

Jupyter Notebook と MySQL で ゼロからはじめるデータサイエンス

株式会社インフィニットループ 技術研究グループ

波多野 信広

Twitter : @nobuhatano



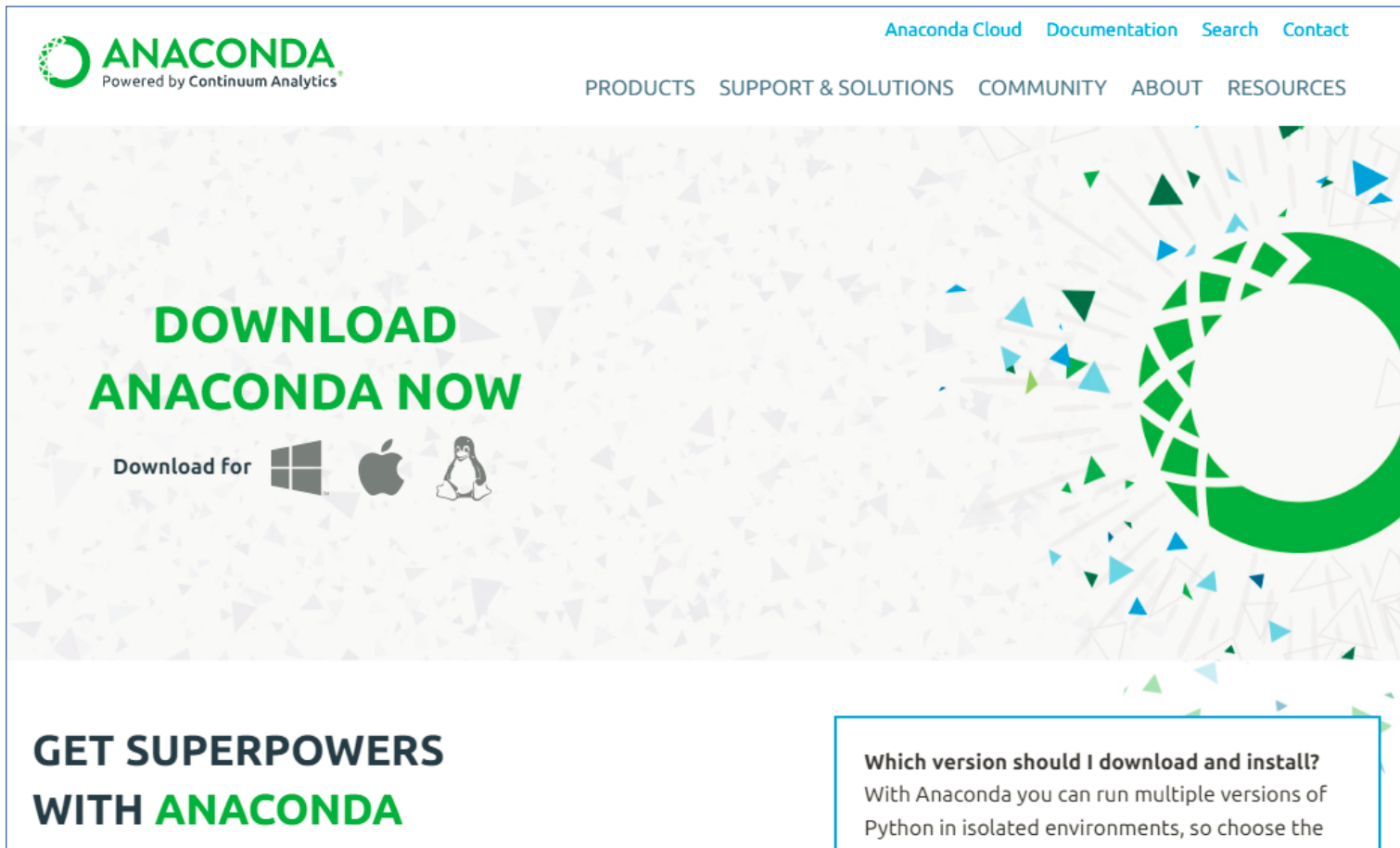
本セッションの内容

Web サービスやゲームを運用している DBA やプログラマの方々に、データ分析に興味がある方向け

- Jupyter Notebook + MySQL
 - SQL と Python で自由にデータ分析
- はじめてのデータ分析
 - ソーシャルゲームのデータ分析事例

まずはインストール

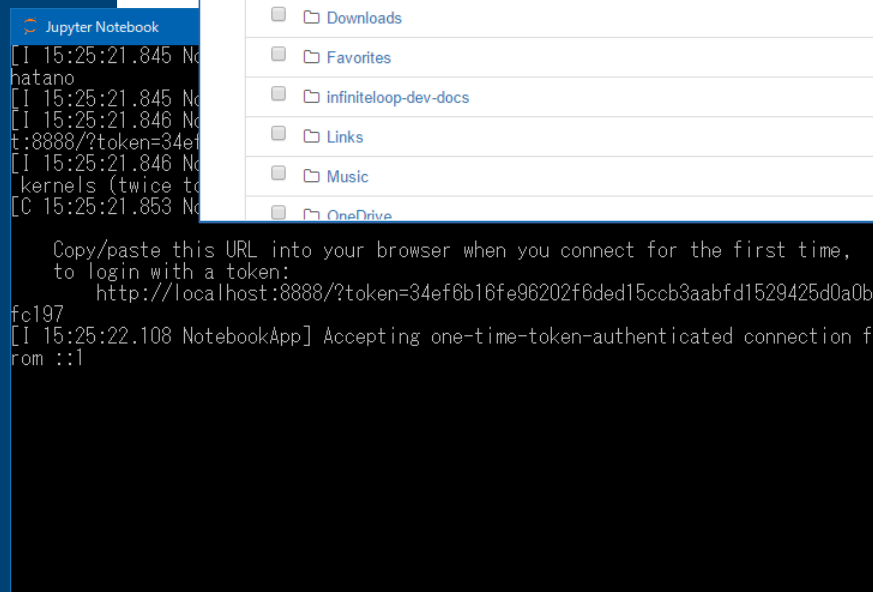
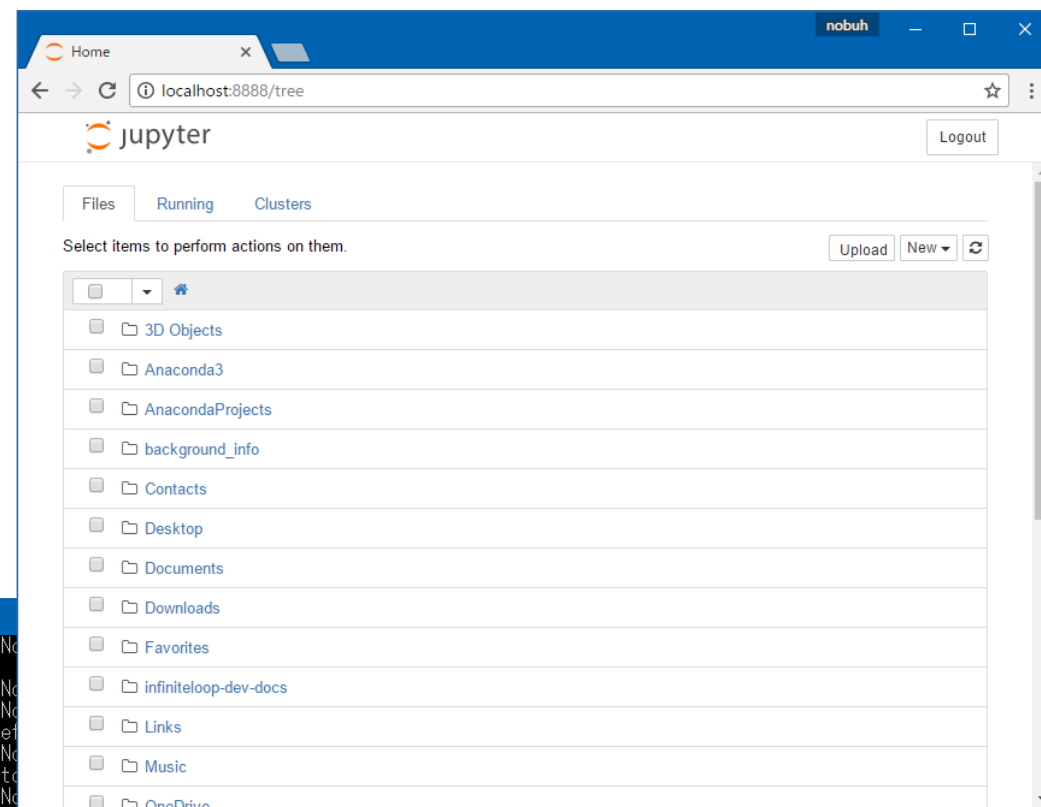
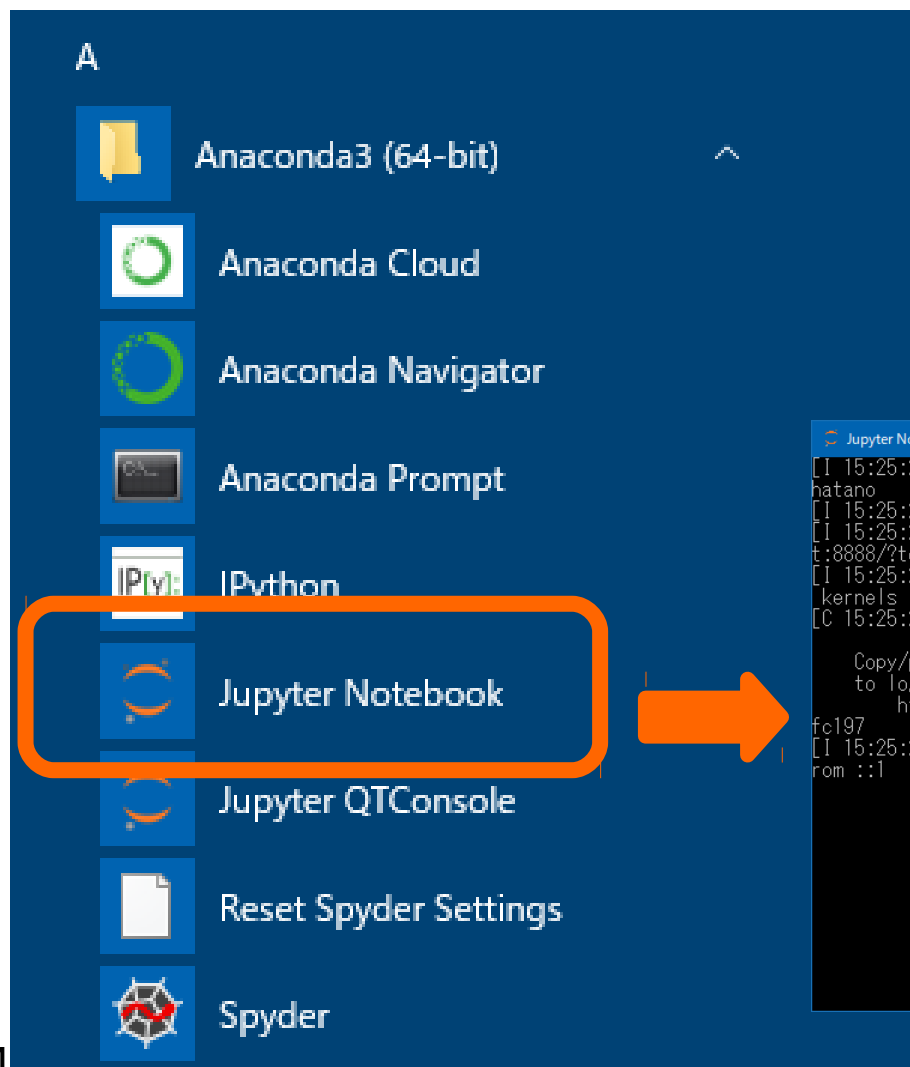
Python 全部入り Continuum Analytics 社の Anaconda
Anaconda 4.3.1 for Windows 64 bit Python 3.6 version



