در بخش اول پروژه داده های عددی را تشخیص داده ایم در بخش تشخیص الگوهای مکرر چون نیاز به داده های کتگوری داشتیم ستون هایی که به کتگوری تبدیل شده بودند را حذف کردیم مثل 'Rating', 'Reviews', 'Size','Minimum Installs', 'Rating Count', "Installs" و همچنین price را به کتگوری تبدیل کردیم:

```
df_drop['Price'] = pd.cut(df_drop['Price'], bins=[-np.inf, 0.33,
          0.66, np.inf], labels=['free', 'cheap', 'expensive'],
                                              ("duplicates="drop
```

و داده هایی که تکست بودند و به نظرمون اهمیت زیادی نداشتند بر روی rating رو حذف کردیم:

```
df_drop=df_drop.drop(columns= [ 'Price', 'Last Updated', 'Current
                                        ,'Ver', 'App Id', 'Currency
       Developer Id', 'Developer'
                  ,'Website', 'Developer Email', 'Privacy Policy
            Summary','Developer'
   Address','Android Ver', 'Minimum Android', 'Released','Android
                                                ,'version Text
          (['Developer','Version'
```

از df_encoded = pd.get_dummies(df_drop)برای binary کردن داده ها استفاده کردیم و سپس apirori بر روی داده ها اعمال کردیم برای پیدا کردن ایتم ست های پرتکرار با استفاده از این کد:

```
        frequent_itemsets = apriori(df_encoded, min_support=0.3,
                                              (use_colnames=True
                                    (print(frequent_itemsets
```

با اینکه شاید min support 0.5 بزرگ به نظر برسد ولی بر روی داده های ما نتایج قابل قبولی داشت

	support	itemsets
0	0.35136	(In app purchases)
1	0.93584	(Type_Free)
2	0.85088	(Content Rating_Everyone)
3	0.6848	(Ad Supported_True)
4	0.39328	(Size_category_Medium)
5	0.3024	(Minimum_installs_category_Very Low)
6	0.33728	(Type_Free, In app purchases)
7	0.79632	(Type_Free, Content Rating_Everyone)
8	0.67216	(Type_Free, Ad Supported_True)
9	0.37472	(Type_Free, Size_category_Medium)
10	0.5744	(Content Rating_Everyone, Ad Supported_True)
11	0.32192	(Content Rating_Everyone, Size_category_Medium)
12	0.56416	...Type_Free, Content Rating_Everyone, Ad Suppor)
13	0.30688	…Type_Free, Content Rating_Everyone, Size_cate)

```
        frequent_itemsets = apriori(df_encoded, min_support=0.5,
                                              (use_colnames=True
```

	support	itemsets
0	0.93584	(Type_Free)
1	0.85088	(Content Rating_Everyone)
2	0.6848	(Ad Supported_True)
3	0.79632	(Type_Free, Content Rating_Everyone)
4	0.67216	(Type_Free, Ad Supported_True)
5	0.5744	(Content Rating_Everyone, Ad Supported_True)
6	0.56416	…Type_Free, Content Rating_Everyone, Ad Suppor)

همینطور که میبیند با min_support 0.3 , 0.5 زیاد تفاوتی حاصل نشده و از هردو میتوان این نتیجه را گرفت که اکثر برنامه‌های رایگان دارای تبلیغات و مناسب برای همه سنین هستند، از support بالاتری برخوردار هستند.

در قسمت بعد rule هارا با association_rules با ایتم های پرتکرار شناسایی شده محاسبه میکنیم که نتایج به صورت زیر است

	\ antecedents
0	(Type_Free)
1	(Content Rating_Everyone)
2	(Type_Free)
3	(Ad Supported_True)
4	(Ad Supported_True)
5	(Type_Free, Content Rating_Everyone)
6	(Type_Free, Ad Supported_True)
7	(Content Rating_Everyone, Ad Supported_True)
8	(Ad Supported_True)

