11]:
```
pip install scikit-learn
```

```
Collecting scikit-learn
  Downloading scikit_learn-1.6.1-cp313-cp313-win_amd64.whl.metadata (15 kB)
Requirement already satisfied: numpy>=1.19.5 in c:\users\laban\appdata\local\programs\python\python313\lib\site-packages (from scik
Collecting scipy>=1.6.0 (from scikit-learn)
  Downloading scipy-1.15.2-cp313-cp313-win_amd64.whl.metadata (60 kB)
Collecting joblib>=1.2.0 (from scikit-learn)
  Downloading joblib-1.4.2-py3-none-any.whl.metadata (5.4 kB)
Collecting threadpoolctl>=3.1.0 (from scikit-learn)
  Downloading threadpoolctl-3.6.0-py3-none-any.whl.metadata (13 kB)
Downloading scikit_learn-1.6.1-cp313-cp313-win_amd64.whl (11.1 MB)
   ---------------------------------------- 0.0/11.1 MB ? eta -:--:--
    --------------------------------------- 0.3/11.1 MB ? eta -:--:--
   - -------------------------------------- 0.5/11.1 MB 1.7 MB/s eta 0:00:07
   --- ------------------------------------ 1.0/11.1 MB 2.0 MB/s eta 0:00:06
   ----- ---------------------------------- 1.6/11.1 MB 2.2 MB/s eta 0:00:05
   -------- ------------------------------- 2.4/11.1 MB 2.7 MB/s eta 0:00:04
   ---------- ----------------------------- 2.9/11.1 MB 2.5 MB/s eta 0:00:04
   ----------- ---------------------------- 3.1/11.1 MB 2.6 MB/s eta 0:00:04
   ------------- -------------------------- 3.7/11.1 MB 2.3 MB/s eta 0:00:04
```

[7]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, roc_auc_score
```

[12]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, roc_auc_score
```

[13]:
```
df = pd.read_csv('Flipkart Mobile - 2.csv')
```

[12]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, roc_auc_score
```

[13]:
```
df = pd.read_csv('Flipkart Mobile - 2.csv')
```

[14]:
```
print(df.head())
print(df.info())
print(df.describe())
```

	brand	model	base_color	processor	screen_size	ROM	RAM	\
0	Apple	iPhone SE	Black	Water	Very Small	64	2	
1	Apple	iPhone 12 Mini	Red	Ceramic	Small	64	4	
2	Apple	iPhone SE	Red	Water	Very Small	64	2	
3	Apple	iPhone XR	Others	iOS	Medium	64	3	
4	Apple	iPhone 12	Red	Ceramic	Medium	128	4	

	display_size	num_rear_camera	num_front_camera	battery_capacity	ratings	\
0	4.7	1	1	1800	4.5	
1	5.4	2	1	2815	4.5	
2	4.7	1	1	1800	4.5	
3	6.1	1	1	2942	4.6	
4	6.1	2	1	2815	4.6	

	num_of_ratings	sales_price	discount_percent	sales
0	38645	32999	0.17	127.52
1	244	57149	0.04	1.39
2	38645	32999	0.17	127.52
3	5366	42999	0.10	23.07
4	745	69149	0.02	5.15

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 430 entries, 0 to 429
Data columns (total 16 columns):
```

```
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   brand             430 non-null    object
 1   model             430 non-null    object
 2   base_color        430 non-null    object
 3   processor         430 non-null    object
 4   screen_size       430 non-null    object
 5   ROM               430 non-null    int64
 6   RAM               430 non-null    int64
 7   display_size      430 non-null    float64
 8   num_rear_camera   430 non-null    int64
 9   num_front_camera  430 non-null    int64
 10  battery_capacity  430 non-null    int64
 11  ratings           430 non-null    float64
 12  num_of_ratings    430 non-null    int64
 13  sales_price       430 non-null    int64
 14  discount_percent  430 non-null    float64
 15  sales             430 non-null    float64
dtypes: float64(4), int64(7), object(5)
memory usage: 53.9+ KB
None
                ROM         RAM  display_size  num_rear_camera  \
count    430.000000  430.000000    430.000000       430.000000
mean     105.748837    5.320930      6.369767         2.904651
std       63.164064    2.182635      0.369549         0.952350
min        8.000000    1.000000      4.700000         1.000000
25%       64.000000    4.000000      6.300000         2.000000
50%      128.000000    4.000000      6.500000         3.000000
75%      128.000000    6.000000      6.500000         4.000000
max      512.000000   12.000000      7.600000         4.000000
```

