Business Case

1. Project Title :- Internship Project

Project Description :- On the basis of the mobile Specification like Battery power, 4G or 3G enabled , wifi ,Bluetooth, Ram etc predict the Price range of the mobile.Prepare a complete data analysis report on the given data.

2. Project Team Id , Project Id and Deadlines

- **Project Team ID** : PTID-CDS-NOV-25-3358
- **Project ID** : PRCP-1009
- **Institute** : DataMites

3.Problem Statements and Goals

3.1 Problem Statements :

Task 1 :

- Prepare a complete data analysis report on the given data.

Task 2 :

- On the basis of the mobile Specification like Battery power, 3G enabled , wifi ,Bluetooth, Ram etc predict the Price range of the mobile.

Task3 :

- Prepare the analysis report stating how model will help expanding the business by stating several factors including feature importance.

3.2 Goals :

- Perform detailed data analysis to understand trends, correlations, and key features affecting mobile prices.
- Build and test a machine learning model to predict the price range of mobiles using specifications like RAM, battery, and connectivity features.
- Use feature importance analysis to identify the most influential specs driving price.
- Prepare a business insight report explaining how model predictions can guide pricing, marketing, and product design decisions.
- Overall goal: enable data-driven business expansion through predictive insights and smart product positioning.

Insides of Dataset

1. What is this dataset about?

This is a **mobile phone specification** dataset used for **mobile price range prediction.**

- **Rows:**- 2000 mobile phones
- **Columns (features):**- 21
- **Problem type:**- Multiclass classification
- **Target column:**- price_range

price_range has **4** classes:

- **0:**– low cost
- **1:**– medium cost
- **2:**– high cost
- **3:**– very high cost

Each class has exactly **500 samples,** so the target is **perfectly balanced.**

2. Structure & data quality

Shape

- **Rows:**- 2000
- **Columns:**- 21

Missing values

- No missing values

Duplicates

- No duplicate rows

Data types

All columns are numeric: int64 or float64. No categorical/string columns.

3. Feature-by-feature domain meaning

I'll group them logically as they relate to a smartphone.

A. Target variable

- **price_range (0–3)** Indicates price segment (low → high). It's **ordinal** (0 < 1 < 2 < 3), but usually treated as multiclass labels.

B. Hardware performance & memory

- **ram (int)-:** RAM size (likely in MB).
  - Min:- 256
  - Max:- 3998
  - Strongest positive correlation with price_range (~0.92).
  - Higher RAM → Higher price segment.
- **int_memory (int)-:** Internal storage in GB.
  - Range: 2 to 64
  - Weak positive correlation with price.
  - More storage tends to mean slightly higher price, but not as strong as RAM.
- **clock_speed (float)-:** Processor clock speed in GHz.
  - Range: 0.5 to 3.0
  - Slight negative correlation with price (basically no clear trend).
  - Surprisingly, clock speed alone does not clearly separate price ranges in this dataset.
- **n_cores (int)-:** Number of CPU cores.
  - Range: 1 to 8
  - Very low correlation with price.
  - Manufacturers across price ranges might use similar core counts.

C. Battery & power

- **battery_power (int)-:** Battery capacity in mAh.
  - Range: 501 to 1998
  - Moderately positively correlated (~0.20) with price.
  - Higher-priced phones tend to have bigger batteries, but not as strongly as RAM.

D. Camera features

- **fc (int)-:** Front camera megapixels.
  - Range: 0 to ~19
  - Very weak positive correlation.
- **pc (int)-:** Primary (rear) camera megapixels.
  - Range: 0 to 20
  - Weak positive correlation.
  - Better cameras are somewhat associated with higher price, but not strongly – maybe because multiple phones across segments share similar MP counts.

E. Display & design

- **px_height (int)-:** Pixel height of the screen (resolution).
  - Range: 0 to 1960
  - Positive correlation with price (~0.15).
- **px_width (int)-:** Pixel width of the screen.
  - Range: 500 to 1998
  - Positive correlation with price (~0.17).
  - Higher resolution screens are more common in higher price segments.
- **sc_h (int)-:** Screen height in cm.
  - Range: 5 to 19
- **sc_w (int)-:** Screen width in cm.
  - Range: 0 to 18
  - Slight positive correlation.
  - Bigger screens tend to be slightly more expensive, but not a strong signal.
- **mobile_wt (int)-:** Weight of the mobile in grams.
  - Range: 80 to 200
  - Slight negative correlation with price.
  - Heavier phones are not necessarily more expensive; light-weight designs might correspond to premium devices.
- **m_dep (float):-** Mobile depth (thickness) in cm.
  - Range: 0.1 to 1.0
  - Almost no correlation with price.

F. Connectivity & extra features (binary flags)

All these are **0/1** features:

- **blue** – Bluetooth present or not
- **dual_sim** – Dual SIM support
- **four_g** – 4G support
- **three_g** – 3G support
- **touch_screen** – Touch screen vs non-touch
- **wifi** – Wi-Fi capability

All of them show very low positive/negative correlations with price_range (near zero).

Interpretation:

- **3G, 4G, Wi-Fi, Bluetooth, dual SIM, touch screen** are fairly standard across price ranges in this dataset.
- These features **do not strongly separate** cheap vs expensive phones here, probably because even low-cost devices in this data often include them. **G. Battery usage**
- **talk_time (int):**- Maximum talk time in hours.
  - Range: 2 to 20
  - Very weak positive correlation with price.

4. Statistical & domain insights

4.1 Class-wise averages (selected features)

Average values per price_range:

- **RAM (most important feature):**-
  - Class 0: ~785 MB
  - Class 1: ~1680 MB
  - Class 2: ~2583 MB
  - Class 3: ~3449 MB
- **Battery power:**-
  - Class 0: ~1117 mAh
  - Class 1: ~1229 mAh
  - Class 2: ~1228 mAh
  - Class 3: ~1380 mAh
- **Screen resolution (px width & height):**-
  - Width (avg):
    - 0: ~1150
    - 1: ~1252
    - 2: ~1234
    - 3: ~1370
  - Height (avg):
    - 0: ~536
    - 1: ~667
    - 2: ~632
    - 3: ~745
- **Internal memory:**-
  - Averages hover around 31–34 GB across all classes, weak trend. **Key takeaway:**-

The **single strongest driver** of price range in this dataset is **RAM,** followed by **battery power and screen resolution.** Basic connectivity features (3G/4G/Wi-Fi/Bluetooth) don't differentiate price much.

5. Modeling view (how this domain behaves for ML)

From a machine learning / domain perspective:

- **Task:**-
  - Multi-class classification with 4 balanced classes.
  - Can also be treated as ordinal classification if you want to use the ordering (0 < 1 < 2 < 3).
- **Most predictive features (based on correlation with target):**-
  - Very high: ram
  - Medium: battery_power, px_width, px_height
  - Low but somewhat useful: int_memory, sc_w, pc
- **Weak features (low information about price):**-
  - clock_speed, mobile_wt, m_dep, and all binary flags (blue, dual_sim, three_g, four_g, touch_screen, wifi) have very low correlation with price.
  - They may still be useful in non-linear models, but they're not strong standalone indicators.
- **Preprocessing needs:**-
  - No missing values,no heavy imputation needed.
  - All numeric → no label encoding/categorical encoding needed.
  - Feature scales differ a lot (e.g., ram up to 4000 vs m_dep < 1), so **scaling/normalization** is recommended for distance-based or gradient-based models (e.g., KNN, SVM, logistic regression, neural nets). Tree-based models cope better without scaling. **No obvious data leakage:**-
  - All features are typical specs someone could know **before** deciding a price.
  - No weird "ID-like" or "post-price" features.

6. Plain-language domain summary

- This file represents **smartphone technical specs** for 2000 phones.
- We want to **predict the phone's price segment** (0–3) based only on specs.
- In this data:
  - **RAM is the main deciding factor** for price.
  - **Battery capacity and screen resolution** also influence price.
  - **Connectivity flags** (3G/4G/Wi-Fi/Bluetooth/dual SIM/touch screen) are almost commoditized and don't strongly separate low vs high price phones in this dataset.
  - Data is **clean, balanced, and well-suited** for training classification models.

