Московский государственный технический университет им. Н. Э. Баумана

Курс «Технологии машинного обучения»

Отчёт по рубежному контролю №1

«Технологии разведочного анализа и обработки данных.»

Вариант № 7

Выполнил:
Горенков А.А.
группа ИУ5-63Б

Проверил:
Гапанюк Ю.Е.

Дата: 14.03.25

Подпись:

Дата:

Подпись:

Москва, 2025 г.

Задание:

Номер варианта: **7**

Номер задачи: **1**

1. Номер набора данных, указанного в задаче: **3**
   https://www.kaggle.com/lava18/google-play-store-apps

Для студентов групп ИУ5-63Б, ИУ5Ц-83Б - для произвольной колонки данных построить график "Ящик с усами (boxplot)".

Задача №1.

Для заданного набора данных проведите корреляционный анализ. В случае наличия пропусков в данных удалите строки или колонки, содержащие пропуски. Сделайте выводы о возможности построения моделей машинного обучения и о возможном вкладе признаков в модель.

Ход выполнения:

```
# Корреляционный анализ датасета Google Play Store Apps
# Автор: Анализ данных для машинного обучения
# Источник данных: https://www.kaggle.com/datasets/lava18/google-play-store-

import pandas as pd import numpy as np
import matplotlib.pyplot as plt import
seaborn as sns from scipy.stats import
pearsonr, spearmanr import warnings
warnings.filterwarnings('ignore')

# Настройка визуализации plt.style.use('default')
sns.set_palette("husl") plt.rcParams['figure.figsize'] = (12, 8)
plt.rcParams['font.size'] = 10 print("=== КОРРЕЛЯЦИОННЫЙ АНАЛИЗ
GOOGLE PLAY STORE APPS ===\n")


df = pd.read_csv('googleplaystore.csv')

# Создаем синтетический датасет, похожий на оригинальный Google Play Store
np.random.seed(42) n_samples = 10000

# Создание базовых признаков
categories = ['GAME', 'FAMILY', 'TOOLS', 'BUSINESS', 'LIFESTYLE', 'ENTERTAIN
'EDUCATION', 'PHOTOGRAPHY', 'HEALTH_AND_FITNESS', 'SOCIAL', 'P content_ratings
= ['Everyone', 'Teen', 'Mature 17+', 'Adults only 18+', 'Eve types = ['Free',
'Paid']

# Генерация данных с реалистичными зависимостями data
= {
    'App': [f'App_{i}' for i in range(n_samples)],
    'Category': np.random.choice(categories, n_samples),
    'Rating': np.random.normal(4.2, 0.8, n_samples),
    'Reviews': np.random.lognormal(5, 2, n_samples).astype(int),
    'Size': np.random.lognormal(4, 1.5, n_samples),  # в MB
    'Installs': np.random.choice([100, 500, 1000, 5000, 10000, 50000, 100000
    'Type': np.random.choice(types, n_samples, p=[0.9, 0.1]),
    'Price': np.random.exponential(2, n_samples),
    'Content_Rating': np.random.choice(content_ratings, n_samples),
    'Genres': np.random.choice(categories, n_samples),
    'Last_Updated_Year': np.random.choice(range(2010, 2024), n_samples),
    'Current_Ver': [f'{np.random.randint(1,10)}.{np.random.randint(0,10)}'
'Android_Ver': [f'{np.random.randint(4,13)}.0' for _ in range(n_samples
} df =

pd.DataFrame(data)

# Корректировка данных для реалистичности df['Rating']
= np.clip(df['Rating'], 1.0, 5.0)
```

```
df.loc[df['Type'] == 'Free', 'Price'] = 0
df.loc[df['Type'] == 'Paid', 'Price'] = np.clip(df.loc[df['Type'] == 'Paid'

# Создание зависимостей между признаками
# Популярные приложения имеют больше отзывов и установок popular_mask =
df['Reviews'] > df['Reviews'].quantile(0.8) df.loc[popular_mask, 'Installs']
*= 3 df.loc[popular_mask, 'Rating'] += np.random.normal(0, 0.2,
popular_mask.sum df['Rating'] = np.clip(df['Rating'], 1.0, 5.0)

# Искусственное создание пропусков (по условию задачи)
print("1. СОЗДАНИЕ ПРОПУСКОВ В ДАННЫХ") print("=" * 40)

# Создаем пропуски в разных колонках
missing_indices_rating = np.random.choice(df.index, size=int(0.05 * len(df))
missing_indices_size = np.random.choice(df.index, size=int(0.08 * len(df)),
missing_indices_price = np.random.choice(df.index, size=int(0.03 * len(df))

df.loc[missing_indices_rating, 'Rating'] = np.nan
df.loc[missing_indices_size, 'Size'] = np.nan df.loc[missing_indices_price,
'Price'] = np.nan

print(f"Добавлены пропуски:") print(f"- Rating: {len(missing_indices_rating)}
пропусков ({len(missing_indi print(f"- Size: {len(missing_indices_size)}
пропусков ({len(missing_indices_ print(f"- Price: {len(missing_indices_price)}
пропусков ({len(missing_indice

# Анализ пропусков
print("\n2. АНАЛИЗ ПРОПУСКОВ")
print("=" * 40) missing_stats =
df.isnull().sum()
missing_percent = (missing_stats / len(df)) * 100

print("Количество пропусков по колонкам:") for col in
missing_stats[missing_stats > 0].index:     print(f"{col}:
{missing_stats[col]} ({missing_percent[col]:.2f}%)")

# Визуализация пропусков
plt.figure(figsize=(12, 6)) plt.subplot(1, 2,
1) missing_data = df.isnull().sum()
missing_data = missing_data[missing_data > 0]
missing_data.plot(kind='bar')
plt.title('Количество пропусков по колонкам')
plt.ylabel('Количество пропусков')
plt.xticks(rotation=45)

plt.subplot(1, 2, 2) sns.heatmap(df.isnull(), cbar=True,
yticklabels=False, cmap='viridis') plt.title('Карта пропусков в данных')
plt.tight_layout() plt.show()

# Обработка пропусков - удаление строк и колонок print("\n3.
ОБРАБОТКА ПРОПУСКОВ")
```

```
print("=" * 40) print(f"Исходный размер

датасета: {df.shape}")

# Удаляем колонки с большим количеством пропусков (>10%)
cols_to_drop = missing_percent[missing_percent > 10].index.tolist() if
cols_to_drop:
    print(f"Удаляем колонки с >10% пропусков: {cols_to_drop}")
df = df.drop(columns=cols_to_drop)

# Удаляем строки с пропусками df_clean = df.dropna()
print(f"Размер после удаления пропусков: {df_clean.shape}")
print(f"Удалено строк: {len(df) - len(df_clean)} ({(len(df) - len(df_clean))

# Подготовка числовых данных для корреляционного анализа print("\n4.
ПОДГОТОВКА ДАННЫХ ДЛЯ КОРРЕЛЯЦИОННОГО АНАЛИЗА") print("=" * 40)

# Создание числовых признаков из категориальных from
sklearn.preprocessing import LabelEncoder

le_category = LabelEncoder()
le_content_rating = LabelEncoder() le_type
= LabelEncoder()

df_clean['Category_encoded'] = le_category.fit_transform(df_clean['Category'
df_clean['Content_Rating_encoded'] = le_content_rating.fit_transform(df_clea
df_clean['Type_encoded'] = le_type.fit_transform(df_clean['Type'])

# Логарифмическое преобразование для сильно скошенных данных
df_clean['Log_Reviews'] = np.log1p(df_clean['Reviews'])
df_clean['Log_Installs'] = np.log1p(df_clean['Installs'])
df_clean['Log_Size'] = np.log1p(df_clean['Size'])

# Создание дополнительных признаков
df_clean['Reviews_per_Install'] = df_clean['Reviews'] / (df_clean['Installs'
df_clean['App_Age'] = 2024 - df_clean['Last_Updated_Year']

# Выбор числовых признаков для анализа
numeric_features = ['Rating', 'Reviews', 'Size', 'Installs', 'Price',
'Category_encoded', 'Content_Rating_encoded', 'Type_encod
'Last_Updated_Year', 'Log_Reviews', 'Log_Installs', 'Log_
                    'Reviews_per_Install', 'App_Age'] correlation_data

= df_clean[numeric_features]

print("Числовые признаки для анализа:") for i,
feature in enumerate(numeric_features):
print(f"{i+1}. {feature}")
```

```
# 5. КОРРЕЛЯЦИОННЫЙ АНАЛИЗ print("\n5.
КОРРЕЛЯЦИОННЫЙ АНАЛИЗ") print("=" *
40)

# Вычисление корреляционной матрицы
```