	num_front_camera	battery_capacity	ratings	num_of_ratings	\
count	430.000000	430.000000	430.000000	430.000000	
mean	1.044186	4529.397674	4.339302	23567.944186	
std	0.227280	986.907252	0.151494	56096.277784	
min	1.000000	1800.000000	3.000000	4.000000	
25%	1.000000	4000.000000	4.300000	745.000000	
50%	1.000000	4500.000000	4.300000	5197.500000	
75%	1.000000	5000.000000	4.400000	21089.250000	
max	3.000000	7000.000000	4.600000	642373.000000	

	sales_price	discount_percent	sales
count	430.000000	430.000000	430.000000
mean	25433.234884	0.108000	29.752326
std	22471.926588	0.073432	58.399588
min	5742.000000	0.010000	0.000000
25%	11999.000000	0.060000	1.640000
50%	16989.500000	0.090000	9.655000
75%	28999.000000	0.160000	29.717500
max	157999.000000	0.440000	550.190000

[15]:
```
print(df.isnull().sum())
```

```
brand              0
model              0
base_color         0
processor          0
screen_size        0
ROM                0
RAM                0
display_size       0
num_rear_camera    0
num_front_camera   0
battery_capacity   0
ratings            0
num_of_ratings     0
sales_price        0
discount_percent   0
sales              0
dtype: int64
```

[16]:
```
df['screen_size'] = df['screen_size'].astype('category')
df['base_color'] = df['base_color'].astype('category')
df['processor'] = df['processor'].astype('category')
```

[17]:
```
# For this analysis, we'll simulate return rates based on product features
np.random.seed(42)
df['return_rate'] = np.random.normal(loc=0.1, scale=0.03, size=len(df))
df['return_rate'] = df['return_rate'].clip(0, 0.3)  # Keep between 0-30%
df['returns'] = (df['sales'] * df['return_rate']).round()
```

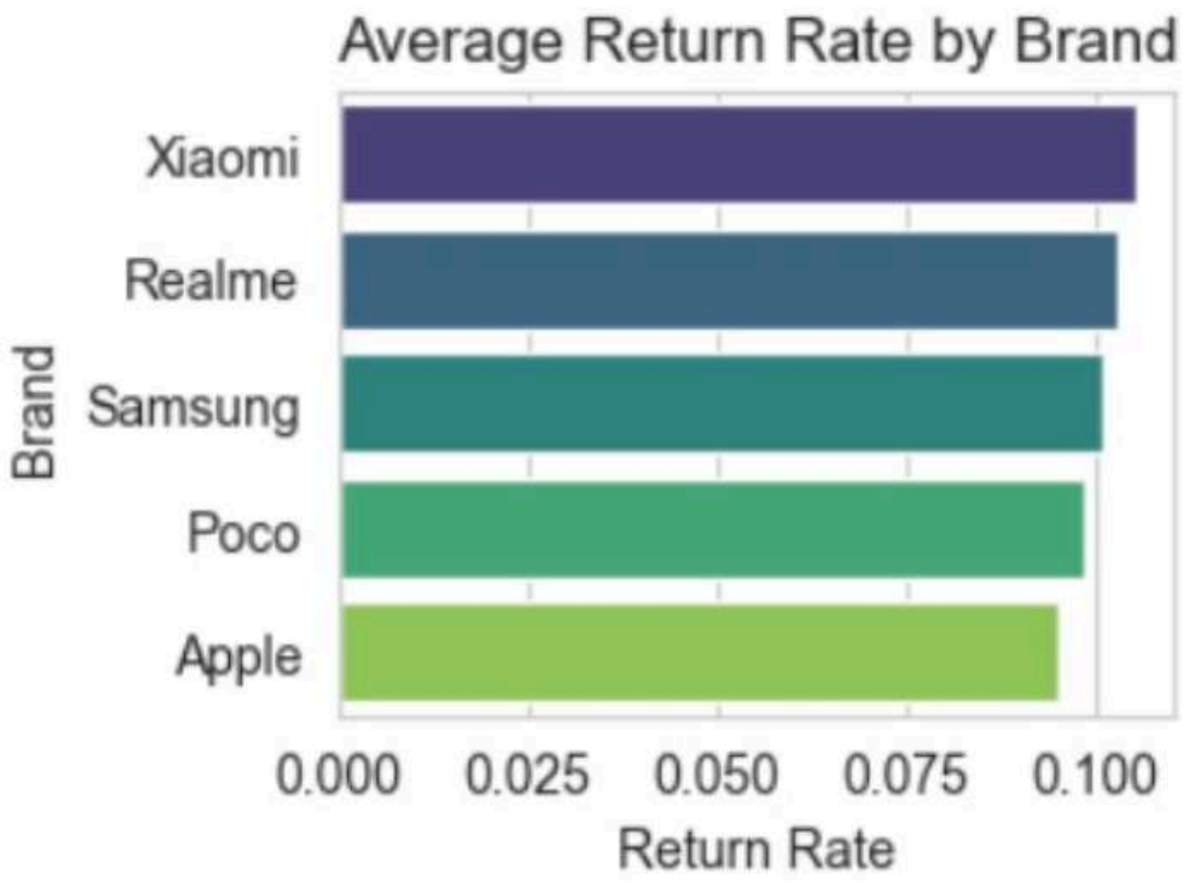
[18]:
```
df['high_return_risk'] = (df['return_rate'] > df['return_rate'].quantile(0.75)).astype(int)
```