Domain Analysis

Dataset Overview

Property & Value

- Total rows 2000
- Total features 21
- Target column price_range
- Type of ML Task Multiclass Classification (0–3)
- Missing values None
- Duplicate rows None
- Data type distribution 100% numeric

- The dataset is clean, balanced, and ready for modeling

🎯 **Target Variable Analysis:**- price_range

Values:- Price Range Meaning & Count

- 0 Low Budget:- 500
- 1 Medium Budget:- 500
- 2 High Budget:- 500
- 3 Premium:- 500

Feature Importance Insights (Based on statistical & expected model behavior)

Rank of Features with Reasons

1 ram:-Very High

- More RAM = higher price category

2 battery_power:- Moderate

- Premium phones tend to have bigger batteries

3 px_width/px_height:- Moderate

- High-resolution screens mostly in expensive models

4 int_memory:- Moderate

- More storage slightly raises price

5 pc/fc(camera):- Low–Moderate

- Higher megapixels mostly found in price range 2–3

6 mobile_wt:- Very Weak

- Slightly heavier phones in lower price because of older design

7 three_g, four_g, wifi, dual_sim, touch_screen, blue:- Minimal Impact

- These are common features across all price segments

Feature Behavior and Trends

1 RAM — Most predictable feature

Price Range with Avg RAM (approx)

- 0:- 800 MB
- 1:- 1600 MB
- 2:- 2500 MB
- 3:- 3500 MB

2 Screen Resolution (px_width × px_height) Premium phones have clearly higher pixel dimensions.

Price Range & Avg Screen Resolution

- 0:- Lowest
- 3:- Highest

3 Battery Power Growth trend, but not as clean as RAM.

4 Camera Features

- Primary/Front camera megapixel counts increase slightly with price, but overlap exists.
- Not reliable for hard classification — secondary feature.

5 Mobile Weight & Depth

- No meaningful pattern. Expensive phones are not necessarily heavier.

Statistical Insights Highlights

Numeric feature summary (Key)

Features

- **Battery Power**
  - Mean:-1238 mAh
  - Min:-501
  - Max:-1998
  - Variance worth nothiing:-Moderate spread
- **RAM**
  - Mean:-2000 MB
  - Min:-256
  - Max:-3998
  - Variance worth noting:-High spread & strong signal
- **Internal Memory**
  - Mean:-32GB
  - Min:-2
  - Max:-64
  - Variance worth noting:-Wide range
- **Talk Time**
  - Mean:-11 hrs
  - Min:-2
  - Max:-20
  - Variance worth noting:-Good variation

Correlations (Expected ML Behavior)

Feature & Correlation with Price

- **ram:**- Very Strong
- **px_width/px_height:**- Medium
- **battery_powe:**- Medium
- **front/primary camera:**- Low
- **Connectivity (3G/4G/WiFi/etc):**- Almost zero
- **mobile_wt/m_dep:**- Near zero

Imports & Configurations

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler,LabelEncoder,OneHotEncoder,OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer
import pickle
from sklearn.pipeline import Pipeline
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split , GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import r2_score, mean_absolute_error , root_mean_squared_error , mean_squared_error
```

[2]:
```
df=pd.read_csv('datasets_11167_15520_train.csv')
df
```

[2]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	t
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	19	0	
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	7	1	
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	9	1	
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	11	1	
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2	15	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1995	794	1	0.5	1	0	1	2	0.8	106	6	...	1222	1890	668	13	4	19	1	
1996	1965	1	2.6	1	0	0	39	0.2	187	4	...	915	1965	2032	11	10	16	1	
1997	1911	0	0.9	1	1	1	36	0.7	108	8	...	868	1632	3057	9	1	5	1	
1998	1512	0	0.9	0	4	1	46	0.1	145	5	...	336	670	869	18	10	19	1	
1999	510	1	2.0	1	5	1	45	0.9	168	6	...	483	754	3919	19	4	2	1	

2000 rows × 21 columns

Basic Checks:-

[3]:
```
df.head(5)
```

[3]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	19	0	
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	7	1	
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	9	1	
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	11	1	
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2	15	1	

5 rows × 21 columns

[4]:
```
df.tail()
```

[4]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	to
1995	794	1	0.5	1	0	1	2	0.8	106	6	...	1222	1890	668	13	4	19	1	
1996	1965	1	2.6	1	0	0	39	0.2	187	4	...	915	1965	2032	11	10	16	1	
1997	1911	0	0.9	1	1	1	36	0.7	108	8	...	868	1632	3057	9	1	5	1	
1998	1512	0	0.9	0	4	1	46	0.1	145	5	...	336	670	869	18	10	19	1	
1999	510	1	2.0	1	5	1	45	0.9	168	6	...	483	754	3919	19	4	2	1	

5 rows × 21 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   battery_power  2000 non-null   int64
 1   blue           2000 non-null   int64
 2   clock_speed    2000 non-null   float64
 3   dual_sim       2000 non-null   int64
 4   fc             2000 non-null   int64
 5   four_g         2000 non-null   int64
 6   int_memory     2000 non-null   int64
 7   m_dep          2000 non-null   float64
 8   mobile_wt      2000 non-null   int64
```

```
 9   n_cores        2000 non-null   int64
 10  pc             2000 non-null   int64
 11  px_height      2000 non-null   int64
 12  px_width       2000 non-null   int64
 13  ram            2000 non-null   int64
 14  sc_h           2000 non-null   int64
 15  sc_w           2000 non-null   int64
 16  talk_time      2000 non-null   int64
 17  three_g        2000 non-null   int64
 18  touch_screen   2000 non-null   int64
 19  wifi           2000 non-null   int64
 20  price_range    2000 non-null   int64
dtypes: float64(2), int64(19)
memory usage: 328.3 KB
```

[6]:
```
df.isnull()
```

[6]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1995	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False
1996	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False
1997	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False
1998	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False
1999	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False

2000 rows × 21 columns

[7]:
```
df.isnull().sum()
```

[7]:
```
battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               0
four_g           0
int_memory       0
m_dep            0
mobile_wt        0
n_cores          0
pc               0
px_height        0
px_width         0
ram              0
sc_h             0
sc_w             0
talk_time        0
three_g          0
touch_screen     0
wifi             0
price_range      0
dtype: int64
```

[8]:
```
df.describe(include=[np.number]).T
```

[8]:

	count	mean	std	min	25%	50%	75%	max
battery_power	2000.0	1238.51850	439.418206	501.0	851.75	1226.0	1615.25	1998.0
blue	2000.0	0.49500	0.500100	0.0	0.00	0.0	1.00	1.0
clock_speed	2000.0	1.52225	0.816004	0.5	0.70	1.5	2.20	3.0
dual_sim	2000.0	0.50950	0.500035	0.0	0.00	1.0	1.00	1.0
fc	2000.0	4.30950	4.341444	0.0	1.00	3.0	7.00	19.0
four_g	2000.0	0.52150	0.499662	0.0	0.00	1.0	1.00	1.0
int_memory	2000.0	32.04650	18.145715	2.0	16.00	32.0	48.00	64.0
m_dep	2000.0	0.50175	0.288416	0.1	0.20	0.5	0.80	1.0
mobile_wt	2000.0	140.24900	35.399655	80.0	109.00	141.0	170.00	200.0
n_cores	2000.0	4.52050	2.287837	1.0	3.00	4.0	7.00	8.0
pc	2000.0	9.91650	6.064315	0.0	5.00	10.0	15.00	20.0
px_height	2000.0	645.10800	443.780811	0.0	282.75	564.0	947.25	1960.0
px_width	2000.0	1251.51550	432.199447	500.0	874.75	1247.0	1633.00	1998.0
ram	2000.0	2124.21300	1084.732044	256.0	1207.50	2146.5	3064.50	3998.0
sc_h	2000.0	12.30650	4.213245	5.0	9.00	12.0	16.00	19.0
sc_w	2000.0	5.76700	4.356398	0.0	2.00	5.0	9.00	18.0
talk_time	2000.0	11.01100	5.463955	2.0	6.00	11.0	16.00	20.0
three_g	2000.0	0.76150	0.426273	0.0	1.00	1.0	1.00	1.0
touch_screen	2000.0	0.50300	0.500116	0.0	0.00	1.0	1.00	1.0
wifi	2000.0	0.50700	0.500076	0.0	0.00	1.0	1.00	1.0
price_range	2000.0	1.50000	1.118314	0.0	0.75	1.5	2.25	3.0

