

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

sns.set_style("darkgrid")
plt.rcParams['figure.figsize'] = (12, 6)

df = pd.read_csv('/content/drive/MyDrive/datat/amazon.csv') # Corrected file path
```

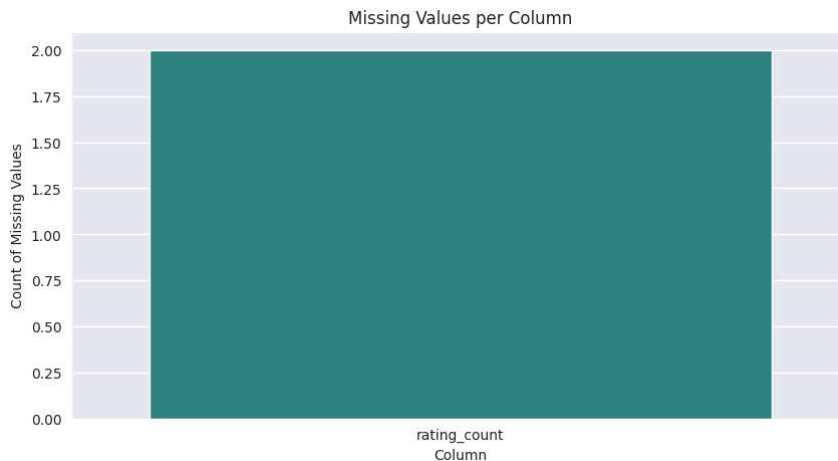
```
missing_values = df.isnull().sum()
missing_values = missing_values[missing_values > 0].sort_values(ascending=False)

plt.figure(figsize=(10, 5))
sns.barplot(x=missing_values.index, y=missing_values.values, palette="viridis")
plt.title('Missing Values per Column')
plt.xlabel('Column')
plt.ylabel('Count of Missing Values')
plt.show()
#
```

/tmp/ipython-input-2646896200.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x=missing_values.index, y=missing_values.values, palette="viridis")
```



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

sns.set_style("darkgrid")
plt.rcParams['figure.figsize'] = (12, 6)
```

```
import pandas as pd
import numpy as np
def clean_and_convert_price(price_series):
    return pd.to_numeric(price_series.astype(str).str.replace('$', '', regex=False).str.replace(',', '', regex=False), errors='coerce')
def clean_and_convert_percentage(percentage_series):
    return pd.to_numeric(percentage_series.astype(str).str.replace('%', '', regex=False), errors='coerce')
def clean_and_convert_rating(rating_series):
    rating_series = rating_series.astype(str).apply(lambda x: x.split(' ')[0] if ' out of ' in x else x)
    return pd.to_numeric(rating_series, errors='coerce')
def clean_and_convert_rating_count(count_series):
    return pd.to_numeric(count_series.astype(str).str.replace(',', '', regex=False), errors='coerce')
df['discounted_price'] = clean_and_convert_price(df['discounted_price'])
df['actual_price'] = clean_and_convert_price(df['actual_price'])
df['discount_percentage'] = clean_and_convert_percentage(df['discount_percentage'])
df['rating'] = clean_and_convert_rating(df['rating'])
df['rating_count'] = clean_and_convert_rating_count(df['rating_count'])
numerical_cols = ['discounted_price', 'actual_price', 'discount_percentage', 'rating', 'rating_count']
categorical_cols = ['category']
for col in numerical_cols:
    if col in df.columns and df[col].isnull().any():
        median_value = df[col].median()
        df.loc[:, col] = df[col].fillna(median_value)
for col in categorical_cols:
    if col in df.columns and df[col].isnull().any():
        mode_value = df[col].mode()
        # Use .loc to avoid FutureWarning with inplace=True and ensuring mode_value is not empty
        df.loc[:, col] = df[col].fillna(mode_value[0] if not mode_value.empty else 'Unknown')

print("Missing value imputation completed after cleaning.")
print(df.isnull().sum())
```

