فاز دوم پروژه داده کاوی

1. بارگذاری داده‌های پاکسازی شده

```
clean_data = pd.read_csv('clean_dataset.csv')
clean_data = clean_data.dropna()
clean_data.head()
```

در این مرحله، داده‌ها از فایل CSV بارگذاری و ردیف‌های ناقص حذف می‌شوند. سپس، پنج ردیف اول داده‌ها نمایش داده می‌شوند.

2. انتخاب ستون‌های با دو مقدار یکتا

```
def find_columns_with_two_unique_values(df):
    cols_with_two_unique = [col for col in df.columns if
df[col].nunique() == 2 and col != 'Genres' and col != 'Category']
    return cols_with_two_unique

columns_with_two_unique_values =
find_columns_with_two_unique_values(clean_data)

print("Columns with exactly 2 unique values:")
print(columns_with_two_unique_values)

selected_data = clean_data[columns_with_two_unique_values]
```

این قسمت تابعی تعریف می‌کند که ستون‌هایی با دو مقدار یکتا را پیدا می‌کند و سپس این ستون‌ها را از داده‌های پاکسازی شده انتخاب و نمایش می‌دهد.

3. استخراج الگوهای مکرر و قوانین انجمنی

```
from mlxtend.frequent_patterns import apriori,
association_rules

min_support = 0.6
min_confidence = 0.8
desired_patterns_count = 12
```

```
support_step = 0.05
while True:
    frequent_itemsets = apriori(selected_data,
min_support=min_support, use_colnames=True)
    if len(frequent_itemsets[frequent_itemsets['support'] >
min_support]) >= desired_patterns_count:
        break
    min_support -= support_step

rules = association_rules(frequent_itemsets,
metric="confidence", min_threshold=min_confidence)

print("Frequent Itemsets:")
print(frequent_itemsets)

print("\nAssociation Rules:")
print(rules)

frequent_patterns =
frequent_itemsets[frequent_itemsets['support'] > min_support]
```

تحلیل خروجی‌ها

- الگوهای مکرر (Frequent Itemsets):

- در اینجا، الگوهای مکرر با استفاده از الگوریتم Apriori و پارامتر `min_support` اولیه ۰.۶ استخراج می‌شوند.

- اگر تعداد الگوهای مکرر استخراج شده کمتر از ۱۲ باشد، مقدار `min_support` به اندازه ۰.۰۵ کاهش می‌یابد و دوباره محاسبه انجام می‌شود تا زمانی که حداقل ۱۲ الگوی مکرر یافت شود.

- قوانین انجمنی (Association Rules):

- پس از استخراج الگوهای مکرر، قوانین انجمنی با استفاده از پارامتر `min_confidence` برابر ۰.۸ استخراج می‌شوند.

خروجی:

- Frequent Itemsets: نمایش الگوهای مکرری که در داده‌ها تکرار شده‌اند.
- Association Rules: نمایش قوانینی که رابطه بین آیتم‌ها را با مقدار اطمینان مشخص می‌کند.

```
Frequent Patterns (at least 12):
     support                                                itemsets
0   0.938915                                                 (Type)
1   0.854995                                       (Content Rating)
2   0.657469                                     (In app purchases)
3   0.969743                                       (Editor Choice)
4   0.803996                                (Content Rating, Type)
5   0.610276                              (In app purchases, Type)
6   0.909800                                (Editor Choice, Type)
7   0.594101                  (Content Rating, In app purchases)
8   0.837488                    (Content Rating, Editor Choice)
9   0.653473                   (Editor Choice, In app purchases)
10  0.552997  (Content Rating, In app purchases, Type)
11  0.786870    (Content Rating, Editor Choice, Type)
```

4. خوشه بندی با استفاده K-Means

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import seaborn as sns

# Step 1: Determine the optimal number of clusters using the Elbow
method and Silhouette score
def determine_optimal_clusters(data, max_k):
    inertia = []
    silhouette_scores = []

    for k in range(2, max_k+1):
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(data)
        inertia.append(kmeans.inertia_)
```

```
        silhouette_scores.append(silhouette_score(data, kmeans.labels_))

    # Plot the Elbow method graph
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.plot(range(2, max_k+1), inertia, marker='o')
    plt.title('Elbow Method')
    plt.xlabel('Number of clusters')
    plt.ylabel('Inertia')

    # Plot the Silhouette score graph
    plt.subplot(1, 2, 2)
    plt.plot(range(2, max_k+1), silhouette_scores, marker='o')
    plt.title('Silhouette Score')
    plt.xlabel('Number of clusters')
    plt.ylabel('Silhouette Score')

    plt.tight_layout()
    plt.show()

    return inertia, silhouette_scores

# Run the function to determine optimal clusters
inertia, silhouette_scores = determine_optimal_clusters(clean_data, 10)

# Step 2: Choose the optimal number of clusters (k) based on the Elbow
method and Silhouette score
optimal_k = np.argmax(silhouette_scores) + 2
print(f'Optimal number of clusters: {optimal_k}')