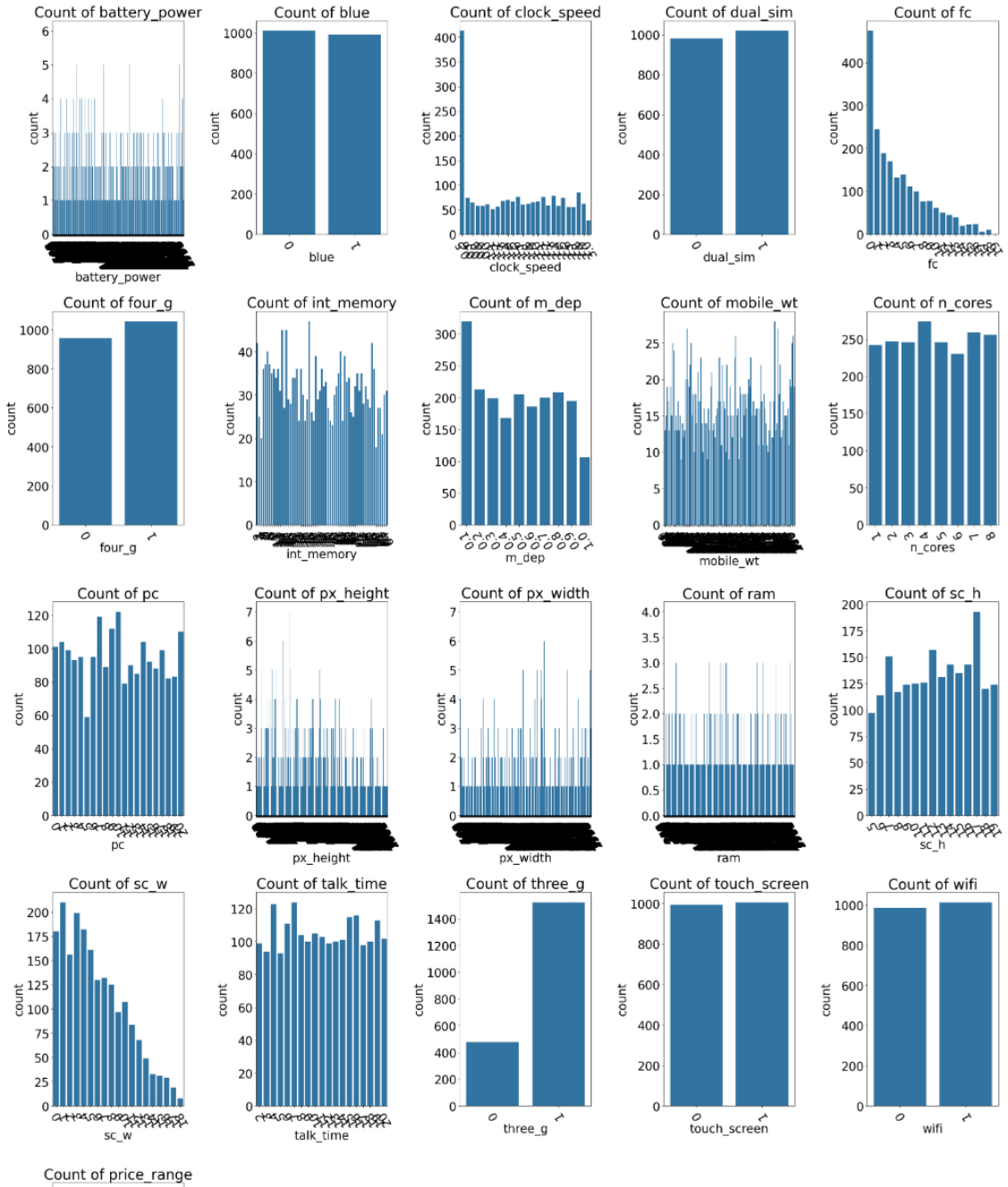
[9]:
```
unique_counts = df.nunique().sort_values(ascending=True)
unique_counts
```

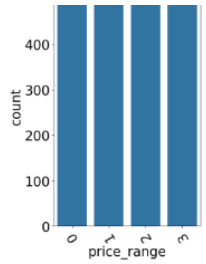
[9]:
```
blue             2
dual_sim         2
four_g           2
```

```
three_g            2
touch_screen       2
wifi               2
price_range        4
n_cores            8
m_dep             10
sc_h              15
sc_w              19
talk_time         19
fc                20
pc                21
clock_speed       26
int_memory        63
mobile_wt        121
battery_power    1094
px_width         1109
px_height        1137
ram              1562
dtype: int64
```