```
Missing value imputation completed after cleaning.
product_id      0
product_name    0
category        0
discounted_price 0
actual_price    0
discount_percentage 0
rating          0
rating_count    0
about_product   0
user_id         0
user_name       0
review_id       0
review_title    0
review_content  0
img_link        0
product_link    0
```

dtype: int64

```
plt.figure(figsize=(8, 6))
sns.boxplot(y=df['discounted_price'])
plt.title('Box Plot for Discounted Price (Before Treatment)')
plt.ylabel('discounted_price')
plt.show()
```



```
col_to_clean = 'discounted_price'
df['price_cleaned'] = df[col_to_clean]

Q1 = df['price_cleaned'].quantile(0.25)
Q3 = df['price_cleaned'].quantile(0.75)
IQR = Q3 - Q1

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

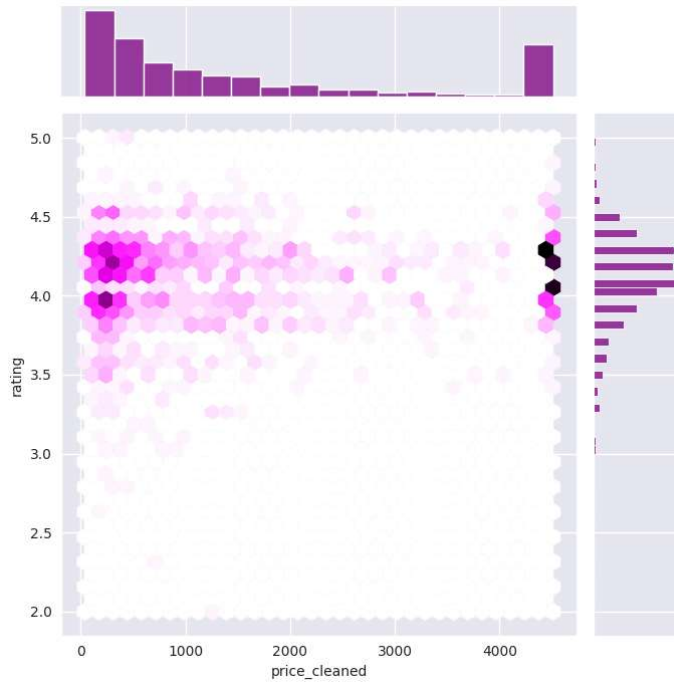
df['price_cleaned'] = np.where(df['price_cleaned'] > upper_bound, upper_bound, df['price_cleaned'])
df['price_cleaned'] = np.where(df['price_cleaned'] < lower_bound, lower_bound, df['price_cleaned'])

plt.figure(figsize=(8, 6))
sns.boxplot(y=df['price_cleaned'])
plt.title('Box Plot for Discounted Price (After Treatment)')
plt.ylabel('price_cleaned')
plt.show()
#
```



```
sns.jointplot(x='price_cleaned', y='rating', data=df, kind='hex', height=7, color='purple')
plt.suptitle('Joint Distribution: Cleaned Price vs. Rating', y=1.02)
plt.show()
#
```

Joint Distribution: Cleaned Price vs. Rating



```
# Select the top 5 categories for a clearer visualization
top_categories = df['category'].value_counts().nlargest(5).index
df_top_cat = df[df['category'].isin(top_categories)]

plt.figure(figsize=(14, 7))
sns.violinplot(x='category', y='price_cleaned', data=df_top_cat, palette='muted')
plt.title('Distribution of Discounted Price Across Top 5 Categories')
plt.xlabel('Product Category')
plt.ylabel('Cleaned Discounted Price')
plt.xticks(rotation=15)
plt.tight_layout()
plt.show()
#
```

/tmp/ipython-input-1282630306.py:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.violinplot(x='category', y='price_cleaned', data=df_top_cat, palette='muted')
```



