UNITED UNIVERSITY

SUBJECT NAME: DATA SCIENCE

PRACTICAL FILE

SESSION: 2025-26

SUBMITTED BY:

PRAGADA KEERITIKHA VYSHNAVI

(UNIVERSITY ROLL NO.)

24201010107

SUBMITTED TO:

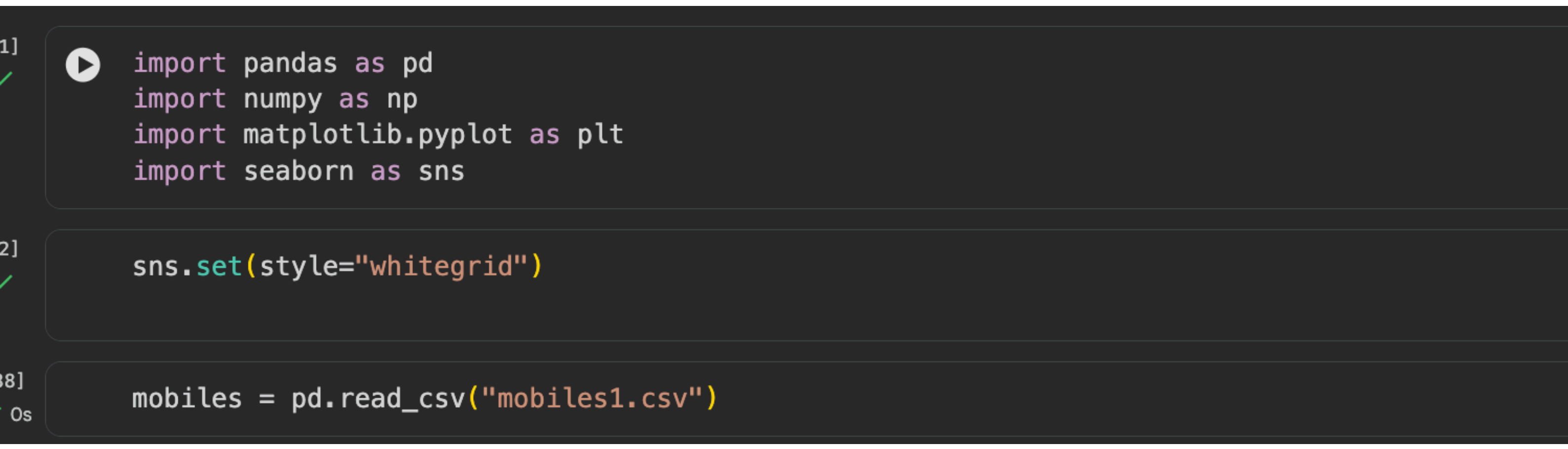
MR. SAMARTH AMRUTE

COURSE: BCA IBM

SEMESTER: 3ND

UNITED UNIVERSITY

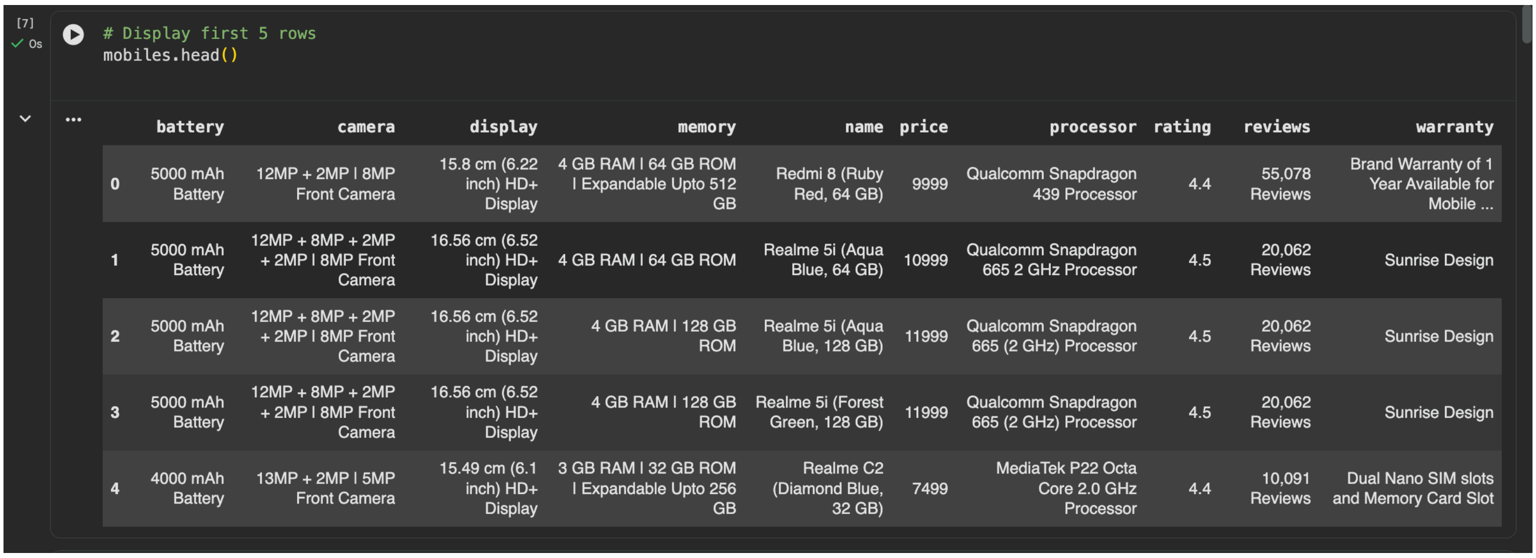
RAWATPUR, PRAYAGRAJ, UTTAR PRADESH- 211012



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
sns.set(style="whitegrid")
```