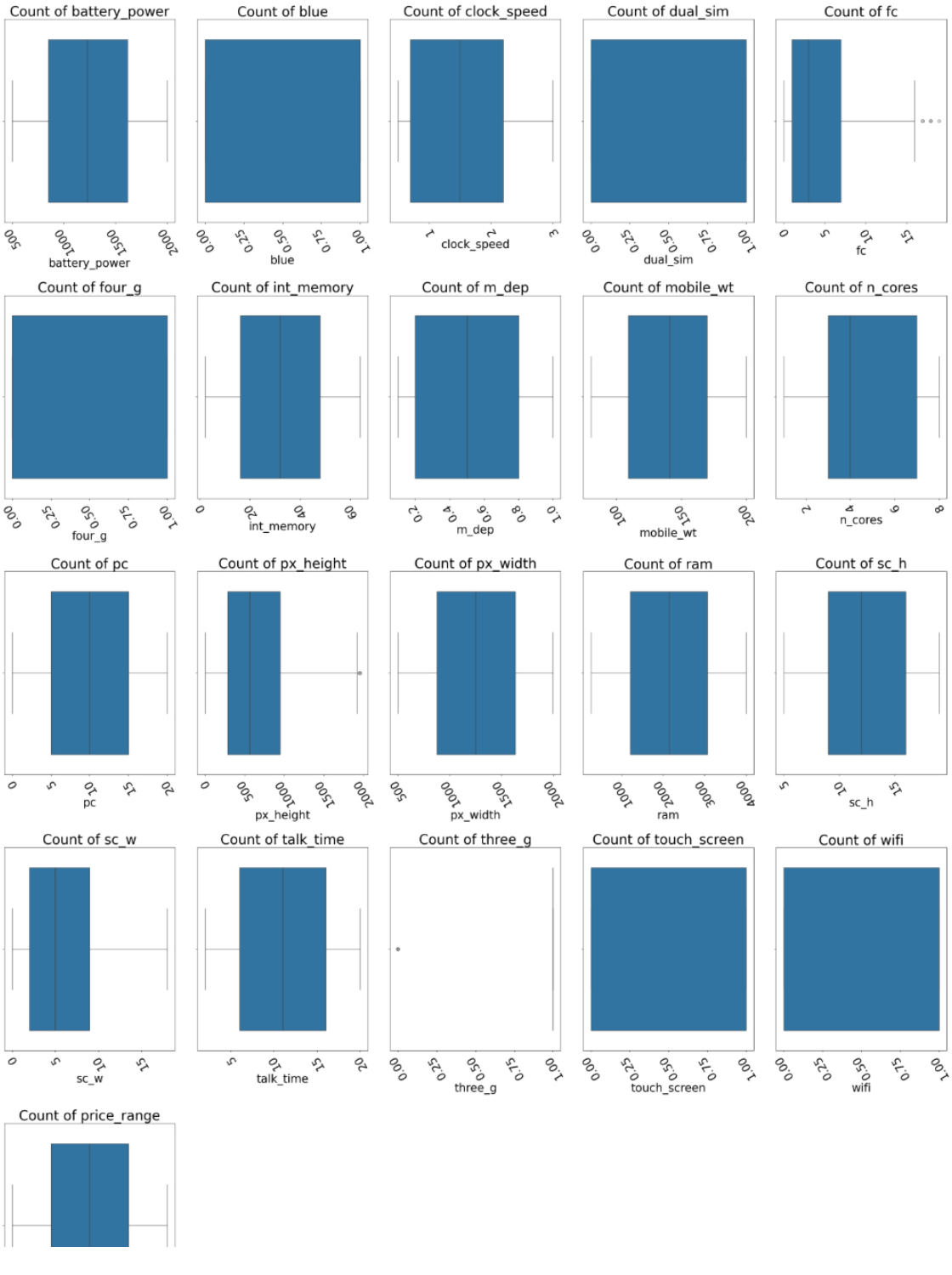
Exploratory data analysis (EDA):-

[10]:
```
plt.figure(figsize=(30,50))
plt.rcParams.update({'font.size': 25})
plot = 1

for i in df:
    if plot <= 21:
        plt.subplot(6, 5, plot)
        sns.countplot(x=df[i])
        plt.title(f'Count of {i}')
        plt.xticks(rotation=120)
        plot += 1

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

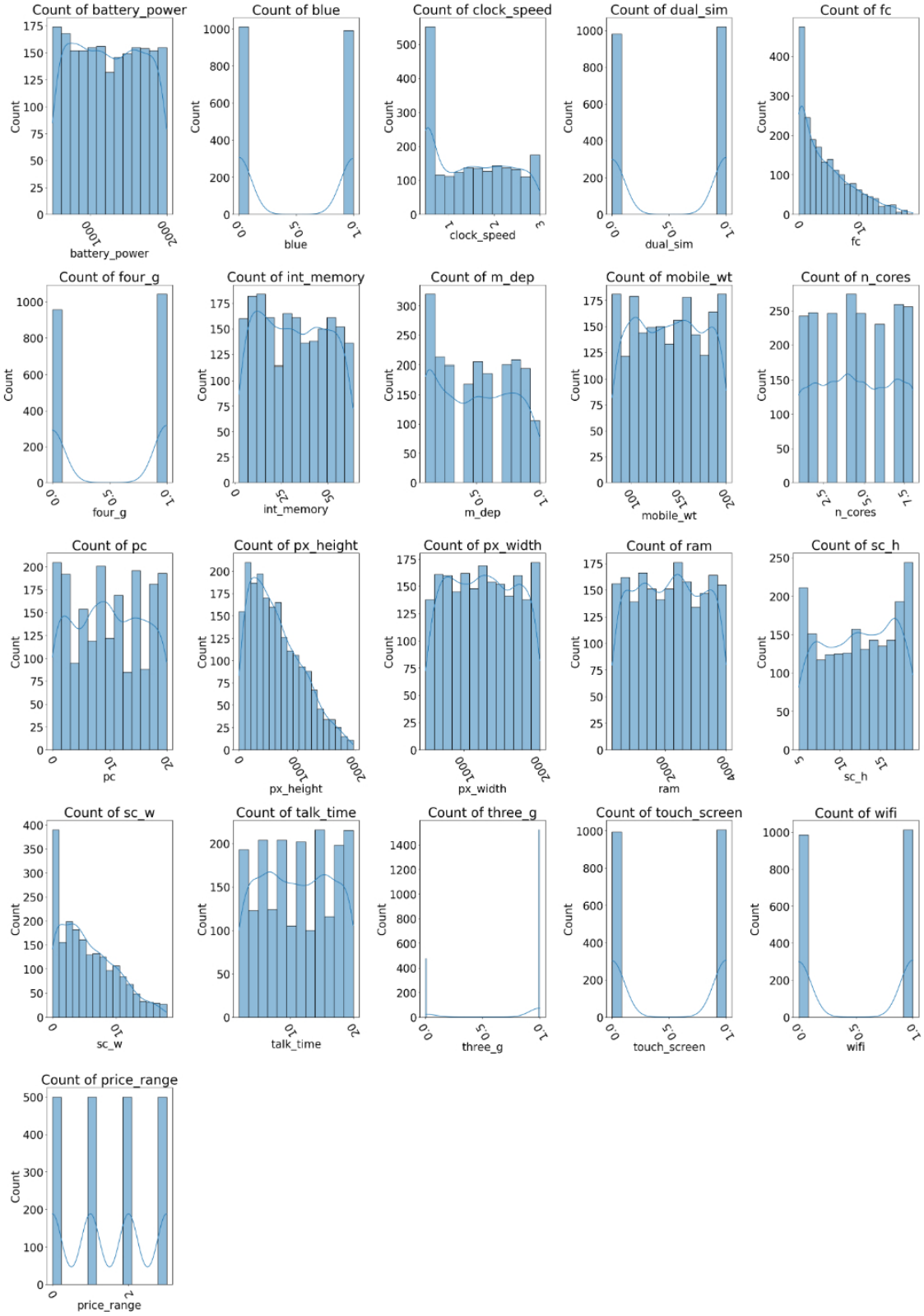




[11]:
```
plt.figure(figsize=(30,50))
plt.rcParams.update({'font.size': 25})
plot = 1

for i in df:
    if plot <= 21:
        plt.subplot(6, 5, plot)
        sns.boxplot(x=df[i])
        plt.title(f'Count of {i}')
        plt.xticks(rotation=120)
        plot += 1

