



优达学城
挑战不可能

探索数据集

DAND VIP班公开课

数据分析过程



优达学城
UDACITY

示例数据集:

Mobile App Store (7200 apps)

Analytics for Mobile Apps

<https://www.kaggle.com/ramamet4/app-store-apple-data-set-10k-apps>

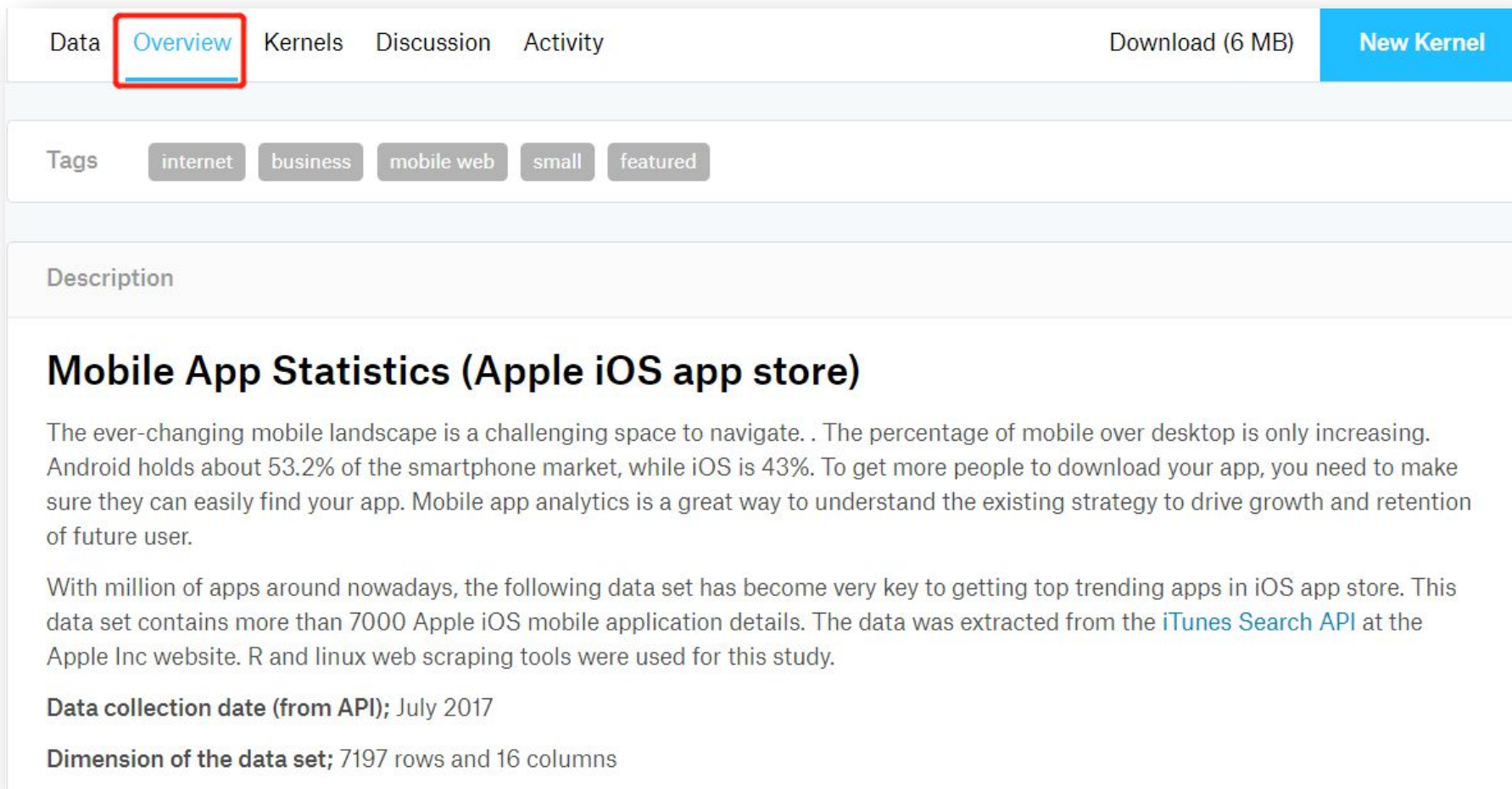


优达学城
UDACITY

1

提出问题——初步了解数据集

在数据集的出处找到相关描述：



The screenshot shows a web interface for a dataset. At the top, there are tabs: 'Data', 'Overview' (highlighted with a red box), 'Kernels', 'Discussion', and 'Activity'. To the right of these tabs are links for 'Download (6 MB)' and a blue button labeled 'New Kernel'. Below the tabs is a 'Tags' section with buttons for 'internet', 'business', 'mobile web', 'small', and 'featured'. The main content area is titled 'Description' and contains the following text:

Mobile App Statistics (Apple iOS app store)

The ever-changing mobile landscape is a challenging space to navigate. . The percentage of mobile over desktop is only increasing. Android holds about 53.2% of the smartphone market, while iOS is 43%. To get more people to download your app, you need to make sure they can easily find your app. Mobile app analytics is a great way to understand the existing strategy to drive growth and retention of future user.

With million of apps around nowadays, the following data set has become very key to getting top trending apps in iOS app store. This data set contains more than 7000 Apple iOS mobile application details. The data was extracted from the [iTunes Search API](#) at the Apple Inc website. R and linux web scraping tools were used for this study.

Data collection date (from API); July 2017

Dimension of the data set; 7197 rows and 16 columns

1 提出问题——初步了解数据集

在数据集的出处找到数据字典对数据中各列的简介：

appleStore.csv

- 1."id" : App ID
- 2."track_name" : App 名称
- 3."size_bytes": 大小 (以 Bytes 为单位)
- 4."currency": 价格货币单位
- 5."price": APP 价格
- 6."rating_count_tot": 用户评分数量 (所有版本的总计)
- 7."rating_count_ver": 用户评分数量 (当前版本)
- 8."user_rating" : 用户平均评分 (所有版本)
- 9."user_rating_ver": 用户平均评分 (当前版本)
- 10."ver" : 最新版本号
- 11."cont_rating": 内容评级
- 12."prime_genre": 主要类型
- 13."sup_devices.num": 支持设备数量
- 14."ipadSc_urls.num": 展示截图的数量
- 15."lang.num": 支持语言的数量
- 16."vpp_lic": Vpp Device Based Licensing Enabled

appleStore_description.csv

- 1.id : App ID
- 2.track_name: App 名称
- 3.size_bytes: 大小 (以 Bytes 为单位)
- 4.app_desc: APP 描述文字



1

提出问题——初步了解数据集

了解数据，最直观的方式还是观察数据本身。

接下来，我们将进入到 Jupyter Notebook 中，开始使用 Python 进行数据探索！

准备工作，导入常用库和读取数据：

```
In [1]: # 导入常用库
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
In [2]: # 读取数据集
df_desc = pd.read_csv('appleStore_description.csv')
df_app = pd.read_csv('AppleStore.csv')
```



1

提出问题——初步了解数据集

使用 head、tail 或者 sample 方法简单查看一下数据集的内容：

```
In [3]: df_desc.head(3)
```

```
Out[3]:
```

	id	track_name	size_bytes	app_desc
0	281656475	PAC-MAN Premium	100788224	SAVE 20%, now only \$3.99 for a limited time!\n...
1	281796108	Evernote - stay organized	158578688	Let Evernote change the way you organize your ...
2	281940292	WeatherBug - Local Weather, Radar, Maps, Alerts	100524032	Download the most popular free weather app pow...

```
In [4]: df_app.head(3)
```

```
Out[4]:
```