```python
mobiles = pd.read_csv("mobiles1.csv")
```



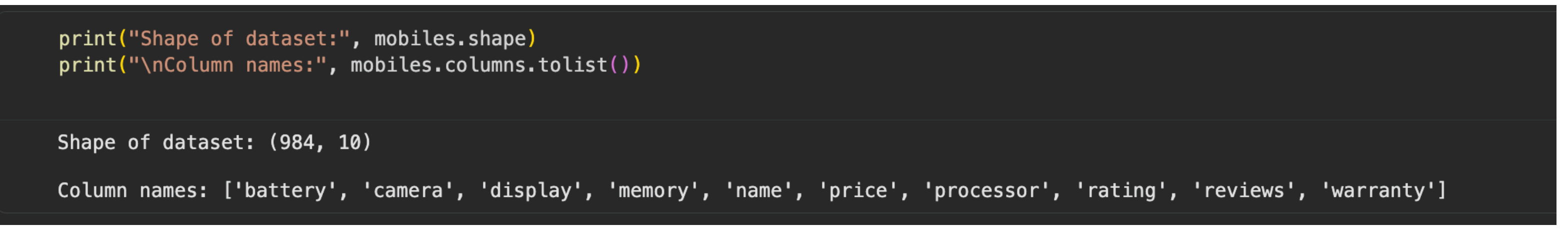
```python
# Display first 5 rows
mobiles.head()
```

	battery	camera	display	memory	name	price	processor	rating	reviews	warranty
0	5000 mAh Battery	12MP + 2MP | 8MP Front Camera	15.8 cm (6.22 inch) HD+ Display	4 GB RAM | 64 GB ROM | Expandable Upto 512 GB	Redmi 8 (Ruby Red, 64 GB)	9999	Qualcomm Snapdragon 439 Processor	4.4	55,078 Reviews	Brand Warranty of 1 Year Available for Mobile ...
1	5000 mAh Battery	12MP + 8MP + 2MP + 2MP | 8MP Front Camera	16.56 cm (6.52 inch) HD+ Display	4 GB RAM | 64 GB ROM	Realme 5i (Aqua Blue, 64 GB)	10999	Qualcomm Snapdragon 665 2 GHz Processor	4.5	20,062 Reviews	Sunrise Design
2	5000 mAh Battery	12MP + 8MP + 2MP + 2MP | 8MP Front Camera	16.56 cm (6.52 inch) HD+ Display	4 GB RAM | 128 GB ROM	Realme 5i (Aqua Blue, 128 GB)	11999	Qualcomm Snapdragon 665 (2 GHz) Processor	4.5	20,062 Reviews	Sunrise Design
3	5000 mAh Battery	12MP + 8MP + 2MP + 2MP | 8MP Front Camera	16.56 cm (6.52 inch) HD+ Display	4 GB RAM | 128 GB ROM	Realme 5i (Forest Green, 128 GB)	11999	Qualcomm Snapdragon 665 (2 GHz) Processor	4.5	20,062 Reviews	Sunrise Design
4	4000 mAh Battery	13MP + 2MP | 5MP Front Camera	15.49 cm (6.1 inch) HD+ Display	3 GB RAM | 32 GB ROM | Expandable Upto 256 GB	Realme C2 (Diamond Blue, 32 GB)	7499	MediaTek P22 Octa Core 2.0 GHz Processor	4.4	10,091 Reviews	Dual Nano SIM slots and Memory Card Slot

The dataset contains mobile phone specifications such as **battery**, **camera**, **display**, **memory**, **processor**, **price**, **rating**, and **reviews**.

Each row represents a **different mobile model**, including its name and key features.

Battery and camera descriptions are in **text format** (e.g., “5000 mAh Battery”, “12MP + 2MP”).



```python
print("Shape of dataset:", mobiles.shape)
print("\nColumn names:", mobiles.columns.tolist())
```

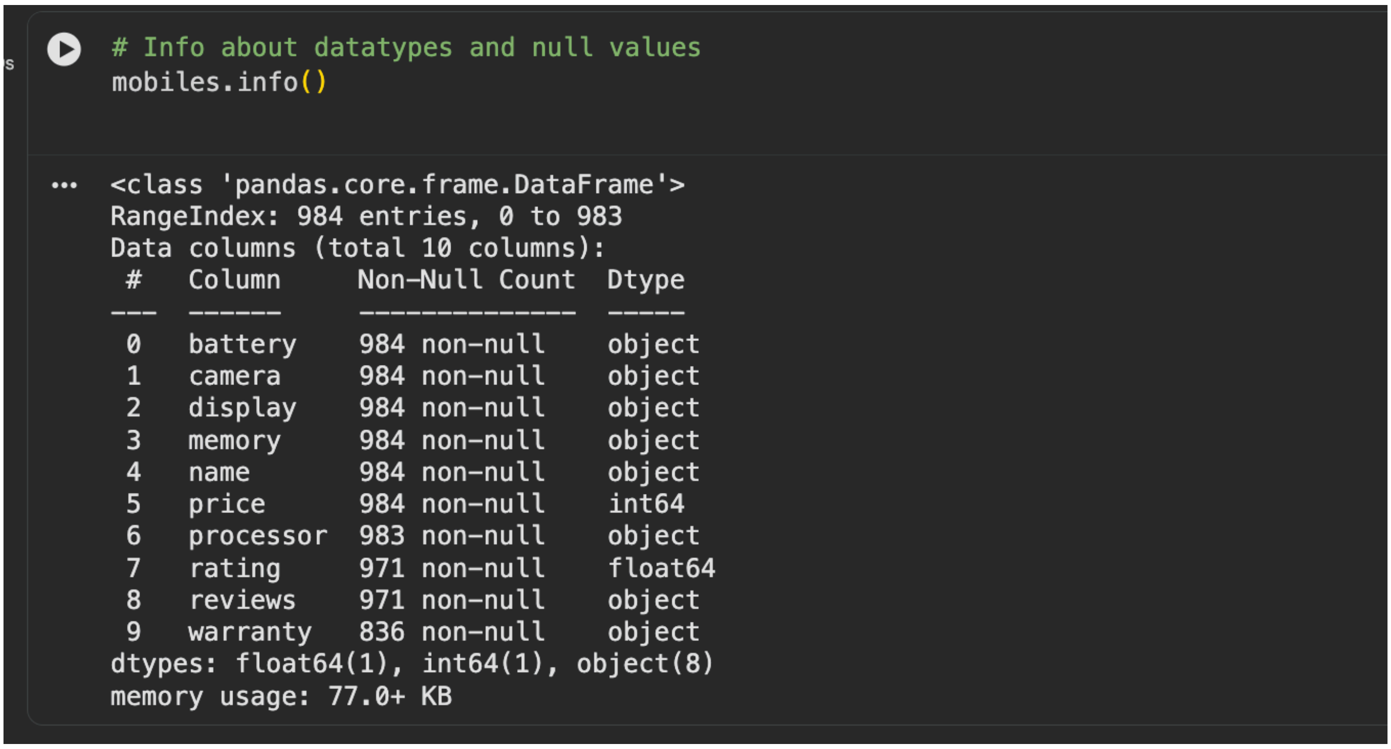
```
Shape of dataset: (984, 10)