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



[12]:
```
plt.figure(figsize=(30,50))
plt.rcParams.update({'font.size': 25})
plot = 1

for i in df:
    if plot <= 21:
        plt.subplot(6, 5, plot)
        sns.histplot(x=df[i],  kde=True)
        plt.title(f'Count of {i}')
        plt.xticks(rotation=120)
        plot += 1

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



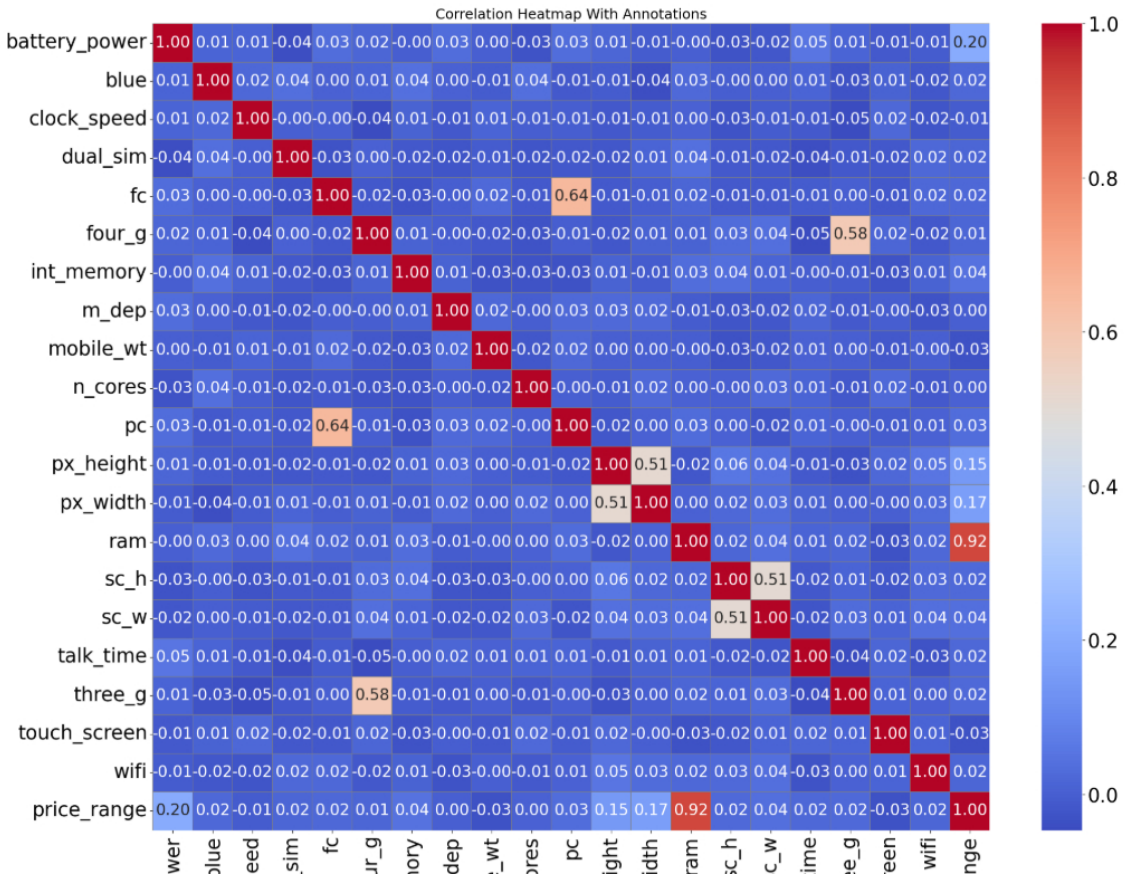
[13]:
```
df.corr()
```

[13]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram
battery_power	1.000000	0.011252	0.011482	-0.041847	0.033334	0.015665	-0.004004	0.034085	0.001844	-0.029727	...	0.014901	-0.008402	-0.000653
blue	0.011252	1.000000	0.021419	0.035198	0.003593	0.013443	0.041177	0.004049	-0.008605	0.036161	...	-0.006872	-0.041533	0.026351
clock_speed	0.011482	0.021419	1.000000	-0.001315	-0.000434	-0.043073	0.006545	-0.014364	0.012350	-0.005724	...	-0.014523	-0.009476	0.003443
dual_sim	-0.041847	0.035198	-0.001315	1.000000	-0.029123	0.003187	-0.015679	-0.022142	-0.008979	-0.024658	...	-0.020875	0.014291	0.041072
fc	0.033334	0.003593	-0.000434	-0.029123	1.000000	-0.016560	-0.029133	-0.001791	0.023618	-0.013356	...	-0.009990	-0.005176	0.015099
four_g	0.015665	0.013443	-0.043073	0.003187	-0.016560	1.000000	0.008690	-0.001823	-0.016537	-0.029706	...	-0.019236	0.007448	0.007313
int_memory	-0.004004	0.041177	0.006545	-0.015679	-0.029133	0.008690	1.000000	0.006886	-0.034214	-0.028310	...	0.010441	-0.008335	0.032813
m_dep	0.034085	0.004049	-0.014364	-0.022142	-0.001791	-0.001823	0.006886	1.000000	0.021756	-0.003504	...	0.025263	0.023566	-0.009434
mobile_wt	0.001844	-0.008605	0.012350	-0.008979	0.023618	-0.016537	-0.034214	0.021756	1.000000	-0.018989	...	0.000939	0.000090	-0.002581
n_cores	-0.029727	0.036161	-0.005724	-0.024658	-0.013356	-0.029706	-0.028310	-0.003504	-0.018989	1.000000	...	-0.006872	0.024480	0.004868
pc	0.031441	-0.009952	-0.005245	-0.017143	0.644595	-0.005598	-0.033273	0.026282	0.018844	-0.001193	...	-0.018465	0.004196	0.028984
px_height	0.014901	-0.006872	-0.014523	-0.020875	-0.009990	-0.019236	0.010441	0.025263	0.000939	-0.006872	...	1.000000	0.510664	-0.020352
px_width	-0.008402	-0.041533	-0.009476	0.014291	-0.005176	0.007448	-0.008335	0.023566	0.000090	0.024480	...	0.510664	1.000000	0.004105
ram	-0.000653	0.026351	0.003443	0.041072	0.015099	0.007313	0.032813	-0.009434	-0.002581	0.004868	...	-0.020352	0.004105	1.000000
sc_h	-0.029959	-0.002952	-0.029078	-0.011949	-0.011014	0.027166	0.037771	-0.025348	-0.033855	-0.000315	...	0.059615	0.021599	0.015996
sc_w	-0.021421	0.000613	-0.007378	-0.016666	-0.012373	0.037005	0.011731	-0.018388	-0.020761	0.025826	...	0.043038	0.034699	0.035576
talk_time	0.052510	0.013934	-0.011432	-0.039404	-0.006829	-0.046628	-0.002790	0.017003	0.006209	0.013148	...	-0.010645	0.006720	0.010820
three_g	0.011522	-0.030236	-0.046433	-0.014008	0.001793	0.584246	-0.009366	-0.012065	0.001551	-0.014733	...	-0.031174	0.000350	0.015795
touch_screen	-0.010516	0.010061	0.019756	-0.017117	-0.014828	0.016758	-0.026999	-0.002638	-0.014368	0.023774	...	0.021891	-0.001628	-0.030455
wifi	-0.008343	-0.021863	-0.024471	0.022740	0.020085	-0.017620	0.006993	-0.028353	-0.000409	-0.009964	...	0.051824	0.030319	0.022669
price_range	0.200723	0.020573	-0.006606	0.017444	0.021998	0.014772	0.044435	0.000853	-0.030302	0.004399	...	0.148858	0.165818	0.917046

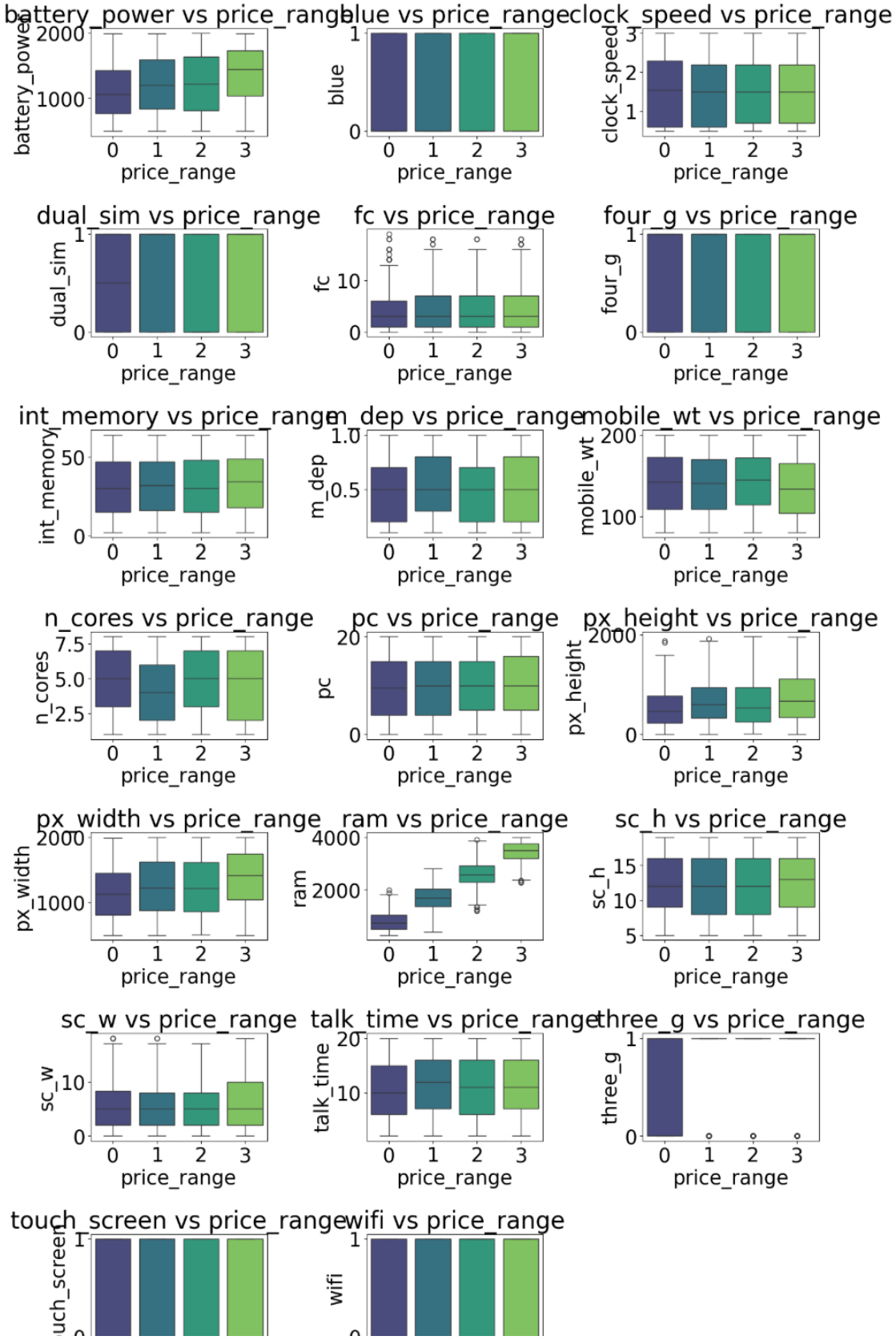
21 rows × 21 columns

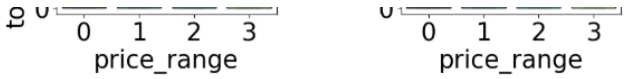
[14]:
```
plt.figure(figsize=(22, 18))
corr = df.corr(numeric_only=True)