The screenshot shows the Anaconda website homepage. At the top left is the Anaconda logo with the text "ANAACONDA Powered by Continuum Analytics". To the right are links for "Anaconda Cloud", "Documentation", "Search", and "Contact". Below these are navigation links: "PRODUCTS", "SUPPORT & SOLUTIONS", "COMMUNITY", "ABOUT", and "RESOURCES". The main content area features a large green "DOWNLOAD ANACONDA NOW" button. Below the button are icons for Windows, macOS, and Linux, with the text "Download for" to the left. On the right side of the main area is a large green circular graphic with a white grid pattern. At the bottom left, the text "GET SUPERPOWERS WITH ANACONDA" is displayed. At the bottom right, a box contains the text: "Which version should I download and install? With Anaconda you can run multiple versions of Python in isolated environments, so choose the".

<https://www.continuum.io/downloads>

起動してみる



インタラクティブな実行環境

The screenshot shows the Jupyter web interface for a notebook named 'FizzBuzz'. The top bar includes the Jupyter logo, the notebook name, and the last checkpoint information. A menu bar with options like File, Edit, View, Insert, Cell, Kernel, Widgets, and Help is visible. A toolbar with icons for file operations and a 'Code' dropdown is also present. The main content area contains a markdown cell with the title '例えば Fizz Buzz 問題' and a list of rules for the Fizz Buzz problem. Below this is a code cell containing a Python list comprehension. The output of the code cell is displayed below the code. Three yellow callout boxes provide additional information: one points to the markdown cell, another points to the code cell, and a third points to the 'Code' dropdown in the toolbar.

セル
(markdown)

例えば **Fizz Buzz** 問題

- 3 の倍数なら Fizz
- 5 の倍数なら Buzz
- 15 の倍数なら FizzBuzz
- それ以外は 1 から 100 まで数字で

```
In [20]: [ (i % 3 == 0) * 'Fizz' + (i % 5 == 0) * 'Buzz' or i for i in range(1, 101) ]
```

Out [20]: [1, 2, 'Fizz', 4, 'Buzz', 'Fizz', 7, 8, 'Fizz', 'Buzz', 11, 'Fizz', 13, 14, 'Fizz', 16, 'Buzz', 19, 'Fizz', 22, 'Fizz', 25, 'Buzz', 28, 'Fizz', 31, 'Buzz', 34, 'Fizz', 37, 38, 'Fizz', 41, 'Buzz', 44, 'Fizz', 47, 48, 'Fizz', 52, 'Buzz', 55, 'Fizz', 58, 59, 'Fizz', 62, 'Buzz', 65, 'Fizz', 68, 69, 'Fizz', 73, 'Buzz', 76, 'Fizz', 79, 80, 'Fizz', 84, 'Buzz', 87, 'Fizz', 91, 'Buzz', 94, 'Fizz', 97, 98, 'Fizz', 100]

セル
(Python3)

Ctrl + Enter
セルのコード実行

グラフを表示するおまじない

```
In [1]: # プロット結果を jupyter 上にインライン表示する
%matplotlib inline

# 慣用句として pyplot は plt に
from matplotlib import pyplot as plt
```

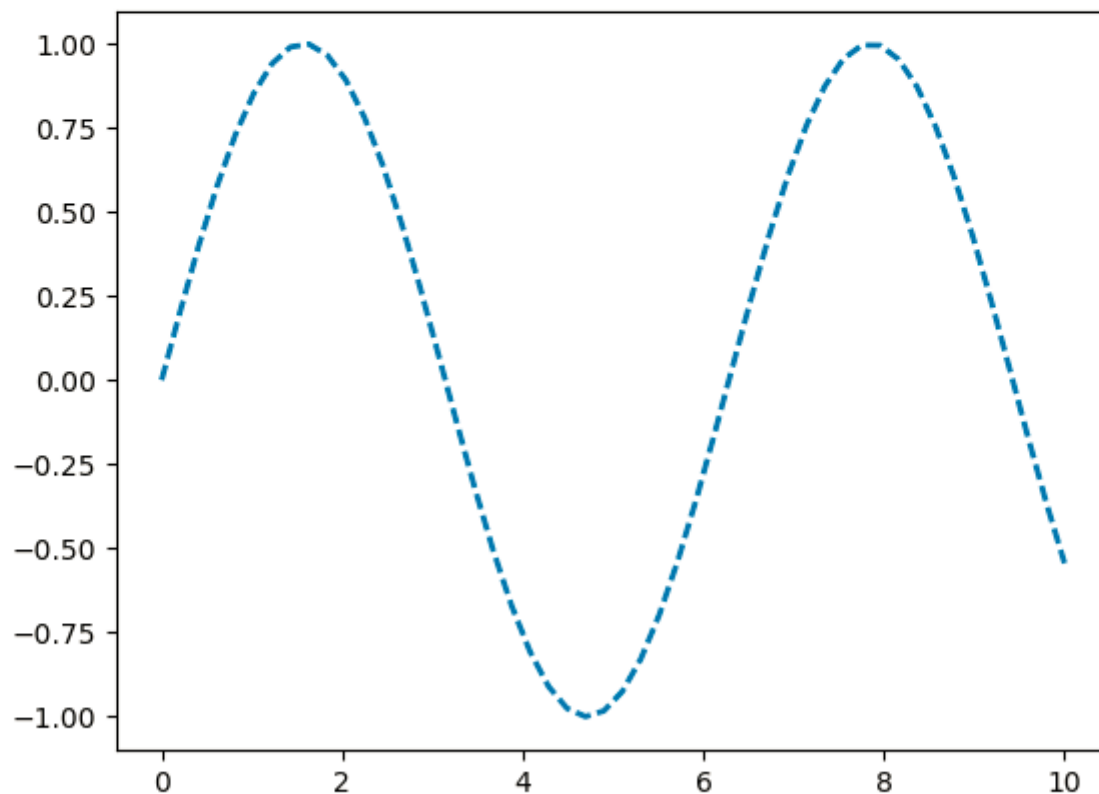
一つの Notebook 上では1回実行すれば OK

sin(x) のグラフ

ドキュメント

```
In [27]: %matplotlib notebook
from matplotlib import pyplot as plt
import numpy as np
x = np.linspace(0,10)
line, = plt.plot(x, np.sin(x), '--', linewidth=2)
```

Figure 1



Stop Interaction

+

コード

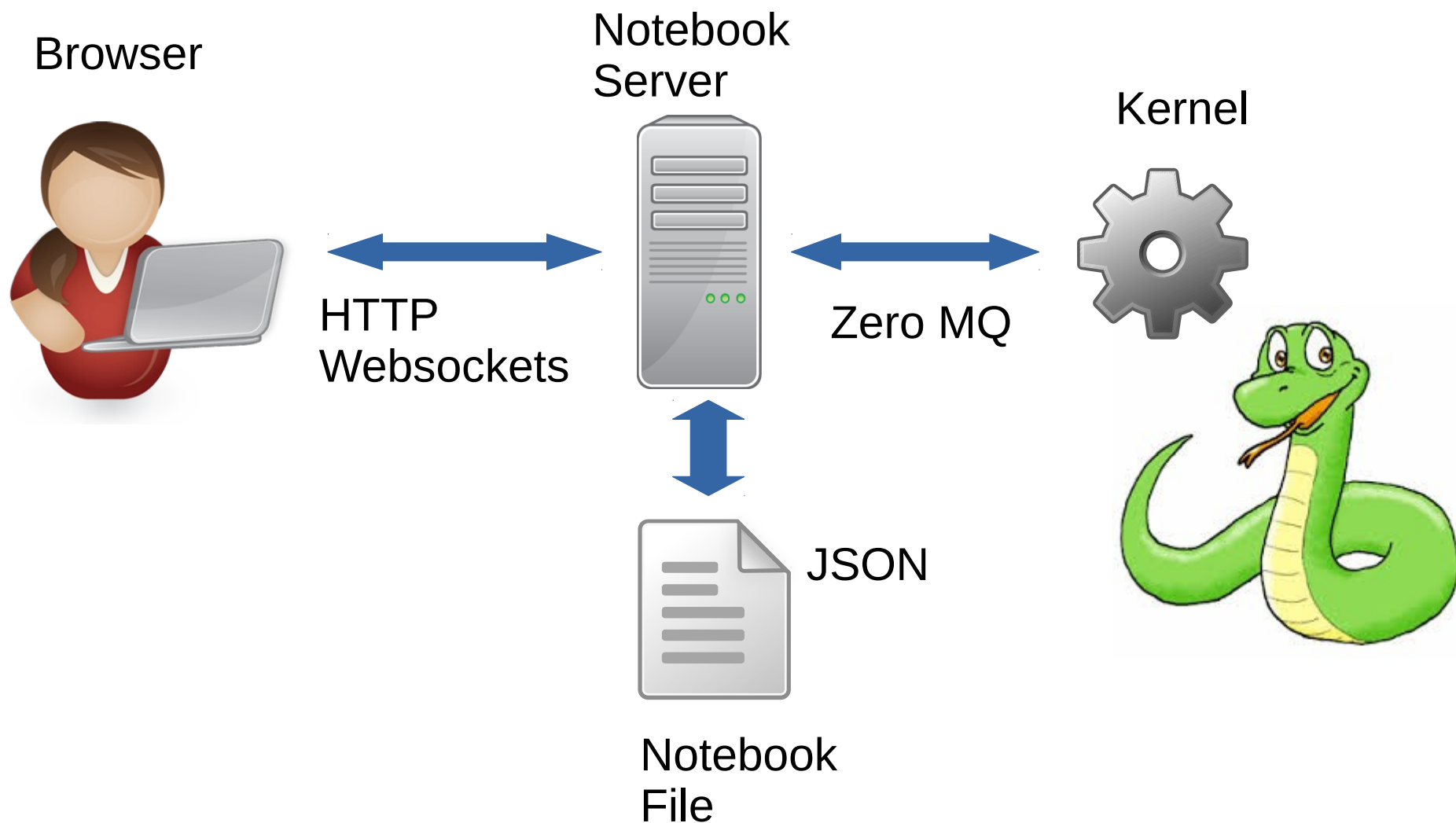
+

実行結果
(可視化)