# Step 3: Cluster the data using the optimal number of clusters
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
clean_data['Cluster'] = kmeans.fit_predict(clean_data)

# Step 4: Analyze and visualize the results
# Plot the distribution of each column for each cluster
plt.figure(figsize=(20, 15))
for i, column in enumerate(clean_data.columns[:-1]):
    plt.subplot(4, 4, i+1)
    sns.boxplot(x='Cluster', y=column, data=clean_data)
    plt.title(f'Distribution of {column}')

plt.tight_layout()
plt.show()

# Plot the clusters (if we can reduce to 2D or 3D using PCA for
```

```
visualization)
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
reduced_data = pca.fit_transform(clean_data.iloc[:, :-1])
plt.figure(figsize=(10, 8))
sns.scatterplot(x=reduced_data[:, 0], y=reduced_data[:, 1],
hue=clean_data['Cluster'], palette='Set1', s=100, alpha=0.7)
plt.title('Clusters Visualization with PCA')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend()
plt.show()
```

این کد برای کلاسترینگ داده‌ها با استفاده از الگوریتم K-Means طراحی شده است و شامل چند مرحله اصلی است:

مرحله 1: تعیین تعداد بهینه خوشه‌ها

در این مرحله، تابع determine_optimal_clusters تعریف شده است. این تابع ابتدا برای اعداد خوشه‌ها از 2 تا `max_k`، مدل K-Means را با هر تعداد خوشه آموزش می‌دهد و دو معیار زیر را برای هر تعداد خوشه ذخیره می‌کند:
- Inertia: جمع مربعات فواصل هر نقطه از مرکز خوشه‌ای که به آن تعلق دارد.
- Silhouette Score: میزان شباهت نقاط درون خوشه و اختلاف با نقاط خوشه‌های مجاور.

سپس، در نمودارهای الگوی (`Elbow Method`) و امتیاز سیلوئت (Silhouette Score) این معیارها را بر حسب تعداد خوشه‌ها رسم می‌کند.

مرحله 2: انتخاب تعداد بهینه خوشه‌ها

بر اساس نمودار Silhouette، تعداد بهینه خوشه‌ها انتخاب می‌شود با استفاده از `np.argmax(silhouette_scores) + 2`، که در آن شمارش از 2 است.

مرحله 3: خوشه‌بندی داده‌ها

در این مرحله، K-Means با تعداد بهینه خوشه‌ها اجرا می‌شود و هر نقطه داده به یک خوشه تخصیص می‌یابد. خروجی این خوشه‌بندی به `clean_data` اضافه می‌شود.

مرحله 4: تحلیل و بصری‌سازی نتایج

- تحلیل توزیع متغیرها بر حسب هر خوشه: برای هر متغیر (ستون) در داده، با استفاده از `sns.boxplot`، توزیع آن متغیر برای هر خوشه را نمایش می‌دهد.

- بصری‌سازی خوشه‌ها با استفاده از PCA: ابتدا داده‌ها را به کمک `PCA` به ابعاد کمتر تبدیل می‌کند (در اینجا به 2 بعد) و سپس با استفاده از `sns.scatterplot`، نموداری از داده‌ها با رنگ‌آمیزی بر اساس خوشه‌ها ایجاد می‌کند.

5. خوشه بندی با استفاده از روش Hierarchical

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from sklearn.decomposition import PCA
import seaborn as sns

Z = linkage(clean_data, method='ward')

plt.figure(figsize=(15, 10))
dendrogram(Z, truncate_mode='level', p=5)
plt.title('Dendrogram')
plt.xlabel('Sample index or (Cluster size)')
plt.ylabel('Distance')
plt.show()

optimal_k = 3

clean_data['Cluster'] = fcluster(Z, t=optimal_k, criterion='maxclust')
```

```
plt.figure(figsize=(20, 15))
for i, column in enumerate(clean_data.columns[:-1]):
    plt.subplot(4, 4, i+1)
    sns.boxplot(x='Cluster', y=column, data=clean_data)
    plt.title(f'Distribution of {column}')

plt.tight_layout()
plt.show()

pca = PCA(n_components=2)
reduced_data = pca.fit_transform(clean_data.iloc[:, :-1])
plt.figure(figsize=(10, 8))
sns.scatterplot(x=reduced_data[:, 0], y=reduced_data[:, 1],
hue=clean_data['Cluster'], palette='Set1', s=100, alpha=0.7)
plt.title('Clusters Visualization with PCA')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend()
plt.show()
```

این کد برای خوشه‌بندی سلسله مراتبی (Hierarchical Clustering) داده‌ها طراحی شده است، و شامل چند مرحله اصلی است:

مرحله 1: ایجاد دندروگرام (Dendrogram)

در این مرحله، ابتدا از تابع linkage از کتابخانه scipy.cluster.hierarchy استفاده می‌شود تا یک ماتریس اتصال (Linkage matrix) برای داده‌ها با استفاده از روش Ward ایجاد شود. سپس با استفاده از تابع `dendrogram`، دندروگرام داده‌ها رسم می‌شود. این نمودار نشان‌دهنده اتصال سلسله‌مراتبی داده‌ها در زمان اجرای خوشه‌بندی است.