sns.heatmap(
    corr,
    annot=True,
    fmt=".2f",
    cmap="coolwarm",
    annot_kws={"size": 20},
    linewidths=0.7,
    linecolor="gray"
)

plt.title("Correlation Heatmap With Annotations", fontsize=18)
plt.tight_layout()
plt.show()
```



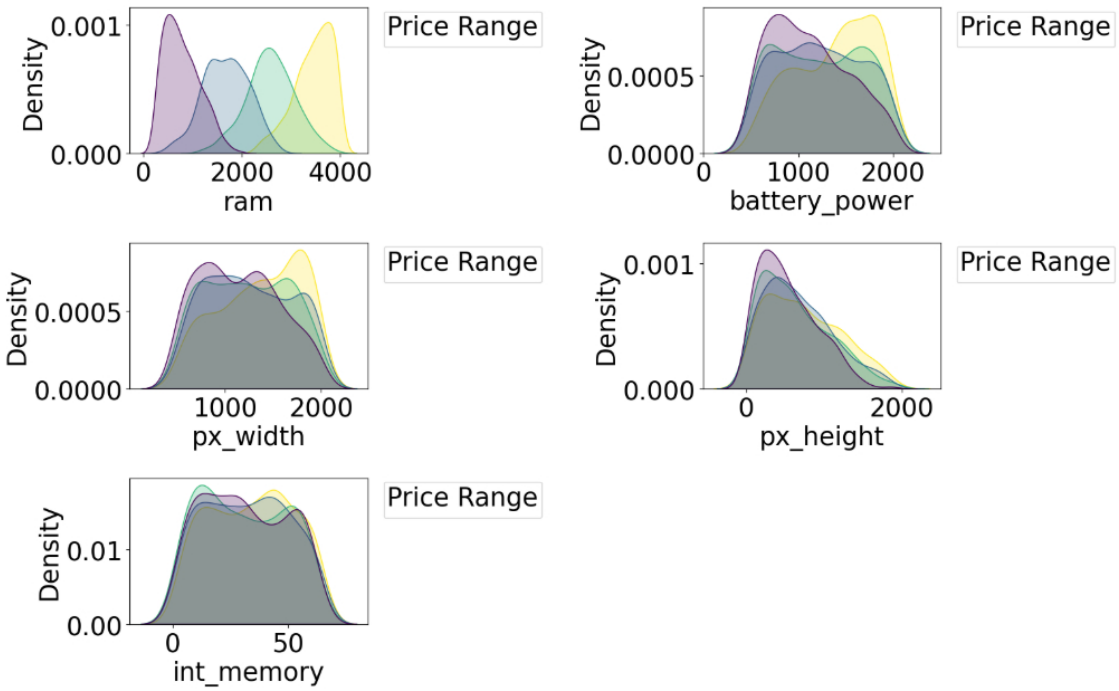
battery_po t clock_sp dual_ fo int_mem m_ mobile n_c px_hei px_wi s talk_t thre touch_scr price_ra

[15]:
```
num_features = df.drop(columns=["price_range"]).columns

plt.figure(figsize=(15,35))
for i, col in enumerate(num_features, 1):
    plt.subplot(10, 3, i)
    sns.boxplot(x=df["price_range"], y=df[col], palette="viridis")
    plt.title(f"{col} vs price_range")

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

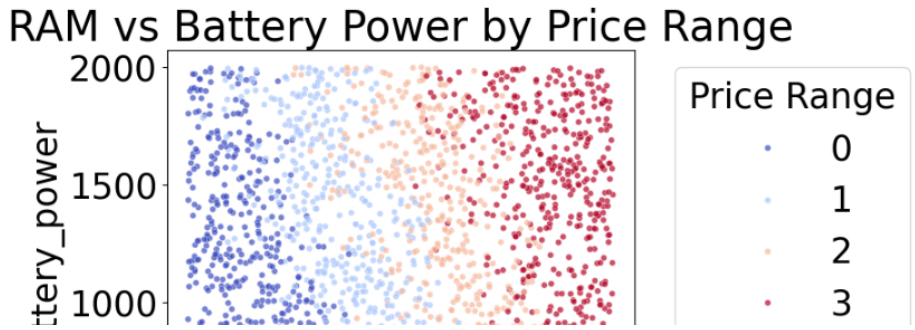


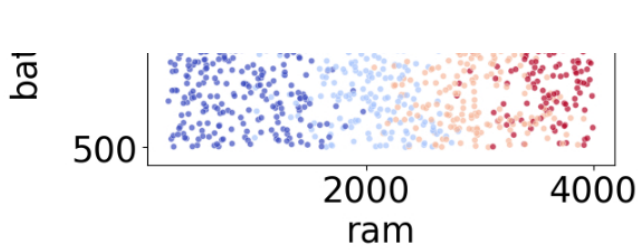


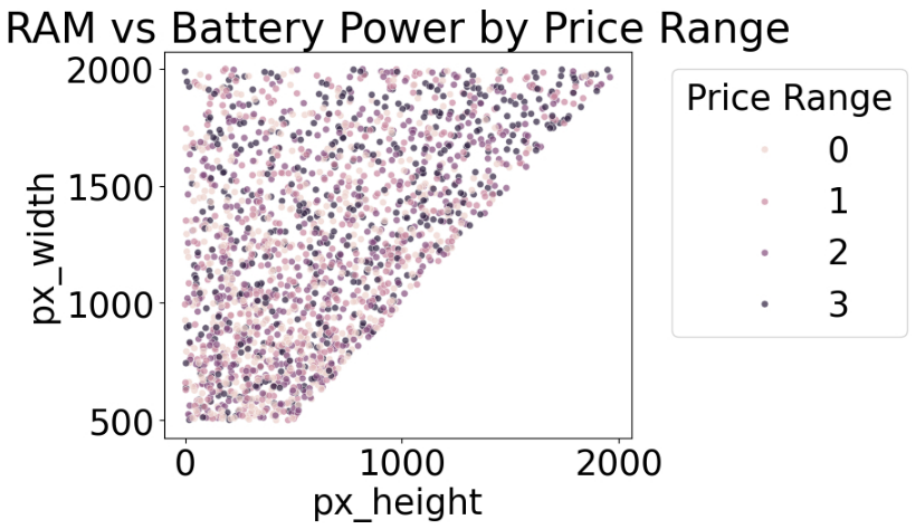
[16]:
```
important_features = ["ram", "battery_power", "px_width", "px_height", "int_memory"]

plt.figure(figsize=(16,10))
for i, col in enumerate(important_features, 1):
    plt.subplot(3, 2, i)
    sns.kdeplot(data=df, x=col, hue=df["price_range"], fill=True, common_norm=False, palette="viridis")

    # Legend right side
    plt.legend(
        title="Price Range",
        bbox_to_anchor=(1.05, 1),
        loc="upper left",
        fontsize=8
    )



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



[17]:
```
plt.figure(figsize=(6,5))
sns.scatterplot(
    x=df["ram"],
    y=df["battery_power"],
    hue=df["price_range"],
    palette="coolwarm",
    s=20,
    alpha=0.7
)

plt.title("RAM vs Battery Power by Price Range")