	\ consequents	antecedent support
0	Content Rating_Everyone)	0.93584)
1	Type_Free)	0.85088)
2	Ad Supported_True)	0.93584)
3	Type_Free)	0.68480)
4	Content Rating_Everyone)	0.68480)
5	Ad Supported_True)	0.79632)
6	Content Rating_Everyone)	0.67216)
7	Type_Free)	0.57440)
8	Type_Free, Content Rating_Everyone)	0.68480)

	\ consequent support	support	confidence	lift	leverage	conviction
0	1.000233	0.000032	1.000041	0.850915	0.79632	0.85088
1	1.000595	0.000032	1.000041	0.935878	0.79632	0.93584
2	1.073325	0.031529	1.049021	0.718373	0.67216	0.68480
3	3.353270	0.031237	1.048721	0.981510	0.67216	0.93584
4	0.977983	0.008177-	0.985766	0.838785	0.57440	0.85088
5	1.386970	0.056899	1.092933	0.844207	0.67216	0.68480
6	0.997663	0.002273-	0.996016	0.854610	0.57440	0.85088
7	3.493735	0.026359	1.048947	0.982015	0.56416	0.93584
8	1.160198	0.018855	1.034651	0.823962	0.56416	0.79632

از این نتایج میتوانیم نتیجع بگیریم که قوانین استخراج شده نشان‌دهنده الگوهای قوی در داده‌ها هستند. به طور خاص، برنامه‌های رایگان اغلب دارای رتبه‌بندی "برای همه" و تبلیغات هستند.
مقادیر بالای `confidence` (اغلب بیش از 0.7) نشان‌دهنده اعتماد بالا به این قوانین است.
`lift` نزدیک به 1 در بسیاری از قوانین نشان‌دهنده رابطه تقریباً مستقل بین برخی از ویژگی‌ها است، اما مقدار بالای `lift` (بیش از 1) در برخی دیگر نشان‌دهنده افزایش احتمال مشترک وقوع آنهاست.

2.clustring

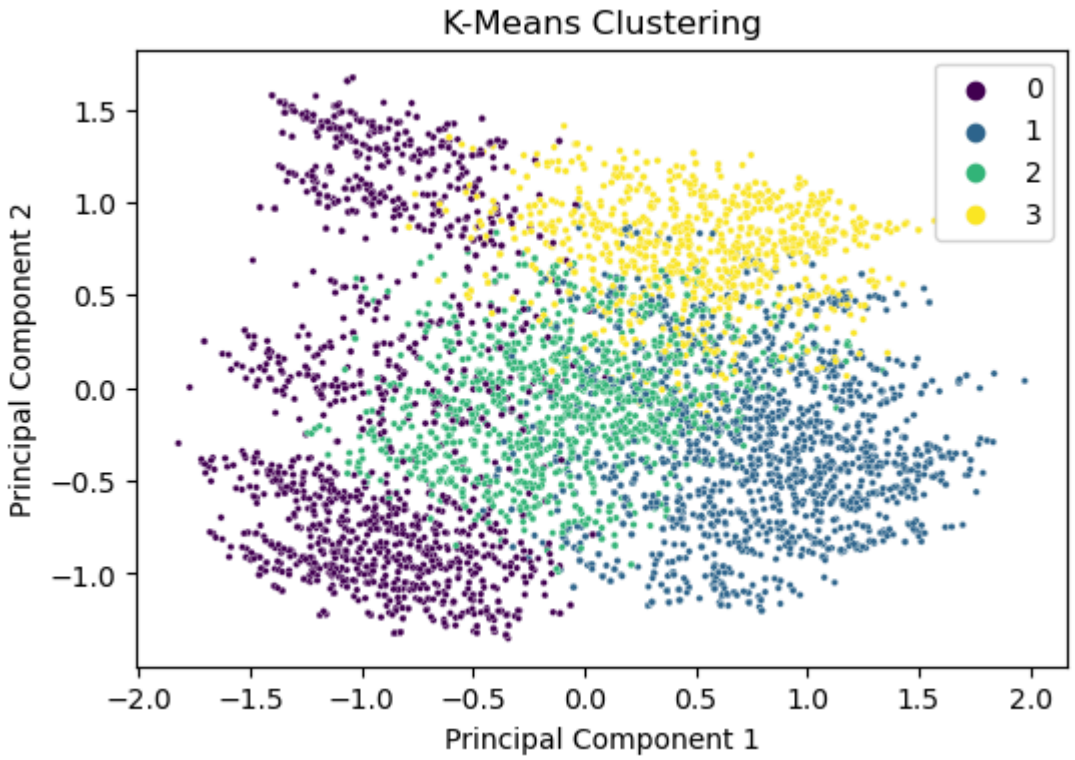
در این بخش ابتدا kmeans را با number cluster 4 و با این کد بر روی دیتاهای encode شده از فاز قبل اعمال کردیم