مرحله 2: خوشه‌بندی با استفاده از دندروگرام

با توجه به دندروگرام و تعیین تعداد بهینه خوشه‌ها (در اینجا `optimal_k = 3`)، از تابع fcluster برای تخصیص هر نقطه به یک خوشه استفاده می‌شود. معیار maxclust به عنوان معیار تعیین تعداد خوشه‌ها انتخاب شده است.

مرحله 3: تحلیل و بصری‌سازی نتایج

- تحلیل توزیع متغیرها بر حسب هر خوشه: با استفاده از `sns.boxplot`، توزیع هر متغیر برای هر خوشه رسم می‌شود. این کمک می‌کند تا ویژگی‌های هر خوشه مورد بررسی قرار گیرند.

- بصری‌سازی خوشه‌ها با استفاده از PCA: داده‌ها با استفاده از `PCA` به فضای کمتر ابعاد تبدیل می‌شوند (در اینجا به 2 بعد) و سپس با استفاده از `sns.scatterplot`، نموداری از داده‌ها با رنگ‌آمیزی بر اساس خوشه‌ها نمایش داده می‌شود. این کار برای مشاهده توزیع و الگوهای مختلف بین نقاط داده‌ها در خوشه‌های مختلف استفاده می‌شود.

6. طبقه بندی

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score

# Define the parameter grids for each model
param_grids = {
    'Random Forest': {
        'n_estimators': [100, 300],
        'max_depth': [None, 10, 30],
        'min_samples_split': [2, 10],
        'min_samples_leaf': [1, 4]
    },
    'Decision Tree': {
        'max_depth': [None, 10, 30],
        'min_samples_split': [2, 10],
```

```
        'min_samples_leaf': [1, 4]
    },
    'SVM': {
        'C': [0.1, 1, 10],
        'gamma': [1, 0.1, 0.01],
        'kernel': ['rbf', 'linear']
    },
    'Naive Bayes': {
        # No parameters to tune for GaussianNB
    }
}

# Train and evaluate classifiers with hyperparameter tuning
models = {
    'Random Forest': RandomForestClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'SVM': SVC(),
    'Naive Bayes': GaussianNB()
}

best_model_name = None
best_model_score = 0
best_model = None
best_params = None

for model_name, model in models.items():
    print(f"Training {model_name}...")
    if model_name in param_grids and param_grids[model_name]:
        grid_search = GridSearchCV(model, param_grids[model_name], cv=5,
scoring='accuracy')
        grid_search.fit(X_train, y_train)
        mean_score = grid_search.best_score_
        print(f'{model_name} Best Cross-Validation Accuracy:
{mean_score:.4f}')
        print(f'{model_name} Best Parameters:
{grid_search.best_params_}')
        if mean_score > best_model_score:
            best_model_score = mean_score
            best_model_name = model_name
            best_model = grid_search.best_estimator_
            best_params = grid_search.best_params_
    else:
        model.fit(X_train, y_train)
        scores = cross_val_score(model, X_train, y_train, cv=5,
scoring='accuracy')
        mean_score = scores.mean()
```

```
        print(f'{model_name} Mean Cross-Validation Accuracy:
{mean_score:.4f}')
        if mean_score > best_model_score:
            best_model_score = mean_score
            best_model_name = model_name
            best_model = model
            best_params = None

print(f'\nBest Model: {best_model_name} with Accuracy:
{best_model_score:.4f}')
if best_params:
    print(f'Best Parameters: {best_params}')

# Train the best model on the entire training set
best_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = best_model.predict(X_test)