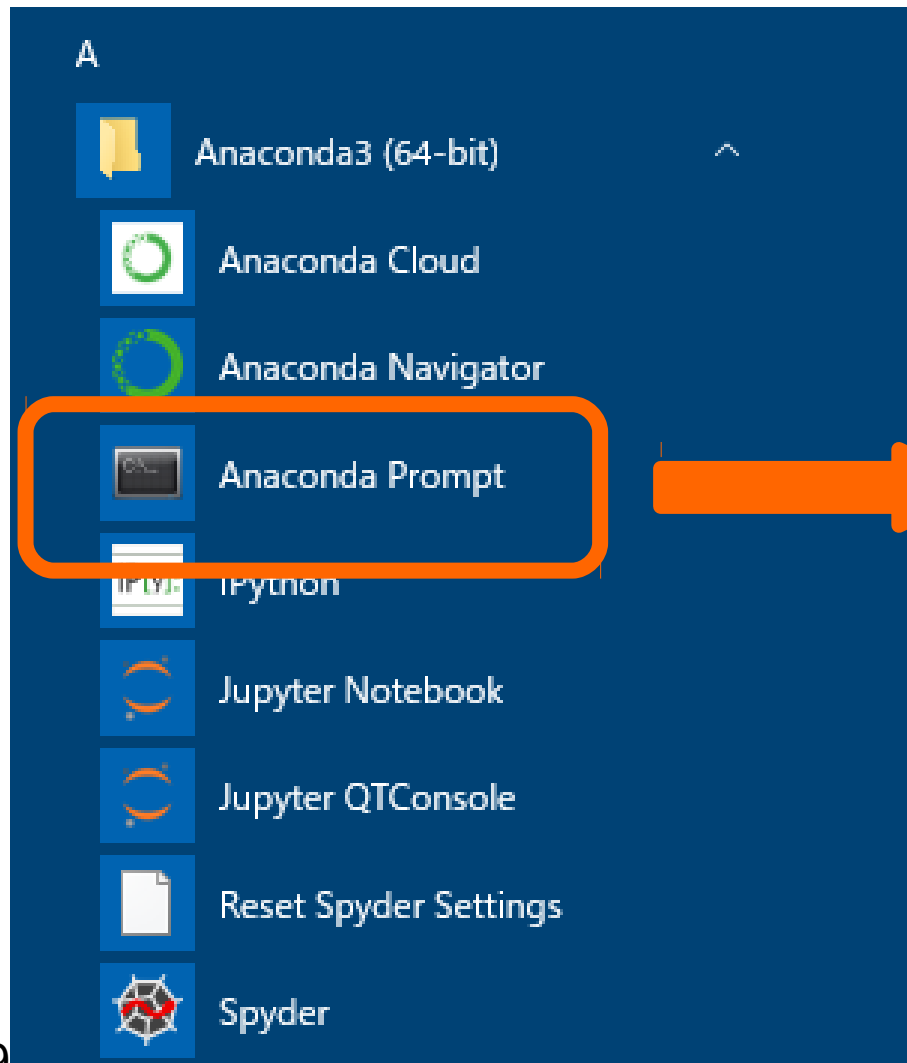
=

Notebook

Jupyter の構造



Python Kernel から MySQL へ



```
>conda install pymysql
```

クエリを実行して結果をフェッチ

```
In [4]: import pymysql
        con = pymysql.connect(host='172.16.111.2',
                               port=3306,
                               user='USER',
                               password='PASS',
                               db='kpistudy',
                               charset='utf8')

        cur = con.cursor()
```

```
In [5]: cur.execute("SHOW tables")
```

```
Out[5]: 3
```

```
In [6]: cur.fetchall()
```

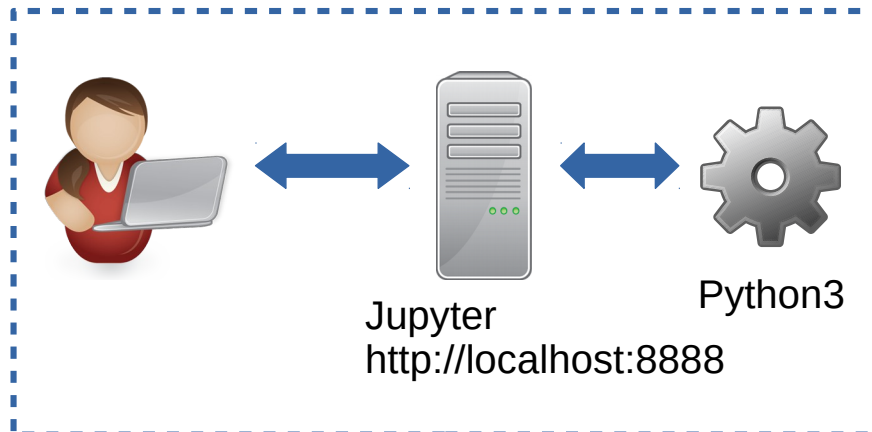
```
Out[6]: (('log_login',), ('tbl_receipt',), ('user_login',))
```

本番 MySQL との連携例

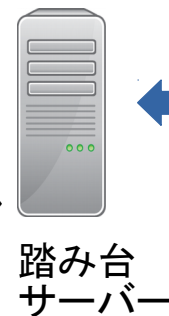
予備スレーブ

- 管理画面
- データ可視化
- OLAP
- ダンプ取得
- スレーブ複製の種
- Jupyter でデータ分析

社内PC



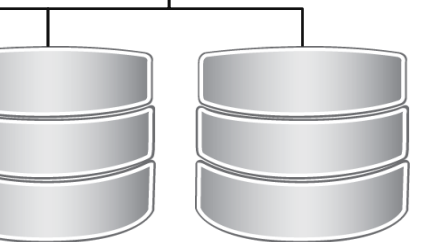
ssh
トンネル



予備
スレーブ

アプリ

マスタ



リードレプリカ

この構成のメリット

- 慣れた SQL で分析
- 最初は全部 SQL で OK

- Python でデータ分析
- DB から PC へ処理を分散
- データ分析から機械学習へと発展

- グラフ化は jupyter 上 matplotlib で一緒

SQL は使ってる
Python は...

これだけ知ってれば OK!

Python のデータ型: リストとタプル

```
In [1]: list1 = [ 'a', 'b', 'c' ]  
list1[2] = 'X'  
list1
```

```
Out[1]: [ 'a', 'b', 'X' ]
```

```
In [2]: tuple1 = ( 'a', 'b', 'c' )  
tuple1[0] = 'OK?'  
tuple1
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-2-9033d2f33d91> in <module>()  
      1 tuple1 = ( 'a', 'b', 'c' )  
----> 2 tuple1[0] = 'OK?'  
      3 tuple1  
  
TypeError: 'tuple' object does not support item assignment
```

リストのスライス

覚え方 最初が0で [以上:未満]

```
In [1]: list1 = ('A', 'B', 'C', 'D', 'E')
```

```
In [4]: list1[0:4]
```

```
Out[4]: ('A', 'B', 'C', 'D')
```

```
In [7]: list1[:4]
```

```
Out[7]: ('A', 'B', 'C', 'D')
```

```
In [9]: list1[3:]
```

```
Out[9]: ('D', 'E')
```

リストとタプルの違い

```
In [37]: b = (1)
         type(b)
```

```
Out[37]: int
```

```
In [43]: a = [1]
         type(a)
```

```
Out[43]: list
```

```
In [38]: c = (1,)
         type(c)
```

```
Out[38]: tuple
```

```
In [41]: c + 1
```

```
TypeError                                Traceback (most recent call last)
<ipython-input-41-993d34e85845> in <module>()
----> 1 c + 1
```

```
TypeError: can only concatenate tuple (not "int") to tuple
```


列データの抜き出し: リスト内包表記

フェッチしたデータは1行が1タプル、全体がタプルのタプル

```
In [17]: fetched_data = ( (1, 'A'), (2, 'B'))
```

[③要素にする変数 **for** ②取り出した変数 **in** ①元リスト]

①、②、③の順で
読んでください

```
[ i for i, _ in fetched_data ] # _ は読み捨てるダミー変数の慣用句
```

```
[1, 2]
```

[④要素にする変数 **for** ②取り出した変数 **in** ①元リスト **if** ③条件]

```
[ j for _, j in fetched_data if _ > 1]
```

```
['B']
```

zip で複数リストの要素を結合

```
In [39]: col1 = (1, 2, 3)
         col2 = ("a", "b", "c")
```

```
In [40]: [(x, y) for (x, y) in zip(col1, col2)]
```

```
Out[40]: [(1, 'a'), (2, 'b'), (3, 'c')]
```

いよいよ ソーシャルゲームの データ分析へ

データ分析とは

- 単変量解析
 - 原因と結果1対1。直感で判定可能
 - 平均値、相関係数、回帰直線
- 多変量解析
 - 多特性データの原因と結果。直感で判定出来ない
 - 重回帰分析、ロジスティック回帰
- データの分析(統計によるモデリング)の用途
 - 分類や未来の予測
 - PDCAやフィードバックに活用

モデルの例：アイス販売

【目的】 アイスの売上を増やしたい

【モデル】 気温が高くなると売れて、値段が高いと売上が減る

【目的変数】 アイスの販売量

【説明変数】 販売量 = 気温 × 値段

【操作変数】 値段

【データ分析】

統計的なモデルを選定（モデル化、データの説明、分類）

気温を条件に売上を最大化する価格設定を見つける（予測）

モデルの例：ゲームアプリの売上

【目的】 ゲームアプリの課金売上を増やしたい

【モデル】 ゲーム自体の魅力(質)、宣伝で売上があがる

【目的変数】 売上 【説明変数】 質 × 宣伝

【データ分析】

質は定量化が困難。「質 = 売上」で説明する程

目的と説明が逆転。説明変数につかえない

売上 = 宣伝？

質(新機能、追加イベント)向上抜きモデル??