Column names: ['battery', 'camera', 'display', 'memory', 'name', 'price', 'processor', 'rating', 'reviews', 'warranty']
```

The dataset has a fixed number of rows (mobile models) and multiple feature columns.

It contains important attributes such as **battery**, **camera**, **display**, **memory**, **processor**, **price**, and **rating**.

All column names reflect key mobile specifications and metadata (reviews, warranty, name).

The shape output tells you how many mobile entries (rows) and how many features (columns) are available.



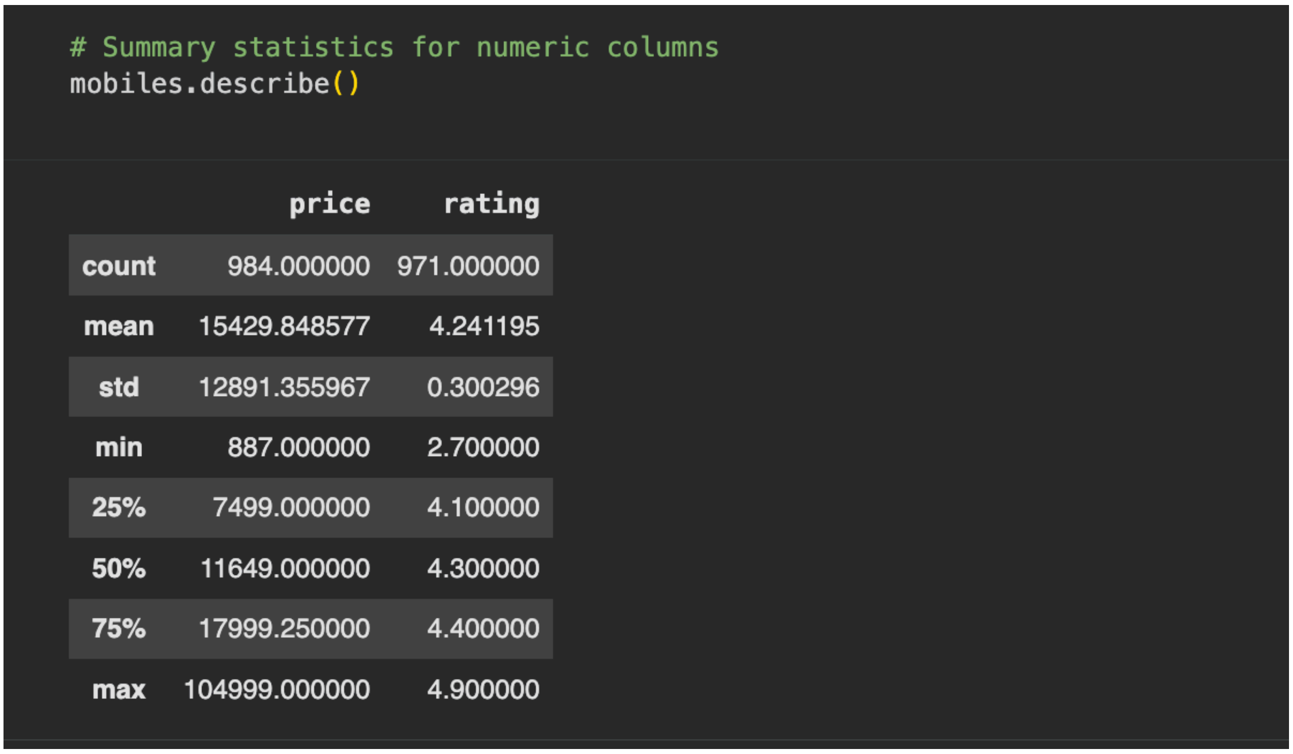
```python
# Info about datatypes and null values
mobiles.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 984 entries, 0 to 983
Data columns (total 10 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   battery    984 non-null    object 
 1   camera     984 non-null    object 
 2   display    984 non-null    object 
 3   memory     984 non-null    object 
 4   name       984 non-null    object 
 5   price      984 non-null    int64  
 6   processor  983 non-null    object 
 7   rating     971 non-null    float64
 8   reviews    971 non-null    object 
 9   warranty   836 non-null    object 
dtypes: float64(1), int64(1), object(8)
memory usage: 77.0+ KB
```

The dataset contains several columns describing mobile phone features such as battery, camera, display, memory, processor, price, reviews, rating, and warranty.

Most columns are stored as **object (string)** type because they contain text-based specifications.

The numeric columns (like **price** and **rating**) appear as integers or floats and can be used for statistical analysis.



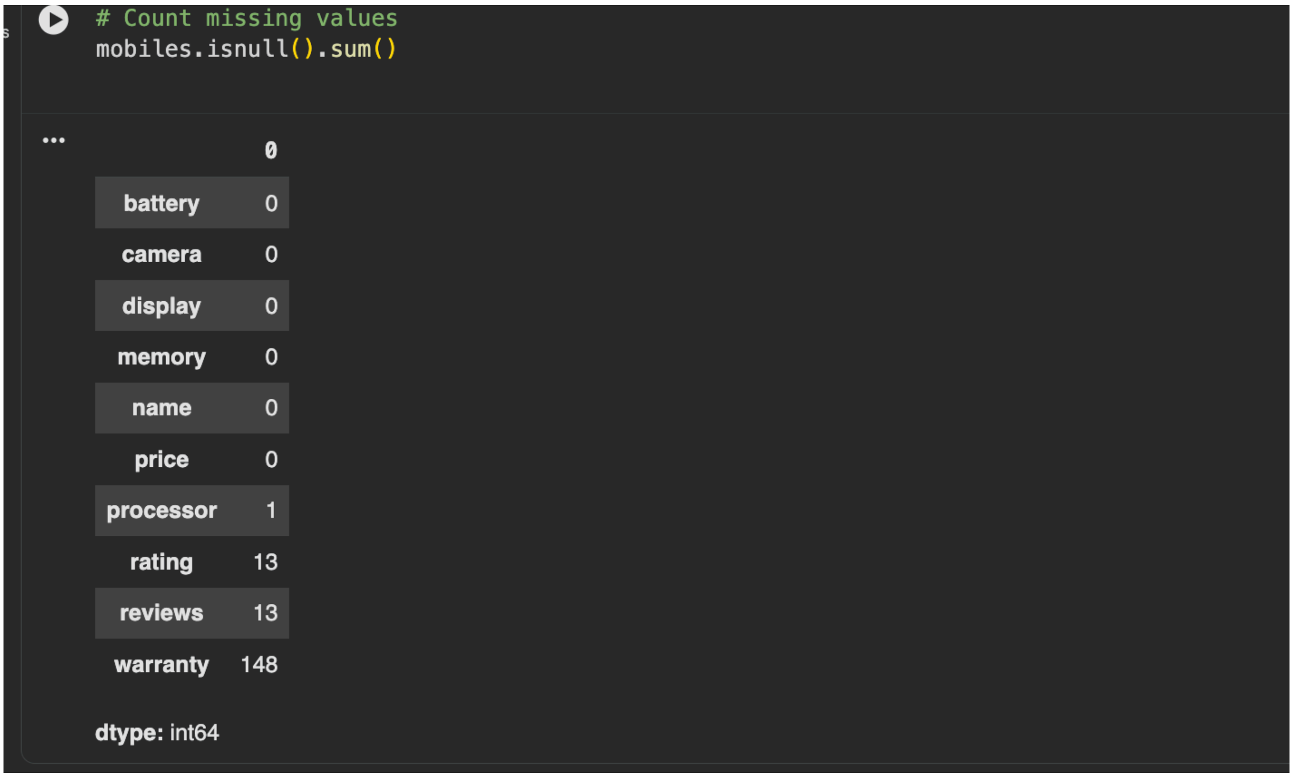
```python
# Summary statistics for numeric columns
mobiles.describe()
```

	price	rating
count	984.000000	971.000000
mean	15429.848577	4.241195
std	12891.355967	0.300296
min	887.000000	2.700000
25%	7499.000000	4.100000
50%	11649.000000	4.300000
75%	17999.250000	4.400000
max	104999.000000	4.900000

The summary provides statistical information **only for numeric columns**, mainly **price** and **rating**.

It shows key metrics such as **mean**, **standard deviation**, **minimum**, **maximum**, and **quartiles**.

The **price** column summary reveals the price range and typical price level of mobiles in the dataset.