# Evaluate the best model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f'\nEvaluation of the Best Model ({best_model_name}):')
print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1-score: {f1:.4f}')
```

این کد به منظور آموزش مدل‌های مختلف دسته‌بندی با استفاده از انتخاب بهینه پارامترها (hyperparameter tuning) و ارزیابی آن‌ها طراحی شده است.

مرحله 1: وارد کردن کتابخانه‌ها و تعریف پارامترهای مدل

- وارد کردن کتابخانه‌ها: کتابخانه‌های مورد نیاز مانند pandas برای کار با داده‌ها، numpy برای عملیات عددی، و کلاس‌ها و توابع مربوط به مدل‌های دسته‌بندی از sklearn وارد می‌شوند.

- تعریف پارامترهای برای هر مدل: برای هر مدل دسته‌بندی (Random Forest، Decision Tree، SVM، Naive Bayes) یک دیکشنری از پارامترهای ممکن برای تنظیم مدل با استفاده از GridSearchCV تعریف می‌شود.

مرحله 2: آموزش و ارزیابی مدل‌ها

- تعریف مدل‌ها: چهار مدل مختلف دسته‌بندی با استفاده از کلاس‌های `RandomForestClassifier`، `DecisionTreeClassifier`، `SVC`، `GaussianNB` تعریف می‌شوند.
- انتخاب بهترین مدل و بهینه‌سازی پارامترها: برای هر مدل، اگر پارامترهای مشخص شده در param_grids وجود داشته باشند، از GridSearchCV برای انتخاب بهترین مدل با استفاده از ارزیابی متقاطع (Cross-Validation) استفاده می‌شود. اگر پارامترهایی وجود نداشته باشد، مدل مستقیماً بر روی داده‌های آموزش آموزش داده می‌شود.

- ارزیابی و انتخاب بهترین مدل: مدلی که دقت متوسط متقاطع بیشتری داشته باشد به عنوان بهترین مدل انتخاب می‌شود و جزئیات آن شامل نام، دقت و پارامترهای بهینه چاپ می‌شود.