	Unnamed: 0	id	track_name	size_bytes	currency	price	rating_count_tot	rating_coun
0	1	281656475	PAC-MAN Premium	100788224	USD	3.99	21292	
1	2	281796108	Evernote - stay organized	158578688	USD	0.00	161065	
2	3	281940292	WeatherBug - Local Weather, Radar, Maps, Alerts	100524032	USD	0.00	188583	



1

提出问题——根据初步了解提出问题

在初步了解数据集的含义，以及其中所包含的具体内容之后，我们可以有一个大概的思路或者直觉，提出自己感兴趣的问题，比如说：

- 这些 APP 的价格分布如何，免费应用所占比例是多少？
- 总评分与价格有没有相关性？
- APP 的分类收费情况？
- 各个分类的评分分布差异？

注意：随着对数据集了解的逐渐深入，我们非常有可能对这些问题进行迭代。



Pandas 常用的评估方法

查看数据内容	缺失值、重复项	异常值
df.head(n)	df.info()	df.describe()
df.sample(n)	df.isnull().sum()	df['column'].unique()
df.tail(n)	df.duplicated().sum()	df['column'].value_counts()

附录：

[Pandas 速查手册中文版](#)[Pandas 基础命令速查清单](#)

2 数据整理——清洗方法

Pandas 常用的清洗方法 1

方法示例	含义
<code>pd.merge(df1, df2, on='id')</code>	将 df1 与 df2 按照 id 列进行合并
<code>pd.concat([df1, df2],axis=1)</code>	将 df2 中的列添加到 df1 的尾部
<code>df.drop(column=['c1', 'c2'])</code>	删除 df 中的 c1 和 c2 列
<code>df[['col1', 'col2']]</code>	筛选出 df 中的 col1 和 col2
<code>df.set_index('col1')</code>	更改索引列为 col1
<code>df.reset_index()</code>	将 df 的索引重设为数字, 原来的索引转变为普通列
<code>df.sort_values('col')</code>	按照 col 列进行排序, 默认为从小到大的顺序

2 数据整理——清洗方法

Pandas 常用的清洗方法 2

方法示例	含义
<code>df.dropna(subset=['c1', 'c2'])</code>	删除所有 c1, c2 列中有缺失值的行
<code>df.fillna(value)</code>	将 df 中的所有 NaN 替换为 value
<code>df.fillna({'c1':0, 'c2':False})</code>	按列替换 NaN 为不同的值
<code>df.drop_duplicates()</code>	删除重复项, 也可以设定 subset
<code>df['col'].astype(float)</code>	将 col 列的类型转换为 float 类型
<code>df.rename(columns={'y':'year'})</code>	修改列名
<code>df.replace([1,3],['one','three'])</code>	用'one'代替 1, 用'three'代替 3
<code>df.loc[df['age']<0, 'age'] = 0</code>	选出 df 中所有age列小于0的行, 将其age特征重新赋值

2

数据整理——评估 APP 数据集



优达学城
挑战不可能

```
df_app.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7197 entries, 0 to 7196
Data columns (total 17 columns):
Unnamed: 0      7197 non-null int64
id              7197 non-null int64
track_name      7197 non-null object
size_bytes      7197 non-null int64
currency        7197 non-null object
price           7197 non-null float64
rating_count_tot 7197 non-null int64
rating_count_ver 7197 non-null int64
user_rating     7197 non-null float64
user_rating_ver 7197 non-null float64
ver             7197 non-null object
cont_rating     7197 non-null object
prime_genre     7197 non-null object
sup_devices.num 7197 non-null int64
ipadSc_urls.num 7197 non-null int64
lang.num        7197 non-null int64
vpp_lic         7197 non-null int64
dtypes: float64(3), int64(9), object(5)
memory usage: 955.9+ KB
```

```
df_app.describe()
```

	size_bytes	price	rating_count_tot	rating_count_ver	user_rating	user_rating_v
count	7.197000e+03	7197.000000	7.197000e+03	7197.000000	7197.000000	7197.000000
mean	1.991345e+08	1.726218	1.289291e+04	460.373906	3.526956	3.2535
std	3.592069e+08	5.833006	7.573941e+04	3920.455183	1.517948	1.80936
min	5.898240e+05	0.000000	0.000000e+00	0.000000	0.000000	0.00000
25%	4.692275e+07	0.000000	2.800000e+01	1.000000	3.500000	2.50000
50%	9.715302e+07	0.000000	3.000000e+02	23.000000	4.000000	4.00000
75%	1.819249e+08	1.990000	2.793000e+03	140.000000	4.500000	4.50000
max	4.025970e+09	299.990000	2.974676e+06	177050.000000	5.000000	5.00000

注：这里没办法将全部结果列出来，可以尝试一下自己在 Notebook 操作。



针对最开始提出的问题, 以及对数据的更详细的评估
我们可以对数据进行以下处理:

- 将 df_desc 中的 app_desc 列合并到 df_app 中
- 删除不需要的数据列 ('Unnamed: 0', 'currency', 'vpp_lic')
- 将列名中的 . 修改为 _
- id 列修改为字符串类型
- 对 prize 进行 cut 分段, 创建新的分类变量 price_cut



具体的清理代码如下：

- 将 df_desc 中的 app_desc 列合并到 df_app 中
`df = pd.merge(df_app, df_desc[['id', 'app_desc']], on='id')`
- 删除不需要的数据列 ('Unnamed: 0', 'currency', 'vpp_lic')
`df.drop(['Unnamed: 0', 'currency', 'vpp_lic'], axis=1, inplace=True)`
- 将列名中的 . 修改为 _
`df = df.rename(columns={c: c.replace('.', '_') for c in df_app.columns})`
- id 列修改为字符串类型
`df.id = df.id.astype('str')`
- 对 price 进行 cut 分段，创建新的分类变量 price_cut
`price_labels = ['Free', '$0~$5', '$5~$10', '$10~$50', '$50+']`
`df['price_cut'] = pd.cut(df['price'], bins=[-1, 0, 5, 10, 50, 300], labels=price_labels)`

2

数据整理——清洗过程



优达学城

挑战不可能

```
df = pd.merge(df_app, df_desc[['id', 'app_desc']], on='id')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7197 entries, 0 to 7196
Data columns (total 18 columns):
Unnamed: 0      7197 non-null int64
id              7197 non-null int64
track_name      7197 non-null object
size_bytes      7197 non-null int64
currency        7197 non-null object
price           7197 non-null float64
rating_count_tot 7197 non-null int64
rating_count_ver 7197 non-null int64
user_rating     7197 non-null float64
user_rating_ver 7197 non-null float64
ver             7197 non-null object
cont_rating     7197 non-null object
prime_genre     7197 non-null object
sup_devices.num 7197 non-null int64
ipadSc_urls.num 7197 non-null int64
lang.num        7197 non-null int64
vpp_lic         7197 non-null int64
app_desc        7197 non-null object
dtypes: float64(3), int64(9), object(6)
memory usage: 1.0+ MB
```

```
df.drop(['Unnamed: 0', 'currency', 'vpp_lic'], axis=1, inplace=True)
```

```
df = df.rename(columns={c: c.replace('.', '_') for c in df_app.columns})
df.columns
```

```
Index(['id', 'track_name', 'size_bytes', 'price', 'rating_count_tot',
       'rating_count_ver', 'user_rating', 'user_rating_ver', 'ver',
       'cont_rating', 'prime_genre', 'sup_devices_num', 'ipadSc_urls_num',
       'lang_num', 'app_desc'],
      dtype='object')
```

```
df.id = df.id.astype('str')
df.id.dtype
```

```
dtype('O')
```

```
: price_labels = ['Free', 'USD:0~5', 'USD:5~20', 'USD:20+']
df['price_cut'] = pd.cut(df['price'], bins=[-1, 0, 5, 20, 300], labels=price_labels)
```

```
: df['price_cut'].value_counts()
```

```
Free      4056
USD:0~5    2703
USD:5~20    402
USD:20+     36
Name: price_cut, dtype: int64
```