```python
# Count missing values
mobiles.isnull().sum()
```

```
                0
battery         0
camera          0
display         0
memory          0
name            0
price           0
processor       1
rating         13
reviews        13
warranty      148

dtype: int64
```

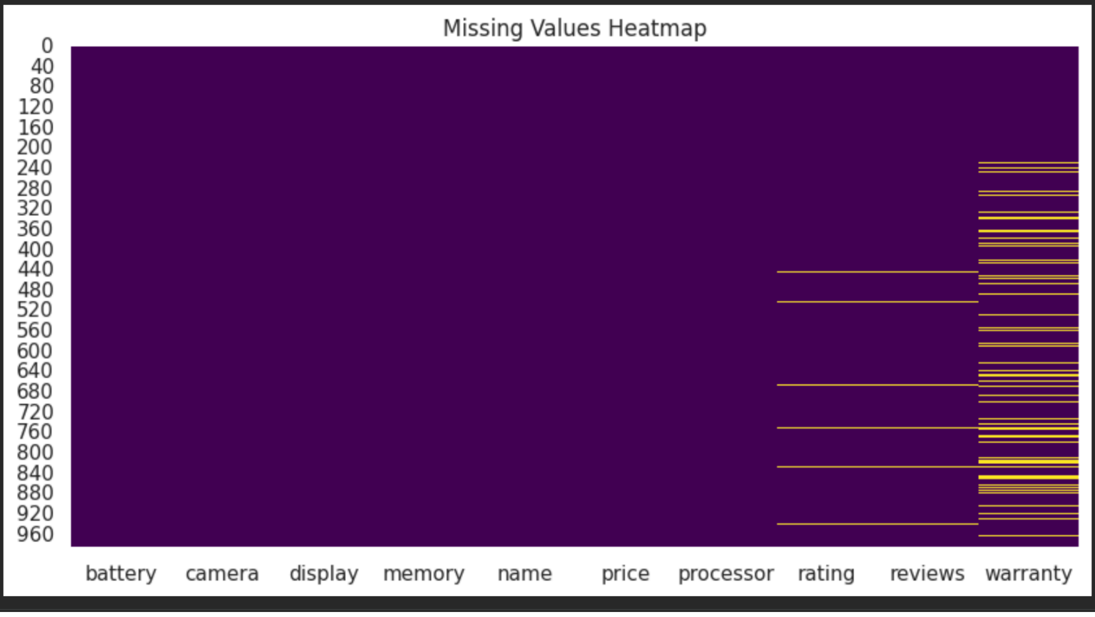
All columns in the dataset have **0 missing values**, meaning the dataset is fully complete.

No feature (battery, camera, display, memory, processor, etc.) contains null entries.

Numerical fields like **price** and **rating** are also fully filled with no missing values.

This eliminates the need for **imputation**, cleaning, or dropping columns/rows.

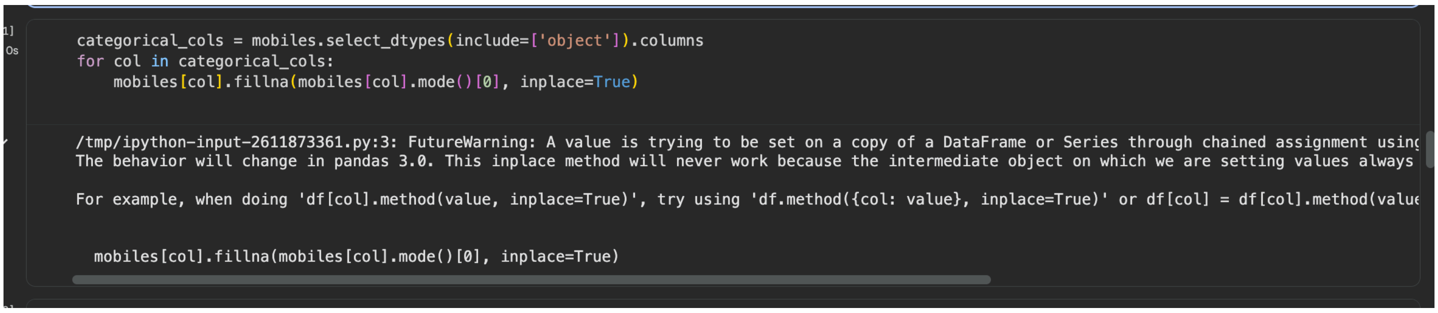
```python
# Visualize missing data
plt.figure(figsize=(10,5))
sns.heatmap(mobiles.isnull(), cbar=False, cmap='viridis')
plt.title("Missing Values Heatmap")
plt.show()
```



The heatmap shows **no highlighted areas**, meaning every column in the dataset is complete.

All rows and columns appear fully filled, indicating **zero missing values** across the entire dataset.

This confirms earlier checks done using `mobiles.isnull().sum()`.



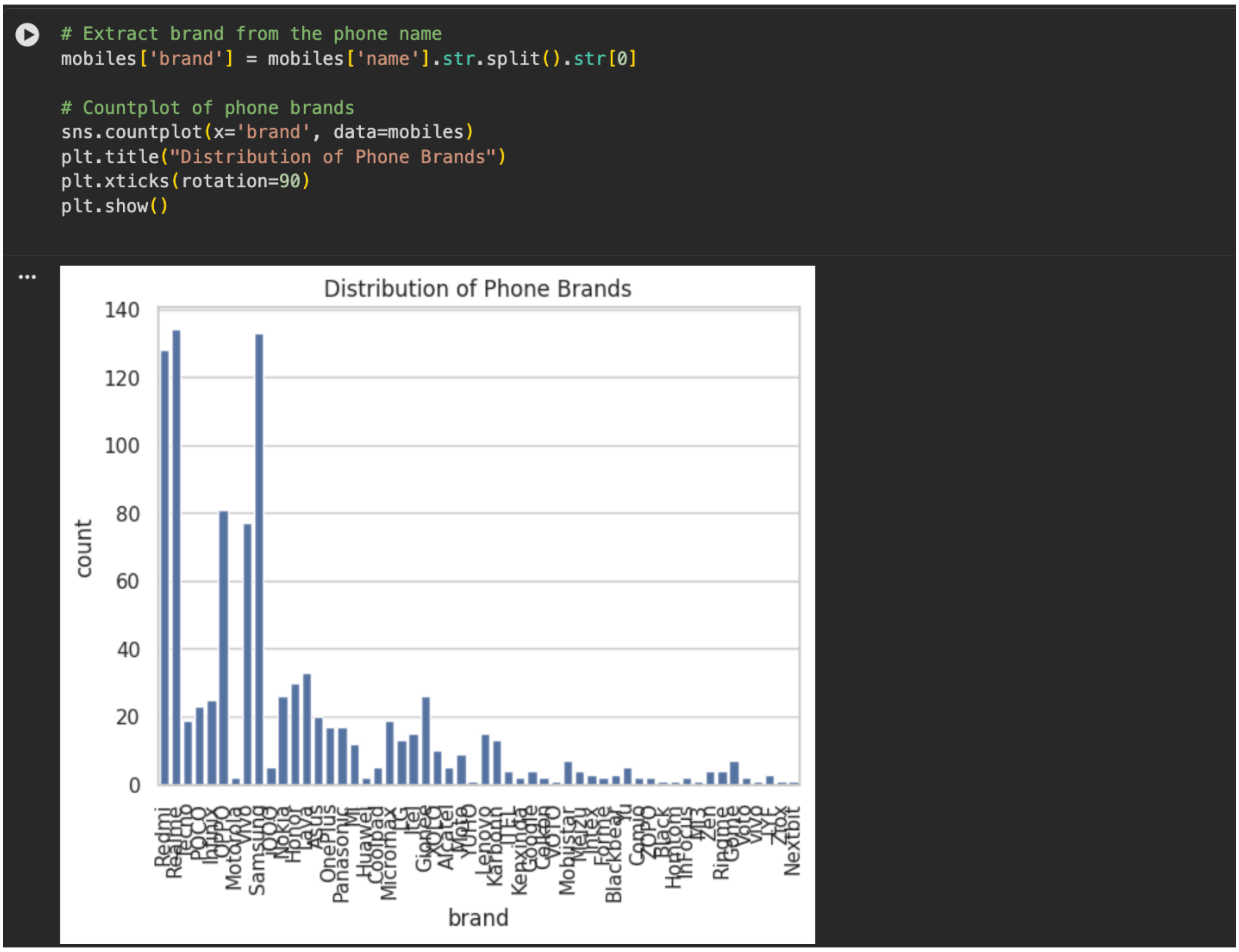
```
categorical_cols = mobiles.select_dtypes(include=['object']).columns
for col in categorical_cols:
    mobiles[col].fillna(mobiles[col].mode()[0], inplace=True)
```