統計モデル～ 以前に
定量化可能な説明変数がまず必要

KGI と基本 KPI

目的変数
*KGI

説明変数
基本*KPI

.....

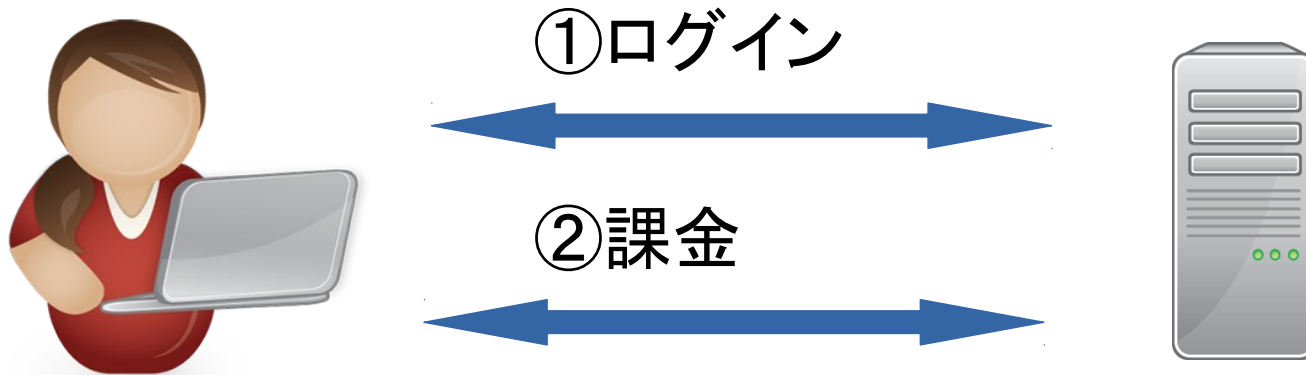
売上 = *DAU x 課金者率 x *ARPPU

定量化は完璧！
ただし操作変数がないのが懸念

- * KGI (Key Goal Index)
- * KPI (Key Performance Index)
- * DAU (Daily Active User)
- * ARPPU (Average Revenue Per Paying User)

KGI と基本 KPI の分析を
進めます

今回の分析対象データ



①ログイン

- ユーザー登録時に記録
- セッション再開毎に記録
- 同じユーザーが1日に何度も
- 約3年で2000万行超のデータ

②課金

- 購入処理毎に記録
- 10万行未満のデータ

スキーマ

log_login		
カラム名	データ型	
login_dt	DATETIME	ログイン日時
user_id	INT	ユーザーID

tbl_receipt		
カラム名	データ型	
user_id	INT	ユーザーID
purchase_dt	DATETIME	購入日時
unit_price	INT	課金額

KGI: 日次の売上

tbl_receipt		
カラム名	データ型	
user_id	INT	ユーザーID
purchase_dt	DATETIME	購入日時
unit_price	INT	課金額

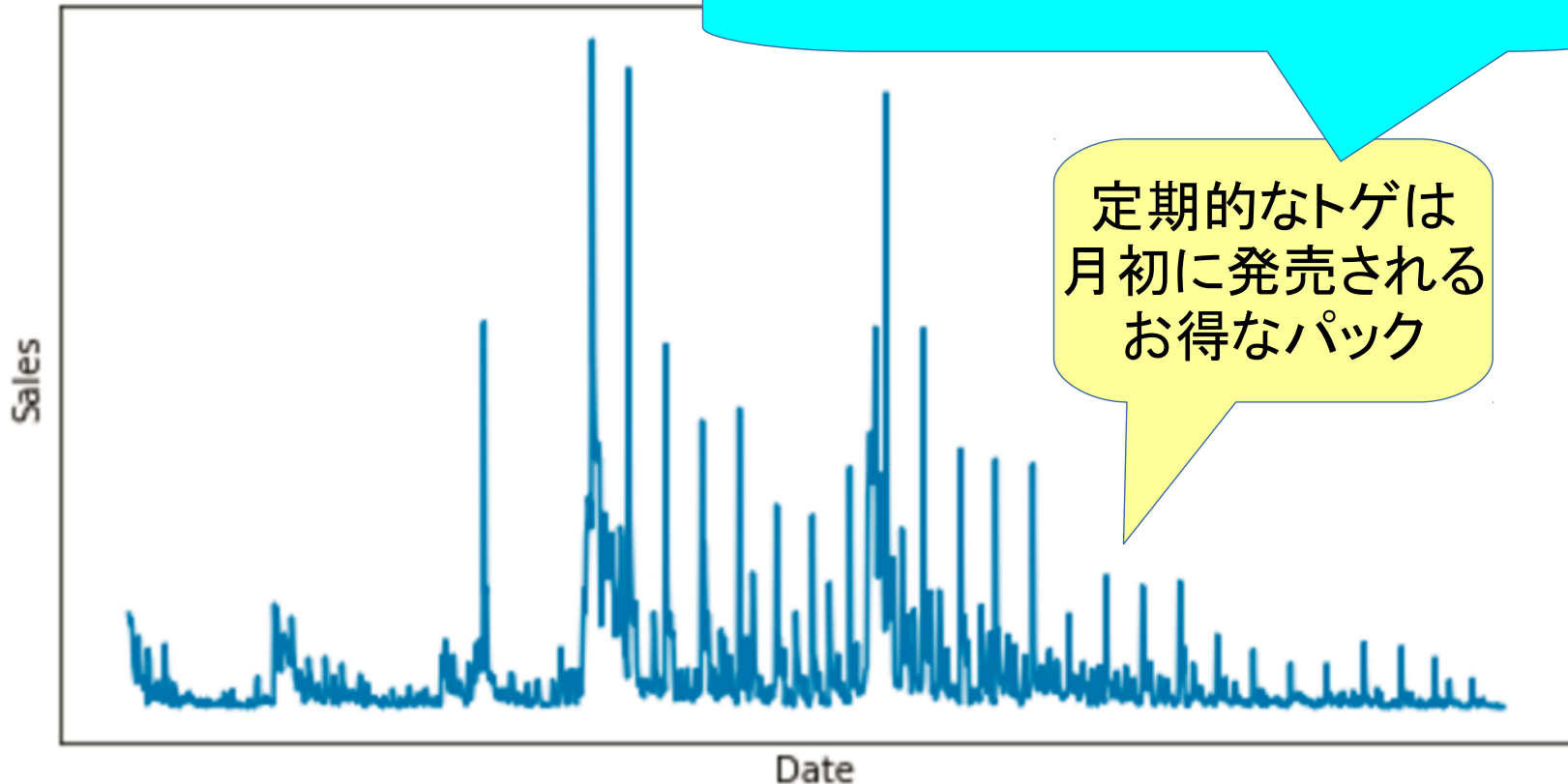
```
SELECT
SUM(unit_price),
DATE(purchase_dt) AS Dt,
FROM tbl_receipt
GROUP BY Dt
ORDER BY Dt
```

```
cur.execute("SELECT SUM(unit_price), DATE(purchase_dt) AS Dt FROM tbl_")
sales = cur.fetchall()
plt.figure(figsize=(8,4))
plt.plot([d for _, d in sales], [s for s, _ in sales])
plt.yticks([], []) # Y軸の目盛を非表示
plt.xticks([], []) # x軸の目盛を非表示
plt.ylabel('Sales')
plt.xlabel('Date')
plt.show()
```

操作の立案にKGI だけで?
もっと効果的な KPI が欲しい

営業策、質の操作と売上が関係している

定期的なトゲは
月初に発売される
お得なパック

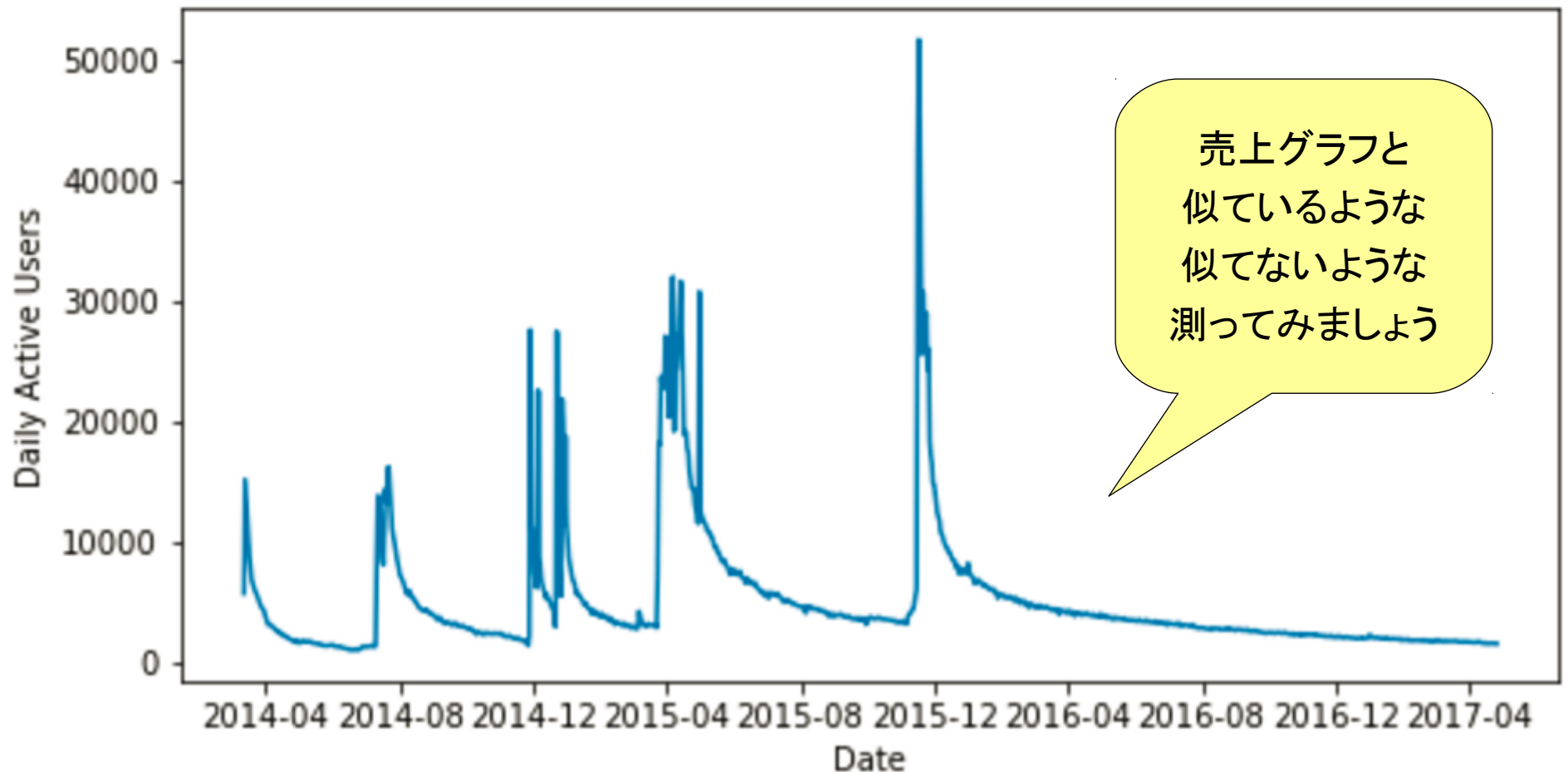


基本KPI: DAU

log_login		
カラム名	データ型	
login_dt	DATETIME	ログイン日時
user_id	INT	ユーザーID

```
SELECT
COUNT(DISTINCT user_id),
DATE(login_dt) AS Dt
FROM log_login
GROUP BY Dt
```

```
plt.figure(figsize=(8,4)) # インチ表記
plt.plot([dt for _, dt in dau], [u for u, _ in dau]) # X軸, Y軸
plt.ylabel('Daily Active Users')
plt.xlabel('Date')
plt.show()
```