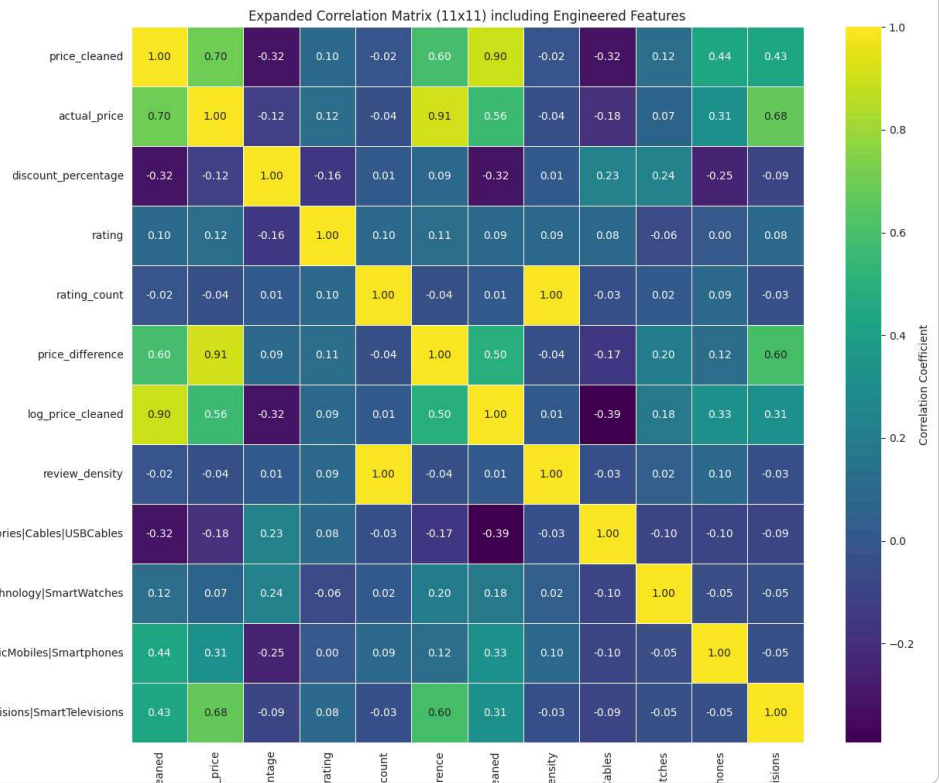
```
df['price_difference'] = df['actual_price'] - df['discounted_price']
df['log_price_cleaned'] = np.log1p(df['price_cleaned'])
df['rating_safe'] = df['rating'].apply(lambda x: x if x > 0 else 1)
df['review_density'] = df['rating_count'] / df['rating_safe']
```

```
top_categories = df['category'].value_counts().nlargest(4).index
df_encoded = pd.get_dummies(df, columns='category', prefix='is', dummy_na=False)
category_cols = [f'is_{cat}' for cat in top_categories]
new_encoded_cols = [col for col in category_cols if col in df_encoded.columns]
```

```
original_cols = ['price_cleaned', 'actual_price', 'discount_percentage', 'rating', 'rating_count']
engineered_cols = ['price_difference', 'log_price_cleaned', 'review_density']
final_correlation_cols = original_cols + engineered_cols + new_encoded_cols
correlation_matrix = df_encoded[final_correlation_cols].corr()
plt.figure(figsize=(14, 12))
sns.heatmap(correlation_matrix,
            annot=True,
            cmap='viridis',
            fmt=".2f",
            linewidths=.5,
            cbar_kws={'label': 'Correlation Coefficient'})

plt.title('Expanded Correlation Matrix (11x11) including Engineered Features')
```