[19]:
```
sns.set_style('whitegrid')
plt.figure(figsize=(12, 8))
```

[19]:
```
<Figure size 1200x800 with 0 Axes>
```
<Figure size 1200x800 with 0 Axes>

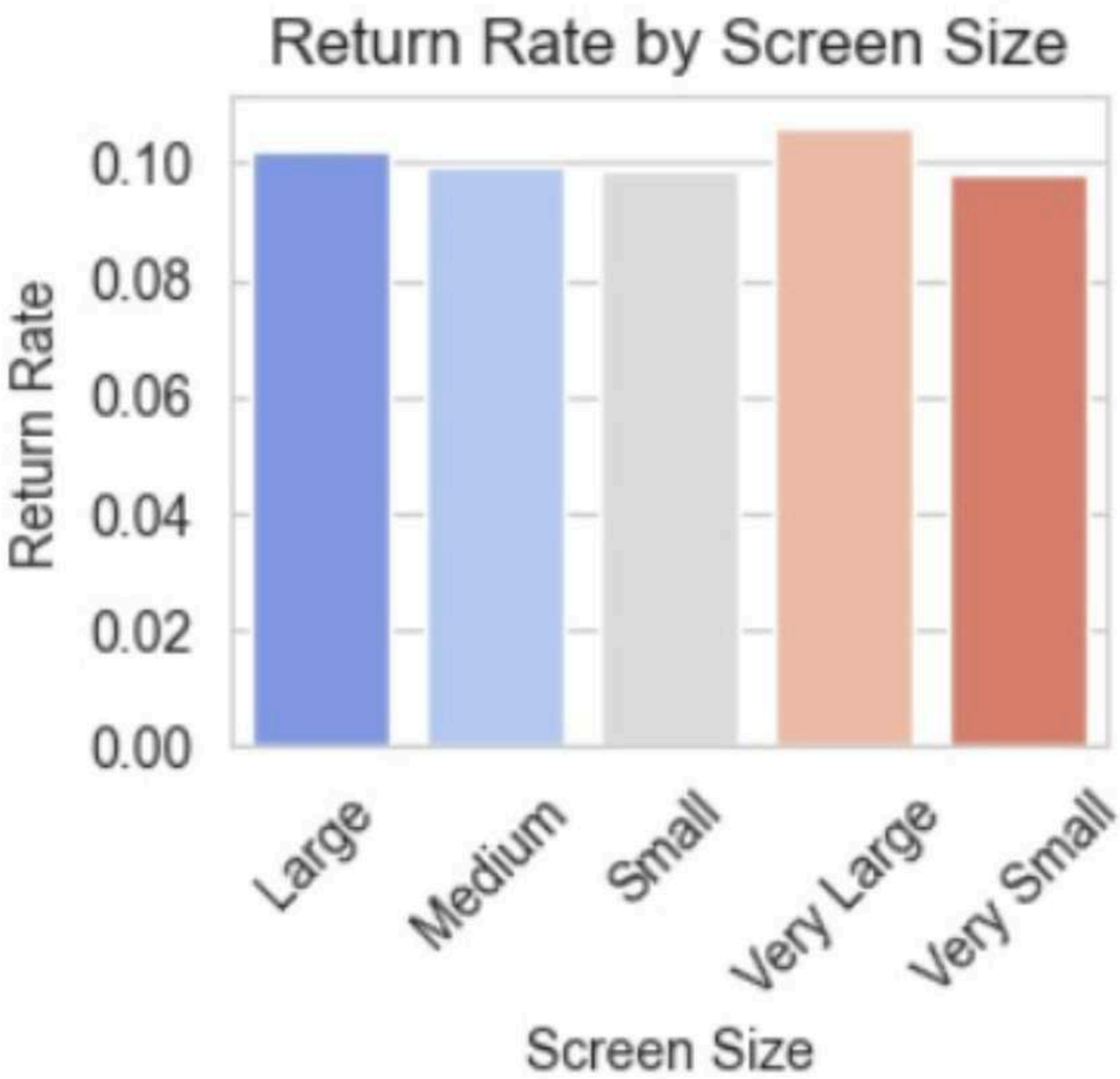
[20]:
```
plt.subplot(2, 2, 1)
brand_return = df.groupby('brand')['return_rate'].mean().sort_values(ascending=False)
sns.barplot(x=brand_return.values, y=brand_return.index, palette='viridis')
plt.title('Average Return Rate by Brand')
plt.xlabel('Return Rate')
plt.ylabel('Brand')
```