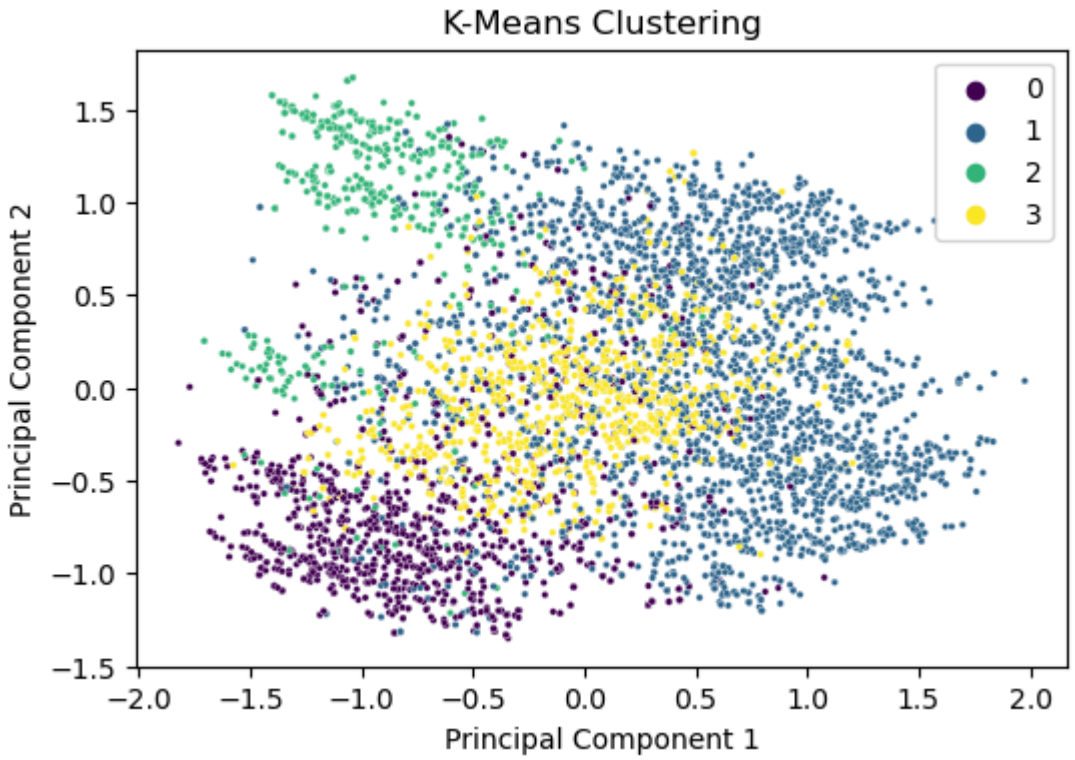
```
(kmeans = KMeans(n_clusters=4, random_state=42
(kmeans_labels = kmeans.fit_predict(df_encoded
```

سپس سلسه مراتبی را اجرا کردیم

```
(hierarchical = AgglomerativeClustering(n_clusters=4
(hierarchical_labels = hierarchical.fit_predict(df_encoded
```

برای کاهش ابعد pca را اعمال کردیم و نتایج را پلات کردیم در بخش دو بعدی نتایج کلاستر زیاد مشخص نبود ولی در پلات سه بعدی خوشه بندی مشخص که در کد قابل مشاهده است و نتایج بهتری داریم





در بخش بعدی برای انتخاب ویژگی ابتدا با استفاده از این کد:

```
()imputer = IterativeImputer
([df[numerical_columns] = imputer.fit_transform(df[numerical_columns
```

