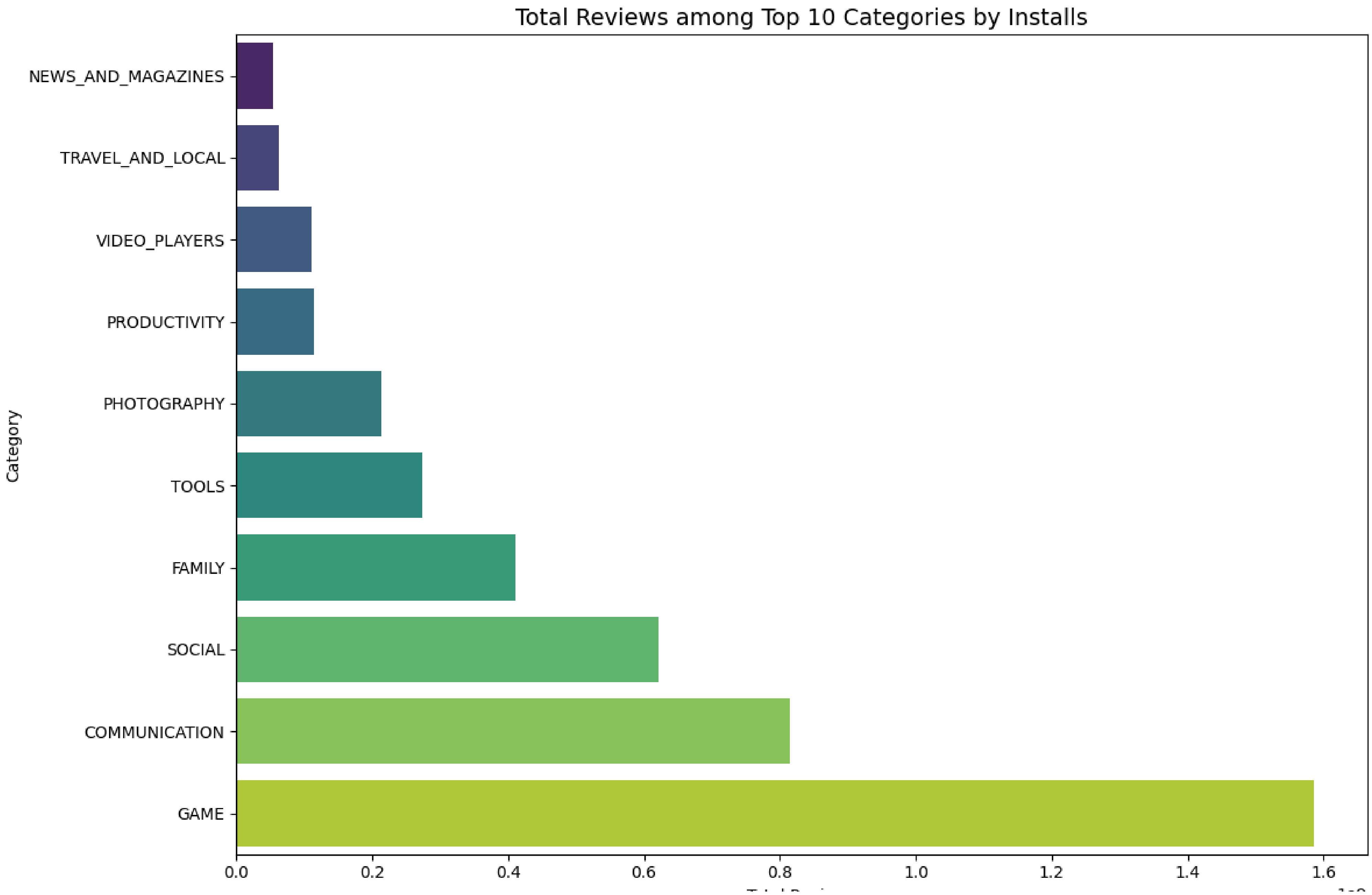
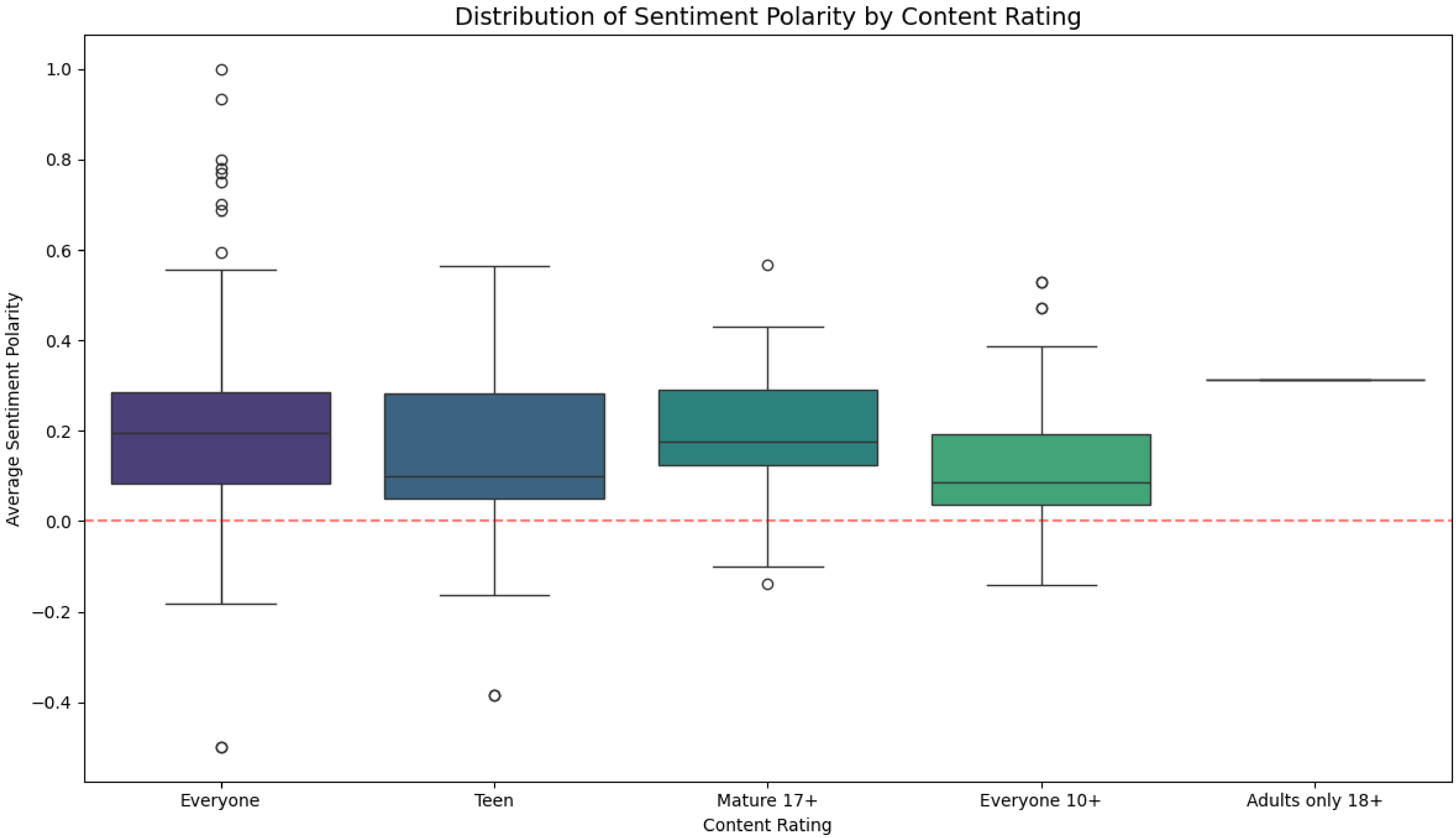
```python
# Group by category
cat_stats = apps_reviews.groupby("Category").agg(
    Apps_Count=("App", "count"),
    Total_Installs=("Installs_Num", "sum"),
    Total_Reviews=("Reviews", "sum")
).sort_values("Total_Installs", ascending=False)

# Top 10 categories by installs
top10_cats = cat_stats.head(10)
```

```python
# Bar chart: Total reviews in top 10 categories
plt.figure(figsize=(12, 8))
sns.barplot(x="Total_Reviews", y=top10_cats.sort_values("Total_Reviews").index,
            data=top10_cats.sort_values("Total_Reviews"), palette="viridis")
plt.xlabel("Total Reviews")
plt.ylabel("Category")
plt.title("Total Reviews among Top 10 Categories by Installs", fontsize=14)
plt.tight_layout()
plt.show()
```

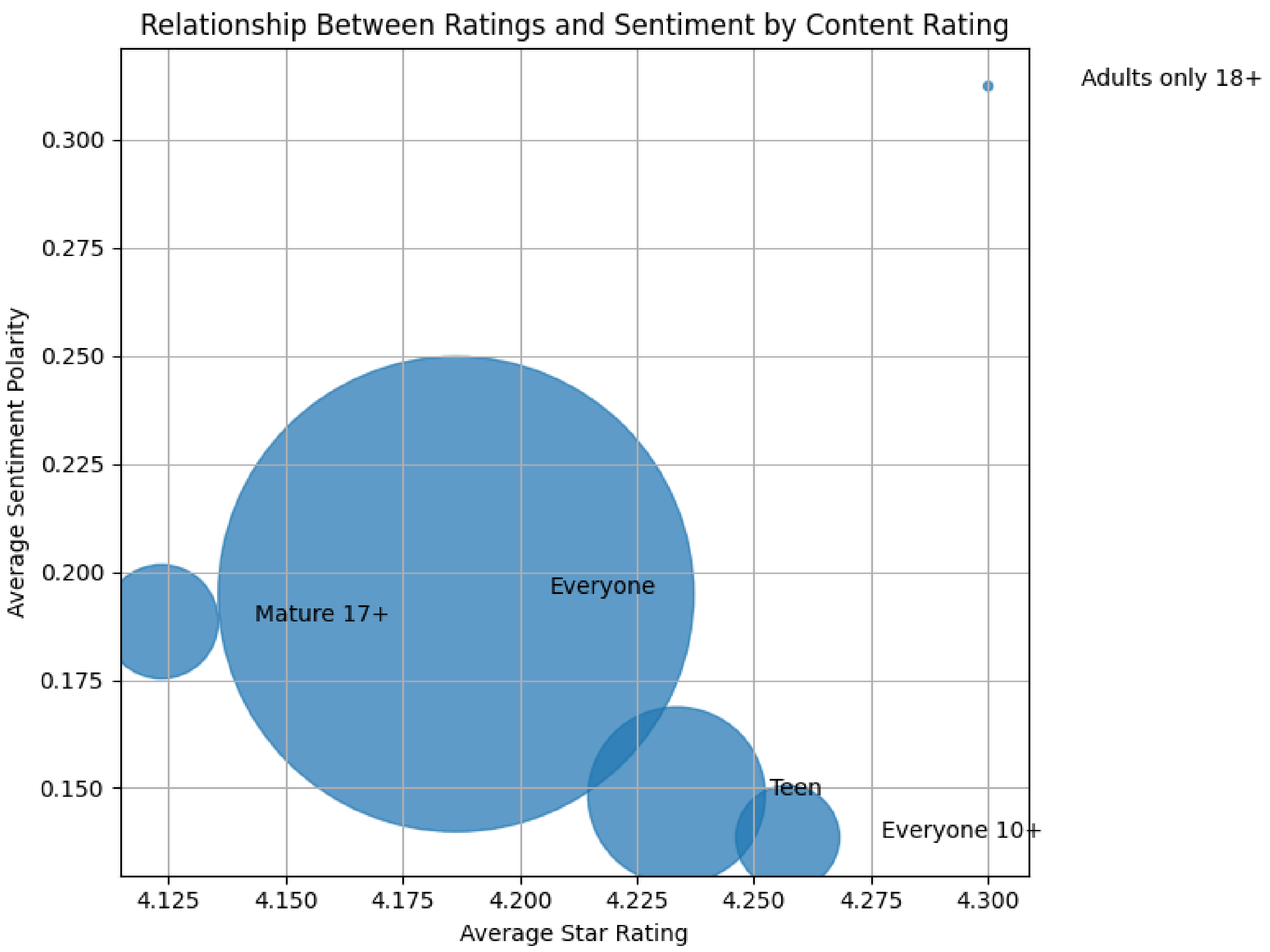


```python
# Filter rows with polarity and content rating available
sentiment_data = apps_reviews.dropna(subset=["Polarity_Mean", "Content Rating"])

plt.figure(figsize=(12,7))
sns.boxplot(
    data=sentiment_data,