[20]:
```
Text(0, 0.5, 'Brand')
```



[21]:
```
plt.subplot(2, 2, 2)
size_return = df.groupby('screen_size')['return_rate'].mean().sort_values()
sns.barplot(x=size_return.index, y=size_return.values, palette='coolwarm')
plt.title('Return Rate by Screen Size')
plt.xlabel('Screen Size')
plt.ylabel('Return Rate')
plt.xticks(rotation=45)
```

[21]:
```
([0, 1, 2, 3, 4],
 [Text(0, 0, 'Large'),
  Text(1, 0, 'Medium'),
  Text(2, 0, 'Small'),
  Text(3, 0, 'Very Large'),
  Text(4, 0, 'Very Small')])
```

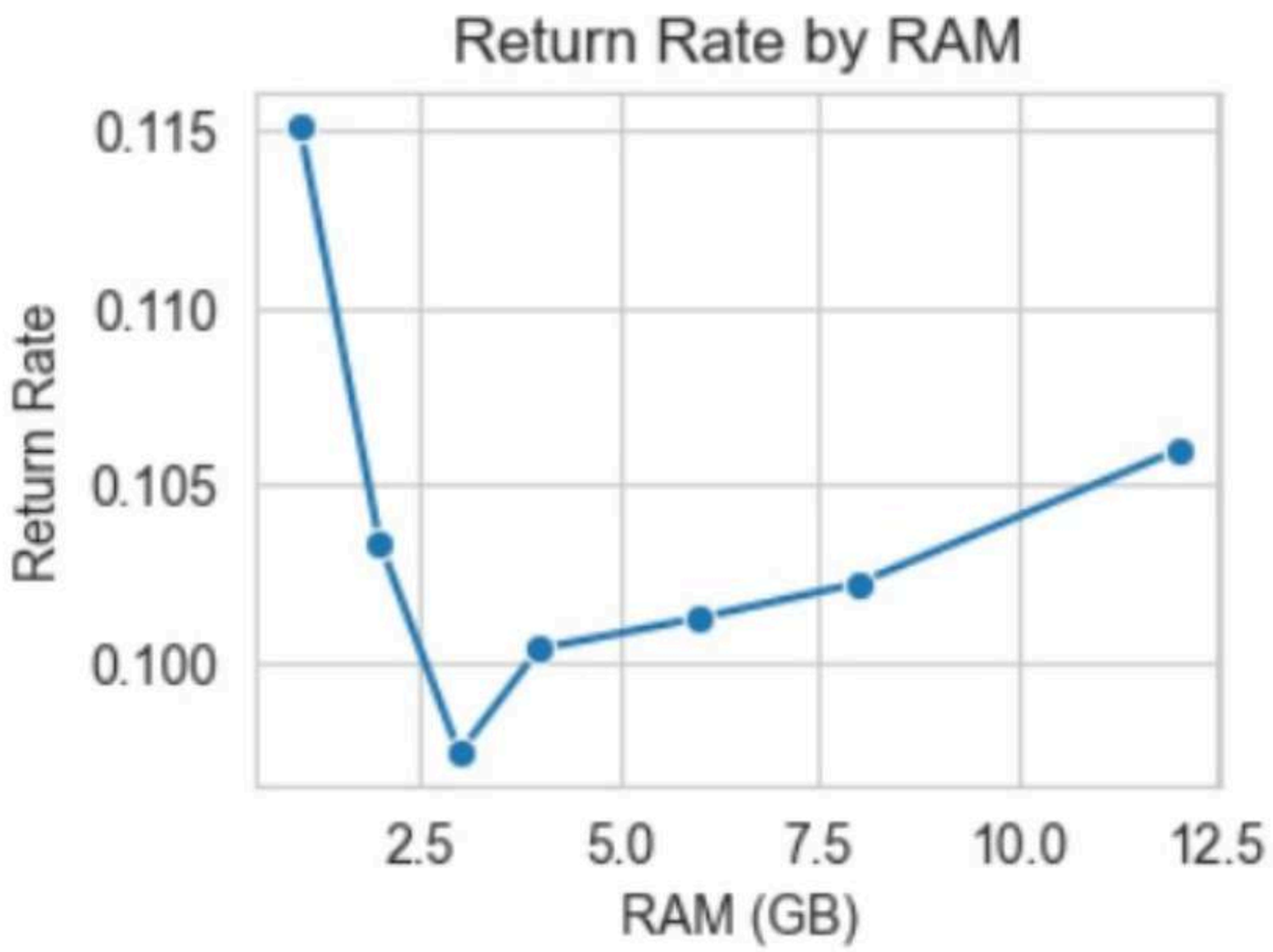


[22]:
```
plt.subplot(2, 2, 3)
sns.scatterplot(x='sales_price', y='return_rate', hue='brand', data=df, alpha=0.6)
plt.title('Return Rate vs Price')
plt.xlabel('Price')
plt.ylabel('Return Rate')
```