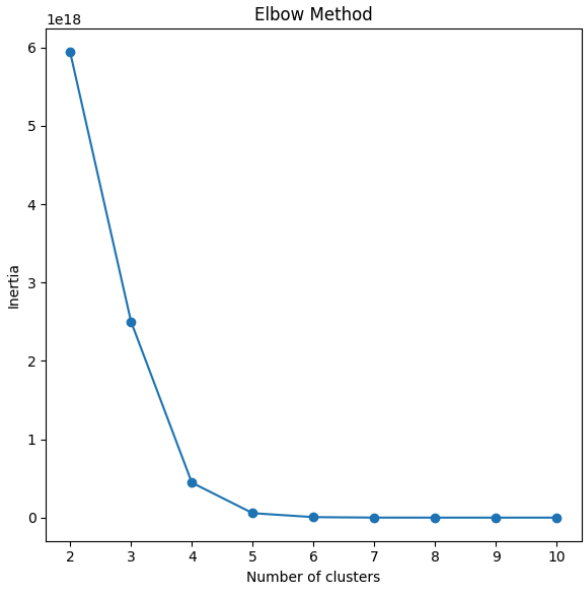
مرحله 3: آموزش بهترین مدل و ارزیابی نهایی

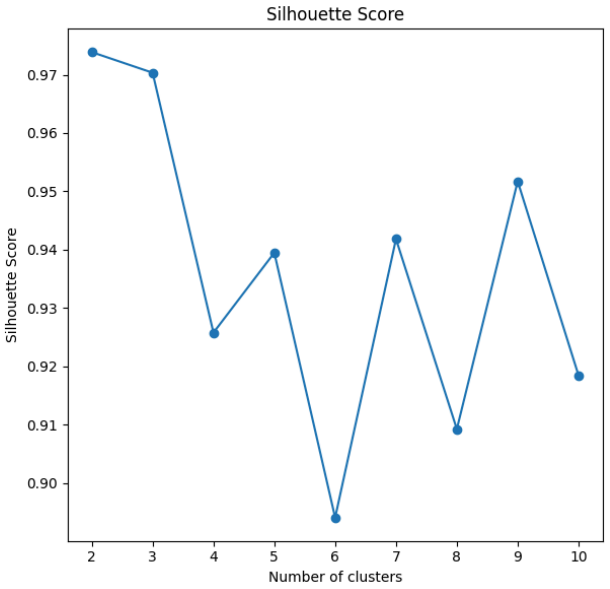
- آموزش بهترین مدل بر روی کل مجموعه‌ی آموزش: مدل انتخاب شده با استفاده از fit بر روی کل مجموعه‌ی آموزش آموزش داده می‌شود.

- پیش‌بینی و ارزیابی روی مجموعه‌ی آزمون: با استفاده از `predict`، مدل بر روی داده‌های آزمون پیش‌بینی انجام می‌دهد و دقت (Accuracy)، دقت تقسیم شده (precision)، بازخوانی (recall) و امتیاز (F1-score) برای بهترین مدل محاسبه و چاپ می‌شود.

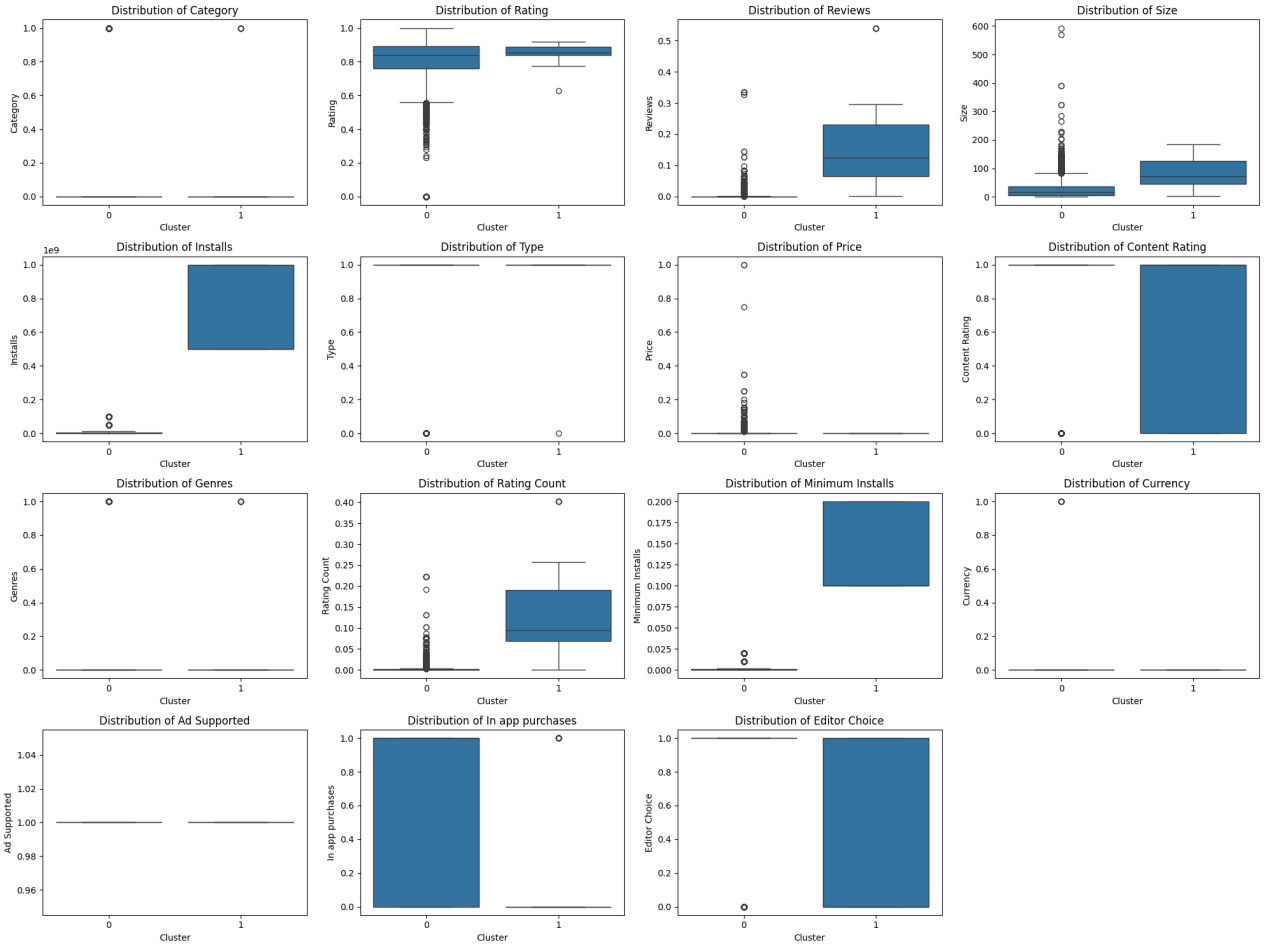
نتیجه گیری بخش خوشه بندی:

توضیح بخش K-Means

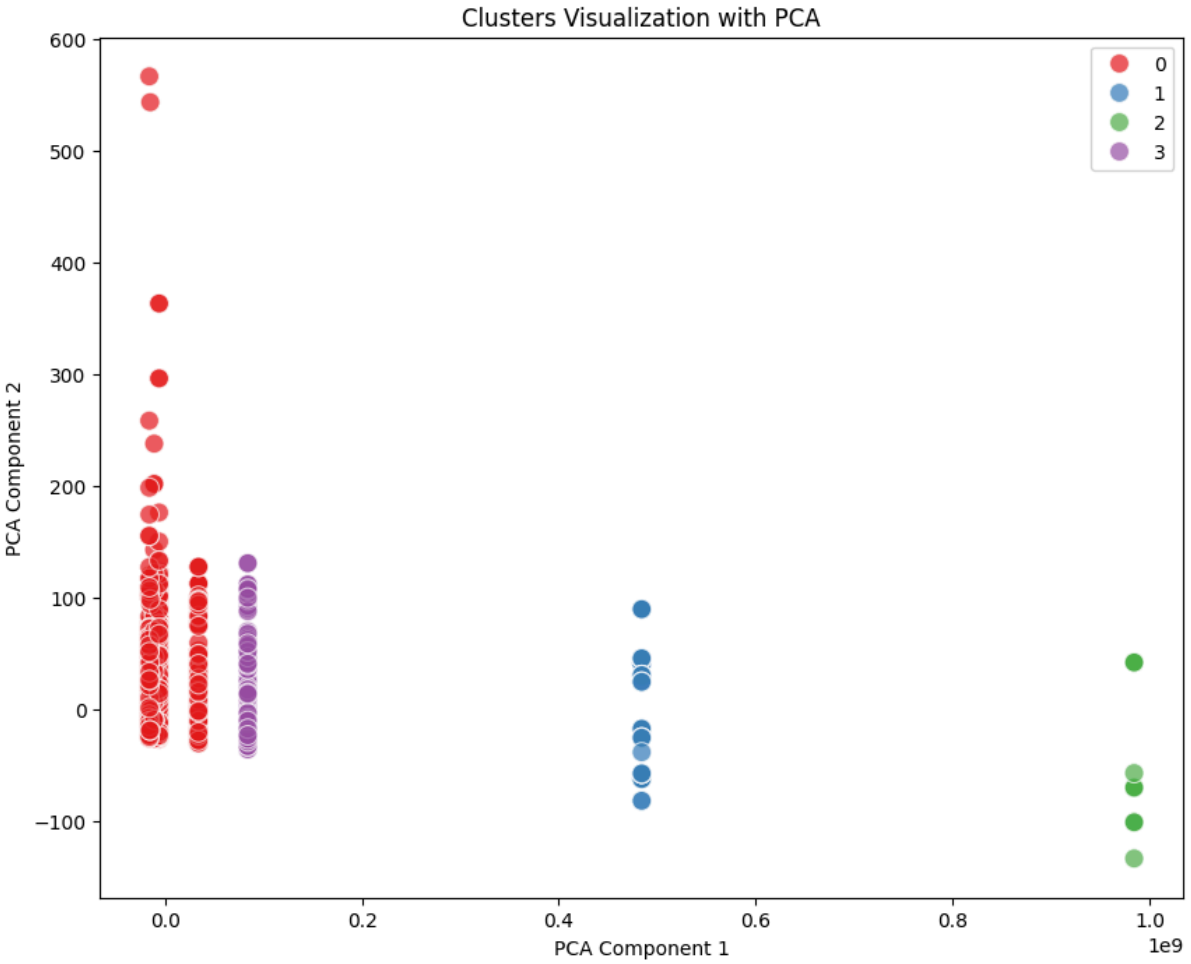




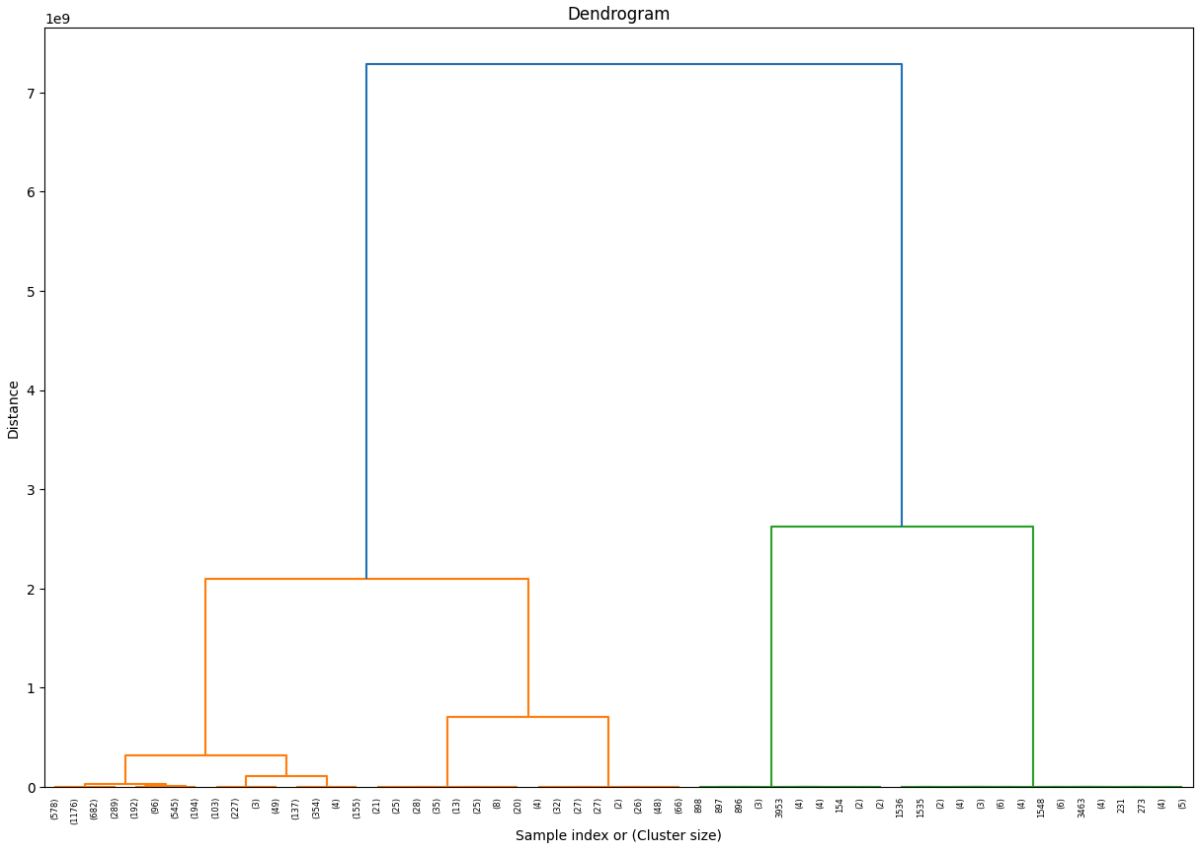