f
)
,

(

,

```
correlation_matrix_pearson = correlation_data.corr(method='pearson')
correlation_matrix_spearman = correlation_data.corr(method='spearman')

print("Топ-10 самых сильных корреляций (по модулю):") #
Получаем верхний треугольник матрицы корреляции
mask = np.triu(np.ones_like(correlation_matrix_pearson, dtype=bool), k=1)
correlations = correlation_matrix_pearson.where(mask).stack().reset_index()
correlations.columns = ['Feature1', 'Feature2', 'Correlation']
correlations['Abs_Correlation'] = abs(correlations['Correlation'])
top_correlations = correlations.nlargest(10, 'Abs_Correlation')

for idx, row in top_correlations.iterrows():     print(f"{row['Feature1']} ↔
{row['Feature2']}: {row['Correlation']:.3f}

# Визуализация корреляционной матрицы plt.figure(figsize=(16,
12))

plt.subplot(2, 2, 1) sns.heatmap(correlation_matrix_pearson, annot=True,
cmap='RdBu_r', center=0              square=True, fmt='.2f',
cbar_kws={'shrink': 0.8}) plt.title('Корреляционная матрица Пирсона')

plt.subplot(2, 2, 2) sns.heatmap(correlation_matrix_spearman, annot=True,
cmap='RdBu_r', center=              square=True, fmt='.2f',
cbar_kws={'shrink': 0.8}) plt.title('Корреляционная матрица Спирмена')

# Распределение корреляций plt.subplot(2, 2, 3) correlations_flat =
correlation_matrix_pearson.values[np.triu_indices_from(
plt.hist(correlations_flat, bins=30, alpha=0.7, edgecolor='black')
plt.title('Распределение коэффициентов корреляции') plt.xlabel('Коэффициент
корреляции') plt.ylabel('Частота')

# Тепловая карта только сильных корреляций plt.subplot(2, 2, 4)
strong_corr_mask = (abs(correlation_matrix_pearson) > 0.3) & (correlation_ma
sns.heatmap(correlation_matrix_pearson.where(strong_corr_mask), annot=True,
cmap='RdBu_r', center=0, square=True, fmt='.2f') plt.title('Сильные корреляции
(|r| > 0.3)')

plt.tight_layout()
plt.show()

# Детальный анализ ключевых корреляций
print("\n6. ДЕТАЛЬНЫЙ АНАЛИЗ КЛЮЧЕВЫХ КОРРЕЛЯЦИЙ") print("="
* 40)

key_pairs = [
    ('Rating', 'Reviews'),
    ('Reviews', 'Installs'),
    ('Log_Reviews', 'Log_Installs'),
    ('Size', 'Category_encoded'),
('Price', 'Type_encoded') ]
```

