

Data Scienceのための Python, その先へ

ゼロからのpandasの使い方とPySparkへの架け橋

株式会社ネットプロテクションズ 関西開発インターン生 上野孝斗

本日の内容

- 自己紹介
- Pythonについて
- Pandasの実際
- Apache Sparkについて
- Pysparkについて
- Pysparkの実際
- PandasとPysparkの比較
- 参考

注意

画像はの引用先は全て同スライドにある参考文献の番号に対応しています

自己紹介

- NPには6月からジョイン
- 担当業務はAFTEE (DS)
- 大学の専攻は強化学習[1], ベイズモデリング
- 趣味は散歩, ゲーム, 映画鑑賞

Pythonについて

Pythonの概要

- Python[2]は1990年代初期オランダで生まれたプログラミング言語
- その習得のしやすさから、入門用として使われる
- 一方、企業でもよく使われている

Pythonの主な利用用途

- データサイエンスおよび数値計算
- Webアプリケーション
- システム管理及びグルー用語
- 教育

フリートーク

- お題：Pythonライブラリについて
- 知っているものでも、好きなものでも
- 実はR派...(ボツッ)みたいなのも

有名なライブラリは数あれどやはり…

- Pythonのライブラリといえばpandas!!(多分)
- ...pandasについて少し紹介します

pandasについて

pandasの概要

- pandas[3]はPythonのライブラリの一つ
- PythonでDataScienceするときはまずはとりあえずpandas
- `pandas.DataFrame` オブジェクトによって、データフレームを作ることができる

補足：データフレームについて

- ・ さまざまな種類のデータフレームが存在(dask,pandas,pickle,parquet...)する
- ・ ざっくり言うと、データフレーム=二次元の表形式のデータ
- ・ 言い換えると、テーブルデータ、ラベル付き二次元配列

データフレームをpandasで作成

- pandasでデータフレームを作成する
- 10 minutes to pandas[4]より

```
In [3]: import pandas as pd
import numpy as np

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

dates = pd.date_range("20130101", periods=6)
df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list("ABCD"))
df
```

```
Out[3]:
```

	A	B	C	D
2013-01-01	0.155043	-0.429780	0.355909	1.094901
2013-01-02	0.518754	-0.619981	-0.352176	-0.214749
2013-01-03	-1.407045	-1.110255	0.244740	-1.343993
2013-01-04	-0.658387	-0.032708	-0.359375	-0.966246
2013-01-05	0.199535	1.237640	-0.302894	-1.948452
2013-01-06	-0.101026	-2.905657	-0.474019	2.090126

pandasでできること

抽出 (WHERE)

- [6]を全体的に参考にした
- まずはデータを作る

```
In [4]: pdf = pd.DataFrame({  
    'student': ['Larissa', 'Jaylan', 'Golda', 'Myriam', 'Fabiola', 'Abigail', 'Hardy', 'Jeromy'],  
    'class': list('ABAACCBC'),  
    'score': [50, 69, 68, 70, 82, 57, 88, 93],  
})  
pdf
```

```
Out[4]:
```

	student	class	score
0	Larissa	A	50
1	Jaylan	B	69
2	Golda	A	68
3	Myriam	A	70
4	Fabiola	C	82
5	Abigail	C	57
6	Hardy	B	88
7	Jeromy	C	93

- このデータからA組のテスト結果の点数と生徒の名前を抽出する。もしSQLでやるならこのようになる。

```
SELECT id, student, class
FROM test_result
WHERE class == 'A'
```

- pandasを使った場合は以下のように書ける

```
In [5]: pdf.loc[pdf['class'] == 'A', ['student', 'score']]
```

```
Out[5]:
```

	student	score
0	Larissa	50
2	Golda	68
3	Myriam	70

pandasでは、`Dataframe.loc` で行について操作することができるようになる

複数の条件で抽出

- 複数の条件を抽出したいときは `&` や `|` を用いる。`and` や `or` ではエラーが出るので注意
- `boolean`型でも抽出ができる
- 「C組のうち、80点以上の生徒」という複数の条件で抽出する

```
In [6]: pdf.loc[(pdf['class'] == 'C') & (pdf['score'] >= 80)]
```

```
Out[6]:
```

	student	class	score
4	Fabiola	C	82
7	Jeromy	C	93

結合 (JOIN)