# Legend ko right side me move
plt.legend(title="Price Range", bbox_to_anchor=(1.05, 1), loc="upper left")

plt.show()


plt.figure(figsize=(6,5))
sns.scatterplot(x=df["px_height"], y=df["px_width"], hue=df["price_range"],  s=25,alpha=0.7 )
plt.title("RAM vs Battery Power by Price Range")

# Legend ko right side me move
plt.legend(title="Price Range", bbox_to_anchor=(1.05, 1), loc="upper left")

plt.show()
```







[18]:
```
df.shape
```

[18]:
```
(2000, 21)
```

[19]:
```
df['price_range'].value_counts()
```

[19]:
```
price_range
1    500
2    500
3    500
0    500
Name: count, dtype: int64
```

[20]:
```
df['four_g'].value_counts()
```

[20]:
```
four_g
1    1043
0     957
Name: count, dtype: int64
```

[21]:
```
df['three_g'].value_counts()
```

[21]:
```
three_g
1    1523
0     477
Name: count, dtype: int64
```

Logistic Regression

[22]:
```
X=df.iloc[:,:-1]
```

[23]:
```
X
```

[23]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	t
0	842	0	2.2	0	1	0	7	0.6	188	2	2	20	756	2549	9	7	19	0	
1	1021	1	0.5	1	0	1	53	0.7	136	3	6	905	1988	2631	17	3	7	1	
2	563	1	0.5	1	2	1	41	0.9	145	5	6	1263	1716	2603	11	2	9	1	
3	615	1	2.5	0	0	0	10	0.8	131	6	9	1216	1786	2769	16	8	11	1	
4	1821	1	1.2	0	13	1	44	0.6	141	2	14	1208	1212	1411	8	2	15	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1995	794	1	0.5	1	0	1	2	0.8	106	6	14	1222	1890	668	13	4	19	1	
1996	1965	1	2.6	1	0	0	39	0.2	187	4	3	915	1965	2032	11	10	16	1	
1997	1911	0	0.9	1	1	1	36	0.7	108	8	3	868	1632	3057	9	1	5	1	
1998	1512	0	0.9	0	4	1	46	0.1	145	5	5	336	670	869	18	10	19	1	
1999	510	1	2.0	1	5	1	45	0.9	168	6	16	483	754	3919	19	4	2	1	

2000 rows × 20 columns

[24]:
```
y=df.iloc[:,-1]
```

[25]:
```
y
```

[25]:
```
0       1
1       2
2       2
3       2
4       1
       ..
1995    0
```

```
1995    0
1996    2
1997    3
1998    0
1999    3
Name: price_range, Length: 2000, dtype: int64
```

[26]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=42,test_size=0.2)
```

[27]:
```
X_train
```

[27]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	t
968	1923	0	0.5	1	7	0	46	0.5	191	1	10	767	1759	1489	10	9	3	1	
240	633	1	2.2	0	0	1	49	0.1	139	8	1	529	1009	3560	11	1	16	1	
819	1236	0	0.9	1	2	1	57	0.1	188	1	14	517	809	1406	14	12	20	1	
692	781	0	1.1	0	2	0	38	0.4	198	5	7	304	1674	3508	13	8	5	0	
420	1456	1	0.5	1	7	0	7	0.4	105	5	12	823	1104	1587	6	5	20	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1130	1975	1	1.9	1	2	0	31	0.9	151	1	17	775	1607	3022	13	5	19	0	
1294	589	1	0.5	0	1	1	59	0.7	146	8	4	759	1858	362	16	10	6	1	
860	1829	1	0.5	0	0	1	15	0.4	160	5	7	729	1267	2080	16	11	12	1	
1459	1927	0	0.9	1	3	0	11	0.4	190	8	12	491	1506	2916	16	11	18	0	
1126	635	1	0.6	1	1	1	50	0.3	97	5	13	193	989	2107	13	12	12	1	

1600 rows × 20 columns

[28]:
```
y_train
```

[28]:
```
968     1
240     2
819     0
692     3
420     1
       ..
1130    3
1294    0
860     2
1459    3
1126    1
Name: price_range, Length: 1600, dtype: int64
```

[29]:
```
X_test
```

[29]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	t
1860	1646	0	2.5	0	3	1	25	0.6	200	2	5	211	1608	686	8	6	11	1	
353	1182	0	0.5	0	7	1	8	0.5	138	8	16	275	986	2563	19	17	19	1	
1333	1972	0	2.9	0	9	0	14	0.4	196	7	18	293	952	1316	8	1	8	1	
905	989	1	2.0	0	4	0	17	0.2	166	3	19	256	1394	3892	18	7	19	1	
1289	615	1	0.5	1	7	0	58	0.5	130	5	8	1021	1958	1906	14	5	5	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
965	1379	0	0.5	1	1	0	19	0.3	134	8	17	387	671	3912	11	2	19	0	
1284	991	0	2.0	0	2	1	12	0.3	158	5	6	1209	1678	2014	11	9	10	1	
1739	1044	0	1.8	0	4	1	12	0.7	104	6	5	1230	1263	1794	18	7	19	1	
261	728	0	2.7	1	0	0	25	0.2	88	4	1	526	1529	2039	5	1	12	1	
535	1185	0	1.9	0	0	0	31	0.4	152	8	7	837	1642	2447	16	2	3	1	

400 rows × 20 columns

[30]:
```
y_test
```

[30]:
```
1860    0
353     2
1333    1
905     3
1289    1
       ..
965     3
1284    2
1739    1
261     1
535     2
Name: price_range, Length: 400, dtype: int64
```

[31]:
```
## Step 3 Model creation

clf=LogisticRegression( C=1.0,class_weight='balanced',)
clf.fit(X_train,y_train)
```

[31]:

▾ LogisticRegression

▶ Parameters

[32]:
```
param_grid = {
    "C": [0.01, 0.1, 1, 10],
    "solver": ["lbfgs", "saga"],
    "class_weight":['balanced']
}
```

[33]:
```
grid = GridSearchCV(
    clf,
    param_grid
```

```
    cv=5,
    n_jobs=-1,
    scoring="accuracy"
)
```

[34]:
```
grid.fit(X_train,y_train)
```

[34]:
```
GridSearchCV
best_estimator_:
LogisticRegression
LogisticRegression
```

[35]:
```
grid.best_params_
```

[35]:
```
{'C': 10, 'class_weight': 'balanced', 'solver': 'lbfgs'}
```

[36]:
```
grid.best_estimator_
```

[36]:
```
LogisticRegression
▶ Parameters
```

[37]:
```
X_test
```

[37]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	t
1860	1646	0	2.5	0	3	1	25	0.6	200	2	5	211	1608	686	8	6	11	1	
353	1182	0	0.5	0	7	1	8	0.5	138	8	16	275	986	2563	19	17	19	1	
1333	1972	0	2.9	0	9	0	14	0.4	196	7	18	293	952	1316	8	1	8	1	
905	989	1	2.0	0	4	0	17	0.2	166	3	19	256	1394	3892	18	7	19	1	
1289	615	1	0.5	1	7	0	58	0.5	130	5	8	1021	1958	1906	14	5	5	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
965	1379	0	0.5	1	1	0	19	0.3	134	8	17	387	671	3912	11	2	19	0	
1284	991	0	2.0	0	2	1	12	0.3	158	5	6	1209	1678	2014	11	9	10	1	
1739	1044	0	1.8	0	4	1	12	0.7	104	6	5	1230	1263	1794	18	7	19	1	
261	728	0	2.7	1	0	0	25	0.2	88	4	1	526	1529	2039	5	1	12	1	
535	1185	0	1.9	0	0	0	31	0.4	152	8	7	837	1642	2447	16	2	3	1	