现在我们已经完成了数据整理的操作，对数据的了解更深入了，这个时候可以对最开始提出的问题迭代(对问题的迭代可能发生在任何一步中)，可能会提出更有价值的问题。

比如在探索了 `prime_genre` 之后了解到游戏类应用占据了数据集的 53.7%，其他分类很多，但是每个类别中数量较少，所以将关于分类的问题修改为游戏类与非游戏类的对比。

迭代后的问题：

- 游戏类与非游戏类 APP 的收费情况？
- 游戏类与非游戏类 APP 的评分分布差异？

迭代后所需要进一步清洗的内容：

```
df['is_Game'] = df['prime_genre'] == 'Games'  
df['is_Game'].replace([True, False], ['Games', 'NotGames'], inplace=True)
```

```
df['is_Game'].value_counts()
```

```
Games      3862  
NotGames   3335  
Name: is_Game, dtype: int64
```



3 探索数据——常用统计方法

简单的统计计算方法：

- `df.describe()`: 查看数据值的汇总统计，也可以直接对某一列使用
- `df.mean()`: 返回所有列的均值，也可以直接对某一列使用
- `df.count()`: 返回每一列中的非空值的个数，也可以直接对某一列使用
- `df.max()`: 返回每一列的最大值，也可以直接对某一列使用
- `df.min()`: 返回每一列的最小值，也可以直接对某一列使用
- `df.median()`: 返回每一列的中位数，也可以直接对某一列使用
- `df.std()`: 返回每一列的标准差，也可以直接对某一列使用
- `df.corr()`: 返回 `df` 中列与列之间的相关系数表
- `df['col1'].corr(df['col2'])`: 返回 `col1` 列与 `col2` 列的相关系数

3

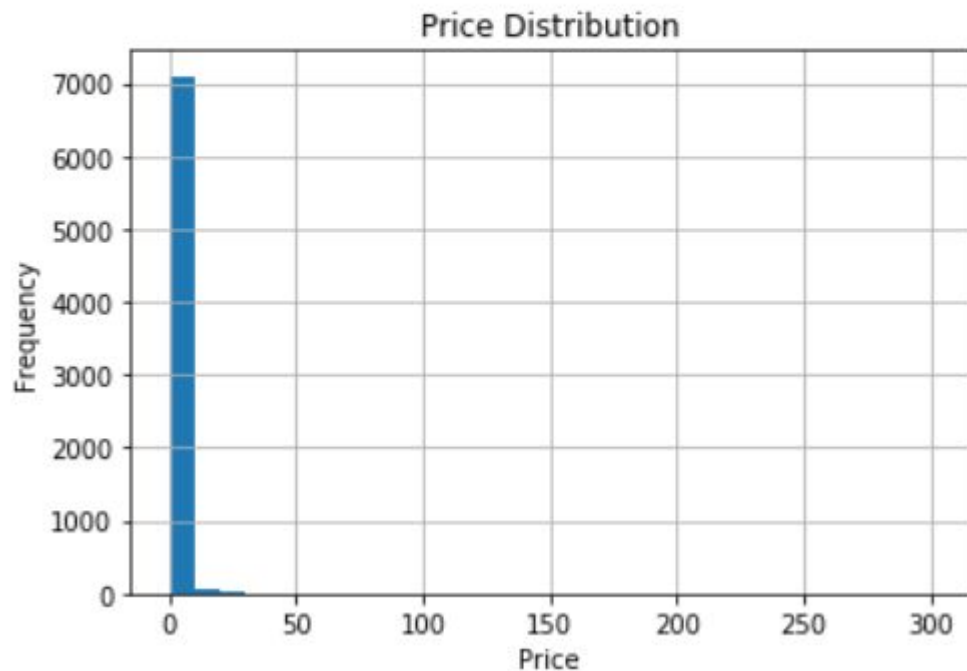
探索数据——可视化探索



优达学城
挑战不可能

单变量探索：连续性数值变量使用直方图或箱线图

```
df['price'].hist(bins=30,figsize=(6,4))  
plt.title('Price Distribution')  
plt.ylabel('Frequency')  
plt.xlabel('Price');
```



```
# showfliers=False 不显示离群值  
df['price'].plot.box(showfliers=False, showmeans=True, figsize=(6,4))  
plt.title('Price Distribution')  
plt.ylabel('USD');
```