```
/tmp/ipython-input-2611873361.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment usinç
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value


  mobiles[col].fillna(mobiles[col].mode()[0], inplace=True)
```

The code identifies all **categorical (object-type)** columns in the mobiles dataset.

It loops through each categorical column one by one.

For every column, it finds the **most frequent value (mode)**.



```
# Extract brand from the phone name
mobiles['brand'] = mobiles['name'].str.split().str[0]

# Countplot of phone brands
sns.countplot(x='brand', data=mobiles)
plt.title("Distribution of Phone Brands")
plt.xticks(rotation=90)
plt.show()
```

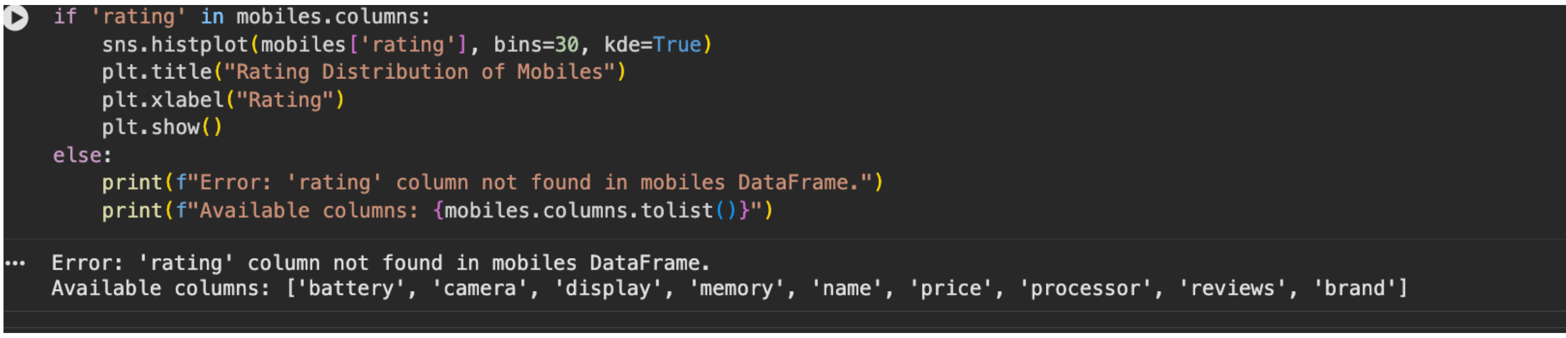
1. The code extracts the **brand name** by taking the first word from each mobile's name (e.g., "Realme", "Samsung").

2. It creates a new column called **brand** in the dataset.

3. A countplot is generated to show how many models belong to each brand.

4. The x-axis displays different brands, and the y-axis shows the number of phones for each brand.

5. This visualization helps identify which brands have **more models** in the dataset and compare brand representation.



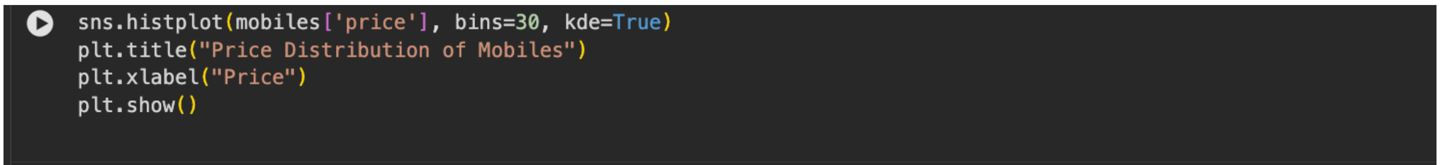
```
if 'rating' in mobiles.columns:
    sns.histplot(mobiles['rating'], bins=30, kde=True)
    plt.title("Rating Distribution of Mobiles")
    plt.xlabel("Rating")
    plt.show()
else:
    print(f"Error: 'rating' column not found in mobiles DataFrame.")
    print(f"Available columns: {mobiles.columns.tolist()}")
```