    x="Content Rating",
    y="Polarity_Mean",
    order=sentiment_data["Content Rating"].value_counts().index,
    palette="viridis"
)
plt.axhline(0, color="red", linestyle="--", alpha=0.5)
plt.title("Distribution of Sentiment Polarity by Content Rating", fontsize=14)
plt.ylabel("Average Sentiment Polarity")
plt.xlabel("Content Rating")
plt.tight_layout()
plt.show()
```



```python
# Group by Content Rating
content_rating_stats = apps_reviews.groupby("Content Rating").agg(
    Avg_Star_Rating=("Rating", "mean"),
    Avg_Sentiment_Polarity=("Polarity_Mean", "mean"),
    Apps_Count=("App", "count")
).sort_values("Avg_Star_Rating", ascending=False)

print(content_rating_stats)
```

```python
# Scatter plot: Star rating vs sentiment
plt.figure(figsize=(8,6))
plt.scatter(
    content_rating_stats["Avg_Star_Rating"],
    content_rating_stats["Avg_Sentiment_Polarity"],
    s=content_rating_stats["Apps_Count"]*5,  # bubble size = number of apps
    alpha=0.7
)
for idx, row in content_rating_stats.iterrows():
    plt.text(row["Avg_Star_Rating"]+0.02, row["Avg_Sentiment_Polarity"], idx)

plt.xlabel("Average Star Rating")
plt.ylabel("Average Sentiment Polarity")
plt.title("Relationship Between Ratings and Sentiment by Content Rating")
plt.grid(True)
plt.tight_layout()
plt.show()
```



```python
# Select features
features = [
    "Reviews_log", "Size_MB", "Price", "Installs_log", "Type_Binary",
    "Polarity_Mean", "Subjectivity_Mean"
] + [col for col in df_model.columns if col.startswith("Content Rating_")]
```

```python
# Drop rows with missing in selected features or target
df_model = df_model.dropna(subset=features + ["Rating"])
```

```python
# Split data
X = df_model[features]
y = df_model["Rating"]

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

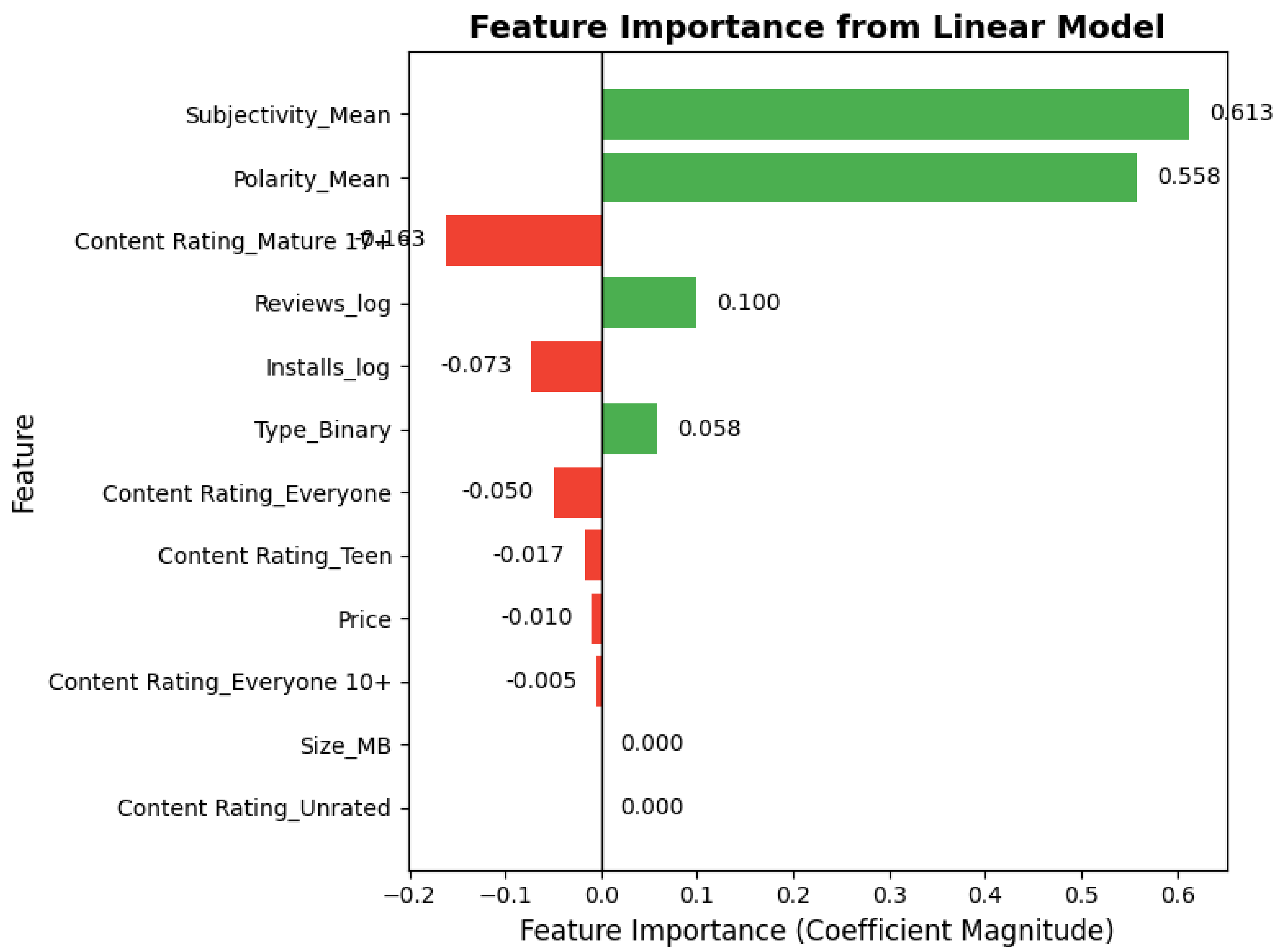
```python
# Train model
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

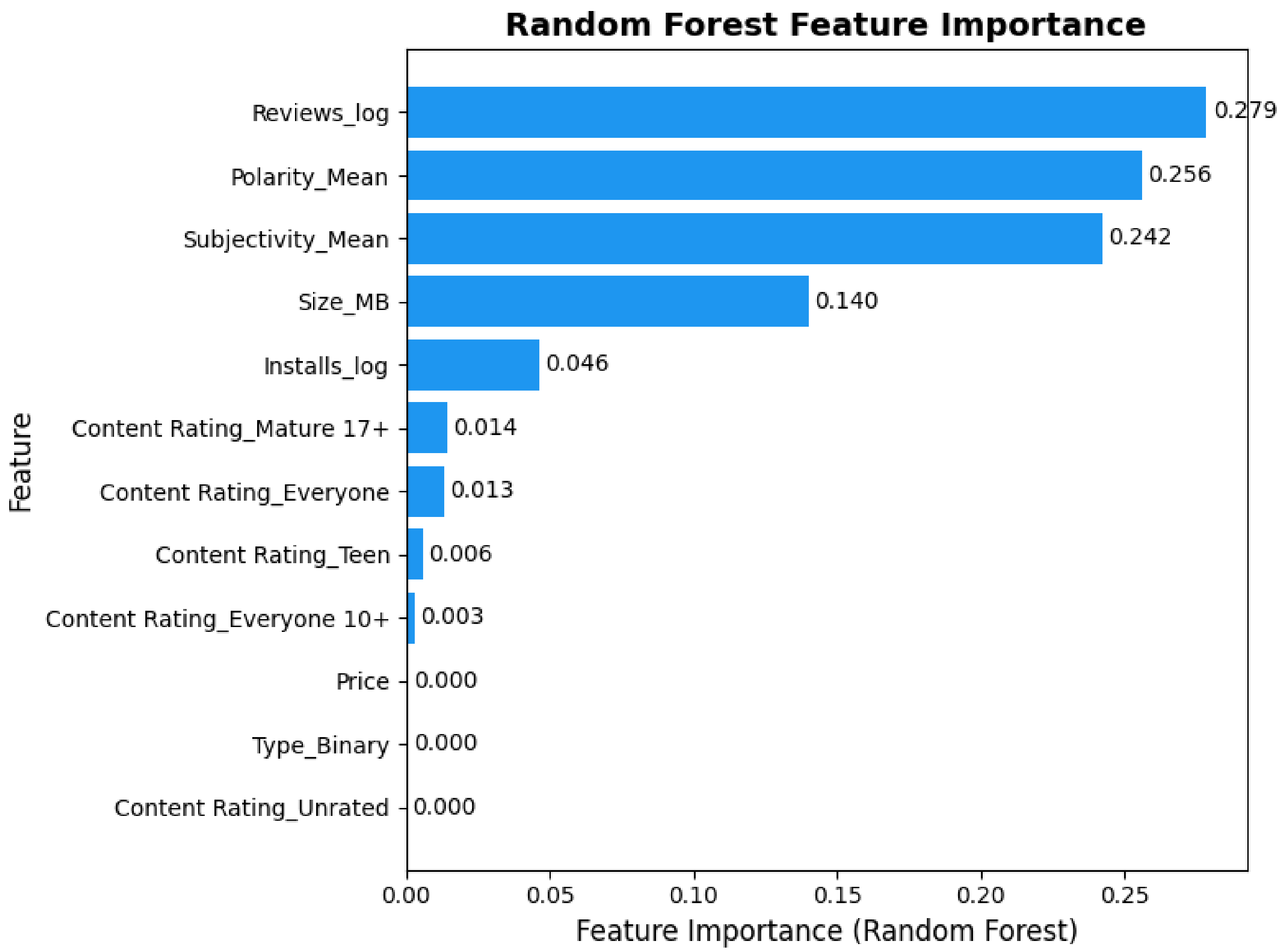
```python
# Predict & evaluate
y_pred = lr.predict(X_test)
print("R² score:", r2_score(y_test, y_pred))
```

```
R² score: 0.2643866646376303
```