با استفاده از این دو معیار مقدار k بهینه انتخاب میشود. سپس الگوریتم k-means را اجرا کردیم و توزیع داده های زیر را در هر ستون خواهیم داشت:



در نهایت خوشه بندی به این صورت خواهد بود (مقدار k چهار می باشد) :

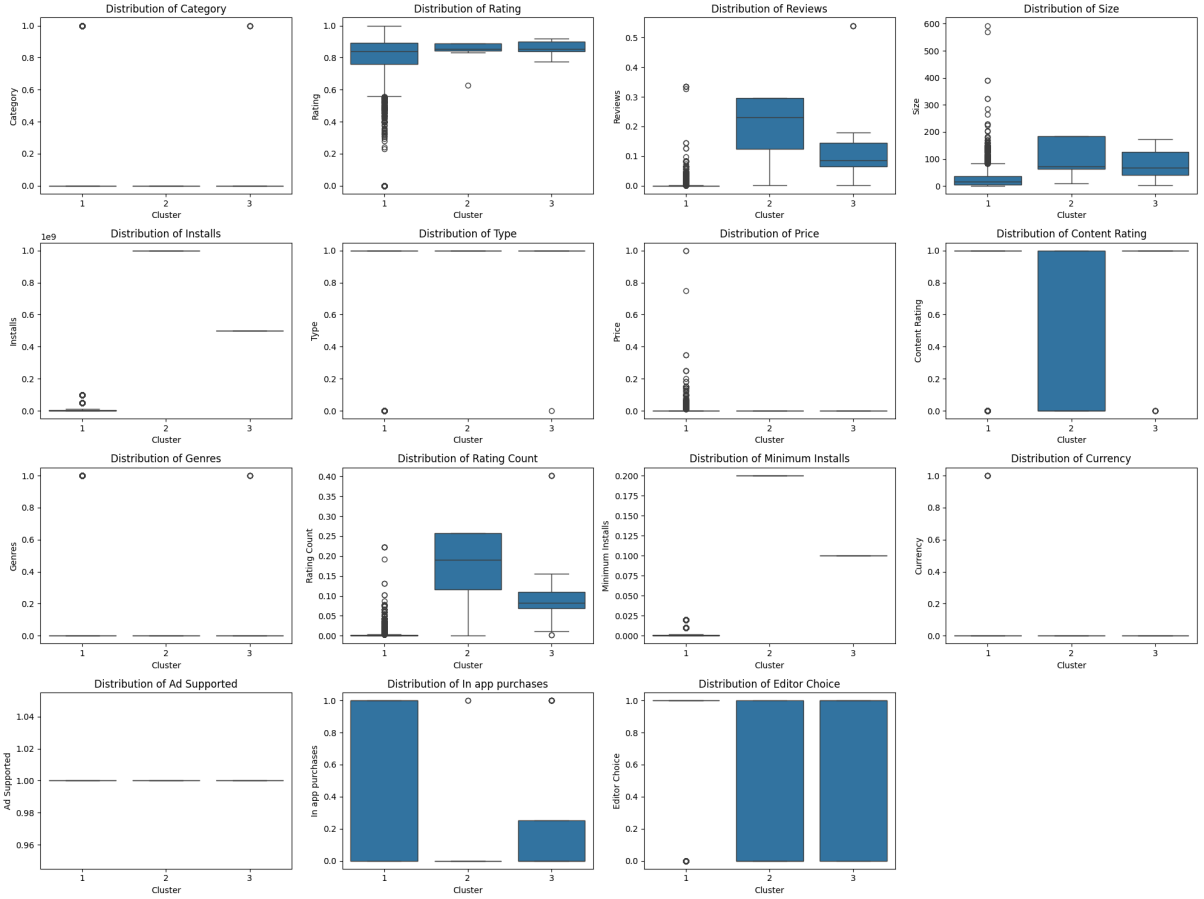


توضیح بخش Hierarchical

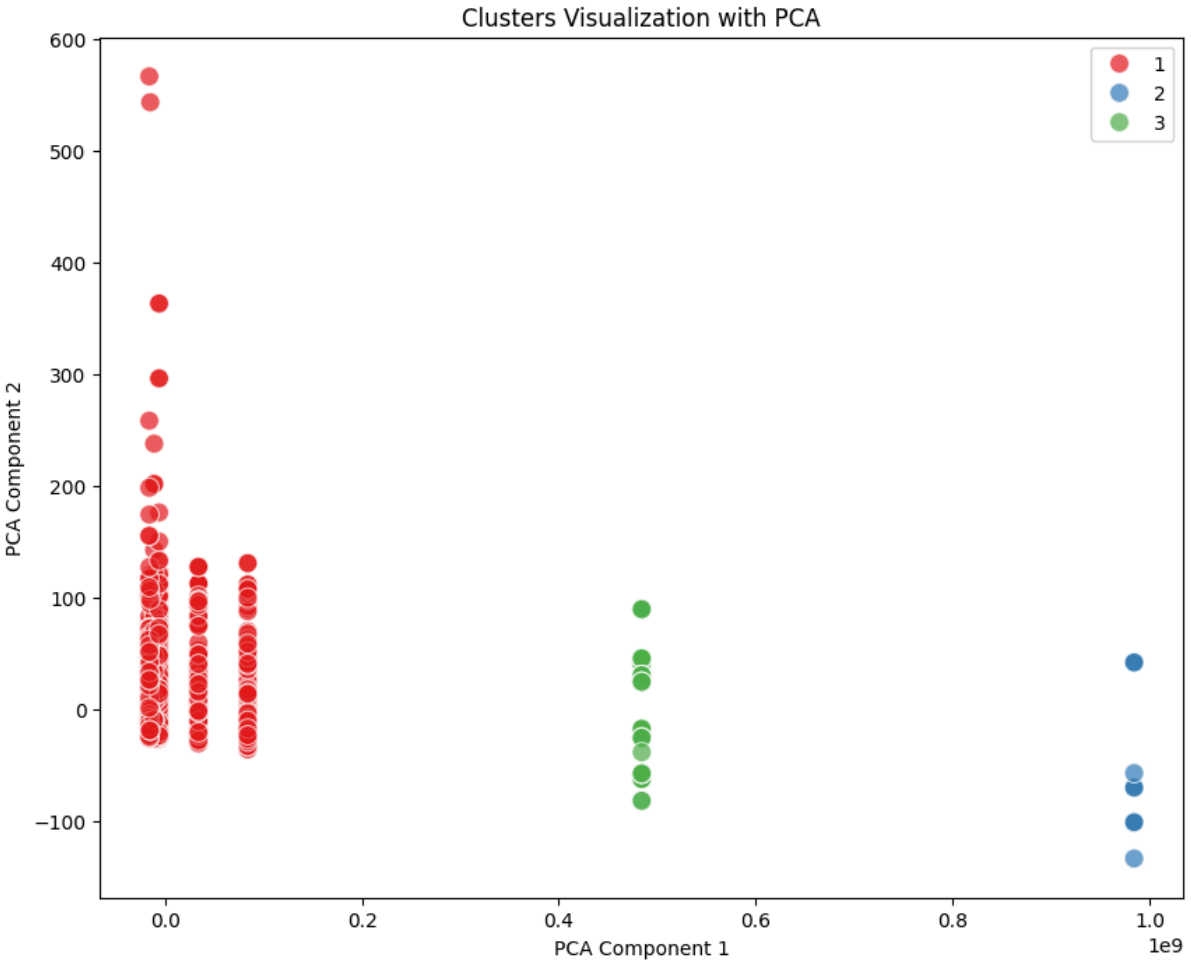


با استفاده از نمودار dendrogram متوجه میشویم که مقدار k بهینه برای خوشه بندی چه مقداری خواهد بود. با پیدا کردن طولانی ترین خط عمودی درون نمودار و رسم یک خط افقی در نقطه شروع آن ما را به عدد بهینه می رساند. در اینجا ما با مقدار 3 خوشه بندی را انجام داده ایم.

توزیع داده ها در هر ستون :



در نهایت خوشه بندی به این صورت خواهد بود (مقدار k سه می باشد) :



نتیجه گیری بخش طبقه بندی:

```
Training Random Forest...
Random Forest Best Cross-Validation Accuracy: 0.9864
Random Forest Best Parameters: {'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}
Training Decision Tree...
Decision Tree Best Cross-Validation Accuracy: 0.9831
Decision Tree Best Parameters: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2}
Training SVM...
SVM Best Cross-Validation Accuracy: 0.9693
SVM Best Parameters: {'C': 10, 'gamma': 1, 'kernel': 'rbf'}
Training Naive Bayes...
Naive Bayes Mean Cross-Validation Accuracy: 0.4622

Best Model: Random Forest with Accuracy: 0.9864
Best Parameters: {'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}

Evaluation of the Best Model (Random Forest):
Accuracy: 0.9867
Precision: 0.9850
Recall: 1.0000
F1-score: 0.9925
```

همانطور که در خروجی می بینیم بعد از تست هایپر پارامترهای مختلف برای مدل های مختلف ، بهترین دقت برای مدل RandomForest شد با پارامترهای زیر:

```
{'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}
```

همچنین ارزیابی این مدل به این صورت خواهد بود:

```
Evaluation of the Best Model (Random Forest):
Accuracy: 0.9867
Precision: 0.9850
Recall: 1.0000
F1-score: 0.9925
```

پس این مدل برای پیشبینی و طبقه بندی مقدار فیچر Rating در این دیتاست مناسب تر از بقیه مدل های طبقه بندی می باشد.