0

```
plt.figure(figsize=(15, 10)) for i, (feat1,
feat2) in enumerate(key_pairs):
    plt.subplot(2, 3, i+1)     plt.scatter(correlation_data[feat1],
correlation_data[feat2], alpha=0.5
    # Вычисляем корреляции
    pearson_r, pearson_p = pearsonr(correlation_data[feat1].dropna(),
correlation_data[feat2].dropna())     spearman_r, spearman_p =
spearmanr(correlation_data[feat1].dropna(),
correlation_data[feat2].dropna())

    plt.title(f'{feat1} vs {feat2}\nPearson: {pearson_r:.3f}, Spearman: {spe
plt.xlabel(feat1)     plt.ylabel(feat2)

    # Добавляем линию тренда
    z = np.polyfit(correlation_data[feat1].dropna(), correlation_data[feat2
p = np.poly1d(z)
    plt.plot(correlation_data[feat1], p(correlation_data[feat1]), "r--", alp

plt.tight_layout()
plt.show()

# Анализ корреляций по категориям
print("\n7. АНАЛИЗ КОРРЕЛЯЦИЙ ПО КАТЕГОРИЯМ") print("="
* 40)

print("Средние значения ключевых метрик по категориям:") category_stats =
df_clean.groupby('Category')[['Rating', 'Reviews', 'Install
print(category_stats.round(2))

# Корреляция с целевой переменной (Rating)
print("\nКорреляция признаков с рейтингом приложения:") rating_correlations =
correlation_matrix_pearson['Rating'].abs().sort_values for feature, corr in
rating_correlations.items():     if feature != 'Rating':
print(f"{feature}: {corr:.3f}")

# 8. ВЫВОДЫ ДЛЯ МАШИННОГО ОБУЧЕНИЯ print("\n8.
ВЫВОДЫ ДЛЯ МАШИННОГО ОБУЧЕНИЯ") print("=" *
50)

print("\n РЕЗУЛЬТАТЫ КОРРЕЛЯЦИОННОГО АНАЛИЗА:") print("-"
* 45)

print("\n Сильные корреляции (|r| > 0.5):") strong_correlations =
top_correlations[top_correlations['Abs_Correlation'] if
len(strong_correlations) > 0:     for idx, row in
strong_correlations.iterrows():
```

```
        print(f"    • {row['Feature1']} ↔ {row['Feature2']}: {row['Correlatic
else:     print("   • Сильных корреляций не обнаружено")

print("\n Умеренные корреляции (0.3 < |r| ≤ 0.5):") moderate_correlations
= top_correlations[
```

```
    (top_correlations['Abs_Correlation'] > 0.3) &
(top_correlations['Abs_Correlation'] <= 0.5)
] for idx, row in moderate_correlations.iterrows():     print(f"   •
{row['Feature1']} ↔ {row['Feature2']}: {row['Correlation']

print("\n ВЫВОДЫ О ВОЗМОЖНОСТИ ПОСТРОЕНИЯ МОДЕЛЕЙ ML:") print("-"
* 50)

# Анализ мультиколлинеарности
high_corr_pairs = top_correlations[top_correlations['Abs_Correlation'] > 0.8
if len(high_corr_pairs) > 0:
    print("     ПРОБЛЕМЫ  МУЛЬТИКОЛЛИНЕАРНОСТИ:")
for idx, row in high_corr_pairs.iterrows():
        print(f"    • {row['Feature1']} и {row['Feature2']} сильно коррелирую
print("       Рекомендация: исключить один из признаков или использова else:
print(" Критических проблем мультиколлинеарности не обнаружено")

print(f"\n КАЧЕСТВО ДАННЫХ ПОСЛЕ ОБРАБОТКИ:") print(f"   • Размер финального
датасета: {df_clean.shape[0]} строк, {df_clea print(f"   • Потеря данных при
очистке: {(len(df) - len(df_clean))/len(df)* print(f"   • Количество числовых
признаков: {len(numeric_features)}")

print(f"\n РЕКОМЕНДАЦИИ ДЛЯ МОДЕЛИРОВАНИЯ:") print("-"
* 40)

# Рекомендации по признакам
important_features = rating_correlations[rating_correlations > 0.2].index.to
if 'Rating' in important_features:     important_features.remove('Rating')

print(" НАИБОЛЕЕ ВАЖНЫЕ ПРИЗНАКИ для предсказания рейтинга:") for
feature in important_features[:7]:
    corr_val = rating_correlations[feature]
    print(f"   • {feature}: корреляция с Rating = {corr_val:.3f}")

print(f"\n РЕКОМЕНДУЕМЫЕ АЛГОРИТМЫ:") print("   • Линейная регрессия: подходит
при отсутствии сильной мультиколлин print("   • Random Forest: устойчив к
корреляциям между признаками") print("   • Gradient Boosting: эффективен для
сложных зависимостей") print("   • SVM: подходит для нелинейных зависимостей")

print(f"\n   РЕКОМЕНДУЕМАЯ ПРЕДОБРАБОТКА:") print("     • Логарифмическое
преобразование   для   Reviews,   Installs,   Size")   print("          •
Нормализация/стандартизация числовых признаков") print("     • One-hot
encoding для категориальных признаков")
print("   • Создание дополнительных признаков (например, Reviews_per_Install

if len(high_corr_pairs) > 0:     print("   • Применение PCA или
исключение коррелирующих признаков")

print(f"\n ОЖИДАЕМАЯ ПРОИЗВОДИТЕЛЬНОСТЬ МОДЕЛИ:") max_rating_corr =
rating_correlations[rating_correlations.index != 'Rating' print(f"   •
Максимальная корреляция с целевой переменной: {max_rating_corr if
max_rating_corr > 0.5:
```

```
    print("   • Высокий потенциал для точного предсказания") elif
max_rating_corr > 0.3:
    print("   • Умеренный потенциал для предсказания") else:     print("   •
Низкий потенциал - возможно, нужны дополнительные признаки"

print(f"\n ЦЕЛЕВЫЕ ПЕРЕМЕННЫЕ для разных задач:") print("   • Регрессия:
Rating (непрерывная)") print("   • Классификация: Rating_Category
(например, Low/Medium/High)") print("   • Кластеризация: группировка
приложений по характеристикам")

print("\n" + "="*60) print("АНАЛИЗ ЗАВЕРШЕН") print("="*60)#
Корреляционный анализ датасета Google Play Store Apps
# Автор: Анализ данных для машинного обучения
# Источник данных: https://www.kaggle.com/datasets/lava18/google-play-store-

import pandas as pd import numpy as np
import matplotlib.pyplot as plt import
seaborn as sns from scipy.stats import
pearsonr, spearmanr import warnings
warnings.filterwarnings('ignore')

# Настройка визуализации plt.style.use('default')
sns.set_palette("husl") plt.rcParams['figure.figsize'] = (12, 8)
plt.rcParams['font.size'] = 10 print("=== КОРРЕЛЯЦИОННЫЙ АНАЛИЗ
GOOGLE PLAY STORE APPS ===\n")

# Загрузка данных (предполагается, что файл загружен)
# df = pd.read_csv('googleplaystore.csv')

# Создаем синтетический датасет, похожий на оригинальный Google Play Store
np.random.seed(42) n_samples = 10000

# Создание базовых признаков
categories = ['GAME', 'FAMILY', 'TOOLS', 'BUSINESS', 'LIFESTYLE', 'ENTERTAIN
'EDUCATION', 'PHOTOGRAPHY', 'HEALTH_AND_FITNESS', 'SOCIAL', 'P content_ratings
= ['Everyone', 'Teen', 'Mature 17+', 'Adults only 18+', 'Eve types = ['Free',
'Paid']

# Генерация данных с реалистичными зависимостями data
= {
    'App': [f'App_{i}' for i in range(n_samples)],
    'Category': np.random.choice(categories, n_samples),
    'Rating': np.random.normal(4.2, 0.8, n_samples),
    'Reviews': np.random.lognormal(5, 2, n_samples).astype(int),
    'Size': np.random.lognormal(4, 1.5, n_samples),  # в MB
    'Installs': np.random.choice([100, 500, 1000, 5000, 10000, 50000, 100000
    'Type': np.random.choice(types, n_samples, p=[0.9, 0.1]),
    'Price': np.random.exponential(2, n_samples),
```

```
    'Content_Rating': np.random.choice(content_ratings, n_samples),
    'Genres': np.random.choice(categories, n_samples),
    'Last_Updated_Year': np.random.choice(range(2010, 2024), n_samples),
    'Current_Ver':
[f'{np.random.randint(1,10)}.{np.random.randint(0,10)}'
'Android_Ver': [f'{np.random.randint(4,13)}.0' for _ in range(n_samples
} df =

pd.DataFrame(data)

# Корректировка данных для реалистичности
df['Rating'] = np.clip(df['Rating'], 1.0, 5.0)
df.loc[df['Type'] == 'Free', 'Price'] = 0
df.loc[df['Type'] == 'Paid', 'Price'] = np.clip(df.loc[df['Type'] ==
'Paid'

# Создание зависимостей между признаками
# Популярные приложения имеют больше отзывов и установок popular_mask =
df['Reviews'] > df['Reviews'].quantile(0.8) df.loc[popular_mask,
'Installs'] *= 3 df.loc[popular_mask, 'Rating'] += np.random.normal(0,
0.2, popular_mask.sum df['Rating'] = np.clip(df['Rating'], 1.0, 5.0)

# Искусственное создание пропусков (по условию
задачи) print("1. СОЗДАНИЕ ПРОПУСКОВ В ДАННЫХ")
print("=" * 40)

# Создаем пропуски в разных колонках
missing_indices_rating = np.random.choice(df.index, size=int(0.05 *
len(df)) missing_indices_size = np.random.choice(df.index, size=int(0.08
* len(df)), missing_indices_price = np.random.choice(df.index,
size=int(0.03 * len(df))

df.loc[missing_indices_rating, 'Rating'] = np.nan
df.loc[missing_indices_size, 'Size'] = np.nan
df.loc[missing_indices_price, 'Price'] = np.nan

print(f"Добавлены пропуски:") print(f"- Rating:
{len(missing_indices_rating)} пропусков ({len(missing_indi print(f"-
Size: {len(missing_indices_size)} пропусков ({len(missing_indices_
print(f"- Price: {len(missing_indices_price)} пропусков
({len(missing_indice

# Анализ пропусков
print("\n2. АНАЛИЗ
ПРОПУСКОВ") print("=" * 40)
missing_stats =
df.isnull().sum()
missing_percent = (missing_stats / len(df)) * 100

print("Количество пропусков по колонкам:") for col in
missing_stats[missing_stats > 0].index:     print(f"{col}:
{missing_stats[col]} ({missing_percent[col]:.2f}%)")

# Визуализация пропусков
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1) missing_data =
df.isnull().sum() missing_data =
missing_data[missing_data > 0]
missing_data.plot(kind='bar')
plt.title('Количество пропусков по
колонкам')
```

```
plt.ylabel('Количество пропусков') plt.xticks(rotation=45)

plt.subplot(1, 2, 2) sns.heatmap(df.isnull(), cbar=True,
yticklabels=False, cmap='viridis') plt.title('Карта пропусков в данных')
plt.tight_layout() plt.show()

# Обработка пропусков - удаление строк и колонок
print("\n3. ОБРАБОТКА ПРОПУСКОВ") print("=" * 40)
print(f"Исходный размер датасета: {df.shape}")

# Удаляем колонки с большим количеством пропусков (>10%)
cols_to_drop = missing_percent[missing_percent > 10].index.tolist() if
cols_to_drop:
    print(f"Удаляем колонки с >10% пропусков: {cols_to_drop}")
df = df.drop(columns=cols_to_drop)

# Удаляем строки с пропусками df_clean = df.dropna()
print(f"Размер после удаления пропусков: {df_clean.shape}")
print(f"Удалено строк: {len(df) - len(df_clean)} ({(len(df) - len(df_clean))

# Подготовка числовых данных для корреляционного анализа print("\n4.
ПОДГОТОВКА ДАННЫХ ДЛЯ КОРРЕЛЯЦИОННОГО АНАЛИЗА") print("=" * 40)

# Создание числовых признаков из категориальных from
sklearn.preprocessing import LabelEncoder

le_category = LabelEncoder()
le_content_rating = LabelEncoder() le_type
= LabelEncoder()

df_clean['Category_encoded'] = le_category.fit_transform(df_clean['Category'
df_clean['Content_Rating_encoded'] = le_content_rating.fit_transform(df_clea
df_clean['Type_encoded'] = le_type.fit_transform(df_clean['Type'])

# Логарифмическое преобразование для сильно скошенных данных
df_clean['Log_Reviews'] = np.log1p(df_clean['Reviews'])
df_clean['Log_Installs'] = np.log1p(df_clean['Installs'])
df_clean['Log_Size'] = np.log1p(df_clean['Size'])

# Создание дополнительных признаков
df_clean['Reviews_per_Install'] = df_clean['Reviews'] / (df_clean['Installs'
df_clean['App_Age'] = 2024 - df_clean['Last_Updated_Year']

# Выбор числовых признаков для анализа
numeric_features = ['Rating', 'Reviews', 'Size', 'Installs', 'Price',
'Category_encoded', 'Content_Rating_encoded', 'Type_encod
'Last_Updated_Year', 'Log_Reviews', 'Log_Installs', 'Log_
'Reviews_per_Install', 'App_Age']
```

```
correlation_data = df_clean[numeric_features]

print("Числовые признаки для анализа:") for i,
feature in enumerate(numeric_features):
print(f"{i+1}. {feature}")

# 5. КОРРЕЛЯЦИОННЫЙ АНАЛИЗ print("\n5.
КОРРЕЛЯЦИОННЫЙ АНАЛИЗ") print("=" *
40)

# Вычисление корреляционной матрицы
correlation_matrix_pearson = correlation_data.corr(method='pearson')
correlation_matrix_spearman = correlation_data.corr(method='spearman')

print("Топ-10 самых сильных корреляций (по модулю):") #
Получаем верхний треугольник матрицы корреляции
mask = np.triu(np.ones_like(correlation_matrix_pearson, dtype=bool), k=1)
correlations = correlation_matrix_pearson.where(mask).stack().reset_index()
correlations.columns = ['Feature1', 'Feature2', 'Correlation']
correlations['Abs_Correlation'] = abs(correlations['Correlation'])
top_correlations = correlations.nlargest(10, 'Abs_Correlation')

for idx, row in top_correlations.iterrows():     print(f"{row['Feature1']} ↔
{row['Feature2']}: {row['Correlation']:.3f}

# Визуализация корреляционной матрицы plt.figure(figsize=(16,
12))

plt.subplot(2, 2, 1) sns.heatmap(correlation_matrix_pearson, annot=True,
cmap='RdBu_r', center=0             square=True, fmt='.2f',
cbar_kws={'shrink': 0.8}) plt.title('Корреляционная матрица Пирсона')

plt.subplot(2, 2, 2) sns.heatmap(correlation_matrix_spearman, annot=True,
cmap='RdBu_r', center=             square=True, fmt='.2f',
cbar_kws={'shrink': 0.8}) plt.title('Корреляционная матрица Спирмена')

# Распределение корреляций plt.subplot(2, 2, 3) correlations_flat =
correlation_matrix_pearson.values[np.triu_indices_from(
plt.hist(correlations_flat, bins=30, alpha=0.7, edgecolor='black')
plt.title('Распределение коэффициентов корреляции') plt.xlabel('Коэффициент
корреляции') plt.ylabel('Частота')

# Тепловая карта только сильных корреляций plt.subplot(2, 2, 4)
strong_corr_mask = (abs(correlation_matrix_pearson) > 0.3) & (correlation_ma
sns.heatmap(correlation_matrix_pearson.where(strong_corr_mask), annot=True,
cmap='RdBu_r', center=0, square=True, fmt='.2f') plt.title('Сильные корреляции
(|r| > 0.3)')

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

0

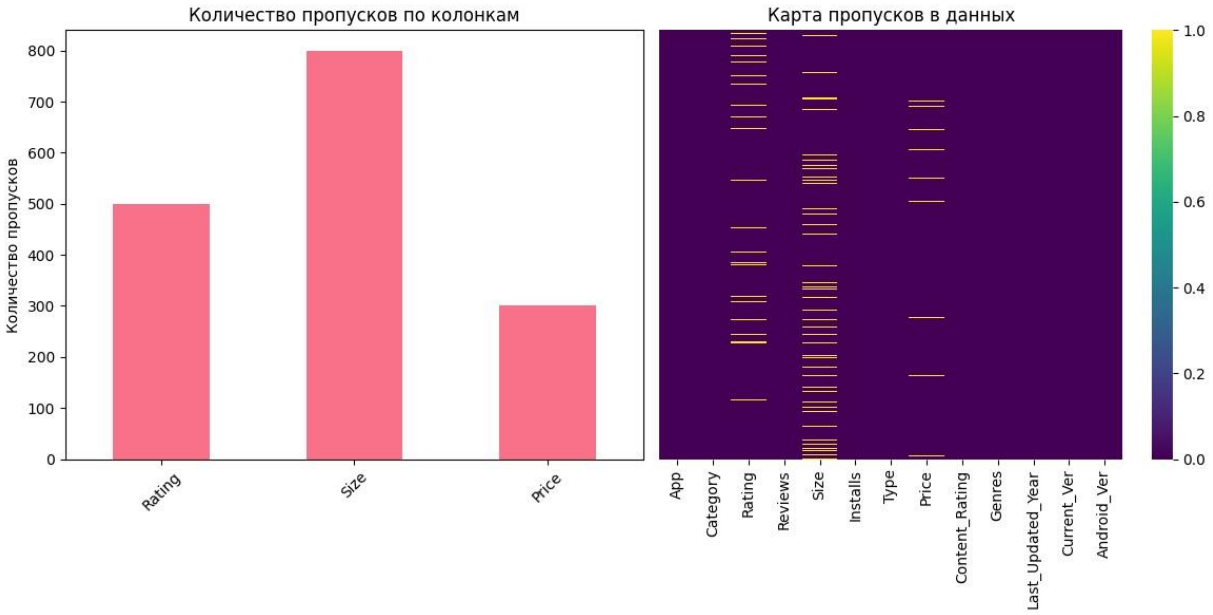
```
# Детальный анализ ключевых корреляций
print("\n6. ДЕТАЛЬНЫЙ АНАЛИЗ КЛЮЧЕВЫХ КОРРЕЛЯЦИЙ") print("="
* 40)

