31% for models with 50.0MP, followed by 19.78% for 64.0MP.
-

21 Primary Camera Front Megapixels:

Univariate Analysis:-

The primary front camera resolutions exhibit a multimodal distribution with multiple peaks. This positively skewed distribution (skewness: 1.43) shows most smartphones have lower resolutions, with a tail towards higher resolutions. Outliers above 30 MP make up about 19.7% of the dataset, with 82.51% from Chinese brands and the rest from Samsung and Nokia.

- The highest peak is at 16MP, covering 32.47% of the dataset.
- The second peak is at 8MP, contributing 17.81%, with 5MP adding 11.61% as a main contribution to the left side of the second peak. Both 8MP and 5MP make up the peak.
- The third peak is at 32MP, accounting for 16.35%.

These three peaks cover approximately 78.24% of the dataset, indicating their prevalence among smartphones.

Bivariate Analysis:-

Primary Camera Front Megapixels And Price:

- The highest average-priced smartphone is the Huawei Mate Xs 2, priced at 115000, featuring a 10.7MP primary front camera.

- The second-highest priced smartphone features a 40MP primary camera, all models belonging to Samsung 5G devices, with 83.3% of these smartphones equipped with 12GB of RAM.
- The third-highest priced category predominantly consists of smartphones with a 12MP primary camera, with 90.70% of them belonging to the Apple brand.
- **Spearman correlation correlation:** The Spearman correlation coefficient of approximately 0.561 indicates a moderately strong monotonic relationship between the two variables.
  The p-value is approximately 1.43×10−74, which is extremely small, providing strong evidence to reject the null hypothesis and conclude that there is a significant monotonic relationship between the variables.

  In summary, there is a statistically significant moderate positive monotonic relationship between the variables.

Primary Camera Front Megapixels And Smartphone Brand:

- **Chi-square test:** The chi-square test statistic (1967.96) is very large, and the p-value (approximately 1.285×10^−242) is extremely small. Therefore, I reject the null hypothesis and conclude that there is a very strong association between Primary Camera Front Megapixels and Smartphone Brand.
- **Cramer's V:** Cramer's V of 0.297 with a very small p-value of 9.767×10−289.767×10−28 indicates a statistically significant moderate association between Primary Camera Front Megapixels and Smartphone Brand.
- Only Samsung manufactures models with 40.0MP (2nd highest median price) and 10.0MP (4th highest median price).
- The Huawei Mate Xs 2 is the only smartphone model with a 10.7MP primary front camera, which has the highest median price.
- 12.0MP, the 3rd highest median price and 5th most populated category, with 90.70% of models belonging to Apple. Samsung follows with 6.98%, and Google has the remaining 2.33%; there are no Chinese models in this category.
- 7.0MP, the 9th highest median price, is exclusive to Apple.
- 10.1MP (6th highest median price) and 11.1MP (5th highest median price) are exclusive to Google.
- 10.8MP, the 7th highest median price, has only 3 models; 2 belong to Google, and 1 belongs to Apple.
- All brands except Apple and Google have models with a 16.0MP primary front camera. Despite being ranked 17th in median price, it makes up 32.47% of models, the highest in models count.
- All brands except Apple have models with an 8.0MP primary front camera. Despite being ranked 19th in median price, it makes up 17.81% of models, the second-highest in count.
- All brands except Apple and Google have models with a 32.0MP primary front camera. Despite being ranked 10th in median price (3rd from least), it makes up 16.35% of models, the third highest in count.

- All brands except Apple and Google have models with a 5.0MP primary front camera. Despite being ranked 20th in median price (2nd from least), it makes up 11.61% of models, the fourth highest in count.

Primary Camera Front Megapixels And Operating System:

iOS:

- There are only four categories of primary rear cameras. 84.78% of iOS models are equipped with a 12.0MP camera, followed by 7.0MP at 10.87%. The remaining models are equally distributed among 13.0MP and 10.8MP cameras.

Android:

- 83.06% of Android models are distributed among four categories of primary rear cameras: 16.0MP at 34.70%, 8.0MP at 18.98%, 32.0MP at 17.05%, and 5.0MP at 12.33%.