400 rows × 20 columns

[38]:
```
y_pred=grid.predict(X_test)
y_pred
```

[38]:
```
array([0, 2, 0, 3, 1, 2, 3, 0, 3, 3, 0, 1, 2, 3, 3, 2, 2, 2, 1, 0, 0, 1,
       0, 2, 1, 1, 3, 3, 3, 0, 1, 1, 3, 0, 2, 3, 2, 1, 3, 0, 1, 2, 3, 0,
       3, 3, 3, 1, 3, 1, 3, 2, 0, 0, 2, 0, 2, 2, 0, 0, 1, 3, 3, 2, 2, 0,
       3, 3, 1, 1, 2, 2, 0, 1, 2, 0, 0, 3, 2, 2, 3, 2, 1, 0, 1, 3, 3, 3,
       3, 0, 3, 3, 3, 0, 3, 2, 2, 3, 2, 1, 0, 1, 0, 0, 0, 3, 3, 0, 0, 1,
       0, 0, 3, 3, 2, 1, 3, 3, 0, 2, 1, 3, 2, 1, 3, 3, 0, 3, 0, 2, 3, 0,
       2, 2, 0, 2, 1, 1, 0, 2, 3, 1, 3, 3, 0, 0, 1, 2, 1, 1, 3, 1, 2, 0,
       2, 3, 0, 1, 0, 1, 3, 3, 2, 2, 1, 0, 0, 2, 1, 3, 3, 1, 0, 0, 3, 1,
       1, 2, 0, 1, 0, 0, 0, 1, 3, 2, 0, 2, 0, 0, 0, 0, 1, 3, 3, 1, 0, 1,
       1, 1, 1, 2, 1, 2, 3, 3, 1, 3, 0, 1, 1, 1, 1, 1, 3, 2, 1, 3, 1, 1,
       3, 2, 3, 0, 0, 3, 0, 2, 0, 0, 1, 0, 2, 3, 2, 1, 0, 2, 3, 1, 3, 3,
       2, 3, 0, 3, 2, 2, 3, 3, 3, 1, 1, 3, 2, 1, 3, 3, 3, 3, 3, 0, 2, 2,
       2, 2, 3, 0, 3, 3, 2, 2, 2, 0, 1, 3, 0, 2, 3, 1, 3, 1, 1, 3, 1, 3,
       0, 0, 3, 0, 1, 2, 3, 2, 2, 0, 0, 0, 0, 3, 3, 0, 1, 1, 2, 0, 3, 3,
       3, 3, 1, 3, 2, 0, 3, 2, 3, 2, 0, 0, 1, 3, 2, 3, 1, 1, 3, 0, 3, 3,
       2, 0, 2, 2, 1, 2, 3, 1, 0, 3, 1, 2, 2, 0, 1, 1, 2, 2, 3, 3, 1, 1,
       1, 2, 2, 0, 3, 0, 0, 2, 0, 0, 1, 2, 2, 3, 1, 1, 2, 3, 3, 3, 2, 3,
       1, 2, 0, 2, 1, 3, 3, 0, 1, 3, 2, 3, 2, 3, 1, 0, 3, 2, 0, 0, 3, 3,
       1, 2, 3, 2])
```

[39]:
```
y_test
```

[39]:
```
1860    0
353     2
1333    1
905     3
1289    1
       ..
965     3
1284    2
1739    1
261     1
535     2
Name: price_range, Length: 400, dtype: int64
```

[40]:
```
from sklearn.metrics import confusion_matrix,accuracy_score,recall_score,precision_score,classification_report,f1_score,average_precision_score
```

[41]:
```
y_test
```

[41]:
```
1860    0
353     2
1333    1
905     3
1289    1
       ..
965     3
1284    2
1739    1
261     1
535     2
Name: price_range, Length: 400, dtype: int64
```

[42]:
```
print(confusion_matrix(y_test,y_pred))
```

```
[[78 26  1  0]
 [17 46 19  9]
 [ 0 17 46 29]
```

```
[ 0  1 26 85]]
```

[43]:
```
accuracy_score(y_test,y_pred)
```

[43]:
```
0.6375
```

[44]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.74      0.78       105
           1       0.51      0.51      0.51        91
           2       0.50      0.50      0.50        92
           3       0.69      0.76      0.72       112

    accuracy                           0.64       400
   macro avg       0.63      0.63      0.63       400
weighted avg       0.64      0.64      0.64       400
```

Random Forest

[45]:
```
rf_best = RandomForestClassifier(
    n_estimators=500,
    max_depth=25,
    min_samples_split=3,
    min_samples_leaf=1,
    bootstrap=True,
    random_state=42,
    n_jobs=-1
)
```

[46]:
```
rf_best.fit(X_train, y_train)
```

[46]:
```
▾ RandomForestClassifier
▶ Parameters
```

[47]:
```
y_pred=rf_best.predict(X_test)
```

[48]:
```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.96      0.94      0.95       107
           1       0.87      0.87      0.87        91
           2       0.82      0.81      0.81        93
           3       0.89      0.92      0.90       109

    accuracy                           0.89       400
   macro avg       0.88      0.88      0.88       400
weighted avg       0.89      0.89      0.89       400
```

[49]:
```
y_pred=rf_best.predict(X_test)
```

XGBoost

[50]:
```
pip install xgboost
```

```
Requirement already satisfied: xgboost in d:\kernal\myenv\lib\site-packages (3.1.2)
Requirement already satisfied: numpy in d:\kernal\myenv\lib\site-packages (from xgboost) (2.3.3)
Requirement already satisfied: scipy in d:\kernal\myenv\lib\site-packages (from xgboost) (1.16.2)
Note: you may need to restart the kernel to use updated packages.

[notice] A new release of pip is available: 25.2 -> 25.3
[notice] To update, run: D:\kernal\myenv\Scripts\python.exe -m pip install --upgrade pip
```

[51]:
```
from xgboost import XGBClassifier
```

[52]:
```
xgb = XGBClassifier(

    n_estimators=600,
    learning_rate=0.05,
    max_depth=8,
    subsample=0.8,
    colsample_bytree=0.8,
    objective="multi:softmax",
    num_class=4,
    random_state=42
)
```

[53]:
```
xgb.fit(X_train,y_train)
```

[53]:
```
▾ XGBClassifier
▶ Parameters
```

[54]:
```
y_pred=xgb.predict(X_test)
```

[55]:
```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.95      0.96      0.96       104
           1       0.93      0.89      0.91        96
           2       0.87      0.85      0.86        94
           3       0.89      0.94      0.92       106

    accuracy                           0.91       400
   macro avg       0.91      0.91      0.91       400
weighted avg       0.91      0.91      0.91       400
```

SVC

SVC

[56]:
```
svc=SVC(random_state=42)
```

[57]:
```
svc.fit(X_train,y_train)
```

[57]:
```
SVC
▶ Parameters
```

[58]:
```
y_pred=svc.predict(X_test)
```

[59]:
```
accuracy_score(y_pred,y_test)
```

[59]:
```
0.965
```

[60]:
```
confusion_matrix(y_pred,y_test)
```

[60]:
```
array([[103,   1,   0,   0],
       [  2,  90,   3,   0],
       [  0,   0,  87,   6],
       [  0,   0,   2, 106]])
```

[61]:
```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       104
           1       0.99      0.95      0.97        95
           2       0.95      0.94      0.94        93
           3       0.95      0.98      0.96       108

    accuracy                           0.96       400
   macro avg       0.97      0.96      0.96       400
weighted avg       0.97      0.96      0.96       400
```

- Conclusion

Your model is performing very well with 96% accuracy.

Key insights:

Class 0 has the strongest performance (F1 = 0.99).

Class 1 has high precision (0.99) but slightly lower recall (0.95), meaning some Class-1 samples are missed.

Class 2 is the weakest among all classes (F1 = 0.94) but still good.

Class 3 also performs strongly with high recall (0.98), meaning the model rarely misses Class-3 cases.

Overall:

✓ The model is balanced, performs well across all classes, and shows no major bias.

[62]:
```
import joblib
```