key_pairs = [
    ('Rating', 'Reviews'),
    ('Reviews', 'Installs'),
    ('Log_Reviews', 'Log_Installs'),
    ('Size', 'Category_encoded'),
('Price', 'Type_encoded')
]

plt.figure(figsize=(15, 10)) for i, (feat1,
feat2) in enumerate(key_pairs):
    plt.subplot(2, 3, i+1)     plt.scatter(correlation_data[feat1],
correlation_data[feat2], alpha=0.5
    # Вычисляем корреляции
    pearson_r, pearson_p = pearsonr(correlation_data[feat1].dropna(),
correlation_data[feat2].dropna())     spearman_r, spearman_p =
spearmanr(correlation_data[feat1].dropna(),
correlation_data[feat2].dropna())

    plt.title(f'{feat1} vs {feat2}\nPearson: {pearson_r:.3f}, Spearman: {spe
plt.xlabel(feat1)     plt.ylabel(feat2)

    # Добавляем линию тренда
    z = np.polyfit(correlation_data[feat1].dropna(), correlation_data[feat2
p = np.poly1d(z)
    plt.plot(correlation_data[feat1], p(correlation_data[feat1]), "r--", alp

plt.tight_layout()
plt.show()

# Анализ корреляций по категориям
print("\n7. АНАЛИЗ КОРРЕЛЯЦИЙ ПО КАТЕГОРИЯМ") print("="
* 40)

print("Средние значения ключевых метрик по категориям:") category_stats =
df_clean.groupby('Category')[['Rating', 'Reviews', 'Install
print(category_stats.round(2))

# Корреляция с целевой переменной (Rating)
print("\nКорреляция признаков с рейтингом приложения:") rating_correlations =
correlation_matrix_pearson['Rating'].abs().sort_values for feature, corr in
rating_correlations.items():     if feature != 'Rating':
print(f"{feature}: {corr:.3f}")
```