3

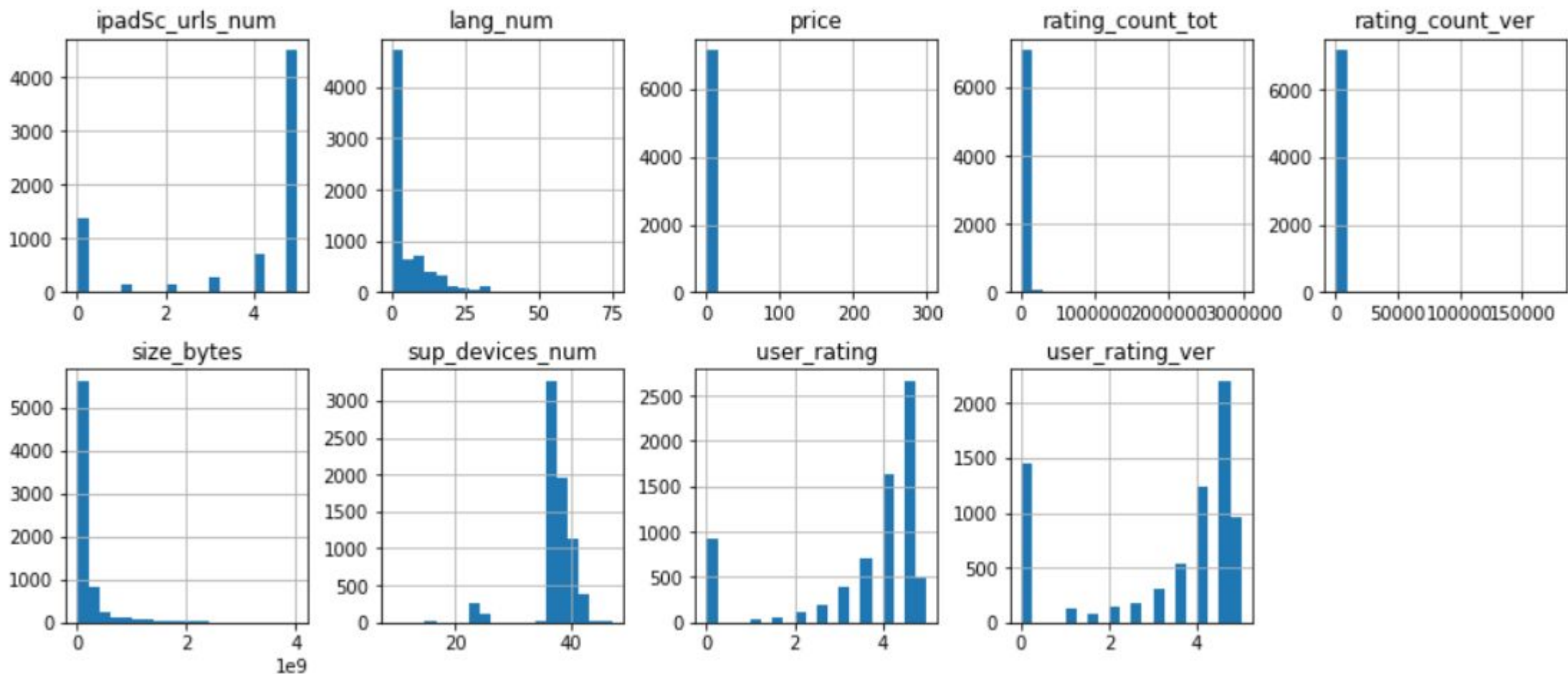
探索数据——可视化探索



优达学城
挑战不可能

单变量探索：直方图矩阵

```
df.hist(figsize=(15, 6), layout=(2, 5), bins=20);
```



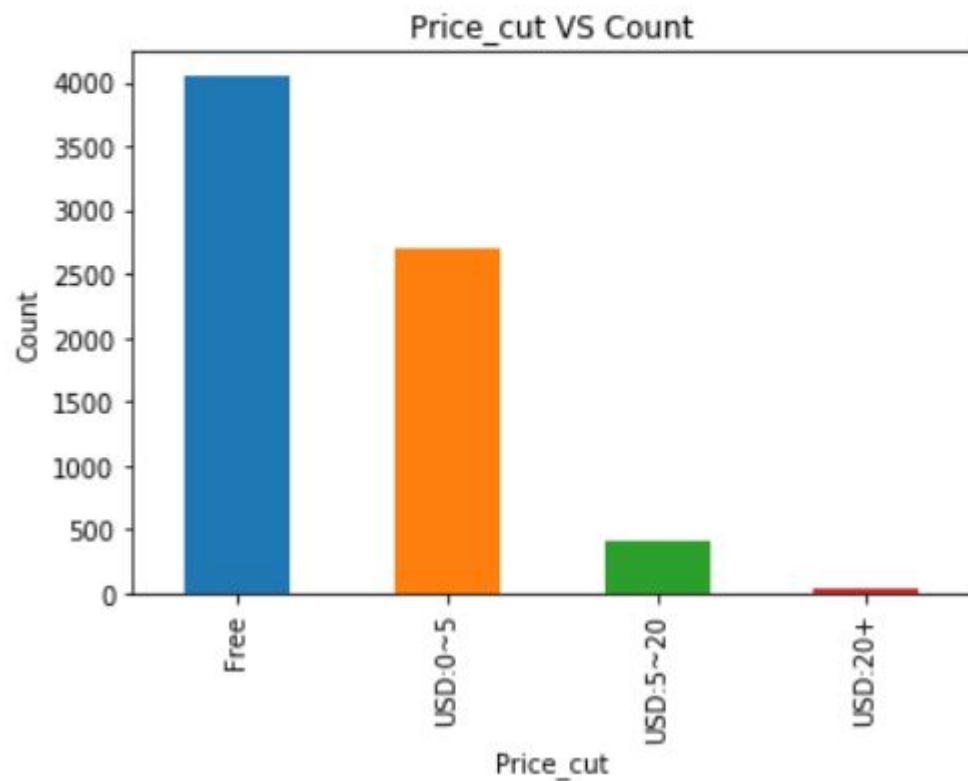


3

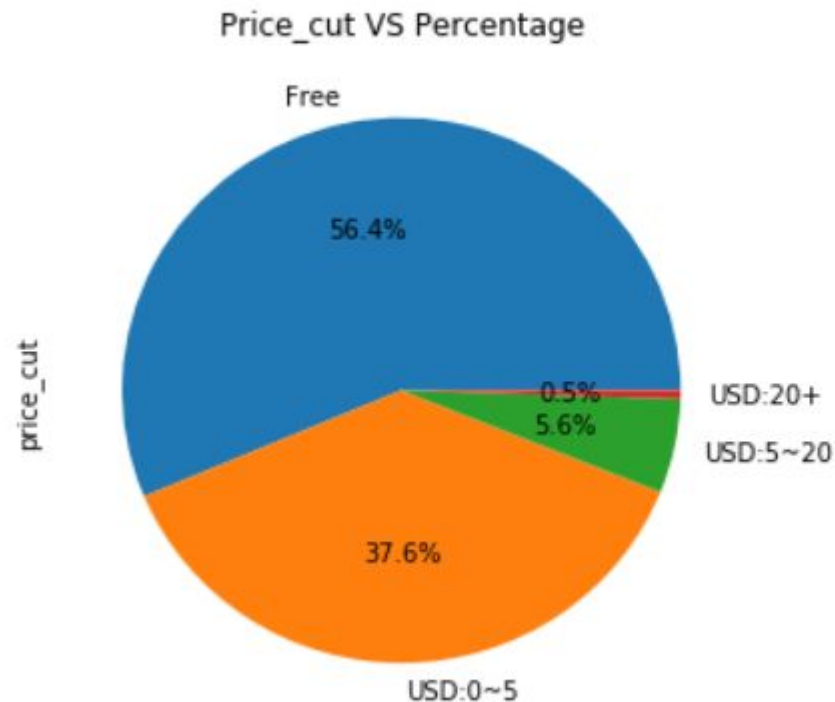
探索数据——可视化探索

单变量探索：离散型/分类变量使用柱状图或饼图

```
df.price_cut.value_counts().plot.bar(figsize=(6,4))  
plt.title('Price_cut VS Count')  
plt.ylabel('Count')  
plt.xlabel('Price_cut');
```



```
df.price_cut.value_counts().plot.pie(autopct='%1f%%',figsize=(5,5))  
plt.title('Price_cut VS Percentage');
```



3

探索数据——可视化探索



优达学城

挑战不可能

双变量探索：两个数值变量的探索——散点图

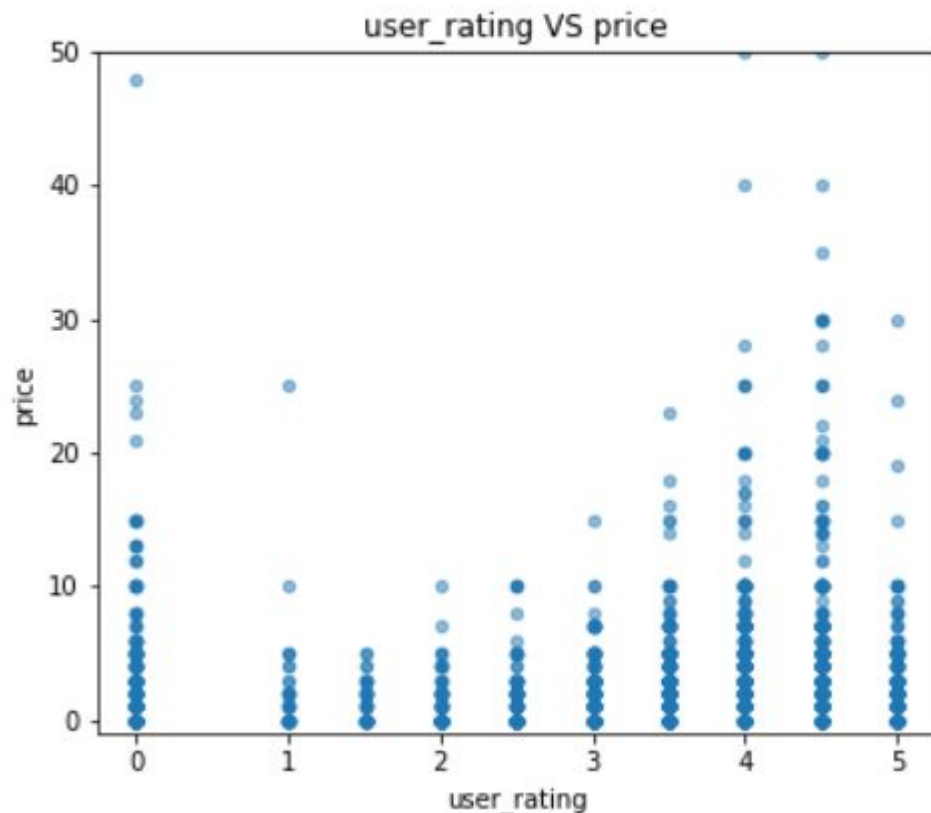
散点图用于探索两个数值变量之间是否存在一些相关性。

本次示例的数据集中，price 的分布存在一些极端值，所以使用 ylim 做了限制；因为 user_rating 实际上是离散数值，所以出现了一条一条的散点分布情况。