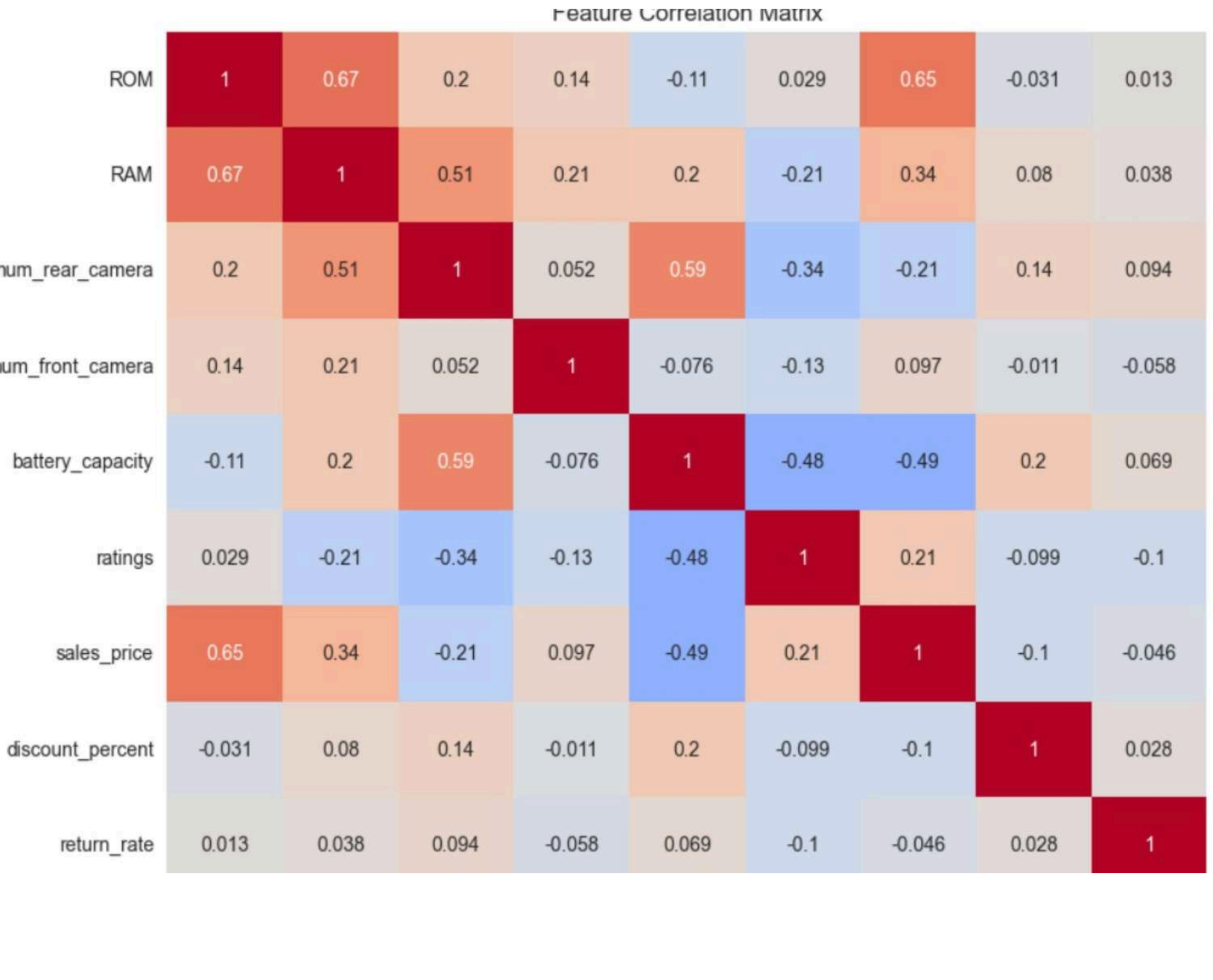
[22]:
```
Text(0, 0.5, 'Return Rate')
```