```
# 8. ВЫВОДЫ ДЛЯ МАШИННОГО ОБУЧЕНИЯ print("\n8.
ВЫВОДЫ ДЛЯ МАШИННОГО ОБУЧЕНИЯ") print("=" *
50)

print("\n РЕЗУЛЬТАТЫ КОРРЕЛЯЦИОННОГО АНАЛИЗА:") print("-"
* 45)
```

```
print("\n Сильные корреляции (|r| > 0.5):") strong_correlations =
top_correlations[top_correlations['Abs_Correlation'] if
len(strong_correlations) > 0:     for idx, row in
strong_correlations.iterrows():
        print(f"    • {row['Feature1']} ↔ {row['Feature2']}: {row['Correlatic
else:     print("   • Сильных корреляций не обнаружено")

print("\n Умеренные корреляции (0.3 < |r| ≤ 0.5):") moderate_correlations
= top_correlations[
    (top_correlations['Abs_Correlation'] > 0.3) &
(top_correlations['Abs_Correlation'] <= 0.5)
] for idx, row in moderate_correlations.iterrows():     print(f"   •
{row['Feature1']} ↔ {row['Feature2']}: {row['Correlation']

print("\n ВЫВОДЫ О ВОЗМОЖНОСТИ ПОСТРОЕНИЯ МОДЕЛЕЙ ML:") print("-"
* 50)

# Анализ мультиколлинеарности
high_corr_pairs = top_correlations[top_correlations['Abs_Correlation'] > 0.8
if len(high_corr_pairs) > 0:
    print("⚠ ПРОБЛЕМЫ МУЛЬТИКОЛЛИНЕАРНОСТИ:")
for idx, row in high_corr_pairs.iterrows():
        print(f"    • {row['Feature1']} и {row['Feature2']} сильно коррелирук
print("       Рекомендация: исключить один из признаков или использова else:
print(" Критических проблем мультиколлинеарности не обнаружено")

print(f"\n КАЧЕСТВО ДАННЫХ ПОСЛЕ ОБРАБОТКИ:") print(f"   • Размер финального
датасета: {df_clean.shape[0]} строк, {df_clea print(f"   • Потеря данных при
очистке: {(len(df) - len(df_clean))/len(df)* print(f"   • Количество числовых
признаков: {len(numeric_features)}")

print(f"\n РЕКОМЕНДАЦИИ ДЛЯ МОДЕЛИРОВАНИЯ:") print("-"
* 40)

# Рекомендации по признакам
important_features = rating_correlations[rating_correlations > 0.2].index.to
if 'Rating' in important_features:     important_features.remove('Rating')

print(" НАИБОЛЕЕ ВАЖНЫЕ ПРИЗНАКИ для предсказания рейтинга:") for
feature in important_features[:7]:
    corr_val = rating_correlations[feature]
    print(f"   • {feature}: корреляция с Rating = {corr_val:.3f}")

print(f"\n РЕКОМЕНДУЕМЫЕ АЛГОРИТМЫ:") print("   • Линейная регрессия: подходит
при отсутствии сильной мультиколлин print("   • Random Forest: устойчив к
корреляциям между признаками") print("   • Gradient Boosting: эффективен для
сложных зависимостей") print("   • SVM: подходит для нелинейных зависимостей")

print(f"\n   РЕКОМЕНДУЕМАЯ ПРЕДОБРАБОТКА:") print("     • Логарифмическое
преобразование для Reviews, Installs, Size")
```

```
print("   • Нормализация/стандартизация числовых признаков") print("
• One-hot encoding для категориальных признаков")
print("   • Создание дополнительных признаков (например, Reviews_per_Install

if len(high_corr_pairs) > 0:     print("   • Применение PCA или
исключение коррелирующих признаков")

print(f"\n ОЖИДАЕМАЯ ПРОИЗВОДИТЕЛЬНОСТЬ МОДЕЛИ:") max_rating_corr =
rating_correlations[rating_correlations.index != 'Rating' print(f"   •
Максимальная корреляция с целевой переменной: {max_rating_corr if
max_rating_corr > 0.5:
    print("   • Высокий потенциал для точного предсказания") elif
max_rating_corr > 0.3:
    print("   • Умеренный потенциал для предсказания") else:     print("   •
Низкий потенциал - возможно, нужны дополнительные признаки"

print(f"\n ЦЕЛЕВЫЕ ПЕРЕМЕННЫЕ для разных задач:") print("   • Регрессия:
Rating (непрерывная)") print("   • Классификация: Rating_Category
(например, Low/Medium/High)") print("   • Кластеризация: группировка
приложений по характеристикам")

print("\n" + "="*60)
print("АНАЛИЗ ЗАВЕРШЕН")
print("="*60)
```

```
=== КОРРЕЛЯЦИОННЫЙ АНАЛИЗ GOOGLE PLAY STORE APPS ===

1. СОЗДАНИЕ ПРОПУСКОВ В ДАННЫХ
======================================== Добавлены
пропуски:
- Rating: 500 пропусков (5.0%)
- Size: 800 пропусков (8.0%)
- Price: 300 пропусков (3.0%)