売上と DAU の相関を測る

相関係数 (correlation coefficient) = xy の共分散 / (x の標準偏差)(y の標準偏差)
2つの確率変数の間の関係を図る指標

```
import numpy as np
np.corrcoef([int(s) for s, _ in sales], [u for u, _ in dau])[0,1]
```

0.67670239908519525

Guilford's Rule of Thumb

- 0 - 0.2 ほとんど相関なし
- 0.2 - 0.4 弱い相関あり
- 0.4 - 0.7 中程度の相関あり
- 0.7 - 0.9 強い相関あり
- 0.9 - 1 非常に強い相関あり

DAU は定義から
売上と因果関係

にもかからず中程度の相関

基本KPI: ARPPU

さらに平均を求める

日付、ユーザー別売上

```
SELECT
DATE(purchase_dt) AS Dt,
user_id,
SUM(unit_price) AS Uriage
FROM tbl_receipt
GROUP BY Dt, user_id
ORDER BY Dt
```



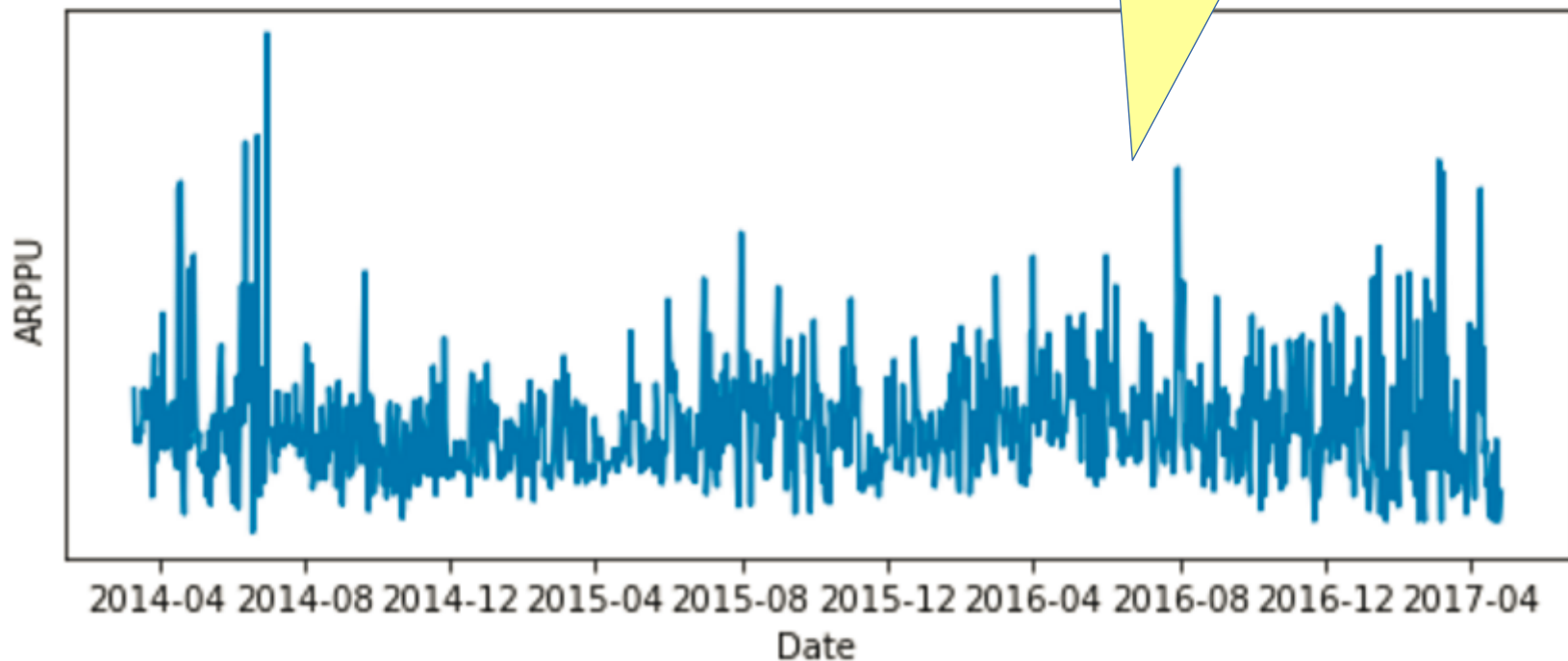
```
SELECT
T.Dt AS Date,
FLOOR(AVG(T.Uriage)) AS ARPPU
FROM (
    SELECT
    DATE(purchase_dt) AS Dt,
    user_id,
    SUM(unit_price) AS Uriage
    FROM tbl_receipt
    GROUP BY Dt, user_id
    ORDER BY Dt
) AS T
GROUP BY Date
```



```
cur.execute("SELECT T.Dt AS Date, FLOOR(AVG(T.Uriage)) AS ARPPU FROM  
arppu = cur.fetchall()
```

```
plt.figure(figsize=(8, 3))  
plt.plot([dt for dt, _ in arppu], [a for _, a in arppu])  
plt.yticks([], [])  
plt.ylabel('ARPPU')  
plt.xlabel('Date')  
plt.show()
```

相関係数 0.168
ほとんど相関なし

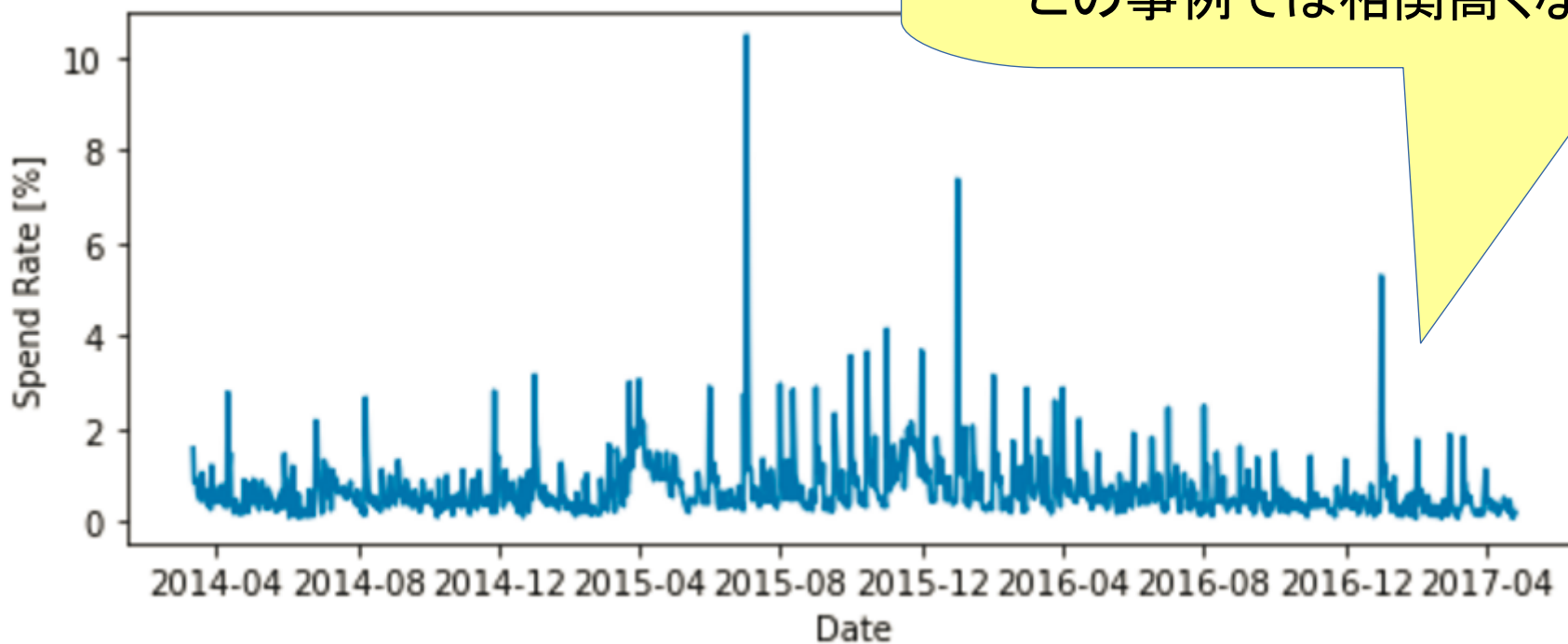


課金者率(スPEND率)

```
spendrate = [s / d / a * 100 for s,d,a in zip([s for s,_ in sales],  
                                              [d for d,_ in dau],  
                                              [a for _,a in arppu])]
```

```
plt.figure(figsize=(8,3))  
plt.plot([dt for _,dt in sales], spendrate)  
plt.ylabel('Spend Rate [%]')  
plt.xlabel('Date')  
plt.show()
```

相関係数 0.623
基本 KPI で重要とも言われる
この事例では相関高くない



基本 KPI は相関が不十分

(基本 KPI はターゲットが売上と同じく全ユーザーのまま)

相関が高いか、
操作の立案や評価に繋がる
よりよい KPI を模索

なにか見たいグラフ
ないですか？

初回ログイン日とそのユーザーのプレイ期間

```
SELECT
    user_id,
    DATEDIFF(MAX(login_dt), MIN(login_dt))
FROM log_login
GROUP BY user_id;
```