图中没有办法观察出明显的相关趋势

:

```
df.plot.scatter(x='user_rating', y='price', alpha = 0.5, figsize=(6, 5))  
plt.title('user_rating VS price')  
plt.ylim(-1, 50);
```



3

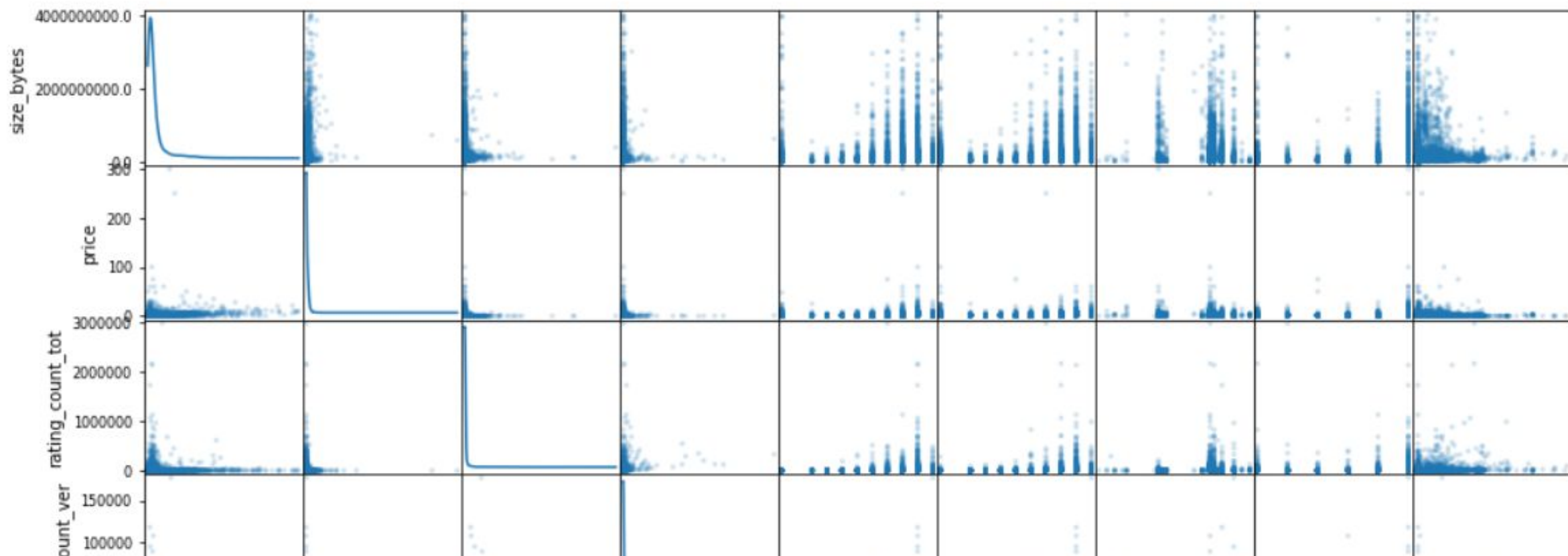
探索数据——可视化探索



优达学城
挑战不可能

双变量探索：散点图矩阵

```
from pandas.plotting import scatter_matrix  
scatter_matrix(df, alpha=0.2, figsize=(15, 15), diagonal='kde');
```



3

探索数据——可视化探索

优达学城
挑战不可能

在探索相关性时，如果散点图给出的趋势不明显，还可以借助相关系数来获得更加明确的探索方向：

```
df.corr()
```

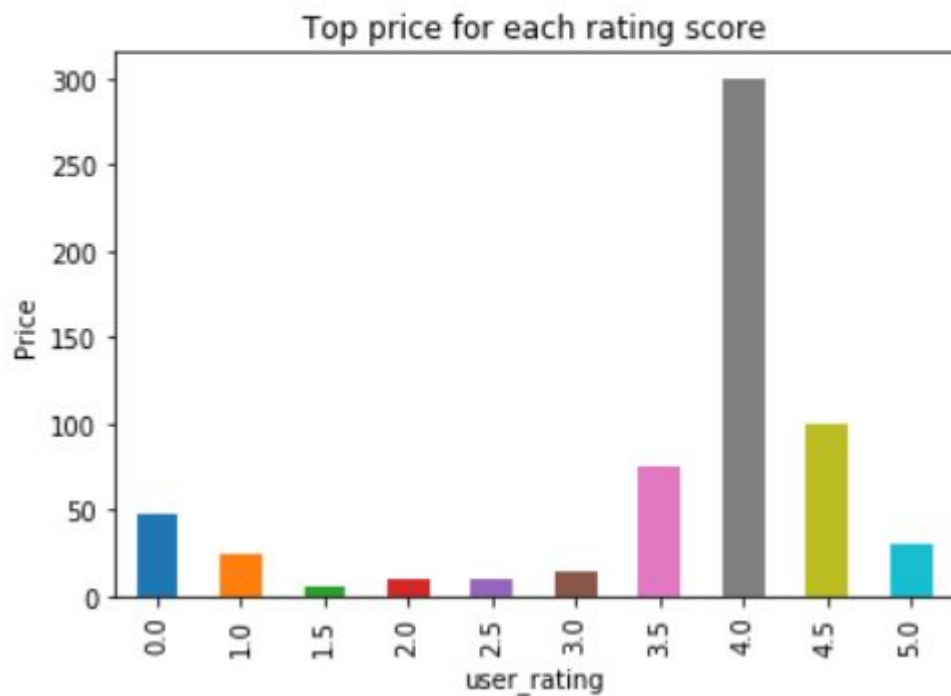
	size_bytes	price	rating_count_tot	rating_count_ver	user_rating	user_rating_ver	sup_devices_num	ipadSc_urls_num	lang_num
size_bytes	1.000000	0.182392	0.004486	0.006337	0.066256	0.086075	-0.118347	0.152697	0.004614
price	0.182392	1.000000	-0.039044	-0.018012	0.046601	0.025173	-0.115361	0.066100	-0.006713
rating_count_tot	0.004486	-0.039044	1.000000	0.163645	0.083310	0.088744	0.008832	0.015734	0.137675
rating_count_ver	0.006337	-0.018012	0.163645	1.000000	0.068754	0.077840	0.037951	0.024333	0.013287
user_rating	0.066256	0.046601	0.083310	0.068754	1.000000	0.774140	-0.042451	0.265671	0.170976
user_rating_ver	0.086075	0.025173	0.088744	0.077840	0.774140	1.000000	-0.018901	0.275737	0.175580
sup_devices_num	-0.118347	-0.115361	0.008832	0.037951	-0.042451	-0.018901	1.000000	-0.037728	-0.041681
ipadSc_urls_num	0.152697	0.066100	0.015734	0.024333	0.265671	0.275737	-0.037728	1.000000	0.088378
lang_num	0.004614	-0.006713	0.137675	0.013287	0.170976	0.175580	-0.041681	0.088378	1.000000