- student, attendance と二つのテーブルを用意する

In [4]:

```
# [7]を参考
from IPython.display import HTML
from jinja2 import Template

student = pd.DataFrame({
    'name': ['Larissa', 'Jaylan', 'Golda', 'Myriam', 'Fabiola', 'Abigail', 'Bernice', 'Hardy', 'Jero
    'class': list('ABAACCBBC'),
}, index=['A1', 'B1', 'A2', 'A3', 'C1', 'C2', 'B2', 'B3', 'C4'])

attendance = pd.DataFrame({
    'student_id': ['C2', 'C1', 'C4', 'A1', 'C2', 'B2', 'A1', 'B1', 'B3'],
    'subject': ['History', 'Japanese', 'English', 'math', 'math', 'History', 'math', 'math', 'Hist
})
```

In [5]:

```
html_tpl = """
<table>
<tr>
<td>{{ student }}</td>
<td>{{ attendance }}</td>
</tr>
</table>
"""

tpl = Template(html_tpl)
html_text = tpl.render({"attendance": attendance.to_html(), "student": student.to_html()})

HTML(html_text)
```

Out [5]:

	name	class	student_id	subject
A1	Larissa	A	0	C2 History
B1	Jaylan	B	1	C1 Japanese
A2	Golda	A	2	C4 English
A3	Myriam	A	3	A1 math
C1	Fabiola	C	4	C2 math
C2	Abigail	C	5	B2 History
B2	Bernice	B	6	A1 math
B3	Hardy	B	7	B1 math
C4	Jeromy	C	8	B3 History

- INNER JOIN はSQLだと以下のようになる

```
SELECT *
FROM attendance INNER JOIN student
    ON attendance.student_id = student.id
```

- pandasではこのようになる

```
In [9]: pd.merge(attendance, student, left_on='student_id', right_index=True)
```

```
Out[9]:
```

	student_id	subject	name	class
0	C2	History	Abigail	C
4	C2	math	Abigail	C
1	C1	Japanese	Fabiola	C
2	C4	English	Jeromy	C
3	A1	math	Larissa	A
6	A1	math	Larissa	A
5	B2	History	Bernice	B
7	B1	math	Jaylan	B
8	B3	History	Hardy	B

- 続いて **LEFT JOIN** をやってみる
 - 「B組の生徒について、歴史（History）への出席情報」を取得する
 - SQLでは以下のようになる

```
SELECT *
FROM student LEFT JOIN attendance
  ON student.id = attendance.student_id
WHERE student.class = 'B' AND attendance.subject = 'History'
```

- pandasでは次のようにする

```
In [10]: # 各DataFrameからB組、およびHistoryの出席情報をそれぞれ抽出  
b_student = student.loc[student['class'] == 'B']  
history_attendance = attendance.loc[attendance['subject'] == 'History']  
  
# LEFT JOINを実行  
pd.merge(  
    b_student,  
    history_attendance,  
    left_index=True,  
    right_on='student_id',  
    how='left',  
)
```

```
Out[10]:
```

	name	class	student_id	subject
NaN	Jaylan	B	B1	NaN
5.0	Bernice	B	B2	History
8.0	Hardy	B	B3	History

- `pd.merge()` では、結合に使う列名を指定できる
- 列名の左右について注意が必要[6]

```
左側に指定した「attendance」は  
結合に「student_id」の値を使う  
  
右側に指定した「student」は  
結合にIndexの値を使う  
  
pd.merge(attendance, student, left_on='student_id', right_index=True)  
  
結合する2つのDataFrame  
指定する順番（左右）に注意。  
結合に使う列を指定するオプションに影響する。  
  
結合に使う列名  
left_onは左側に指定したDataFrameの、  
right_onは右側に指定したDataFrameの列名を指  
定する。  
  
結合にIndexの値を使う  
結合にIndexの値を使いたい場合は、right_index  
もしくはleft_indexをTrueに設定する。
```

集約 (GROUP BY)

以下のような test_result テーブルを例にする

```
In [11]: test_result = pd.DataFrame({  
    'name': np.sort(['Larissa', 'Jaylan', 'Golda', 'Myriam', 'Fabiola', 'Abigail', 'Bernice', 'Harc  
    'class': np.sort(list('AABBCC') * 3),  
    'subject': ['math', 'English'] * 9,  
    'score': [50, 72, 96, 74, 66, 65, 51, 65, 85, 69, 53, 76, 44, 77, 56, 56, 52, 68],  
})
```