مقادیر nan را پر میکنیم و سپس برای نتیجه بهتر ویژگی های جدید ایجاد کردیم مثلا:

```
['df['Reviews_per_Install'] = df['Reviews'] / df['Installs
['df['Price_per_Install'] = df['Price'] / df['Installs
```

یا برای خروجی بهتر ویژگی هارا تبدیل کردیم:

```
(['df['log_Reviews'] = np.log1p(df['Reviews
(['df['sqrt_Installs'] = np.sqrt(df['Installs
```

مقادیر نویز حذف کردیم و داده هارا مقیاس بندی کردیم:

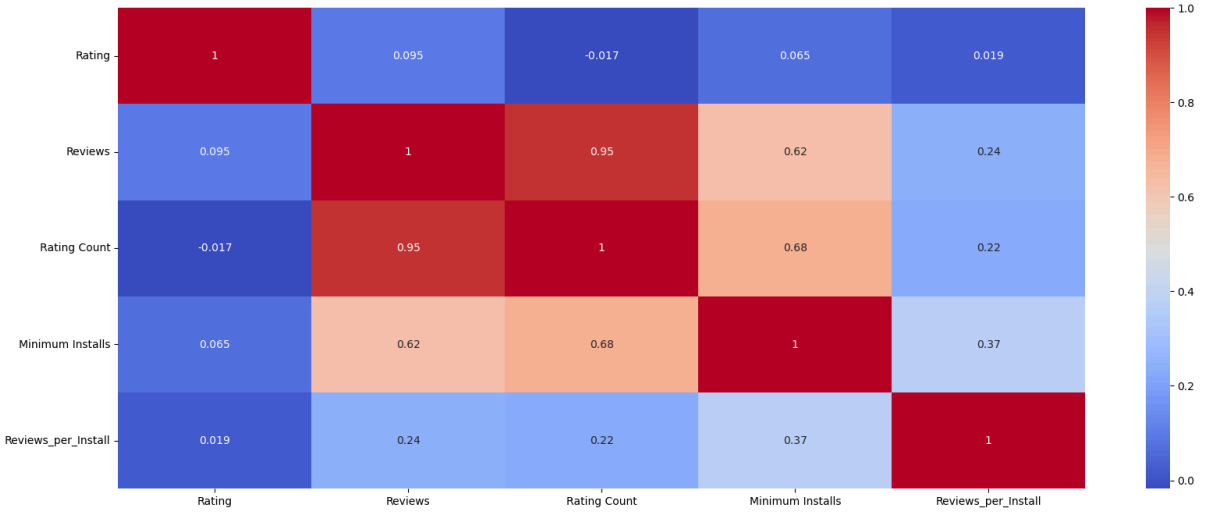
```
[(df = df[(np.abs(stats.zscore(df[numerical_columns])) < 3).all(axis=1

()scaler = StandardScaler
([df[numerical_columns] = scaler.fit_transform(df[numerical_columns
```

برای کاهش ابعد pca زدیم و سپس با selectKBest ویژگی هارا انتخاب کردیم:

```
(selector = SelectKBest(score_func=f_classif, k=5
(['selector.fit(df[numerical_columns], df['Rating
selected_features =
[()df[numerical_columns].columns[selector.get_support
```

در نهایت correlation محاسبه کردیم که به این صورت است



که نتیجه میگیریم:

همبستگی قوی بین **تعداد نظرات و تعداد امتیازات** وجود دارد، که نشان‌دهنده این است که کاربران معمولاً هم نظر می‌دهند و هم امتیاز می‌دهند.

همبستگی متوسط بین **تعداد نصب‌های حداقلی با تعداد نظرات و تعداد امتیازات** وجود دارد، که نشان‌دهنده این است که برنامه‌هایی که تعداد نصب بیشتری دارند معمولاً نظرات و امتیازات بیشتری هم دریافت می‌کنند.

سایر ویژگی‌ها مانند **امتیاز کلی (Rating)** با دیگر ویژگی‌ها همبستگی ضعیفی دارند، که نشان می‌دهد این ویژگی‌ها تأثیر قابل توجهی بر امتیاز کلی ندارند.