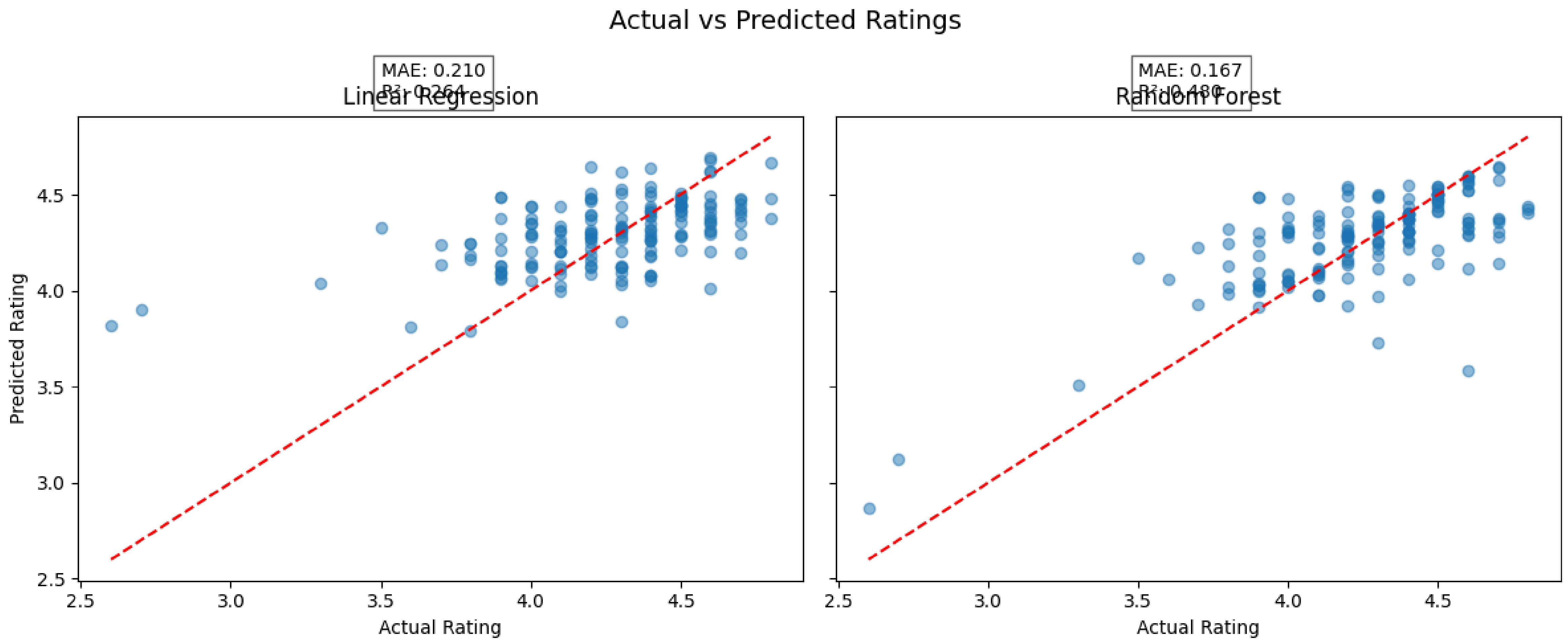
```python
    'Content Rating_Unrated': 0.000000
}

# Convert to DataFrame
df_importance = pd.DataFrame(list(feature_importance.items()), columns=['Feature', 'Importance'])

# Sort by absolute importance
df_importance['Abs_Importance'] = df_importance['Importance'].abs()
df_importance = df_importance.sort_values('Abs_Importance', ascending=True)

# Plot
plt.figure(figsize=(8,6))
colors = ['#4CAF50' if x > 0 else '#F44336' for x in df_importance['Importance']]
bars = plt.barh(df_importance['Feature'], df_importance['Importance'], color=colors)

# Add coefficient annotations
for bar, value in zip(bars, df_importance['Importance']):
    plt.text(
        value + (0.02 if value >= 0 else -0.02),  # offset to right for positive, left for negative
        bar.get_y() + bar.get_height() / 2,
        f"{value:.3f}",
        va='center',
        ha='left' if value >= 0 else 'right',
        fontsize=10
    )

plt.axvline(0, color='black', linewidth=1)
plt.xlabel('Feature Importance (Coefficient Magnitude)', fontsize=12)