22 Extended Memory Feature:

Univariate Analysis:-

- Exactly 63.8% of smartphones in the dataset offer extended memory options.

Bivariate Analysis:-

Extended Memory Feature And Price:

- Smartphones with absence of extended memory feature have 166.68% higher average price than the ones which have
- Both distributions are right-skewed with numerous outliers; skewness is 5 for distributions with the feature and 1.5 for those without.
- The price distribution plot of smartphones with the extended_memory feature is more concentrated around its median compared to those without the feature.
- The standard deviation of the distribution without the extended_memory feature is higher than that with the feature.
- The mean and median of both distributions are not similar.
- Despite there being more models in the distribution with the extended_memory feature, the price range and interquartile range (IQR) of the distribution without the feature are higher.
- IQR of prices without the feature: 28,621.75 - 64,974.25.
- IQR of prices with the feature: 11,499.00 - 19,999.00.

- **Spearman Correlation Coefficient:** The Spearman correlation coefficient of -0.692 indicates a strong negative monotonic relationship between the variables, with a p-value of 6.098073303338923e-128, strongly rejecting the null hypothesis. Therefore, there is a statistically significant strong negative monotonic relationship between the variables analyzed.

Extended Memory Feature And Smartphone Brand:

- Neither Google nor Apple have a single smartphone model with the extended memory feature.
- Infinix and Nokia have all their models equipped with the extended memory feature.
- 68.75% of smartphone brands have more models with the extended memory feature available.
- In smartphone models with the extended memory feature available, Samsung has 18.66%, followed by Xiaomi with 15.32%, Vivo with 13.73%, Realme with 12.85%, and Oppo with 10.56%. All these percentages add up to 71.12%.

Extended Memory Feature And Operating System:

iOS:

- In iOS smartphones, there is not even a single model with extended memory feature.

Android:

- Total number of Android models with extended memory feature: 67.67%.

23 Extended Memory Upto:

Univariate Analysis:-

- The top three extended memory capacities are 1024GB, 512GB, and 256GB, accounting for 98.1% of smartphones in the dataset.
- Approximately 54.1% of smartphones support extended memory up to 1024GB.
- Around 23.7% of smartphones support extended memory up to 512GB.
- Approximately 20.3% of smartphones support extended memory up to 256GB.

Bivariate Analysis:-

Extended Memory Upto And Price:

- The highest median price, 17990.0, is for smartphones with 1000GB of extended memory, represented by the Nokia X100 5G.
- The second highest median price, 16900.0, is for smartphones with 2048GB of extended memory, all from the 'Infinix' brand with 5 models.
- The third highest median price, 16499.0, is for smartphones with 1024GB of extended memory, the most populated category.
- Both 512GB and 256GB have the same median price of 12999.0, with more models in the 512GB category than in the 256GB category.
- The lowest median price, 7499.0, is for smartphones with 128GB of extended memory, represented by three models from Chinese brands.
- **Spearman correlation coefficient:** the Spearman correlation coefficient of 0.269 with a p-value of 3.304e-09 indicates a statistically significant moderate positive monotonic relationship between extended memory available and price.

Extended Memory Upto And Smartphone Brands:

- **Cramer's V:** suggests that there is a moderate to strong correlation between extended memory upto and smartphone brands.

Extended Memory Upto And Operating System:

iOS:

- iOS doesn't produce models with memory extension options.

Android:

- In Android, 98.06% of models fall under the top 3 categories: 54.53% have 1024GB, 23.92% have 512GB, and 19.61% have 256GB.

24 Operating System:

Univariate Analysis:-

- 93.9% of smartphone models are equipped with Android, followed by 5.2% with iOS, and the remaining models belong to other operating systems.

Bivariate Analysis:-

Operating System And Price:

- iOS smartphones have the highest average price, while Android has the lowest average price, likely because Android includes a wide range of priced smartphones, whereas iOS only has high-priced models.
- iOS median price is 347.37% higher than Android
- Android has numerous outliers and is the only distribution with outliers.