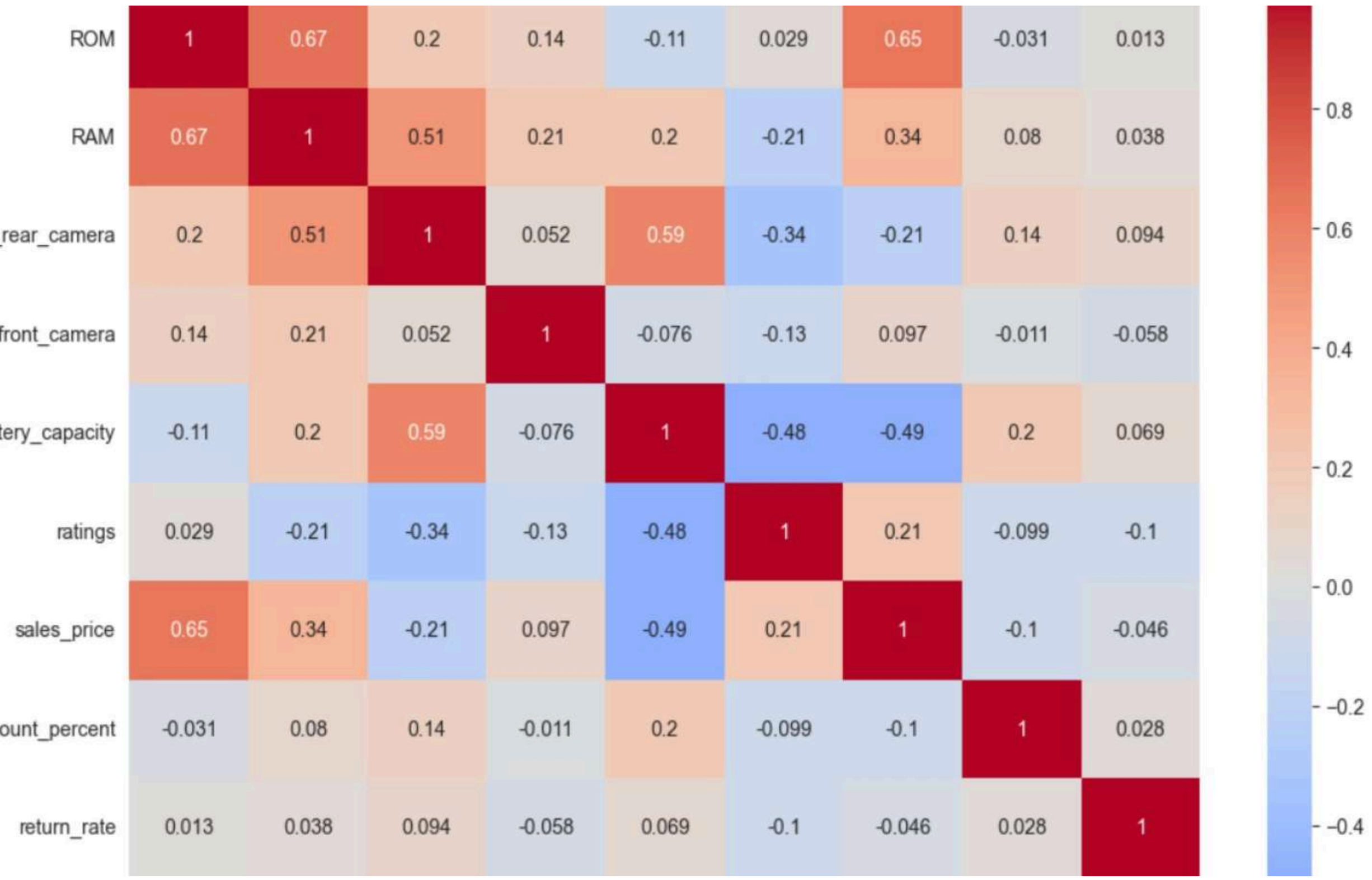
[23]:
```
plt.subplot(2, 2, 4)
ram_return = df.groupby('RAM')['return_rate'].mean()
sns.lineplot(x=ram_return.index, y=ram_return.values, marker='o')
plt.title('Return Rate by RAM')
plt.xlabel('RAM (GB)')
plt.ylabel('Return Rate')

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



[24]:
```
plt.figure(figsize=(12, 8))
numeric_cols = ['ROM', 'RAM', 'num_rear_camera', 'num_front_camera',
                'battery_capacity', 'ratings', 'sales_price', 'discount_percent', 'return_rate']
corr_matrix = df[numeric_cols].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Feature Correlation Matrix')
plt.show()
```


Feature Correlation Matrix
ROM 1 0.67 0.2 0.14 -0.11 0.029 0.65 -0.031 0.013
RAM 0.67 1 0.51 0.21 0.2 -0.21 0.34 0.08 0.038
num_rear_camera 0.2 0.51 1 0.052 0.59 -0.34 -0.21 0.14 0.094
um_front_camera 0.14 0.21 0.052 1 -0.076 -0.13 0.097 -0.011 -0.058
battery_capacity -0.11 0.2 0.59 -0.076 1 -0.48 -0.49 0.2 0.069
ratings 0.029 -0.21 -0.34 -0.13 -0.48 1 0.21 -0.099 -0.1
sales_price 0.65 0.34 -0.21 0.097 -0.49 0.21 1 -0.1 -0.046
discount_percent -0.031 0.08 0.14 -0.011 0.2 -0.099 -0.1 1 0.028