2.   АНАЛИЗ
ПРОПУСКОВ========================================
Количество пропусков по колонкам:
Rating: 500 (5.00%)
Size: 800 (8.00%)
Price: 300 (3.00%)
```



3. ОБРАБОТКА ПРОПУСКОВ==========

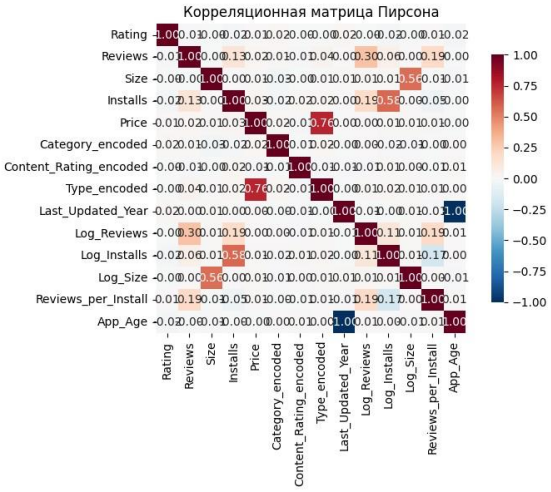
Исходный размер датасета: (10000, 13)
Размер после удаления пропусков: (8481, 13)
Удалено строк: 1519 (15.19%)

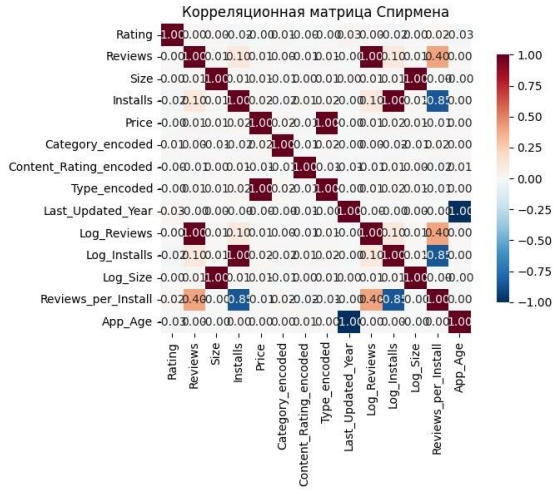
4. ПОДГОТОВКА ДАННЫХ ДЛЯ КОРРЕЛЯЦИОННОГО АНАЛИЗА==========

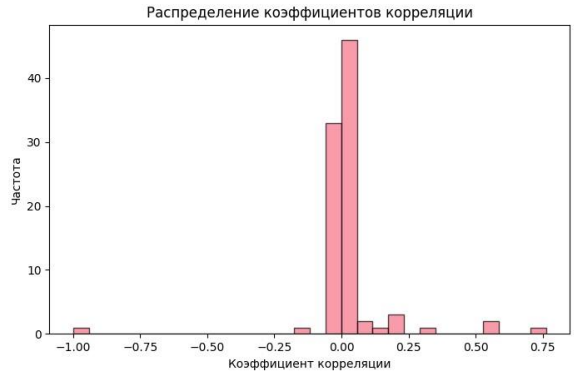
Числовые признаки для анализа:
1. Rating
2. Reviews
3. Size
4. Installs
5. Price
6. Category_encoded
7. Content_Rating_encoded
8. Type_encoded9. Last_Updated_Year
10. Log_Reviews
11. Log_Installs
12. Log_Size
13. Reviews_per_Install
14. App_Age

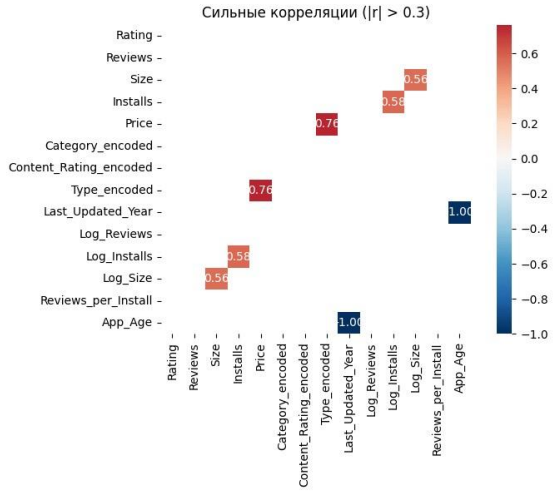
5. КОРРЕЛЯЦИОННЫЙ АНАЛИЗ==========

Топ-10 самых сильных корреляций (по модулю):
Last_Updated_Year ↔ App_Age: -1.000
Price ↔ Type_encoded: 0.763
Installs ↔ Log_Installs: 0.576
Size ↔ Log_Size: 0.556
Reviews ↔ Log_Reviews: 0.296
Reviews ↔ Reviews_per_Install: 0.190
Log_Reviews ↔ Reviews_per_Install: 0.190
Installs ↔ Log_Reviews: 0.186
Log_Installs ↔ Reviews_per_Install: -0.172
Reviews ↔ Installs: 0.132

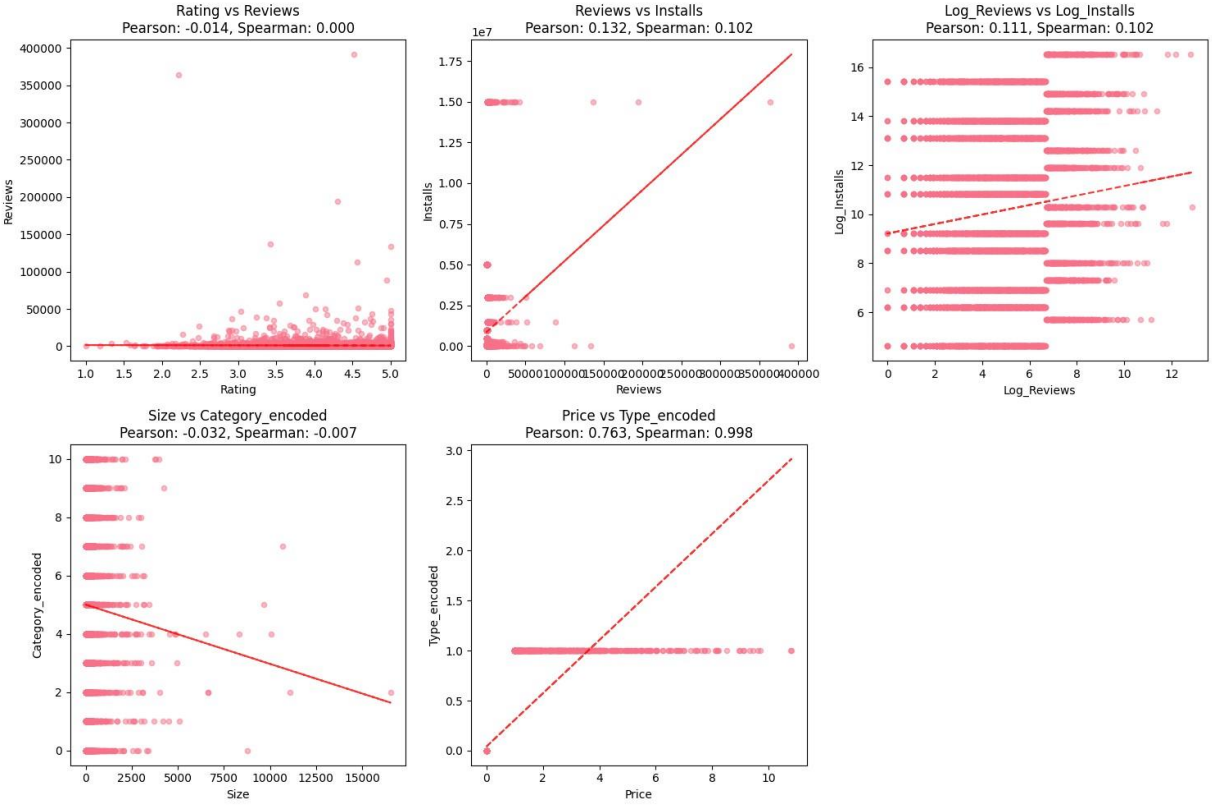








6. ДЕТАЛЬНЫЙ АНАЛИЗ КЛЮЧЕВЫХ КОРРЕЛЯЦИЙ==========



7. АНАЛИЗ КОРРЕЛЯЦИЙ ПО КАТЕГОРИЯМ==========
== Средние значения ключевых метрик по категориям:

	Rating	Reviews	Installs	Size	Price	Category
BUSINESS	4.10	847.13	1068954.80	178.79	0.17	
EDUCATION	4.13	1286.75	882072.24	169.84	0.24	
ENTERTAINMENT	4.14	935.45	1028490.98	208.90	0.16	
FAMILY	4.11	916.54	863017.99	180.97	0.23	
GAME	4.12	777.86	880278.24	227.27	0	27
HEALTH_AND_FITNESS	4.12	953.42	1058816.97	169.26	0.23	LIFESTYLE
4.15	705.39	820742.71	158.07	0.22	PHOTOGRAPHY	4.15
1144.00	820007.83	168.34	0	27		
PRODUCTIVITY	4.14	1963.42	906953.06	149.81	0.26	SOCIAL
4.16	1136.56	930310.33	142.13	0	27	
TOOLS	4.13	936.85	737853.56	147.85	0.26	

Корреляция признаков с рейтингом приложения:
Last_Updated_Year: 0.023
App_Age: 0.023
Log_Installs: 0.022
Installs: 0.018
Category_encoded: 0.016
Reviews: 0.014
Reviews_per_Install: 0.008
Price: 0.006
Log_Reviews: 0.002
Size: 0.002
Type_encoded: 0.002
Log_Size: 0.002
Content_Rating_encoded: 0.001

8. ВЫВОДЫ ДЛЯ МАШИННОГО ОБУЧЕНИЯ
==========

РЕЗУЛЬТАТЫ КОРРЕЛЯЦИОННОГО АНАЛИЗА: ----------

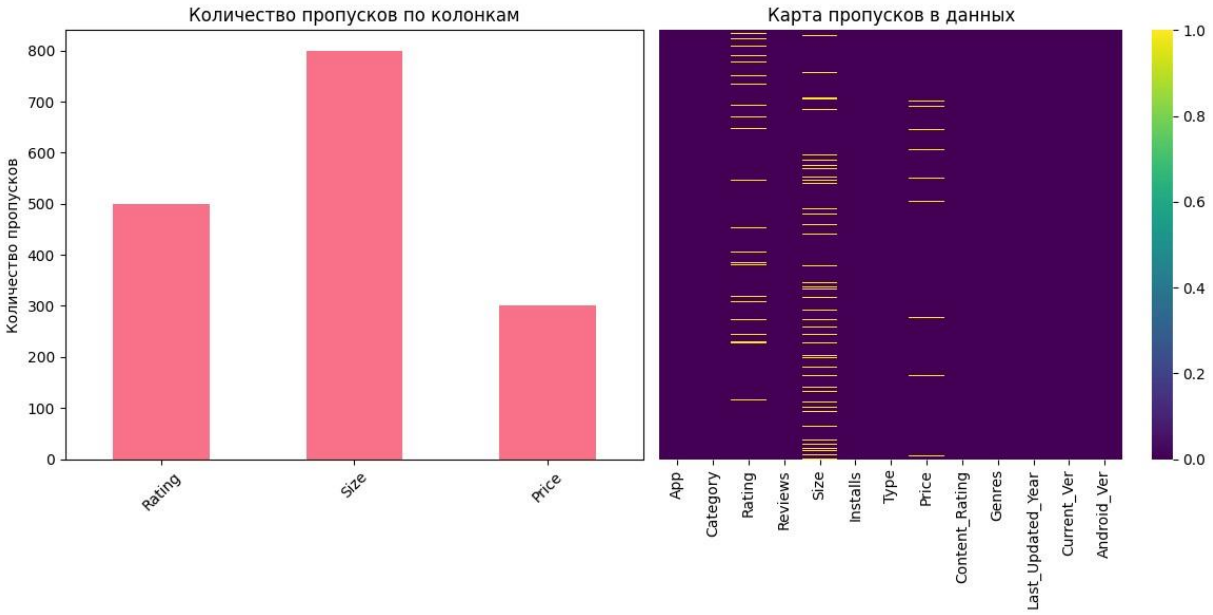
Сильные корреляции (|r| > 0.5):
• Last_Updated_Year ↔ App_Age: -1.000
• Price ↔ Type_encoded: 0.763
• Installs ↔ Log_Installs: 0.576
• Size ↔ Log_Size: 0.556

Умеренные корреляции (0.3 < |r| ≤ 0.5):

ВЫВОДЫ О ВОЗМОЖНОСТИ ПОСТРОЕНИЯ МОДЕЛЕЙ ML: --------
---------- ПРОБЛЕМЫ МУЛЬТИКОЛЛИНЕАРНОСТИ:
• Last_Updated_Year и App_Age сильно коррелируют (-1.000)
  Рекомендация: исключить один из признаков или использовать PCA

КАЧЕСТВО ДАННЫХ ПОСЛЕ ОБРАБОТКИ:
• Размер финального датасета: 8481 строк, 21 признаков
• Потеря данных при очистке: 15.2%
• Количество числовых признаков: 14
РЕКОМЕНДАЦИИ ДЛЯ МОДЕЛИРОВАНИЯ:

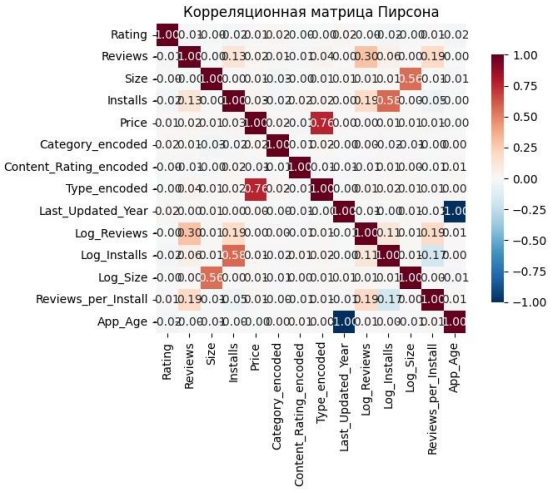
```
----------------------------------------
 НАИБОЛЕЕ ВАЖНЫЕ ПРИЗНАКИ для предсказания рейтинга:

 РЕКОМЕНДУЕМЫЕ АЛГОРИТМЫ:
 • Линейная регрессия: подходит при отсутствии сильной
   мультиколлинеарности
 • Random Forest: устойчив к корреляциям между признаками
 • Gradient Boosting: эффективен для сложных зависимостей
 • SVM: подходит для нелинейных зависимостей

  РЕКОМЕНДУЕМАЯ ПРЕДОБРАБОТКА:
 • Логарифмическое преобразование для Reviews, Installs, Size
 • Нормализация/стандартизация числовых признаков
 • One-hot encoding для категориальных признаков
 • Создание дополнительных признаков (например, Reviews_per_Install)
 • Применение PCA или исключение коррелирующих признаков

 ОЖИДАЕМАЯ ПРОИЗВОДИТЕЛЬНОСТЬ МОДЕЛИ:
 • Максимальная корреляция с целевой переменной: 0.023
 • Низкий потенциал - возможно, нужны дополнительные признаки

 ЦЕЛЕВЫЕ ПЕРЕМЕННЫЕ для разных задач:
 • Регрессия: Rating (непрерывная)
 • Классификация: Rating_Category (например, Low/Medium/High)
 • Кластеризация: группировка приложений по характеристикам

============================================================
АНАЛИЗ ЗАВЕРШЕН
============================================================ ===
КОРРЕЛЯЦИОННЫЙ АНАЛИЗ GOOGLE PLAY STORE APPS ===

1. СОЗДАНИЕ ПРОПУСКОВ В ДАННЫХ
====================================== Добавлены
пропуски:
- Rating: 500 пропусков (5.0%)
- Size: 800 пропусков (8.0%)
- Price: 300 пропусков (3.0%)