- log_login は2200万行+
- 一方全ユーザーは80万未満なのでユーザー別なら少量データ
- ユーザー別集計テーブル user_login を作成

Field	Type	Null	Key	Default	Extra
user_id	int(11)	NO	PRI	NULL	
first_login	datetime	YES		NULL	
last_login	datetime	YES		NULL	

```
mysql> insert into user_login select user_id, MIN(login_dt), MAX(log  
in_dt) FROM log_login GROUP BY user_id;
```

初回ログイン日とそのユーザーのプレイ期間

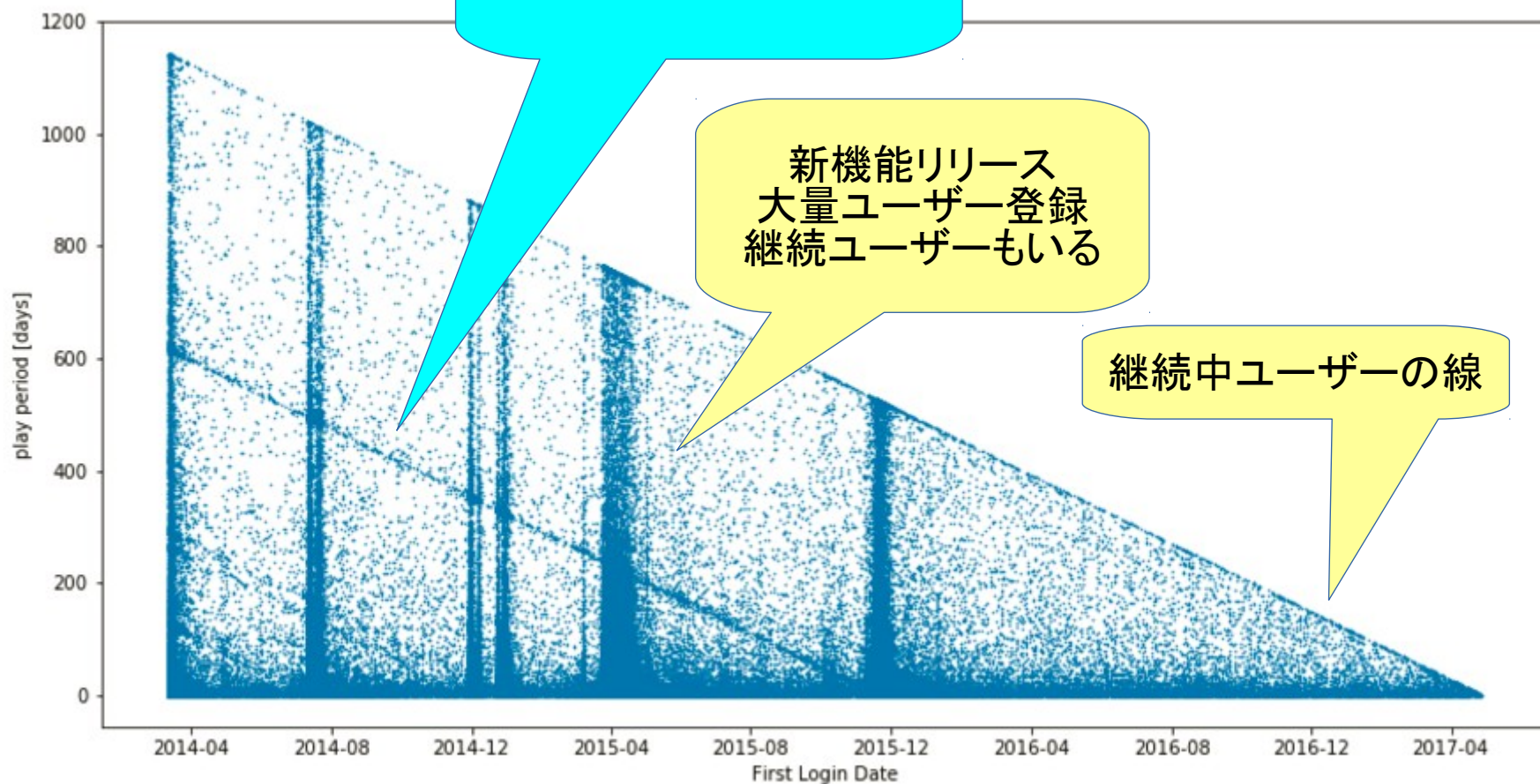
```
SELECT  
    first_login,  
    DATEDIFF(last_login, first_login),  
    user_id  
FROM user_login  
WHERE DATEDIFF(last_login,
```

復帰キャンペーンで
継続ユーザーが大量離脱？
え？課金者はどう？

継続ユーザーと同じ傾き
この線は？

新機能リリース
大量ユーザー登録
継続ユーザーもいる

継続中ユーザーの線



課金ユーザーの初回ログイン日とそのプレイ期間

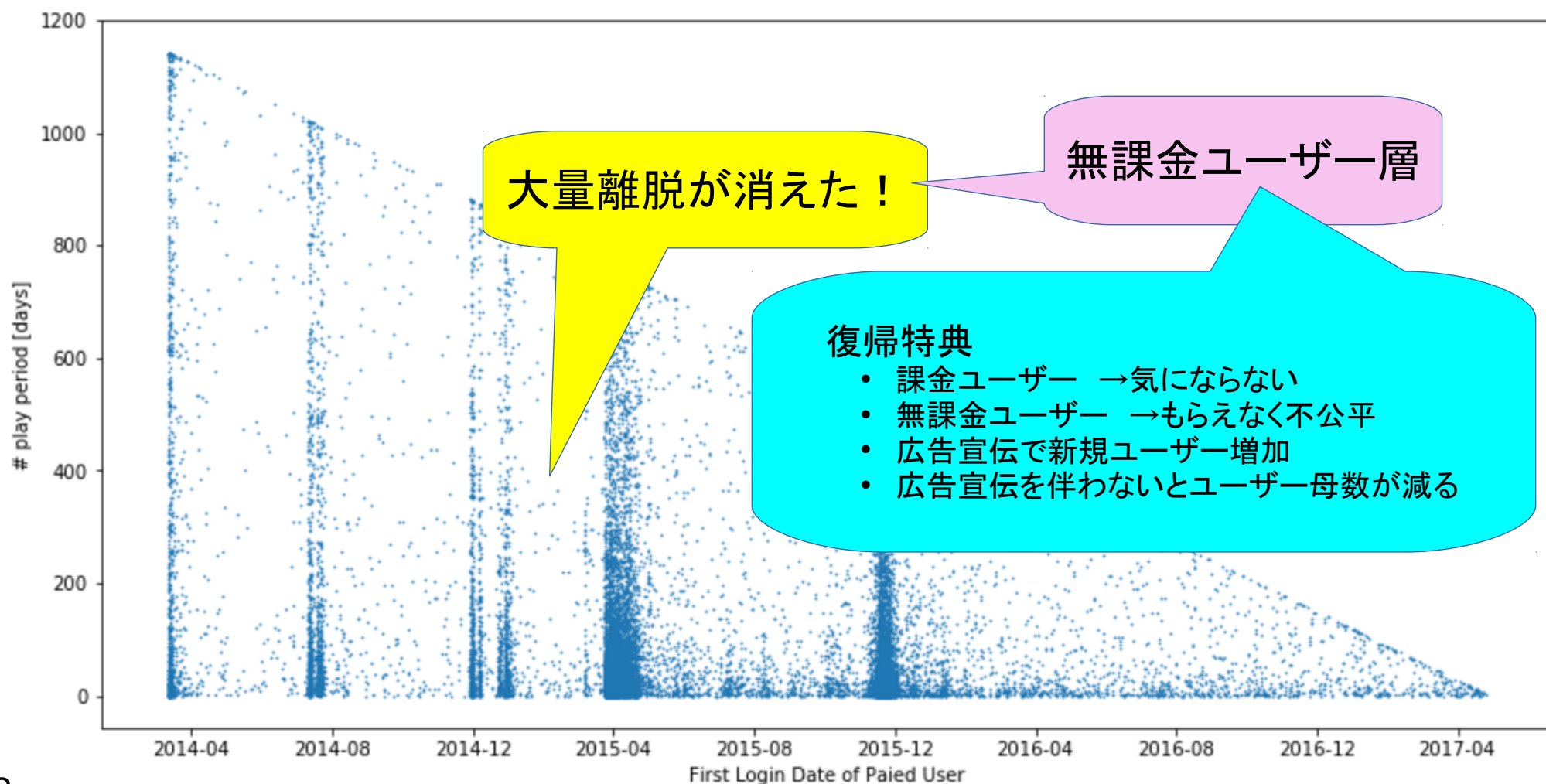
```
SELECT
    first_login,
    DATEDIFF(last_login, first_login)
FROM user_login AS u
    INNER JOIN
        (SELECT DISTINCT user_id FROM tbl_receipt) AS r
    ON u.user_id = r.user_id
```



SQL で簡単に書けます！！

課金ユーザーの初回ログイン日とそのプレイ期間

```
plt.scatter([dt for dt, _ in playperiodpu], [days for _, days in playperiodpu], marker='.', s=1) # 点グラフ, 点のサイズ = 1
plt.ylabel('# play period [days]')
plt.xlabel("First Login Date of Paied User")
plt.show()
```



見たかったグラフ
「初回ログイン日とプレイ期間」は
ユーザー動向把握に有効

ただ KGI と KPI からは脱線だった
引き続き探ります

売上なので...初心にかえり金額で

課金ランキングとその金額

```
SELECT
    user_id,
    SUM(unit_price) as Sougaku
FROM tbl_receipt
GROUP BY user_id
ORDER BY Sougaku DESC
```