return_rate 0.013 0.038 0.094 -0.058 0.069 -0.1 -0.046 0.028 1


ROM 1 0.67 0.2 0.14 -0.11 0.029 0.65 -0.031 0.013
RAM 0.67 1 0.51 0.21 0.2 -0.21 0.34 0.08 0.038
_rear_camera 0.2 0.51 1 0.052 0.59 -0.34 -0.21 0.14 0.094
front_camera 0.14 0.21 0.052 1 -0.076 -0.13 0.097 -0.011 -0.058
tery_capacity -0.11 0.2 0.59 -0.076 1 -0.48 -0.49 0.2 0.069
ratings 0.029 -0.21 -0.34 -0.13 -0.48 1 0.21 -0.099 -0.1
sales_price 0.65 0.34 -0.21 0.097 -0.49 0.21 1 -0.1 -0.046
ount_percent -0.031 0.08 0.14 -0.011 0.2 -0.099 -0.1 1 0.028
return_rate 0.013 0.038 0.094 -0.058 0.069 -0.1 -0.046 0.028 1
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

[25]:
```
df_model = pd.get_dummies(df, columns=['brand', 'screen_size', 'base_color', 'processor'], drop_first=True)
```

[26]:
```
features = [col for col in df_model.columns if col not in ['model', 'high_return_risk', 'return_rate', 'returns', 'sales']]
X = df_model[features]
y = df_model['high_return_risk']
```

[27]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

[28]:
```
model = LogisticRegression(max_iter=1000, class_weight='balanced')
model.fit(X_train, y_train)
```