plt.ylabel('Feature', fontsize=12)
plt.title('Feature Importance from Linear Model', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```



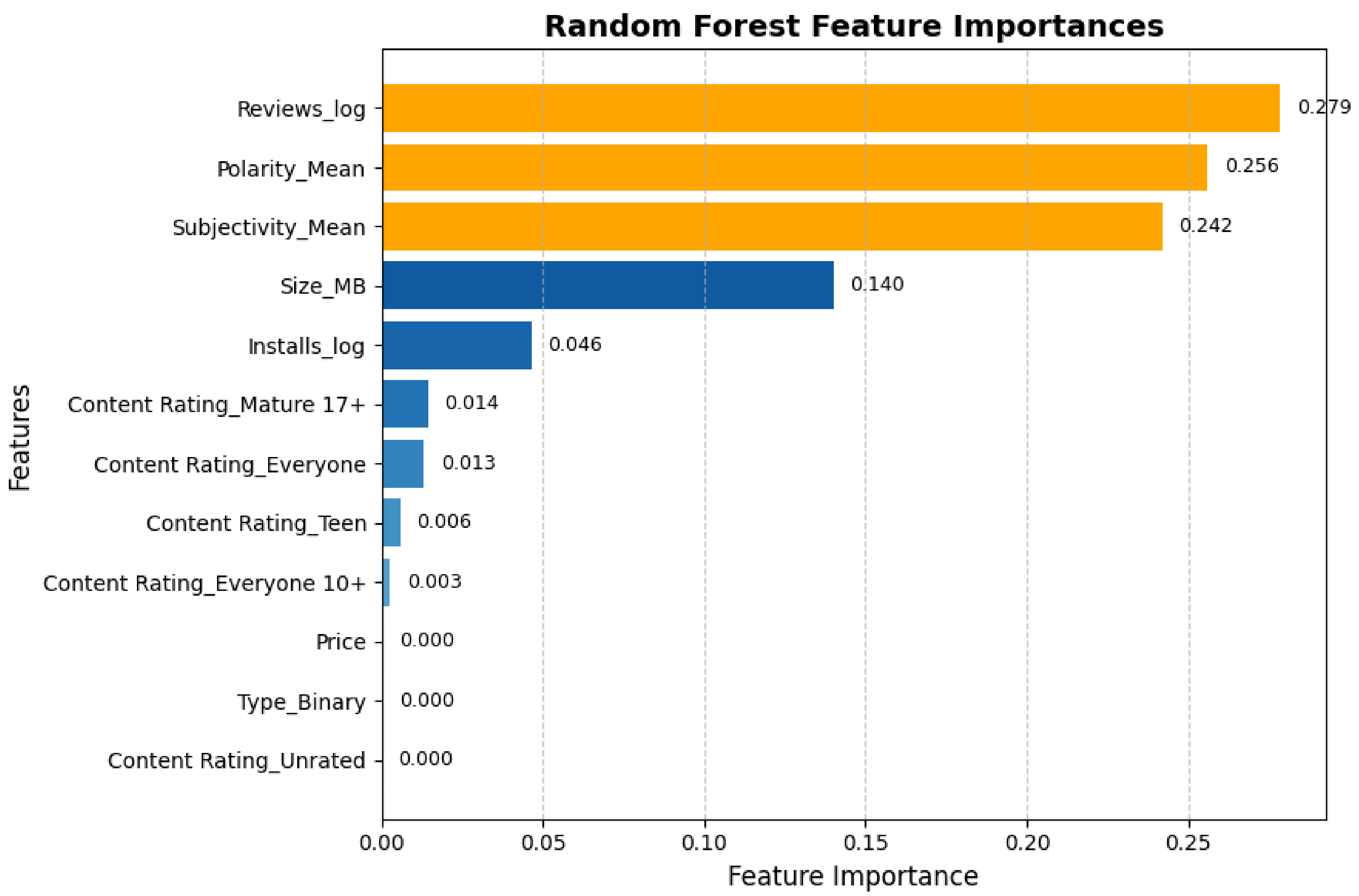
```python
# Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Train Random Forest
rf = RandomForestRegressor(n_estimators=200, random_state=42)
rf.fit(X_train, y_train)

# Get feature importance
rf_importances = pd.DataFrame({
    'Feature': X.columns,
    'Importance': rf.feature_importances_
}).sort_values('Importance', ascending=True)

# Plot
plt.figure(figsize=(8,6))
bars = plt.barh(rf_importances['Feature'], rf_importances['Importance'], color="#2196F3")

# Add annotations
for bar, value in zip(bars, rf_importances['Importance']):
    plt.text(
        value + 0.002,
        bar.get_y() + bar.get_height() / 2,
        f"{value:.3f}",
        va='center'
    )

plt.xlabel('Feature Importance (Random Forest)', fontsize=12)
plt.ylabel('Feature', fontsize=12)
plt.title('Random Forest Feature Importance', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```