- 集約関数が一つのとき
- 例えば、「クラスごと、教科ごとの平均点」を求めてみる
- SQLでは以下のようになる

```
SELECT class, subject, avg(score)
FROM test_result
GROUP BY class, subject
```

```
In [34]: test_result.groupby(['class', 'subject']).mean()
```

```
/var/folders/nr/8kygzlbx7djc1yh15qnzhhgr0000gn/T/ipykernel_7691/2146  
501525.py:1: FutureWarning: The default value of numeric_only in DataFrameG  
roupBy.mean is deprecated. In a future version, numeric_only will default to Fals  
e. Either specify numeric_only or select only columns which should be valid for t  
he function.
```

```
test_result.groupby(['class', 'subject']).mean()
```

```
Out[34]:
```

		score
class	subject	
A	English	70.333333
	math	70.666667
B	English	70.000000
	math	63.000000
C	English	67.000000
	math	50.666667

- 集約関数が複数ある場合
- 平均だけでなく、最大点も表示したいときがある
- その場合は DataFrameGroupBy オブジェクトの agg() メソッドを使うことによって実現できる

```
In [33]: grouped = test_result.groupby(['class', 'subject'])
grouped.agg(['mean', 'max'])
```

```
/var/folders/nr/8kygzlbx7djc1yh15qnzhgr0000gn/T/ipykernel_7691/3838
957712.py:2: FutureWarning: ['name'] did not aggregate successfully. If any er
ror is raised this will raise in a future version of pandas. Drop these columns/ops
to avoid this warning.
grouped.agg(['mean', 'max'])
```

```
Out[33]:
```

		score	
		mean	max
class	subject		
A	English	70.333333	74
	math	70.666667	96
B	English	70.000000	76
	math	63.000000	85
C	English	67.000000	77
	math	50.666667	56

pandasの課題点

- pandasはものすごく便利だけど
- ビッグデータではメモリエラーが出る／処理が遅い
- SQLに習熟している場合は文法がわかりにくいことも

フリートーク（その2）

- メモリエラーを起こさずにデータを扱うにはどんな方法があるか

pandasからApache Sparkへ

- Apache Sparkを使えば、場合によってはpandasの100倍早く処理を行うことができる

Apache Sparkについて

Apache Sparkの概要

- 並列分散処理を行えるデータ処理ツール
- 大規模データ分析を行う際に役立つ！

Python + Apache Spark= ?

- PythonでApache Sparkがもし使えたなら...
- 学習しやすい、使いやすい！！
- もちろんJupyter Lab (Notebook) で使える！
- それがPySpark!![5]



PySparkについて

PySparkの概要

- Sparkをpython上で実行するためのAPI
- python以外にも実はsparkにはたくさんのAPIがあり、たくさんの言語に対応している。

PySparkでできること

PySparkとPandasの互換性

- pandasのDFとPySparkのDFは相互に変換することができる
- 実際のところは、pandasからPySparkの場合がほとんど

```
In [29]: from pyspark.sql import SparkSession  
  
# pandasでつくったDFをSparkのDFへ変換  
spark = SparkSession.builder.getOrCreate()  
 sdf = spark.createDataFrame(pdf)  
  
sdf.show()
```

```
+-----+-----+-----+  
|student|class|score|  
+-----+-----+-----+  
|Larissa| A1 | 50 |  
| Jaylan| B1 | 69 |  
| Goldal| A1 | 68 |  
| Myriam| A1 | 70 |  
| Fabiola| C1 | 82 |  
| Abigail| C1 | 57 |  
| Hardyl| B1 | 88 |  
| Jeromyl| C1 | 93 |  
+-----+-----+-----+
```

抽出 (WHERE)

- データからA組のテスト結果の点数と生徒の名前を抽出する
- PySparkを使って書いた場合このようになる

```
In [15]: sdf.filter(sdf['class'] == 'A').select('student','score').show()
```

```
+-----+-----+
|student|score|
+-----+-----+
|Larissal| 50|
| Goldal | 68|
| Myriam | 70|
+-----+-----+
```

- SQL, pandas, PySparkとコードを比較する

SQL

```
SELECT id, student, class
FROM test_result
WHERE class == 'A'
```

pandas

```
pandas
pdf.loc[pdf['class'] == 'A', ['student', 'score']]b
```

PySpark

```
PySpark
sdf.filter(sdf['class'] == 'A').select('student','score').show()
```

複数の条件で抽出

- 「C組のうち80点以上の生徒」で抽出する

```
In [17]: sdf.filter((sdf['class'] == 'C') & (sdf['score'] >= 80)).show()
```

```
+-----+-----+-----+
|student|class|score|
+-----+-----+-----+
|Fabiola|  C|  82|
|Jeromy|  C|  93|
+-----+-----+-----+
```

pandas

```
pandas  
pdf.loc[(pdf['class'] == 'C') & (pdf['score'] >= 80)]
```

PySpark

```
PySpark  
sdf.filter((sdf['class'] == 'C') & (sdf['score'] >= 80)).show()
```