3

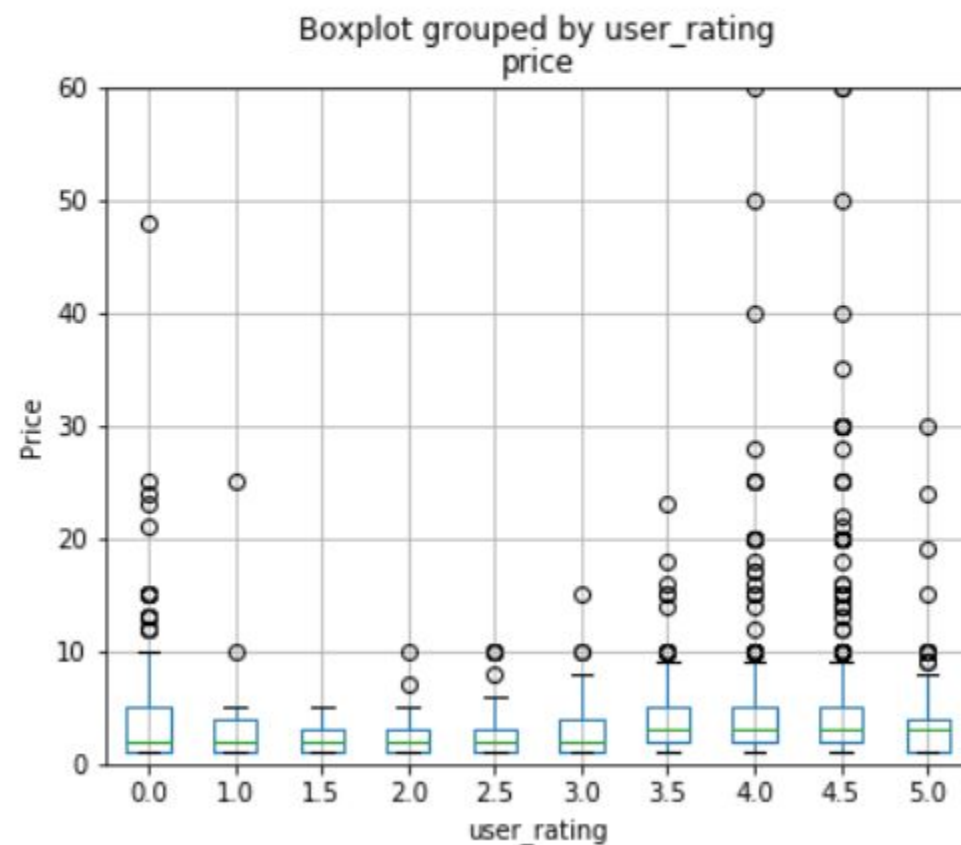
探索数据——可视化探索

双变量探索：离散 VS 连续

```
df.groupby('user_rating')['price'].max().plot.bar()
plt.title('Top price for each rating score')
plt.ylabel('Price');
```



```
df[df['price']>0].boxplot(column=['price'],
                           by='user_rating', figsize=(6, 5))
plt.ylim(0, 60)
plt.ylabel('Price');
```



优达学城
挑战不可能



3

探索数据——可视化探索

双变量探索：两个离散变量，组合柱状图

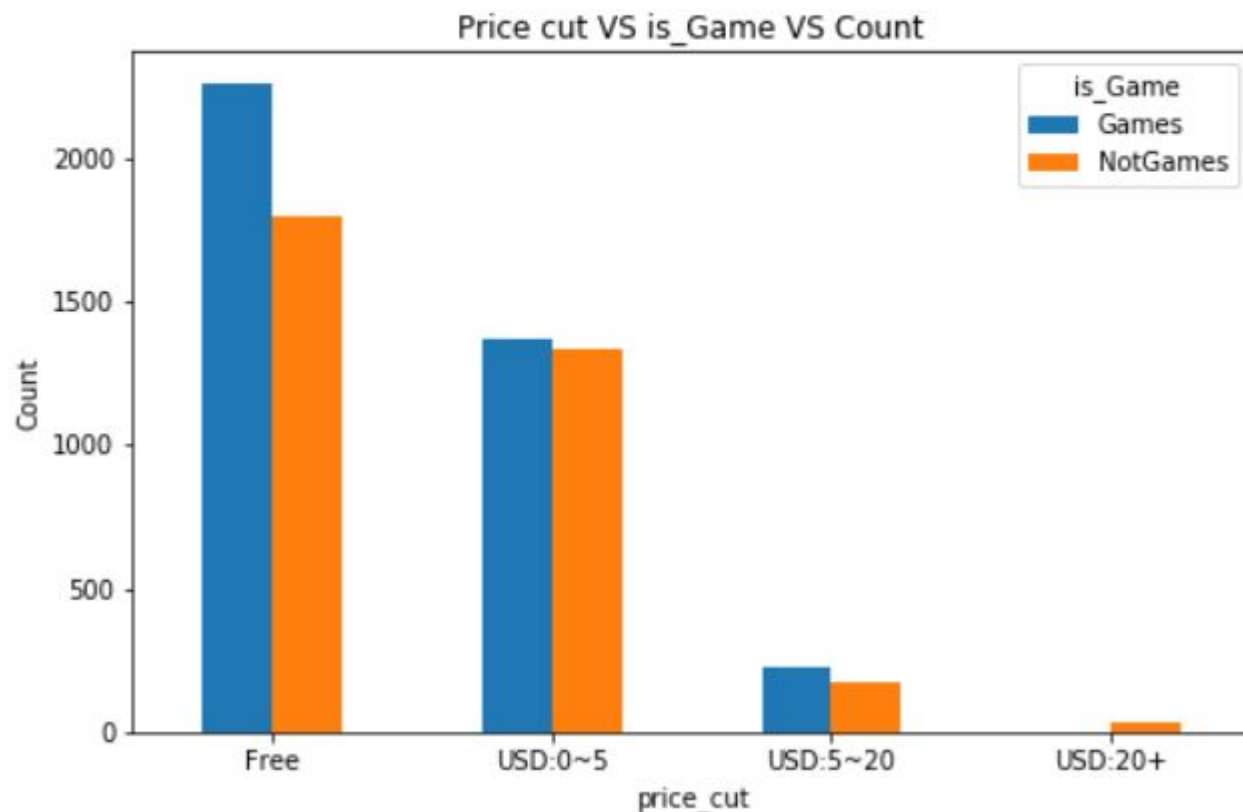
探索游戏应用与非游戏应用的价格差异，可以使用组合柱状图进行对比。

下图代码中的 `unstack` 将数据由双索引的结构转换为宽格式的数据：

```
df.groupby(['price_cut', 'is_Game']).size().unstack()
```

is_Game	Games	NotGames
price_cut		
Free	2257	1799
USD:0~5	1372	1331
USD:5~20	230	172
USD:20+	3	33

```
df.groupby(['price_cut', 'is_Game']).size().unstack().plot.bar(figsize=(8, 5))  
plt.title('Price cut VS is_Game VS Count')  
plt.xticks(rotation=0) # 控制 x 轴刻度文字的角度  
plt.ylabel('Count');
```