در بخش بعدی سراغ ساخت مدل برای طبقه بندی میرویم که در کد زیر انجام شده

```
rating_mean = df['Rating'].mean()
rating_std = df['Rating'].std()

def categorize_rating(rating):
    if rating > rating_mean + rating_std:
        return 'High'
    elif rating < rating_mean - rating_std:
        return 'Low'
    else:
        return 'Medium'

df['Rating_Category'] = df['Rating'].apply(categorize_rating)

X = df[selected_features].drop(columns='Rating')
y = df['Rating_Category']

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

nb_classifier = GaussianNB()
rf_classifier = RandomForestClassifier(random_state=42)
svm_classifier = SVC(random_state=42)

classifiers = [
    ('Naive Bayes', nb_classifier),
    ('Random Forest', rf_classifier),
    ('SVM', svm_classifier)
]

evaluation_metrics = {
    'Model': [],
    'Accuracy': [],
    'F1-Score': [],
    'Recall': [],
    'Precision': []
}

for name, classifier in classifiers:
```

```
                                (classifier.fit(X_train, y_train
                               (y_pred = classifier.predict(X_test

                          (accuracy = accuracy_score(y_test, y_pred
                  ('f1 = f1_score(y_test, y_pred, average='weighted
          ('recall = recall_score(y_test, y_pred, average='weighted
  ('precision = precision_score(y_test, y_pred, average='weighted

                           (evaluation_metrics['Model'].append(name
                    (evaluation_metrics['Accuracy'].append(accuracy
                          (evaluation_metrics['F1-Score'].append(f1
                      (evaluation_metrics['Recall'].append(recall
                  (evaluation_metrics['Precision'].append(precision

                          (df_metrics = pd.DataFrame(evaluation_metrics
                                                  (print(df_metrics
```

و خروجی مدل به این صورت است

Model	Accuracy	F1-Score	Recall	Precision	
Naive Bayes	0.184801	0.171123	0.184801	0.822637	0
Random Forest	0.917098	0.909238	0.917098	0.909452	1
SVM	0.907599	0.883655	0.907599	0.877088	

نتیجه ای که از این خروجی میگیریم به این صورت است:

در اینجا، داده‌ها به سه دسته `High`, `Medium` و `Low` بر اساس مقادیر میانگین و انحراف معیار تقسیم شده‌اند

مدل Naive Bayes عملکرد ضعیفی دارد که می‌تواند به دلیل فرضیات ساده‌ای باشد که این مدل دارد (مانند فرض استقلال ویژگی‌ها). دقت بالا در Precision نشان می‌دهد که وقتی مدل یک دسته را به درستی پیش‌بینی می‌کند، احتمالاً این دسته‌بندی صحیح است، اما بازخوانی پایین نشان‌دهنده این است که مدل قادر به شناسایی همه موارد درست نیست.

مدل Random Forest بهترین عملکرد را در بین سه مدل دارد. این مدل توانایی بالایی در شناسایی درست دسته‌بندی‌ها دارد و تعادل خوبی بین Precision و Recall برقرار کرده است. دقت بالای آن نشان‌دهنده این است که مدل به خوبی توانسته است الگوها را از داده‌ها استخراج کند.

مدل SVM نیز عملکرد خوبی دارد، اما کمی کمتر از Random Forest است. این مدل نیز تعادل خوبی بین Precision و Recall برقرار کرده است و نشان می‌دهد که مدل SVM نیز توانایی خوبی در شناسایی الگوها از داده‌ها دارد.

پس نتیجه میگیریم مدل Random Forest با دقت، امتیاز F1، بازخوانی و دقت بالا بهترین عملکرد را دارد و می‌تواند انتخاب مناسبی برای این مسئله باشد و مدل Naive Bayes عملکرد ضعیفی دارد و نمی‌تواند به خوبی دسته‌بندی‌ها را شناسایی کند، اما دقت بالای Precision نشان می‌دهد که پیش‌بینی‌های صحیح آن اغلب دقیق هستند و مدل SVM نیز عملکرد خوبی دارد و می‌تواند به عنوان یک مدل جایگزین برای Random Forest استفاده شود.