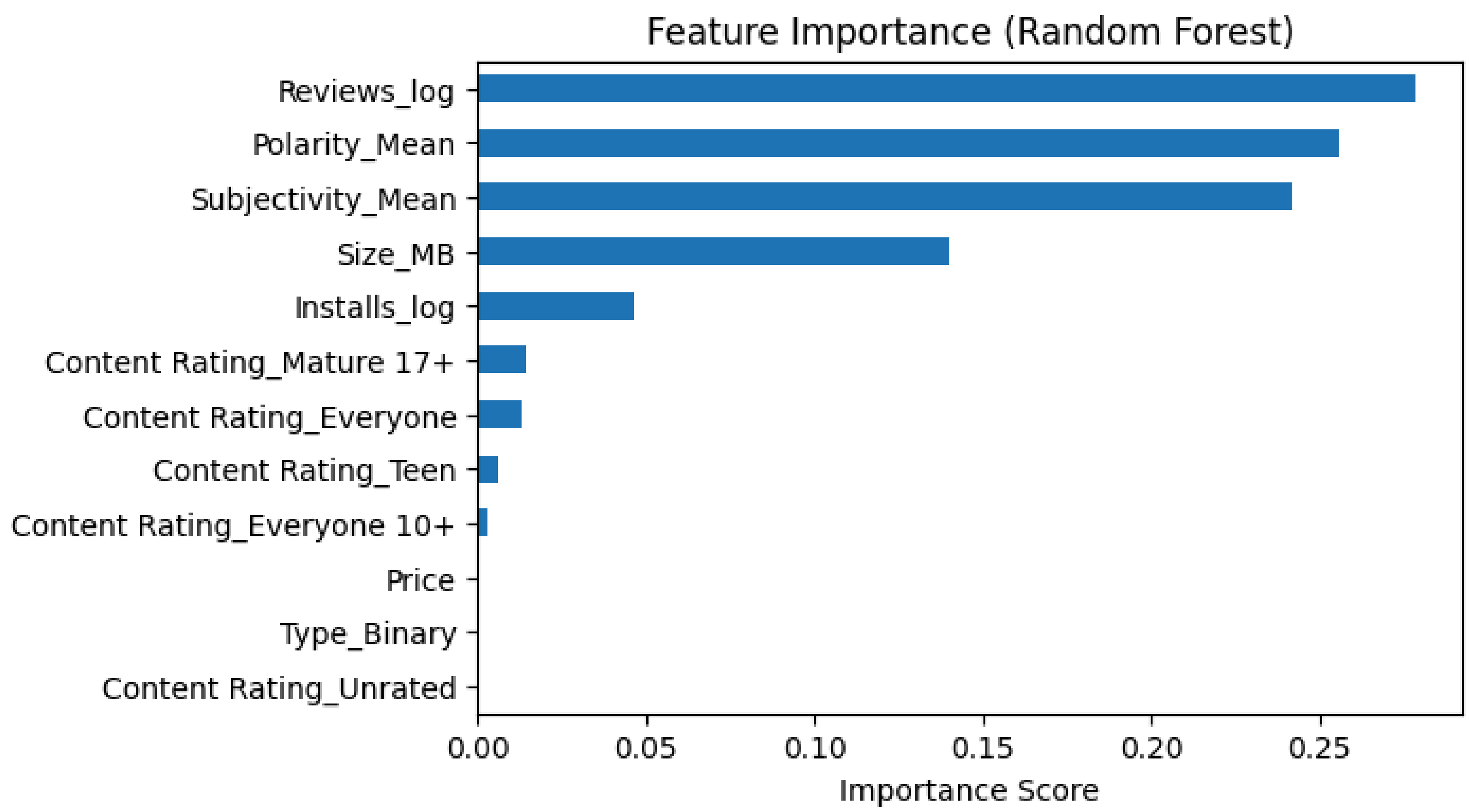
```python
import matplotlib.pyplot as plt

# Predictions
y_pred_lr = lr.predict(X_test)
y_pred_rf = rf.predict(X_test)

# Create subplots
fig, axes = plt.subplots(1, 2, figsize=(12, 5), sharey=True)

# Linear Regression plot
axes[0].scatter(y_test, y_pred_lr, alpha=0.5)
axes[0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
axes[0].set_title('Linear Regression')
axes[0].set_xlabel('Actual Rating')
axes[0].set_ylabel('Predicted Rating')
axes[0].text(3.5, 5.0, f"MAE: {0.210:.3f}\nR²: {0.264:.3f}", fontsize=10,
             bbox=dict(facecolor='white', alpha=0.6))

# Random Forest plot
axes[1].scatter(y_test, y_pred_rf, alpha=0.5)
axes[1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
axes[1].set_title('Random Forest')
axes[1].set_xlabel('Actual Rating')
axes[1].text(3.5, 5.0, f"MAE: {0.167:.3f}\nR²: {0.480:.3f}", fontsize=10,
             bbox=dict(facecolor='white', alpha=0.6))

plt.suptitle('Actual vs Predicted Ratings', fontsize=14)
plt.tight_layout()
plt.show()
```