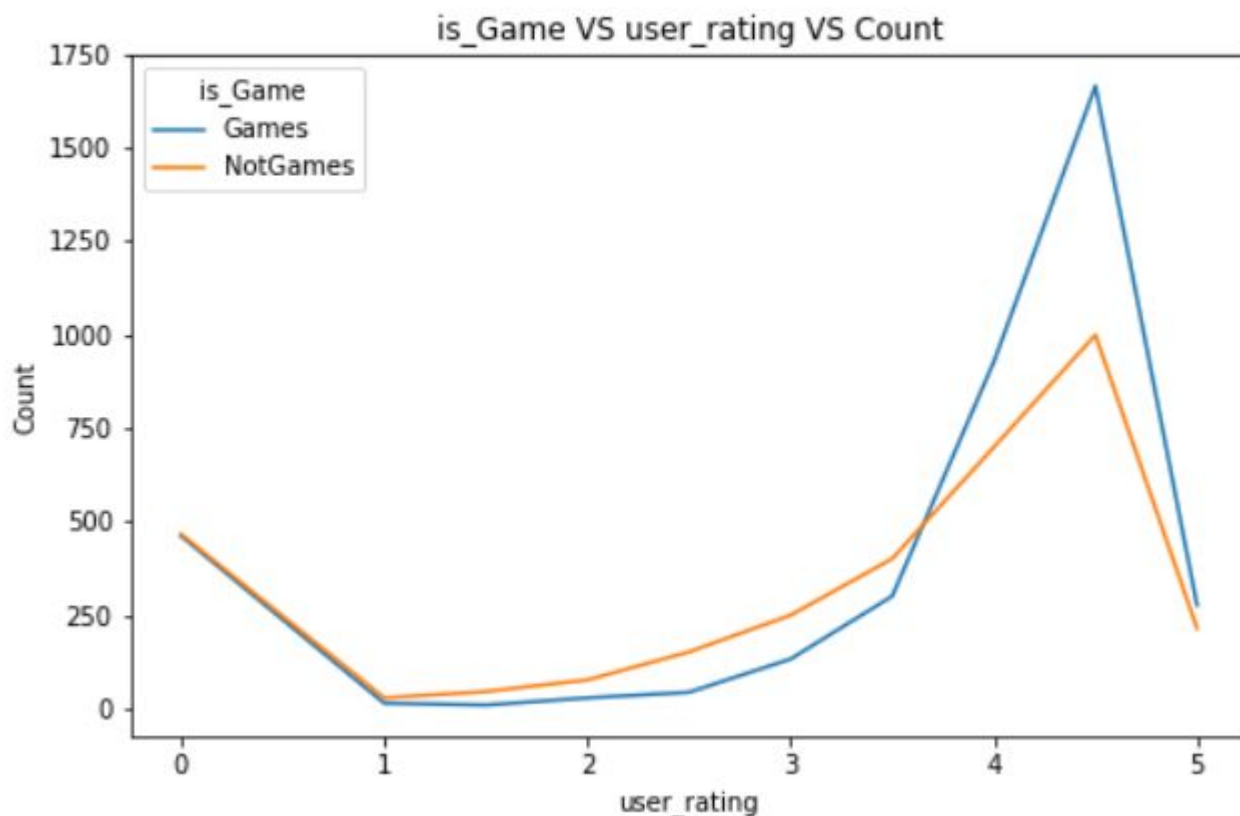
3 探索数据——可视化探索

双变量探索：折线图绘制示例

折线图主要用于绘制有前后关联的数据，最常见的是时间序列。

本次数据集中没有包含时间序列的数据，因为 user_rating 的各个取值之间存在关联，所以尝试使用一下折线图绘制随着 user_rating 的增长，游戏与非游戏应用的数量变化：

```
df.groupby(['user_rating', 'is_Game']).size().unstack().plot(figsize=(8, 5))  
plt.title('is_Game VS user_rating VS Count')  
plt.ylabel('Count');
```



4 得出结论



优达学城
挑战不可能

对探索问题的回答

在数据探索的过程中，陆续得出了一些结论，在报告的最后部分，我们需要与最开始提出的问题相呼应，进行一一回答，也是对中间探索过程的一个总结。

局限性探讨

除了探索结果的总结，还需要对当前分析中存在的限制加以探讨。

- 数据本身的局限

是否存在一些你觉得可能对探索有帮助，但是数据集中不存在的特征？

是否有一些特征你不理解含义，或者不确定理解的是否正确？

- 探索方式的局限

本次项目探索使用的只是一些简单的统计计算和可视化分析。这样得出的结论只能是暂时的，还有进一步验证的需要（使用统计检验或者机器学习建模等）。不过因为这个项目并不要求做其他工作，所以只需要声明这些局限即可。



总结

- 所有 APP 中，大部分都小于 5 美元。免费应用占比为 56.4%，占据一半以上。
- 经过散点图探索和相关系数的计算，可以看到价格和评分没有明显的线性相关性。在深入探索后，可以发现价格较高的 APP 的评分比较极端，0分或者 3.5分以上。猜测可能如果是价格较高的 APP，如果让用户失望，可能会更倾向于评 0 分，而不是一些中间的分。
- 免费应用中游戏应用比例较高，非免费应用中，游戏与非游戏类应用的比例则比较相似。
- 游戏/非游戏类应用与评分的关系：在 3.5分之前，都是非游戏应用较多，3.5分之后则是游戏应用较多。

局限性

- 当前探索的问题主要集中在价格、评分、游戏分类这些特征，其他特征没有进行过多的探索。
- 根据对 APP 商店的了解，一些新上架的应用，评价应该会比较少，有可能有一些偶然情况不能了解到，评分结合评分人数应该会有更加准确的信息。
- 本次分析的过程只是使用了可视化探索和一些简单的统计计算，没有进行任何的显著性检验，所以目前的结论只是暂时的，还有进一步分析的可能性。