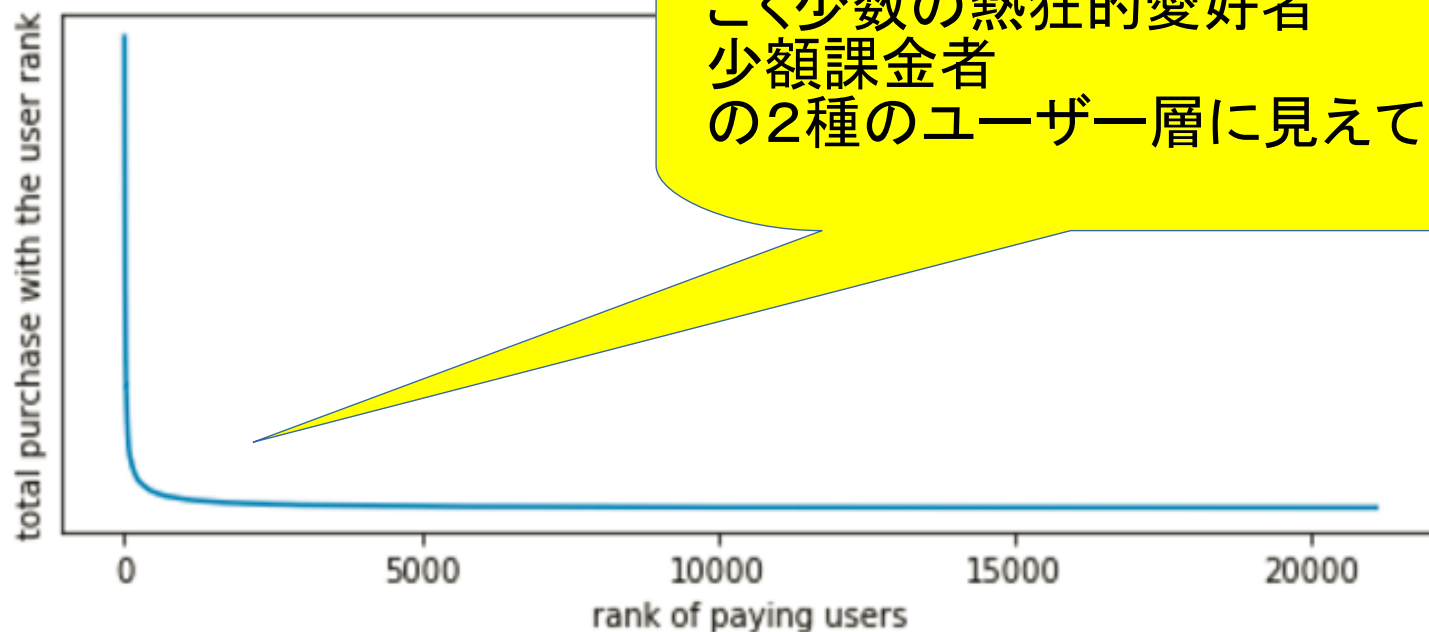
```
In [126]: cur.execute("select user_id, sum(unit_price) as Sougaku  
totalpay = cur.fetchall()
```

```
plt.figure(figsize=(8,3))  
plt.plot([p for _, p in totalpay])  
plt.ylabel('total purchase with  
plt.xlabel('rank of paying user  
plt.yticks([], [])  
plt.show()
```

プレイ継続して高額に
なっているのでは？

総課金額だと偏りが強烈

ごく少数の熱狂的愛好者
少額課金者
の2種のユーザー層に見えてしまう



課金ランキングとプレイ期間から プレイ中の平均日次課金額

SELECT

FLOOR(Sougaku / period) AS DRPU

FROM (

SELECT

user_id,

SUM(unit_price) as Sougaku

FROM tbl_receipt

GROUP BY user_id

ORDER BY Sougaku DESC) AS S

ユーザー毎
総課金額、ランキング順

INNER JOIN (

SELECT

u.user_id as id,

DATEDIFF(last_login, first_login) as period 課金ユーザーのプレイ期間

FROM user_login AS u

INNER JOIN (

SELECT DISTINCT user_id FROM tbl_receipt) AS r

ON u.user_id = r.user_id

WHERE DATEDIFF(last_login, first_login) > 0) AS P

ON S.user_id = P.id

課金者のプレイ中の日次課金ランキング

```
plt.figure(figsize=(8,3))  
plt.plot([i for i in drpu])  
plt.ylabel('Revenue with User / playing period')  
plt.xlabel("User Spend Ranking")  
plt.show()
```



売上に貢献している上位者層を特定

```
totalpay # 総課金額
totalrevenue = sum([p for _, p in totalpay])

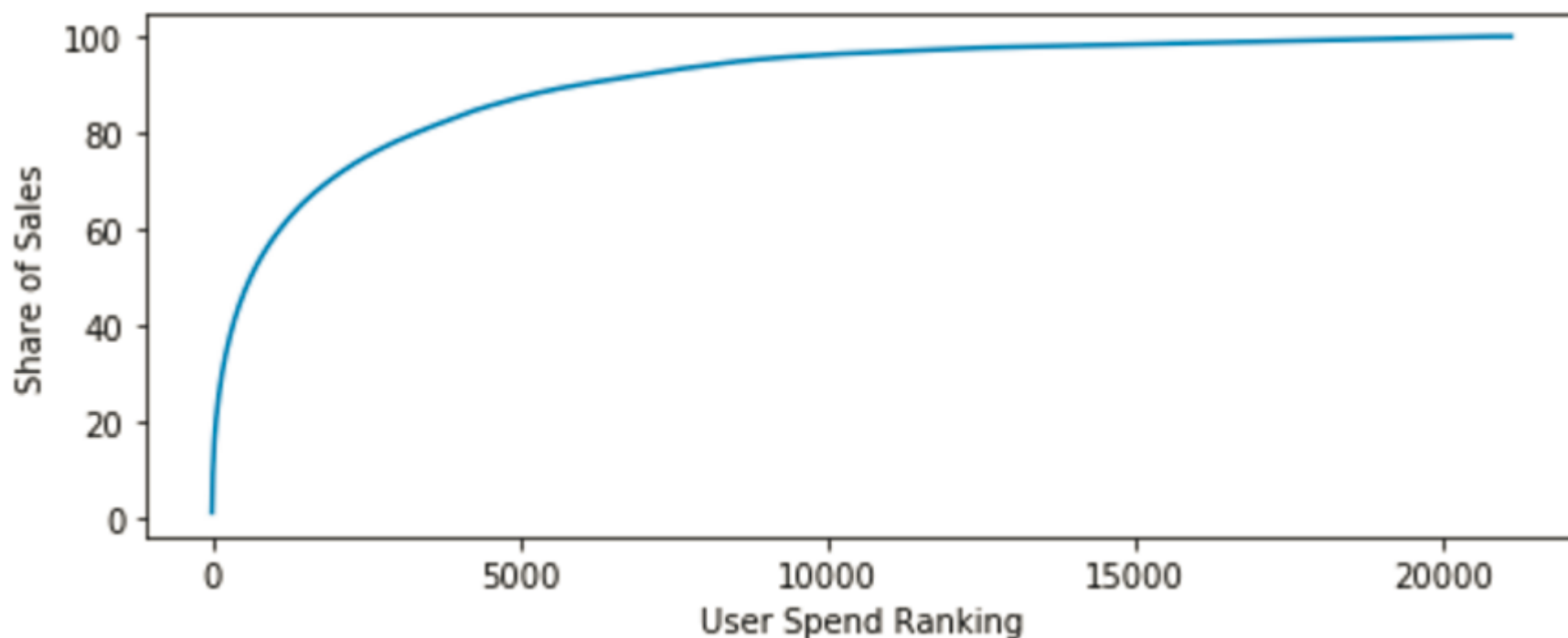
subtotal = 0
p80 = 0
shareofsales = []
for i in [p for _, p in totalpay]:
    subtotal += i
    shareofsales.append((subtotal, subtotal/totalrevenue*100))
    # おまけで占有が8割を超えたらランキングをプリント
    if subtotal/totalrevenue > 0.8 and p80 == 0:
        p80 = len(shareofsales)
        print(p80)
```

3326

課金上位 3326 位で売上の8割貢献

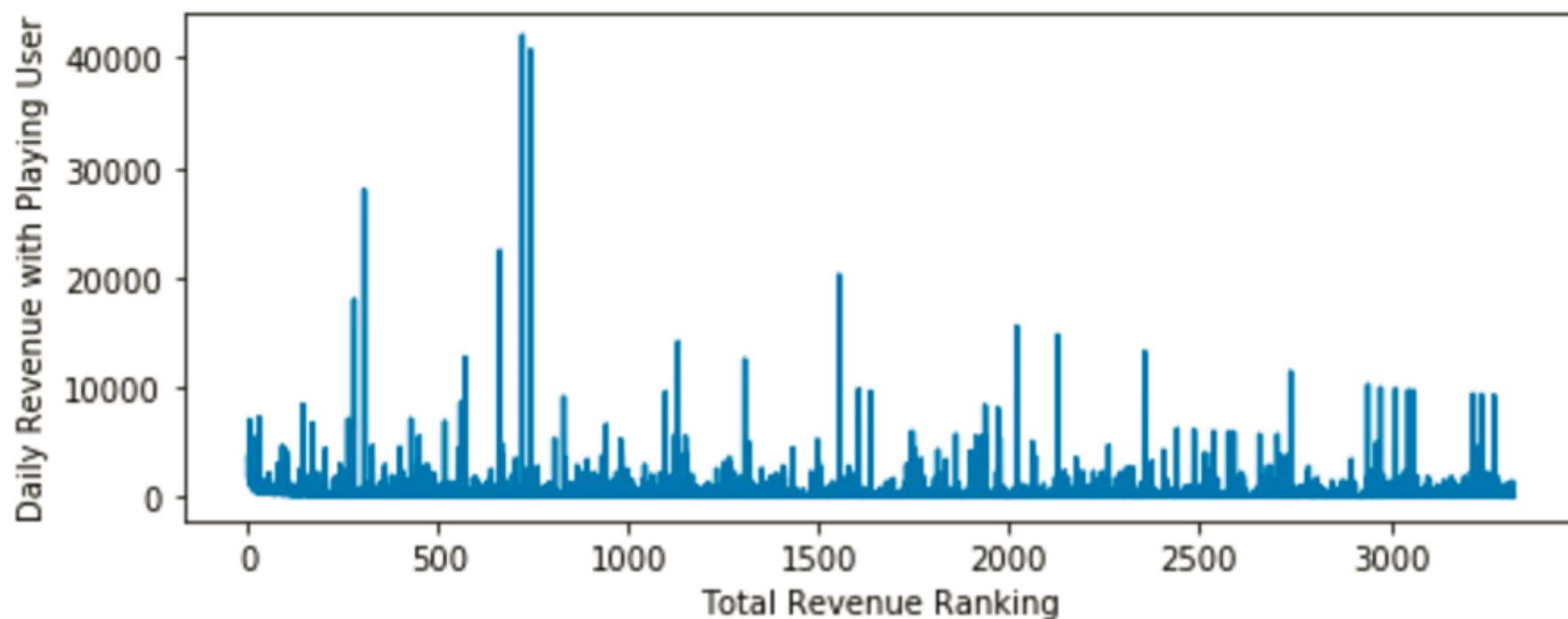
課金ランキングと売上貢献度

```
plt.figure(figsize=(8, 3))  
plt.plot([p for _, p in shareofsales])  
plt.ylabel('Share of Sales')  
plt.xlabel("User Spend Ranking")  
plt.show()
```



上位者層のプレイ中の平均日次課金額

```
plt.figure(figsize=(8, 3))  
plt.plot([i for i in dru[0:3326]])  
plt.ylabel('Daily Revenue with Playing User')  
plt.xlabel('Total Revenue Ranking')  
plt.show()
```