```
  n_iter_i = _check_optimize_result(
```

[28]:

LogisticRegression

```
LogisticRegression(class_weight='balanced', max_iter=1000)
```

[29]:

```
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]
```

[30]:

```
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("\nROC AUC Score:", roc_auc_score(y_test, y_pred_proba))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.57      0.67       100
           1       0.27      0.55      0.36        29

    accuracy                           0.57       129
   macro avg       0.54      0.56      0.52       129
weighted avg       0.69      0.57      0.60       129


ROC AUC Score: 0.5396551724137931
```

1]:
```python
importance = pd.DataFrame({'feature': features, 'coefficient': model.coef_[0]})
importance['abs_coef'] = importance['coefficient'].abs()
importance = importance.sort_values('abs_coef', ascending=False)
print("\nTop 10 Important Features:")
print(importance.head(10))
```

```
Top 10 Important Features:
             feature  coefficient  abs_coef
13      brand_Xiaomi     0.608986  0.608986
18   base_color_Blue    -0.604797  0.604797
4   num_front_camera    -0.549482  0.549482
25    base_color_Red    -0.472414  0.472414
21   base_color_Gray     0.434172  0.434172
33   processor_Water    -0.423080  0.423080
27  base_color_White     0.367750  0.367750
29  processor_Exynos     0.302288  0.302288
10        brand_Poco    -0.295984  0.295984
12     brand_Samsung    -0.275226  0.275226
```

2]:
```python
df['risk_score'] = model.predict_proba(df_model[features])[:, 1]
```

[33]:
```
high_risk_products = df[df['risk_score'] > 0.7].sort_values('risk_score', ascending=False)
print("\nHigh Risk Products:")
print(high_risk_products[['brand', 'model', 'risk_score', 'return_rate']].head())
```

```
High Risk Products:
      brand             model  risk_score  return_rate
380  Xiaomi  Redmi Note 9 Pro    0.849192     0.074808
371  Xiaomi     Redmi 8A Dual    0.824787     0.114940
394  Xiaomi             Mi A3    0.824512     0.135383
391  Xiaomi             Mi A3    0.814694     0.114758
370  Xiaomi            Mi 11X    0.771803     0.100735
```

[34]:
```
high_risk_products.to_csv('high_risk_mobile_products.csv', index=False)
```

[35]:
```
brand_summary = df.groupby('brand').agg({
    'sales': 'sum',
    'returns': 'sum',
    'return_rate': 'mean',
    'risk_score': 'mean'
}).reset_index()

feature_summary = df.melt(id_vars=['return_rate'],
                          value_vars=['RAM', 'ROM', 'num_rear_camera', 'battery_capacity'],
                          var_name='feature', value_name='value')
```

[36]:
```
df.to_csv('mobile_sales_analysis.csv', index=False)
brand_summary.to_csv('brand_summary.csv', index=False)
feature_summary.to_csv('feature_summary.csv', index=False)
```