5 与人交流



优达学城
挑战不可能

以阅读者的角度，整理报告：

- 注意整理中间过程的迭代

中间迭代的问题要整理记得添加到【简介：提出问题】部分；

- 数据整理过程添加思路的说明和所做变更的记录

评估的目的和结果，相应的清洗方案

做每个数据清洗的目的说明，清洗后对结果的说明

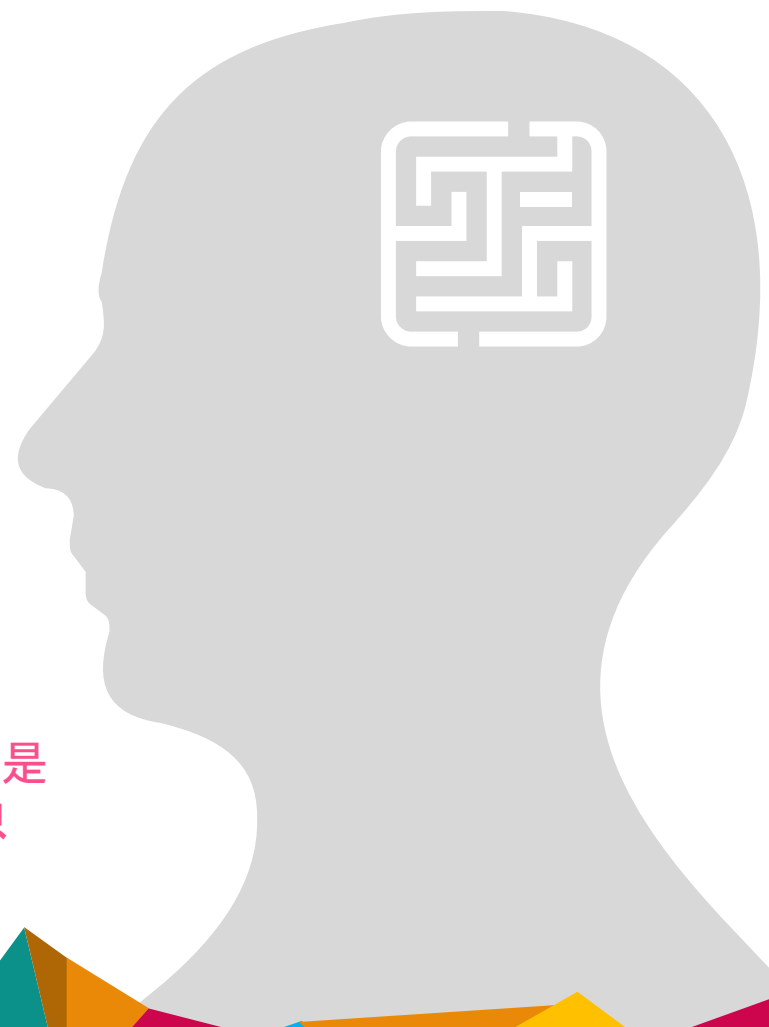
- 可视化图表的各种元素：

可视化图表一定要记得添加标题、坐标轴标题、图例等要素。

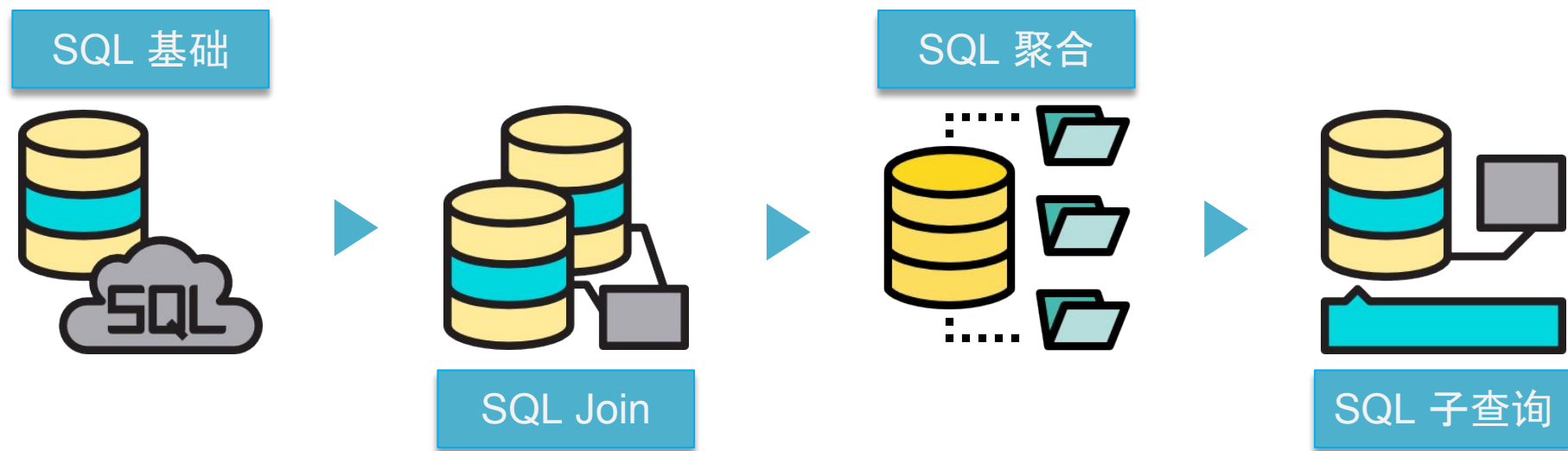
参考资料：[matplotlib 绘图可视化知识点整理](#)

- 每个可视化和统计计算后的推理说明文字

当前图表/统计计算表现了什么信息，从中我们可以推理出什么结果，是否可以回答所要探索的问题等。要使用 Markdown 单元格，代码注释中只是针对代码功能的说明，而不是与分析思路和推理相关的说明。



SQL 知识点回顾





1

SQL 基础

- **SELECT**: 提供你想要的列
- **FROM**: 提供列存在的表
- **LIMIT**: 限制返回的行数
- **ORDER BY**: 根据列对表排序, 与 **DESC** 一起使用。
- **WHERE**: 用于过滤结果的条件语句
- **WHERE** Col **LIKE** '%me%': 仅拉取文本中包含 'me' 的列
- **WHERE** Col **IN** ('Y','N'): 仅过滤包含 'Y' 或 'N' 列的行
- **WHERE** Col **NOT IN** ('Y','N'): **NOT** 经常与 **LIKE** 和 **IN** 一起使用
- **WHERE** Col1 > 5 **AND** Col2 < 3: 过滤两个或多个条件必须为真的行
- **WHERE** Col1 > 5 **OR** Col2 < 3: 过滤至少一个条件必须为真的行
- **WHERE** Col **BETWEEN** 3 **AND** 5: 通常比使用 **AND** 的语法简单

2

SQL Join



优达学城

挑战不可能

- 主键:对于表格中的每行都是唯一的。主键通常是数据库中的第一列(就像Parch & Posey数据库中每个表格的 id 列)
- 外键:是出现在另一个表格中的主键,允许行不是唯一的行。
- JOIN:一种 INNER JOIN, 仅获取在两个表格中都存在的数据。
- LEFT JOIN:用于获取 FROM 中的表格的所有行,即使它们不存在于 JOIN 语句中。
- RIGHT JOIN:用于获取 JOIN 中的表格的所有行,即使它们不存在于 FROM 语句中。
- 其他高级 JOIN: UNION 和 UNION ALL、CROSS JOIN、SELF JOIN
- 别名:使用 AS 或直接对表格和列设定别名。这样可以减少要输入的字符数,同时确保列标题可以描述表格中的数据。

3

SQL 聚合



优达学城

挑战不可能

- **NULL**: 是一种数据类型, 表示 SQL 中没有数据。在聚合中经常会忽略掉
- **COUNT**: 聚合数据的行数。**COUNT** 不会考虑具有 **NULL** 值的行。因此, 可以用来快速判断哪些行缺少数据。
- **SUM**: 对某一系列聚合求和, 你只能针对数字列使用 **SUM**。
- **MIN** 与 **MAX**: 与 **COUNT** 相似, 它们都可以用在非数字列上。**MIN** 将返回最小的数字、最早的日期或按字母表排序的最之前的非数字值, 具体取决于列类型。**MAX** 则正好相反, 返回的是最大的数字、最近的日期, 或与"Z"最接近(按字母表顺序排列)的非数字值。
- **AVG**: 返回的是数据的平均值, 即列中所有的值之和除以列中值的数量。该聚合函数同样会忽略分子和分母中的 **NULL** 值。

3

SQL 聚合



优达学城

挑战不可能

- **GROUP BY**: 可以用来在数据子集中聚合数据。始终在 **WHERE** 和 **ORDER BY** 之间。
- **DISTINCT**: 仅返回特定列的唯一值。在使用 **DISTINCT** 时, 尤其是在聚合函数中使用时, 会让查询速度有所减慢。
- **HAVING**: 是过滤被聚合的查询的"整洁"方式。只要你想对通过聚合创建的查询中的元素执行 **WHERE** 条件, 就需要使用 **HAVING**
- **DATE**: **DATE_TRUNC** 使你能够将日期截取到日期时间列的特定部分。**DATE_PART** 可以用来获取日期的特定部分
- **CASE**: 始终位于 **SELECT** 条件中, 必须包含以下几个部分: **WHEN**、**THEN** 和 **END**。**ELSE** 是可选组成部分, 用来包含不符合上述任一 **CASE** 条件的情况。

4

SQL 子查询



优达学城

挑战不可能

- 子查询(SubQuery)或者说内查询(InnerQuery), 也可以称作嵌套查询(NestedQuery)
- 子查询用于为主查询返回其所需数据, 或者对检索数据进行进一步的限制
- 天气数据集示例, 查询两个城市的数据:

```
SELECT c.year, c.avg_temp hangzhou, s.shanghai
FROM city_data c,
     (SELECT year, avg_temp shanghai
      FROM city_data WHERE city = 'Shanghai') AS s
WHERE c.city = 'Hangzhou'
AND c.year = s.year
```

Q&A
Thanks



优达学城
UDACITY

Acknowledgement

**感谢以下资深助教对本辅导资料的
贡献**

李伟伟



优达学城
UDACITY