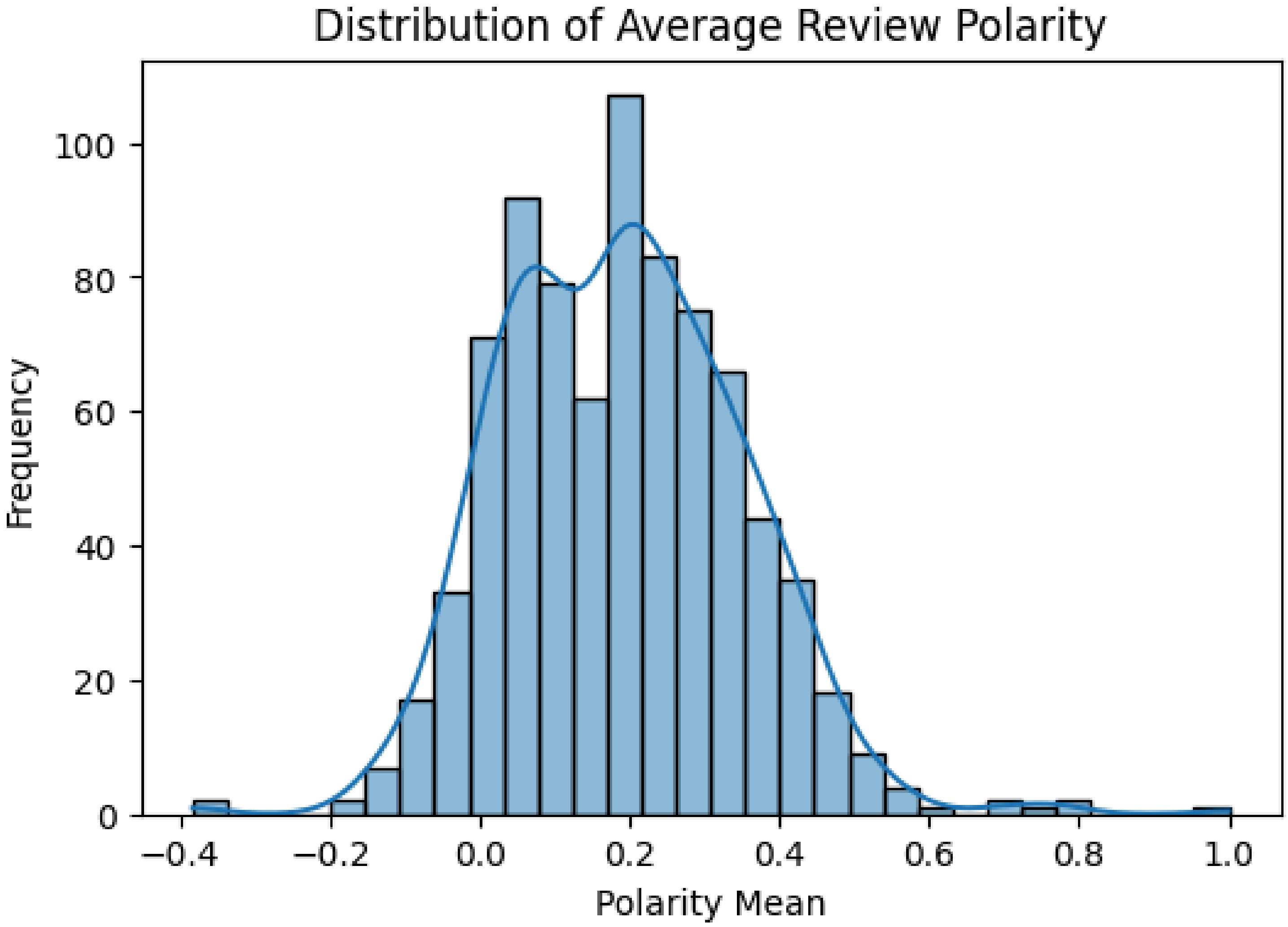


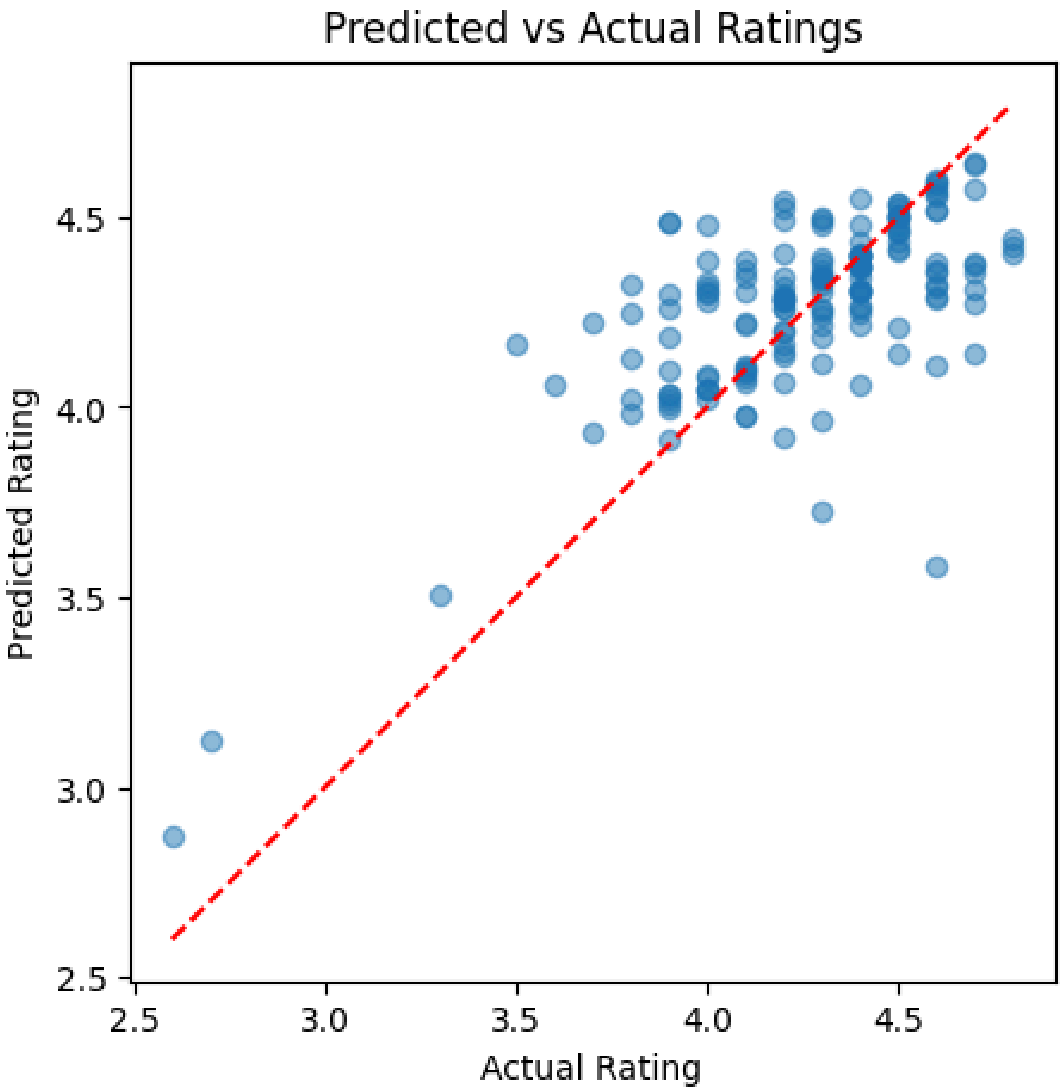
```python
importances = rf.feature_importances_
feature_names = X_train.columns

# Create DataFrame and sort
feat_imp_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=True)

# Color gradient
colors = plt.cm.Blues(np.linspace(0.4, 1, len(feat_imp_df)))

# Highlight top 3 features in orange
top3_idx = feat_imp_df['Importance'].nlargest(3).index
highlight_colors = [
    'orange' if i in top3_idx else colors[j]
    for j, i in enumerate(feat_imp_df.index)
]

# Plot
plt.figure(figsize=(9, 6))
bars = plt.barh(feat_imp_df['Feature'], feat_imp_df['Importance'], color=highlight_colors)

# Add labels
for bar in bars:
    width = bar.get_width()
    plt.text(width + 0.005, bar.get_y() + bar.get_height()/2,
             f'{width:.3f}', va='center', fontsize=9)

plt.xlabel('Feature Importance', fontsize=12)
plt.ylabel('Features', fontsize=12)
plt.title('Random Forest Feature Importances', fontsize=14, weight='bold')
plt.grid(axis='x', linestyle='--', alpha=0.6)
plt.tight_layout()
```



```python
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Sentiment Distribution
plt.figure(figsize=(6,4))
sns.histplot(data=df_model, x='Polarity_Mean', bins=30, kde=True)
plt.title('Distribution of Average Review Polarity')
plt.xlabel('Polarity Mean')
plt.ylabel('Frequency')
plt.show()

# 2. Feature Importance
importances = pd.Series(rf.feature_importances_, index=X.columns).sort_values()
plt.figure(figsize=(6,4))
importances.plot(kind='barh')
plt.title('Feature Importance (Random Forest)')
plt.xlabel('Importance Score')
plt.show()

# 3. Predicted vs Actual
plt.figure(figsize=(5,5))
plt.scatter(y_test, y_pred_rf, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel('Actual Rating')
plt.ylabel('Predicted Rating')
plt.title('Predicted vs Actual Ratings')
plt.show()
```




Distribution of Average Review Polarity
Frequency
0
20
40
60
80
100
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
Polarity Mean


Predicted vs Actual Ratings
Predicted Rating
2.5
3.0
3.5
4.0
4.5
2.5
3.0
3.5
4.0
4.5
Actual Rating