そのデータの特徴を調べる

```
topdailypay = sorted([ i[0] for i in drpu[0:3326]]) # 昇順にソート
```

```
min(topdailypay), max(topdailypay)
```

```
(Decimal('8'), Decimal('42080'))
```

```
import statistics as stat  
stat.mean(topdailypay) # 平均
```

```
Decimal('629.9329524954900781719783524')
```

```
# 中央値, 最頻値, 標準偏差
```

```
stat.median(topdailypay), stat.mode(topdailypay), stat.stdev(topdailypay)
```

```
(Decimal('195.5'), Decimal('20'), Decimal('1757.819299569603193580973526'))
```

「売上の8割を支えるトップユーザー」
だけではまだバラつきが激しい

スパイクを除外する四分位範囲を適用

```
n = len(topdaily pay)
# 四分位範囲 上下 25% の値をカット
interquarter = topdaily pay[ int(n * 1/4) : int(n * 3/4)]
stat.stdev(interquarter)
```

```
Decimal('134.1066537671433402139525056')
```

```
stat.mean(interquarter), stat.median(interquarter)
```

```
(Decimal('235.1467227901383042693926639'), Decimal('195'))
```

```
min(interquarter), max(interquarter)
```

```
(Decimal('73'), Decimal('557'))
```

平均と中央値に近い
トップ層をよくモデル化？

優良課金ユーザーを月単位で

```
drpufixed = [i for i in drpu[0:3326] if i[0] >= 73 and i[0] <= 557]
```

```
# トップ課金ユーザーの月額モデル
```

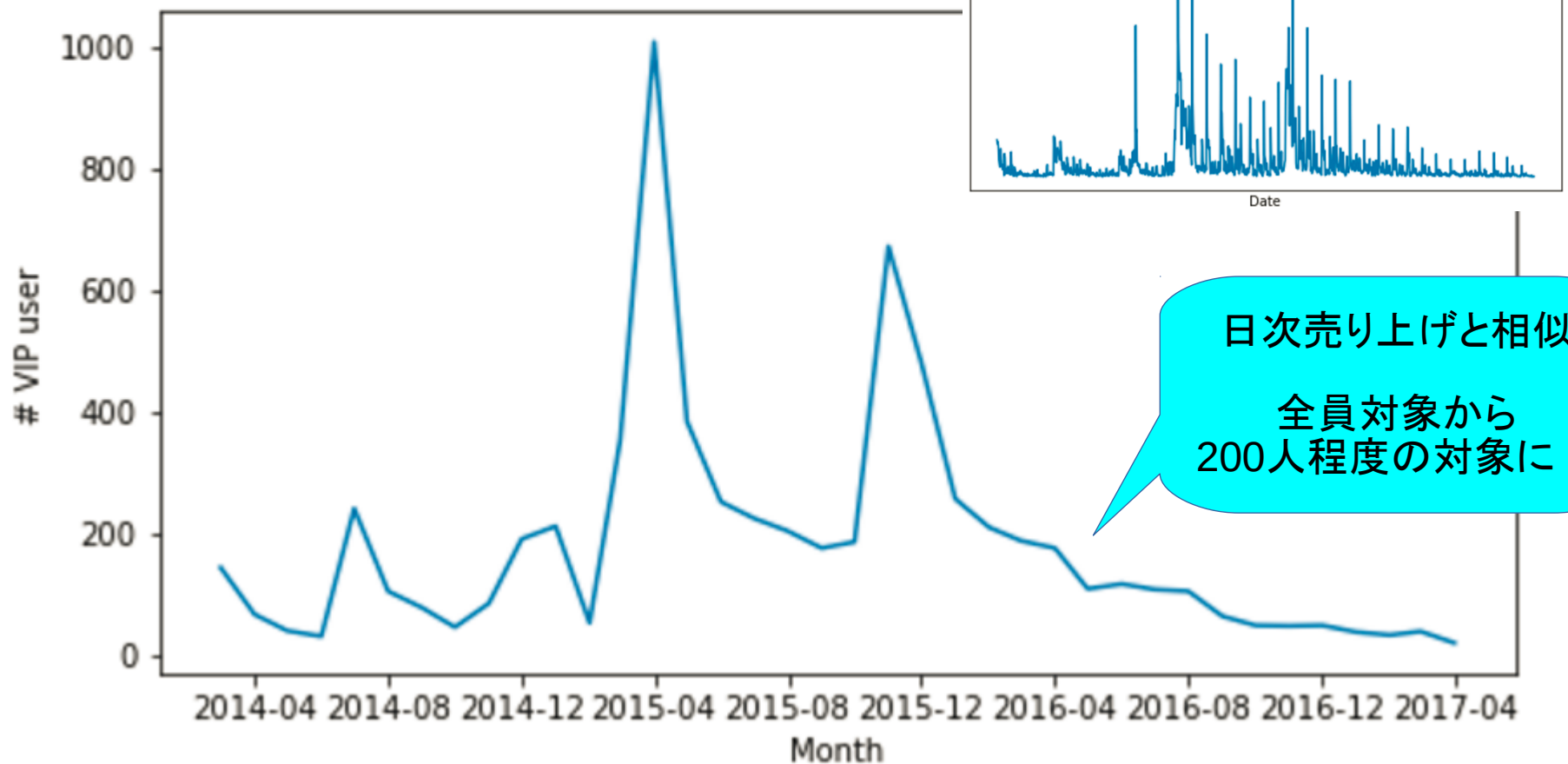
```
mtarget = sum([i[0] for i in drpufixed]) / len(drpufixed) * 30
```

```
SELECT
    count(a.user_id) AS num_good,
    month,
    SUM(kakin)
FROM (SELECT user_id,
    CAST(DATE_FORMAT(purchase_dt, '%Y-%m-01') AS DATE) AS month,
    SUM(unit_price) AS kakin
    FROM tbl_receipt
    GROUP BY user_id, month
    HAVING kakin >= ここに mtarget) a
GROUP BY month
ORDER BY month
```

営業策(イベント)は月単位
月単位の KPI に変換

優良課金ユーザー数

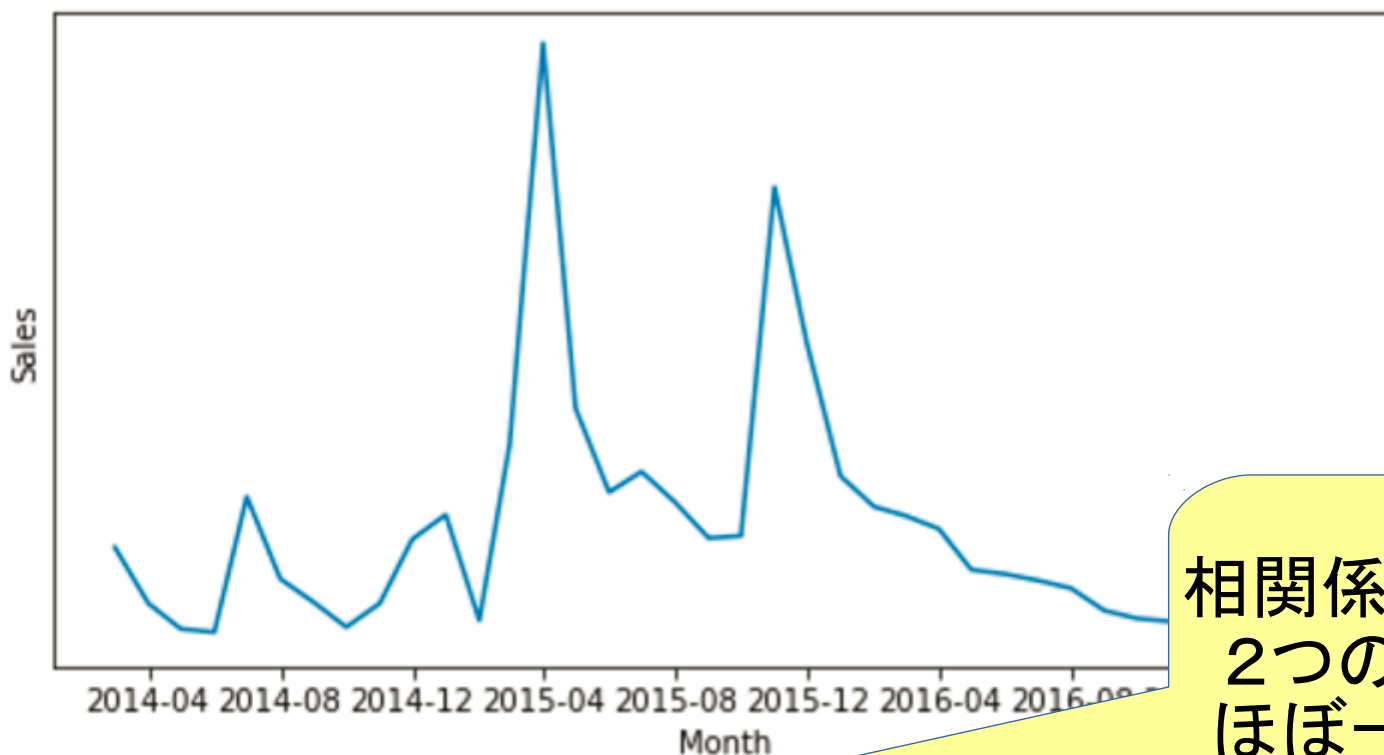
```
plt.figure(figsize=(8, 4))  
plt.plot([m for _, m, _ in good_user], [n for n, _, _ in good_user])  
plt.ylabel('# VIP user')  
plt.xlabel("Month")  
plt.show()
```



優良ユーザー数と月次売上との相関

```
SELECT SUM(unit_price), CAST(DATE_FORMAT(purchase_dt, '%Y-%m-01') AS DATE) AS Month  
FROM tbl_receipt GROUP BY Month ORDER BY Month")
```

```
np.corrcoef([int(s) for s, _ in msales], [n for n, _, _ in good_user])[0,1]
```



相関係数 0.995
2つの変数が
ほぼ一致！！

```
Out[38]: 0.99520517736351888
```

全員での売上(金額)と
優良ユーザー数(人数)が一致
ターゲット絞り込み成功

この事例では KPI として期待！

まとめ

- Jupyter Notebook

MySQL と接続、SQLとPythonで手軽にデータ分析

- ソーシャルゲームのデータ分析

弊社の例

月額課金である額以上の優良ユーザー数が売上と相関

操作(質の改善、宣伝)を売上や課金率で PDCA

↓

絞り込んだユーザー層できめ細かい行動分析が可能
操作の立案に有効

データ分析、ビジネス固有の KPI から初めてみましょう！！

ご清聴ありがとうございました！

ご質問は Twitter : @nobuhatano へ