plt.show()



```
# اختيار الميزات الرقمية والهدف
# بناء على الاسعار والميزات الأخرى (rating) سنحاول التنبؤ بالتقييم

# --- Re-including feature engineering and encoding steps from previous cells ---
# Ensure 'price_cleaned' is created and processed (from cell hWjt7xfGKCS9)
col_to_clean = 'discounted_price'
df['price_cleaned'] = df[col_to_clean]

Q1 = df['price_cleaned'].quantile(0.25)
Q3 = df['price_cleaned'].quantile(0.75)
IQR = Q3 - Q1

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

df['price_cleaned'] = np.where(df['price_cleaned'] > upper_bound, upper_bound, df['price_cleaned'])
df['price_cleaned'] = np.where(df['price_cleaned'] < lower_bound, lower_bound, df['price_cleaned'])

# From cell xI9AD7rGLITk:
df['price_difference'] = df['actual_price'] - df['discounted_price']
df['log_price_cleaned'] = np.log1p(df['price_cleaned'])
df['rating_safe'] = df['rating'].apply(lambda x: x if x > 0 else 1)
df['review_density'] = df['rating_count'] / df['rating_safe']

# From cell cM-UAWpMLMFP:
top_categories = df['category'].value_counts().nlargest(4).index
df_encoded = pd.get_dummies(df, columns=['category'], prefix='is', dummy_na=False)
category_cols = [f'is_{cat}' for cat in top_categories]
new_encoded_cols = [col for col in category_cols if col in df_encoded.columns]

# From cell UaZAS0WJLPSa (modified for X features):
# original_cols now excludes 'rating' as it is the target
original_cols_for_X = ['price_cleaned', 'actual_price', 'discount_percentage', 'rating_count']
engineered_cols = ['price_difference', 'log_price_cleaned', 'review_density']
final_features_for_X = original_cols_for_X + engineered_cols + new_encoded_cols

# Selecting features (X) and target (y)
X = df_encoded[final_features_for_X]
y = df_encoded['rating'] # 'rating' is the target variable

# تقسيم البيانات إلى تدريب واختبار
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# توحيد مقاييس البيانات (Scaling)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("تم تجهيز البيانات وتقسيمها بنجاح.")
```

تم تجهيز البيانات وتقسيمها بنجاح.

```
from sklearn.ensemble import RandomForestRegressor

# تعريف النموذج
rf = RandomForestRegressor(random_state=42)

# تحديد الإعدادات للتجربة
param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}

# تطبيق Grid Search
```

```
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=3, scoring='neg_mean_squared_error')
grid_search.fit(X_train_scaled, y_train)
```

```
best_model = grid_search.best_estimator_
print(f"أفضل إعدادات تم العثور عليها: {grid_search.best_params_}")
```

أفضل إعدادات تم العثور عليها: {'max_depth': 20, 'min_samples_split': 2, 'n_estimators': 50}

```
from sklearn.feature_selection import SelectKBest, f_regression
```

```
# الطريقة الأولى: SelectKBest
selector = SelectKBest(score_func=f_regression, k=5)
X_best_features = selector.fit_transform(X_train_scaled, y_train)
selected_names = X.columns[selector.get_support()]
print(f"الميزات المختارة (SelectKBest): {list(selected_names)}")
```

```
# الطريقة الثانية: Feature Importance من النموذج
importances = best_model.feature_importances_
feat_importances = pd.Series(importances, index=X.columns)
print(f"أهم الميزات حسب نموذج Random Forest:")
print(feat_importances.nlargest(5))
```

الميزات المختارة (SelectKBest): ['price_cleaned', 'actual_price', 'discount_percentage', 'rating_count', 'price_difference']

```
Random Forest: أهم الميزات حسب نموذج
rating_count      0.260535
review_density    0.208067
price_difference   0.143002
discount_percentage 0.128753
actual_price      0.097194
dtype: float64
```

Double-click (or enter) to edit

```
# استخدام أفضل 5 ميزات فقط
top_5_features = feat_importances.nlargest(5).index
X_train_selected = X_train[top_5_features]
X_test_selected = X_test[top_5_features]

# إعادة تدريب النموذج الأفضل
final_model = RandomForestRegressor(n_estimators=100, random_state=42)
final_model.fit(X_train_selected, y_train)

# التنبؤ
y_pred = final_model.predict(X_test_selected)
```

```
from sklearn.metrics import mean_squared_error, r2_score
```

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```
print(f"نتائج النموذج النهائي:")
print(f"Mean Squared Error: {mse:.4f}")
print(f"R-squared Score: {r2:.4f}")
```

```
# رسم بياني للقيم الحقيقية مقابل المتوقعة
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.5, color='teal')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.title('Actual vs Predicted Ratings')
plt.xlabel('Actual Rating')
plt.ylabel('Predicted Rating')
plt.show()
```

نتائج النموذج النهائي:
Mean Squared Error: 0.0634
R-squared Score: 0.2237