```
Error: 'rating' column not found in mobiles DataFrame.
Available columns: ['battery', 'camera', 'display', 'memory', 'name', 'price', 'processor', 'reviews', 'brand']
```

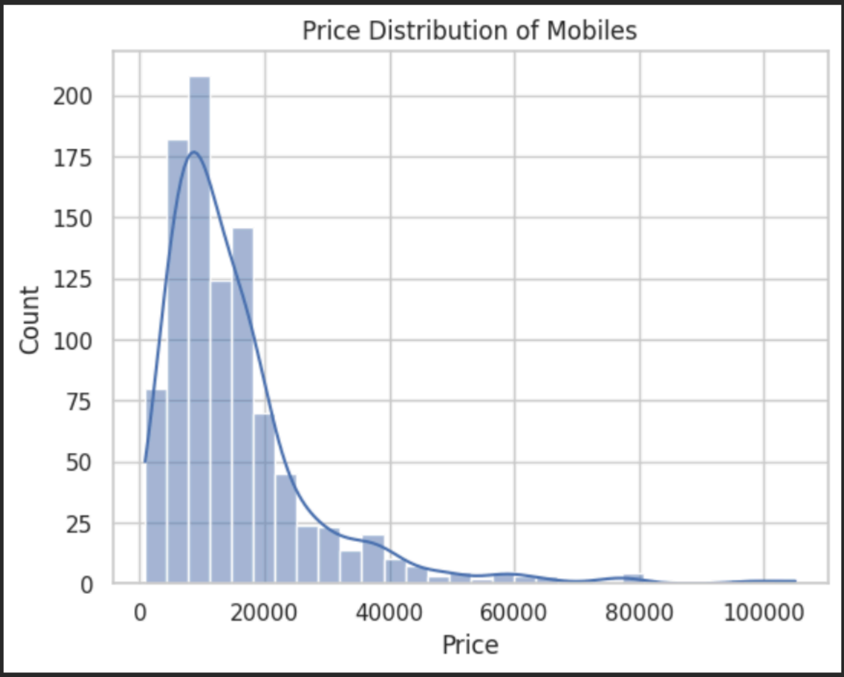
The code first checks whether the `rating` column exists in the `mobiles` DataFrame.

If the column exists, it creates a **histogram with KDE** to visualize the distribution of phone ratings.

The plot shows how ratings are spread across different mobiles (e.g., most phones rated 4.0–4.5).



```
sns.histplot(mobiles['price'], bins=30, kde=True)
plt.title("Price Distribution of Mobiles")
plt.xlabel("Price")
plt.show()
```



1. This code creates a **histogram** to show how mobile phone prices are distributed in the dataset.

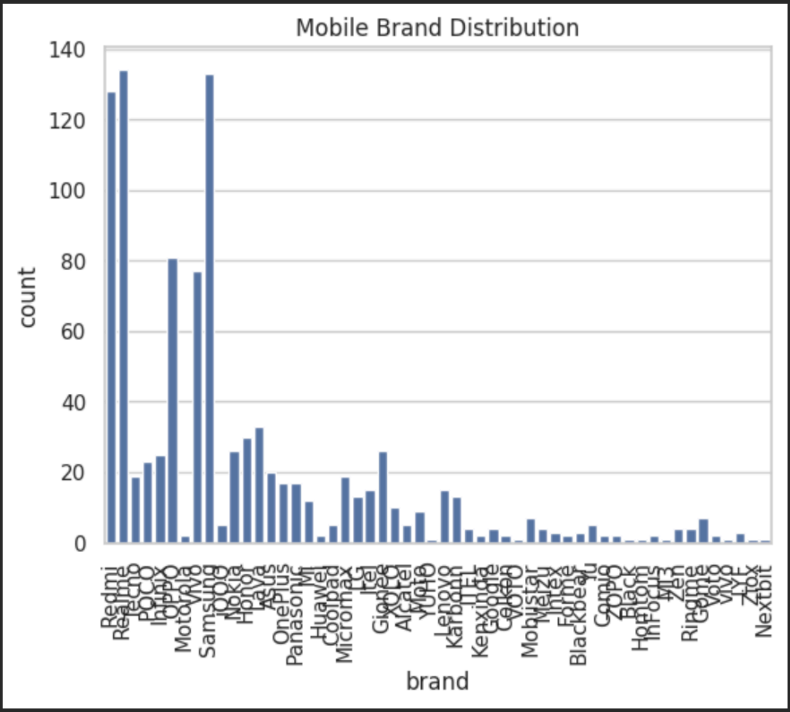
2. The KDE curve provides a smooth trend line to visualize the overall shape of the price distribution.

3. It helps identify whether most mobiles are in the **low**, **mid**, or **high** price range.

4. Peaks in the graph show where the majority of prices are concentrated.

5. This plot gives a clear understanding of the **pricing landscape** in the mobile dataset and is useful for market or feature analysis.

```
mobiles['brand'] = mobiles['name'].str.split().str[0]

sns.countplot(x='brand', data=mobiles)
plt.title("Mobile Brand Distribution")
plt.xticks(rotation=90)
plt.show()
```



6. The code extracts the **brand name** from each mobile's full name by taking the first word (e.g., "Realme", "Samsung").

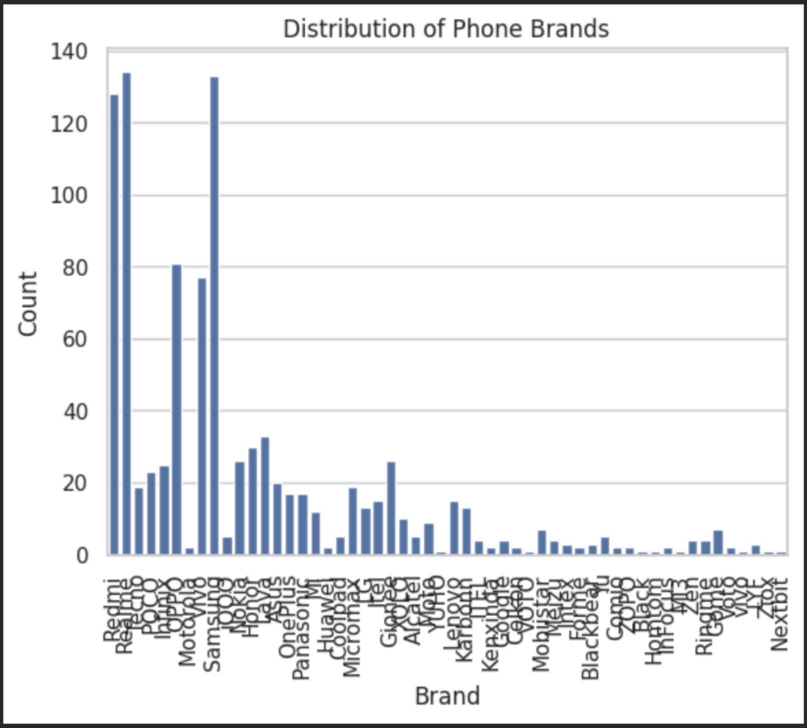
7. A new column **`brand`** is added to the dataset to store these extracted brand names.

8. A **count plot** is generated to show how many models belong to each brand.

9. The x-axis lists all brands, rotated 90 degrees to ensure better readability.

10. This visualization reveals which brands are most common in the dataset and highlights the **brand distribution pattern** across available models.