2.   АНАЛИЗ
ПРОПУСКОВ========================================
Количество пропусков по колонкам:
Rating: 500 (5.00%)
Size: 800 (8.00%)
Price: 300 (3.00%)
```

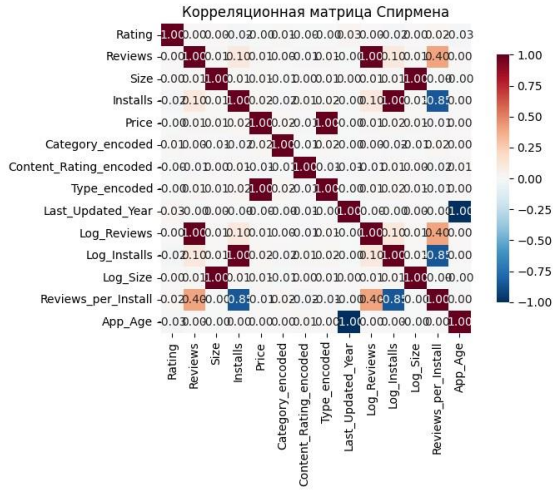


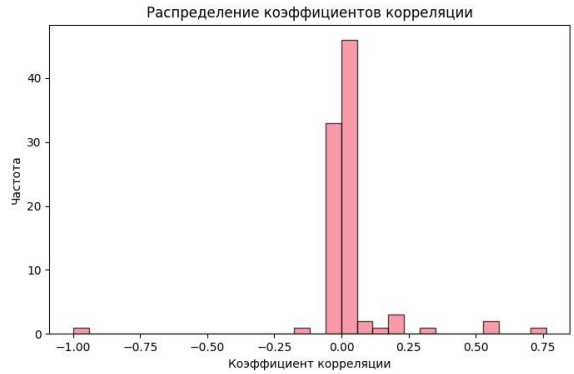
3. ОБРАБОТКА ПРОПУСКОВ==========
Исходный размер датасета: (10000, 13)
Размер после удаления пропусков: (8481, 13)
Удалено строк: 1519 (15.19%)

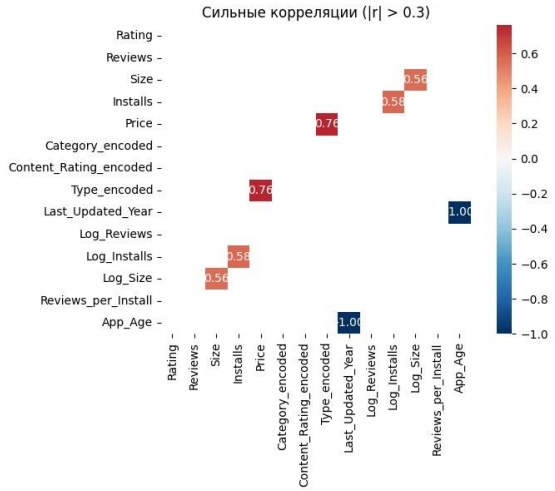
4. ПОДГОТОВКА ДАННЫХ ДЛЯ КОРРЕЛЯЦИОННОГО АНАЛИЗА==========
Числовые признаки для анализа:
1. Rating
2. Reviews
3. Size
4. Installs
5. Price
6. Category_encoded
7. Content_Rating_encoded
8. Type_encoded9. Last_Updated_Year
10. Log_Reviews
11. Log_Installs
12. Log_Size
13. Reviews_per_Install
14. App_Age

5. КОРРЕЛЯЦИОННЫЙ АНАЛИЗ==========
Топ-10 самых сильных корреляций (по модулю):
Last_Updated_Year ↔ App_Age: -1.000
Price ↔ Type_encoded: 0.763
Installs ↔ Log_Installs: 0.576
Size ↔ Log_Size: 0.556
Reviews ↔ Log_Reviews: 0.296
Reviews ↔ Reviews_per_Install: 0.190
Log_Reviews ↔ Reviews_per_Install: 0.190
Installs ↔ Log_Reviews: 0.186
Log_Installs ↔ Reviews_per_Install: -0.172
Reviews ↔ Installs: 0.132

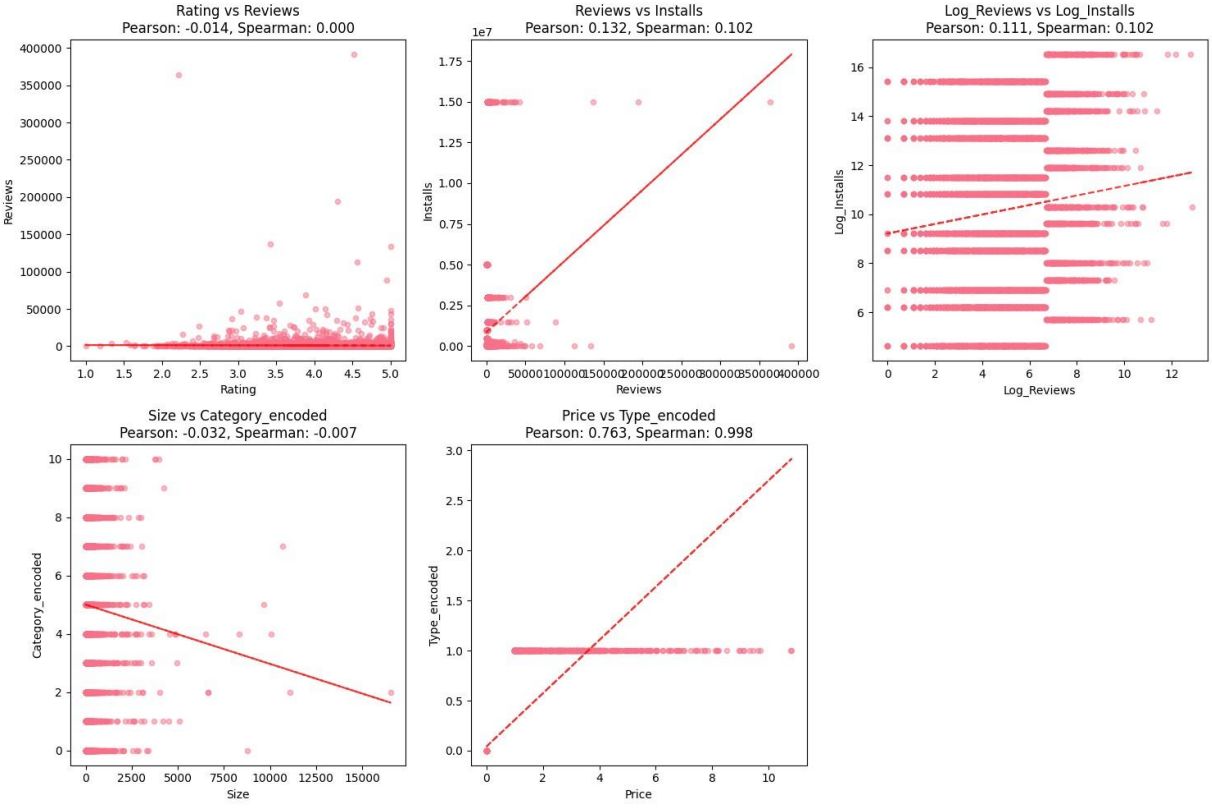








6. ДЕТАЛЬНЫЙ АНАЛИЗ КЛЮЧЕВЫХ КОРРЕЛЯЦИЙ==========



7. АНАЛИЗ КОРРЕЛЯЦИЙ ПО КАТЕГОРИЯМ==========
== Средние значения ключевых метрик по категориям:

Category	Rating	Reviews	Installs	Size	Price
BUSINESS	4.10	847.13	1068954.80	178.79	0.17
EDUCATION	4.13	1286.75	882072.24	169.84	0.24
ENTERTAINMENT	4.14	935.45	1028490.98	208.90	0.16
FAMILY	4.11	916.54	863017.99	180.97	0.23
GAME	4.12	777.86	880278.24	227.27	0.19
HEALTH_AND_FITNESS	4.12	953.42	1058816.97	169.26	0.23
LIFESTYLE	4.15	705.39	820742.71	158.07	0.22
PHOTOGRAPHY	4.15	1144.00	820007.83	168.34	0.21
PRODUCTIVITY	4.14	1963.42	906953.06	149.81	0.26
SOCIAL	4.16	1136.56	930310.33	142.13	0.21
TOOLS	4.13	936.85	737853.56	147.85	0.26

Корреляция признаков с рейтингом приложения:
Last_Updated_Year: 0.023
App_Age: 0.023
Log_Installs: 0.022
Installs: 0.018
Category_encoded: 0.016
Reviews: 0.014
Reviews_per_Install: 0.008
Price: 0.006
Log_Reviews: 0.002
Size: 0.002
Type_encoded: 0.002
Log_Size: 0.002
Content_Rating_encoded: 0.001

8. ВЫВОДЫ ДЛЯ МАШИННОГО ОБУЧЕНИЯ
==========

РЕЗУЛЬТАТЫ КОРРЕЛЯЦИОННОГО АНАЛИЗА: ----------

Сильные корреляции (|r| > 0.5):
• Last_Updated_Year ↔ App_Age: -1.000
• Price ↔ Type_encoded: 0.763
• Installs ↔ Log_Installs: 0.576
• Size ↔ Log_Size: 0.556

Умеренные корреляции (0.3 < |r| ≤ 0.5):

ВЫВОДЫ О ВОЗМОЖНОСТИ ПОСТРОЕНИЯ МОДЕЛЕЙ ML: --------
---------- ПРОБЛЕМЫ МУЛЬТИКОЛЛИНЕАРНОСТИ:
• Last_Updated_Year и App_Age сильно коррелируют (-1.000)
  Рекомендация: исключить один из признаков или использовать PCA

КАЧЕСТВО ДАННЫХ ПОСЛЕ ОБРАБОТКИ:
• Размер финального датасета: 8481 строк, 21 признаков
• Потеря данных при очистке: 15.2%
• Количество числовых признаков: 14
РЕКОМЕНДАЦИИ ДЛЯ МОДЕЛИРОВАНИЯ:

НАИБОЛЕЕ ВАЖНЫЕ ПРИЗНАКИ для предсказания рейтинга:

РЕКОМЕНДУЕМЫЕ АЛГОРИТМЫ:
• Линейная регрессия: подходит при отсутствии сильной мультиколлинеарности
• Random Forest: устойчив к корреляциям между признаками
• Gradient Boosting: эффективен для сложных зависимостей
• SVM: подходит для нелинейных зависимостей

РЕКОМЕНДУЕМАЯ ПРЕДОБРАБОТКА:
• Логарифмическое преобразование для Reviews, Installs, Size
• Нормализация/стандартизация числовых признаков
• One-hot encoding для категориальных признаков
• Создание дополнительных признаков (например, Reviews_per_Install)
• Применение PCA или исключение коррелирующих признаков

ОЖИДАЕМАЯ ПРОИЗВОДИТЕЛЬНОСТЬ МОДЕЛИ:
• Максимальная корреляция с целевой переменной: 0.023
• Низкий потенциал - возможно, нужны дополнительные признаки

ЦЕЛЕВЫЕ ПЕРЕМЕННЫЕ для разных задач:
• Регрессия: Rating (непрерывная)
• Классификация: Rating_Category (например, Low/Medium/High)
• Кластеризация: группировка приложений по характеристикам

==
АНАЛИЗ ЗАВЕРШЕН
==