結合 (JOIN)

- PySparkのDF変換した際 `index` の情報が抜け落ちるので `student_id` として再定義する
- `withColumn` メソッドをうまく使えば解決できそうだが手が及ばなかった

```
In [18]: student = pd.DataFrame({  
    'name': ['Larissa', 'Jaylan', 'Golda', 'Myriam', 'Fabiola', 'Abigail', 'Bernice', 'Hardy', 'Jero',  
    'class': list('ABAACCBBC'),  
    'student_id': ['A1', 'B1', 'A2', 'A3', 'C1', 'C2', 'B2', 'B3', 'C4']  
})  
  
s_attendance = spark.createDataFrame(attendance)  
s_student = spark.createDataFrame(student)
```

- INNER JOIN をPySparkにより実装すると以下のようになる

```
In [19]: s_attendance.join(s_student, 'student_id', 'inner').show()
```

[Stage 16:>

(0 + 1) / 1]

student_id	subject	name	class
A1	math	Larissa	A
A1	math	Larissa	A
B1	math	Jaylan	B
B2	History	Bernice	B
B3	History	Hardy	B
C1	Japanese	Fabiola	C
C2	History	Abigail	C
C2	math	Abigail	C
C4	English	Jeromy	C

- 続いて LEFT JOIN をやってみる
- INNER JOIN と同様に, join メソッドの引数で

In [20]:

```
# LEFTJOIN
sb_student = s_student.filter(s_student['class'] == 'B')
shistory_attendance = s_attendance.filter(s_attendance['subject'] == 'History')
# indexを使っていないため,結合の指定がしやすい(のかも)
sb_student.join(shistory_attendance, "student_id", "left").show()
```

student_id	name	class	subject
B1	Jaylan	B1	null
B2	Bernice	B1	History
B3	Hardy	B1	History

SQL

```
SELECT *
FROM student LEFT JOIN attendance
  ON student.id = attendance.student_id
WHERE student.class = 'B' AND attendance.subject = 'History'
```

pandas

```
pd.merge(b_student,history_attendance, left_index=True, right_on='student_id', how='left', )
```

PySpark

```
sb_student.join(shistory_attendance, "student_id", "left").show()
```

集約 (GROUP BY)

- まずはDFの変換をする

```
In [21]: s_test_result = spark.createDataFrame(test_result)
```

- pandasの場合と同様に「クラスごと、教科ごとの平均点」を求めてみる

```
In [22]: s_test_result.groupBy('class','subject').mean().show()
```

[Stage 23:> (0 + 8) / 8]

class	subject	avg(score)
A	math	70.6666666666667
A	English	70.3333333333333
B	math	63.0
B	English	70.0
C	English	67.0
C	math	50.66666666666664

SQL

```
SELECT class, subject, avg(score)
FROM test_result
GROUP BY class, subject
```

pandas

```
test_result.groupby(['class', 'subject']).mean()
```

PySpark

```
s_test_result.groupBy('class','subject').mean().show()
```

- 集約関数が複数存在する場合
- `agg()` メソッドを使って表現、計算のために `pyspark.sql.functions` ライブラリをインポートしている

```
In [23]: import pyspark.sql.functions as F
```

```
s_test_result.groupBy('class','subject').agg(F.avg('score'), F.max('score')).show()
```

[Stage 26:>

(0 + 8) / 8]

class	subject	avg(score)	max(score)
A1	math	70.66666666666667	96
A1	English	70.33333333333333	74
B1	math	63.0	85
B1	English	70.0	76
C1	English	67.0	77
C1	math	50.66666666666664	56

SparkSQLについて

- PySparkはPandasよりもSQLに近い形で書くことができて便利
- SQL文によって操作することもできる

- 先ほど行ったPySparkでの抽出をSQLで行う[8]

In [24]:

```
# SparkSQLで操作するテーブルを登録
sdf.createOrReplaceTempView('test_result')

# SQLによる操作
query = """
SELECT student, score
    FROM test_result
    WHERE class == 'A'
"""

spark.sql(query).show()
```

```
+-----+-----+
|student|score|
+-----+-----+
|Larissa| 50|
| Goldal| 68|
| Myriam| 70|
+-----+-----+
```