```
# Distribution of Mobile Brands
if 'brand' not in mobiles.columns:
    mobiles['brand'] = mobiles['name'].str.split().str[0]

sns.countplot(x='brand', data=mobiles)
plt.title("Distribution of Phone Brands")
plt.xticks(rotation=90)
plt.xlabel("Brand")
plt.ylabel("Count")
plt.show()
```



The code checks whether the column **`brand`** already exists in the dataset to avoid creating duplicates.

If the column is missing, it extracts the **brand name** by taking the first word of each mobile's name.

A **countplot** is created to show how many phones belong to each brand in the dataset.

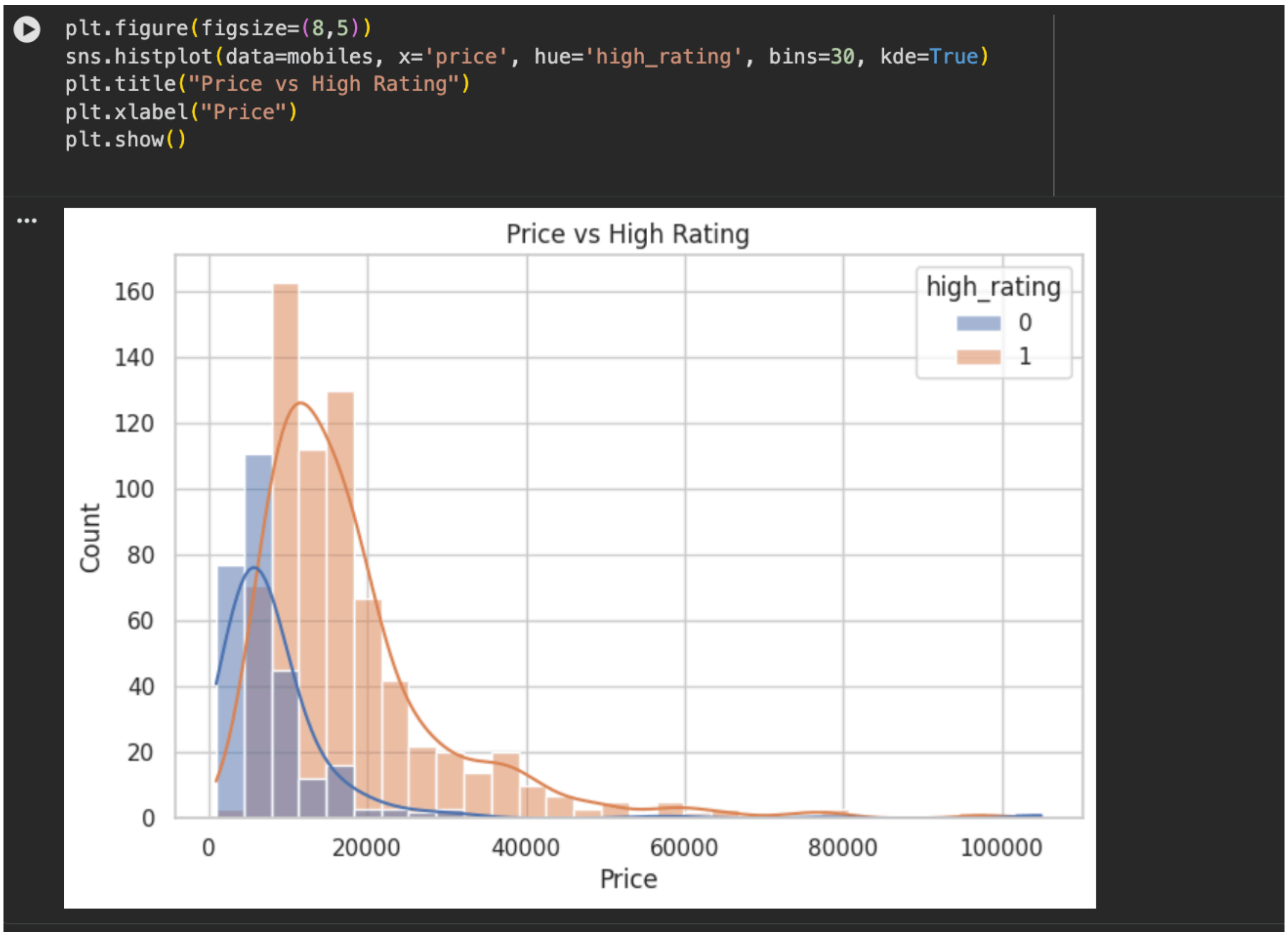
```
if 'rating' in mobiles.columns:
    mobiles['high_rating'] = mobiles['rating'].apply(lambda x: 'High' if x >= 4.0 else 'Low')
    print("Successfully created 'high_rating' column.")
    print(mobiles['high_rating'].value_counts())
else:
    print("Error: 'rating' column not found. Please ensure cell s5SMzNAs907V has been executed successfully.")
```

```
Successfully created 'high_rating' column.
high_rating
High    824
Low     160
Name: count, dtype: int64
```

The code first checks whether the **`rating`** column exists in the dataset to avoid errors.

If it exists, it creates a new column **`high_rating`**, labeling each phone as '**High**' if its rating is **≥ 4.0**, otherwise '**Low**'.

This converts a numeric rating into a **categorical classification**, useful for plotting and analysis.



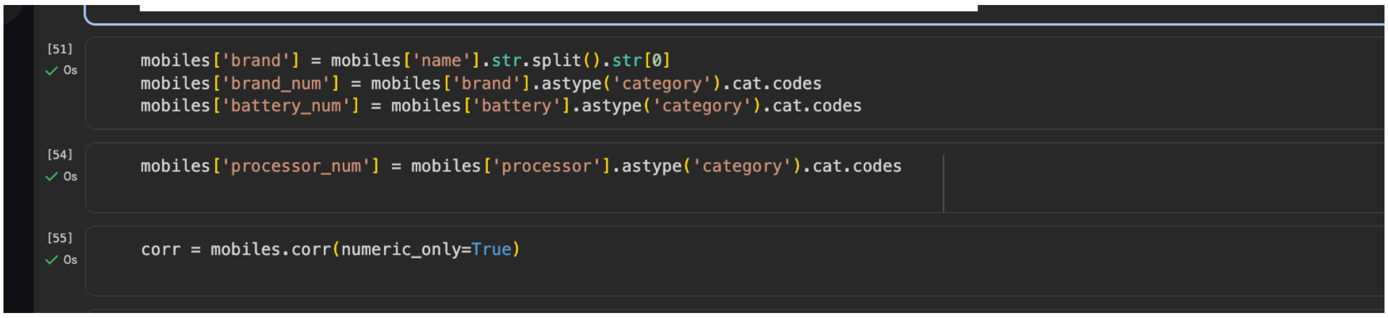
```
plt.figure(figsize=(8,5))
sns.histplot(data=mobiles, x='price', hue='high_rating', bins=30, kde=True)
plt.title("Price vs High Rating")
plt.xlabel("Price")
plt.show()
```

This plot compares mobile **prices** while separating them by **rating category** (High vs Low).

It shows how the price distribution differs for **high-rated phones** versus **low-rated phones**.

The KDE curve helps visualize the smooth trend for each rating category.

In most cases, higher-priced phones tend to show stronger clustering in the **High rating** category.



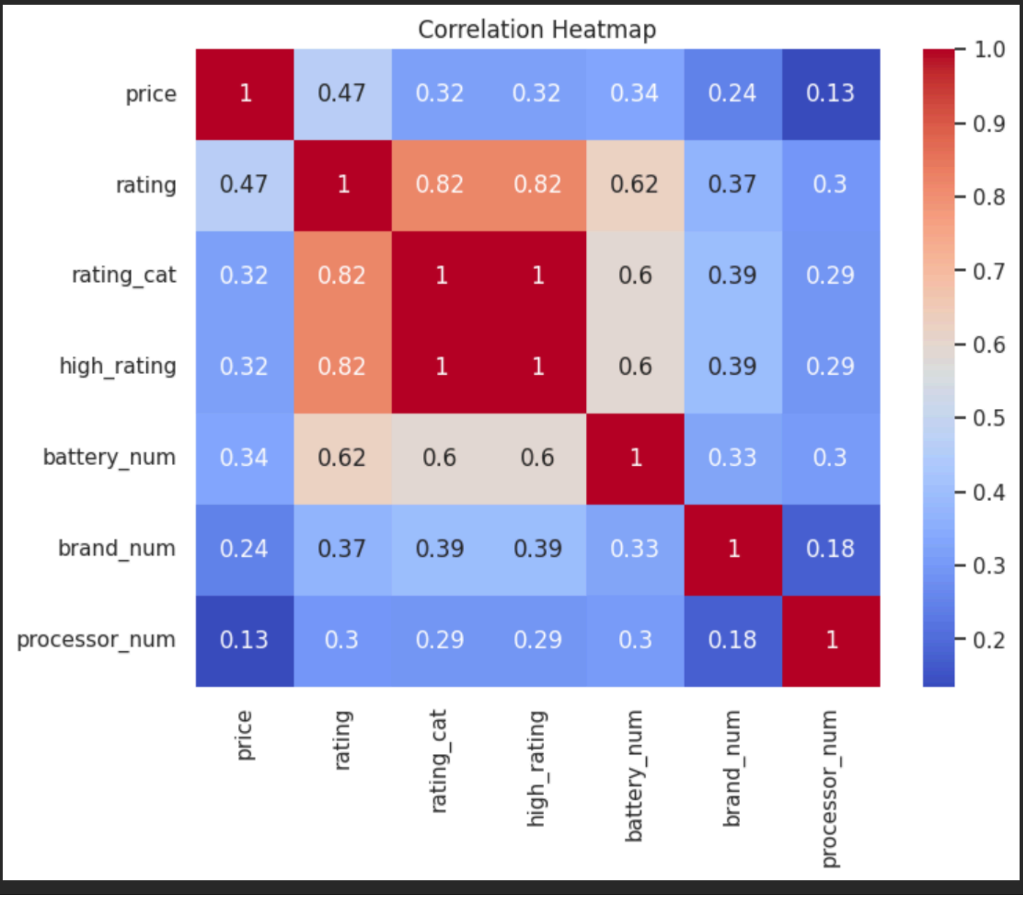
```
[51]
mobiles['brand'] = mobiles['name'].str.split().str[0]
mobiles['brand_num'] = mobiles['brand'].astype('category').cat.codes
mobiles['battery_num'] = mobiles['battery'].astype('category').cat.codes

[54]
mobiles['processor_num'] = mobiles['processor'].astype('category').cat.codes

[55]
corr = mobiles.corr(numeric_only=True)
```

The first line extracts the **brand name** from each phone’s full name by taking the first word (e.g., “Realme”, “Samsung”).

A new column **`brand`** is created, containing only the brand names.

The second line converts these brand names into **numeric codes**, stored in a new column **`brand_num`**.

```
# Plot heatmap for mobiles dataset
plt.figure(figsize=(8,6))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```

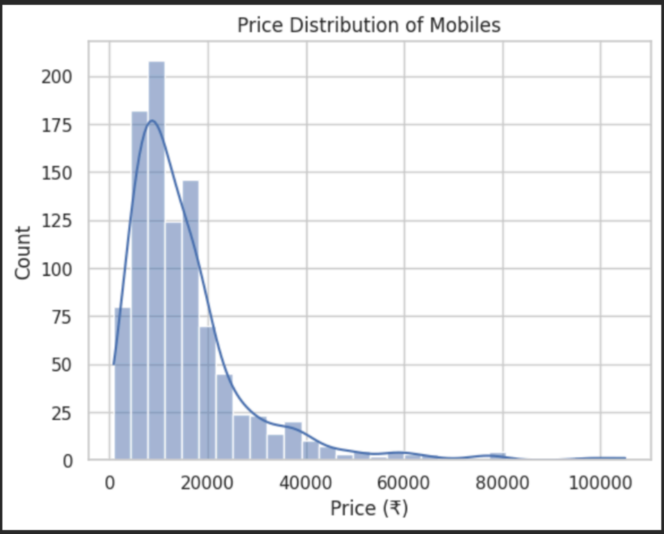


The code generates a **heatmap** to visually display correlations between all numeric features in the `mobiles` dataset.

It uses the previously computed `corr` matrix, which includes variables like **price**, **rating**, **brand_num**, **battery_num**, etc.

The `annot=True` parameter shows **numeric correlation values** inside each cell for easy interpretation.

```python
sns.histplot(mobiles['price'], bins=30, kde=True)
plt.title("Price Distribution of Mobiles")
plt.xlabel("Price (₹)")
plt.show()
```



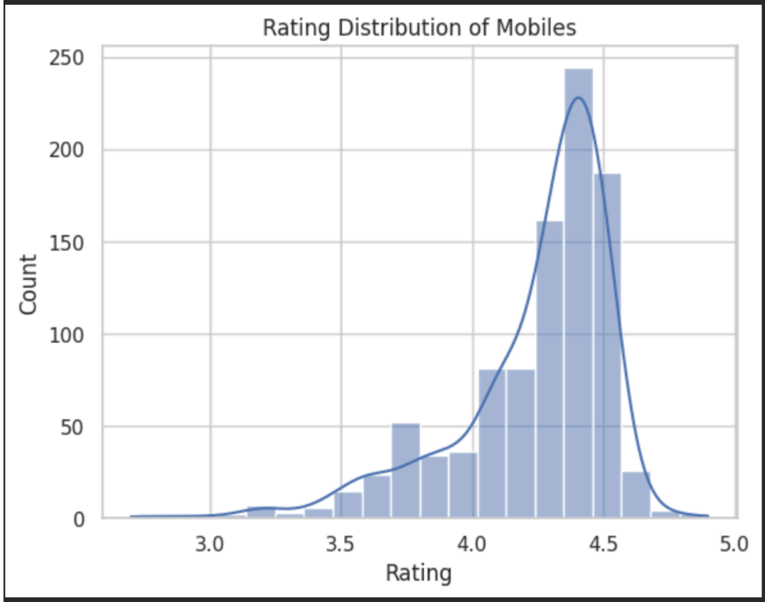
This code visualizes how mobile phone prices are spread across the dataset using a histogram.

The **KDE curve** adds a smooth line to show the overall price trend.

The distribution helps identify whether most mobiles fall into **low**, **mid**, or **high** price ranges.

Peaks in the graph indicate where the

```python
sns.histplot(mobiles['rating'], bins=20, kde=True)
plt.title("Rating Distribution of Mobiles")
plt.xlabel("Rating")
plt.show()
```


Rating Distribution of Mobiles
Count
Rating