PandasとPysparkの比較

乱数によるデータセット作成

- 亂数によってデータを生成[9]する
- データサイズはおよそ80KB

In [25]:

```
np.random.seed(seed=42)
r_pdf = pd.DataFrame(np.random.rand(100,100))
r_sdf = spark.createDataFrame(r_pdf)
#Byte単位でデータサイズを表示
r_pdf.memory_usage().sum()
```

Out [25]: 80128

時間計測

- 一つのカラムに関してソートを行う時間を計測することにより比較する
- pandasの場合

In [26]:

```
import time

start = time.perf_counter()

r_pdf.sort_values([0])

print(time.perf_counter() - start)
```

0.0010171659996558446

- PySparkの場合

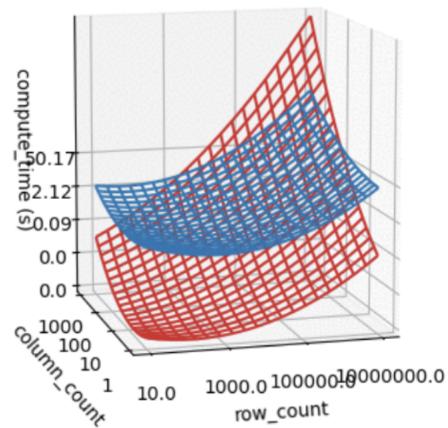
```
In [27]: import time
```

```
start = time.perf_counter()  
r_sdf.sort("0")  
print(time.perf_counter() - start)
```

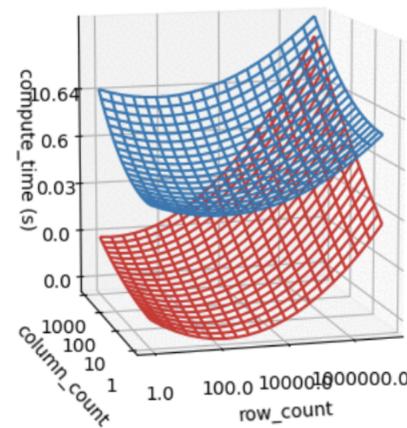
```
0.053194125001027714
```

- PySparkとpandasそれぞれの場合について実行時間を比較すると圧倒的にpandasの方が早い
- [10]ではPySparkとpandasの性能比較をより詳しく行なっている
- [10]内でも、平均に関しては常にpandasの方が早い
- データのサイズが $1,000,000 \times 1000$ を越え、かつ相関を求めるといった複雑な処理を行うとPySparkの性能が上回る
- 下の図では横軸がそれぞれデータの行、列を表し、縦軸では計算時間を表している

DataFrame Corr Computation Time (s)



DataFrame Mean Computation Time (s)



終わりに

- PySparkによるデータ前処理の例を参考文献に数例紹介した
- pandasっぽい文法でSparkを動かそうというコンセプトのKoalasというものもある
- Sparkのほうが処理が早いからと言って全ての分析ケースに対してpandasからPySpark(Spark)に置き換えるのは得策ではない
- プロジェクトとして行なっていく際には、まずは小さなデータからpandasを使ってアドホックな分析を行い、分析の方針が決まってからSparkを扱うといったパターンが想定される。

フリートーク（その3）

Zennのスクラップを貼り付けました。質問いただけすると大変嬉しいです。

<https://zenn.dev/uenotakato/scraps/f3a1f903ee7b6c>

参考

スライド中の参考文献

- [1] <https://bookclub.kodansha.co.jp/product?item=0000275420>
- [2] <https://docs.python.org/ja/3/>
- [3] <https://pandas.pydata.org/docs/>
- [4] https://pandas.pydata.org/docs/user_guide/10min.html
- [5] [https://sparkbyexamples.com/pyspark/pandas-vs-pyspark-dataframe-with-examples/amp/](https://sparkbyexamples.com/pyspark/pandas-vs-pyspark-dataframe-with-examples/)
- [6] <https://www.ohitori.fun/entry/basic-data-analysis-in-pandas>
- [7] <https://qiita.com/driller/items/ef8a16be03e146ce2183>
- [8] <https://blog.serverworks.co.jp/introducing-pyspark-6>
- [9] <https://qiita.com/yukifddd/items/5668483705ef9d89a0c9>
- [10] <https://towardsdatascience.com/parallelize-pandas-dataframe-computations-with-spark-dataframe-bba4c924487c>

PySparkによるデータ前処理の例

<https://techblog.nhn-techorus.com/archives/7301>

<https://www.ariseanalytics.com/activities/report/20211210/>