Extract(抽出)Transform(変換)Load(書出)（Python、petl）

目次

Extract(抽出)Transform(変換)Load(書出)（Python、petl）

はじめに

システムの実装単位はプログラム（呼び方は他にも種々ありますが）で、内容はアルゴリズムです。アルゴリズムは特許法上の定義とは別に、ネイティブスピーカは「Algorithm = Logic + Control」という認識のようです。構成要素の Logic に関しては大量な関連語の中から“Logic programming”をみると「問題領域に関する事実と規則を表現し...言語に Prolog」と説明されています（詳しくはネットで！）。

もう一つの制御（Control）の部分は条件判定・分岐、I/O 等で、必要な試験のバリエーションはこの量に依存します。したがって、制御部分を省略して「事実と規則」だけで実装できれば品質と生産性が上がり保守性も向上します。Prolog 言語で業務システムを開発するのはハードルが高いですが、制御部分をフレームワークやツールに任せる方法があり、その一つが ETL です。ETL はデータ統合(DWH)の前処理として説明されますが、データ処理の汎用型であり業務処理でも使うことができます。

1. ETL ツール／ライブラリ

ETL のフレームワークやツールの開発は盛んに行われています（GitHub をで管理するオープンソースのプロジェクト数は万の単位です[1]）。GUI で操作ができる製品レベルのツールもありますがそれらは開発・保守の主体（Contributors）が営利企業であることが多く、大企業に買収されてライセンスや開発の方針が変わってくることがあります。特定企業が開発しているとユーザ登録が必要等の制約があるため、ここでは個人を中心に比較的活発に活動していて高評価（星が多い）の petl を使います。

2. petl のインストール

以下 Windows 上で Python 3.10.5 と petl(MIT License-商用可)ver-1.7.11 の使用を前提に説明します。

Python には pip というパッケージャー（packager）が標準モジュールとして添付されているので、これを使って petl 及び petl が使う（依存）ライブラリをインストールします。

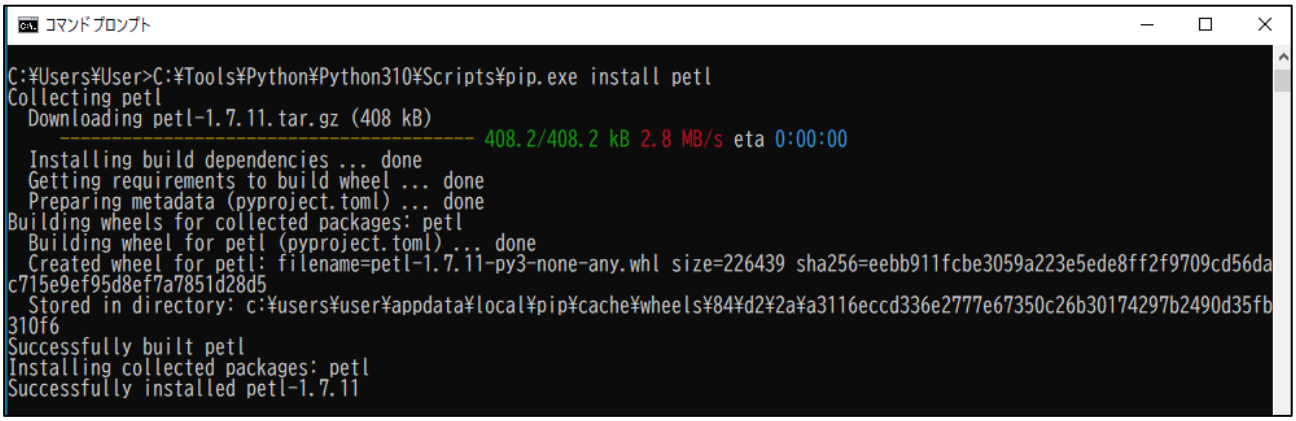
2.1. petl パッケージ

petl 本体を Windows 環境でインストールするためのコマンドは以下のとおりです。

> py -m pip install petl　または、*<Python インストール先パス¥>*pip.exe install petl

※ py は Windows の Python 起動コマンドで、Linux の場合は python3 になります

問題なくインストールが終わると以下のように表示されます。

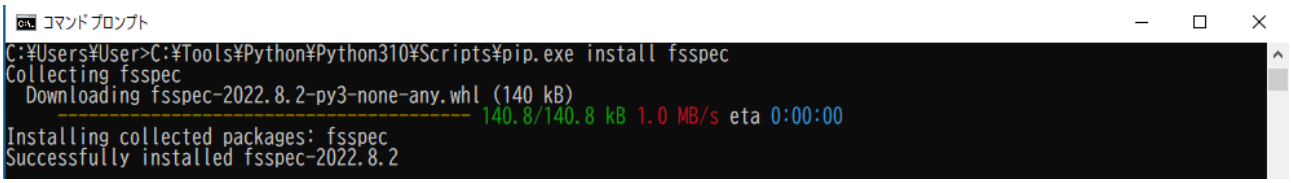


[1] GitHub-ETL プロジェクトの検索 https://github.com/search?q=etl&type=Repositories

2.2. 依存ライブラリ

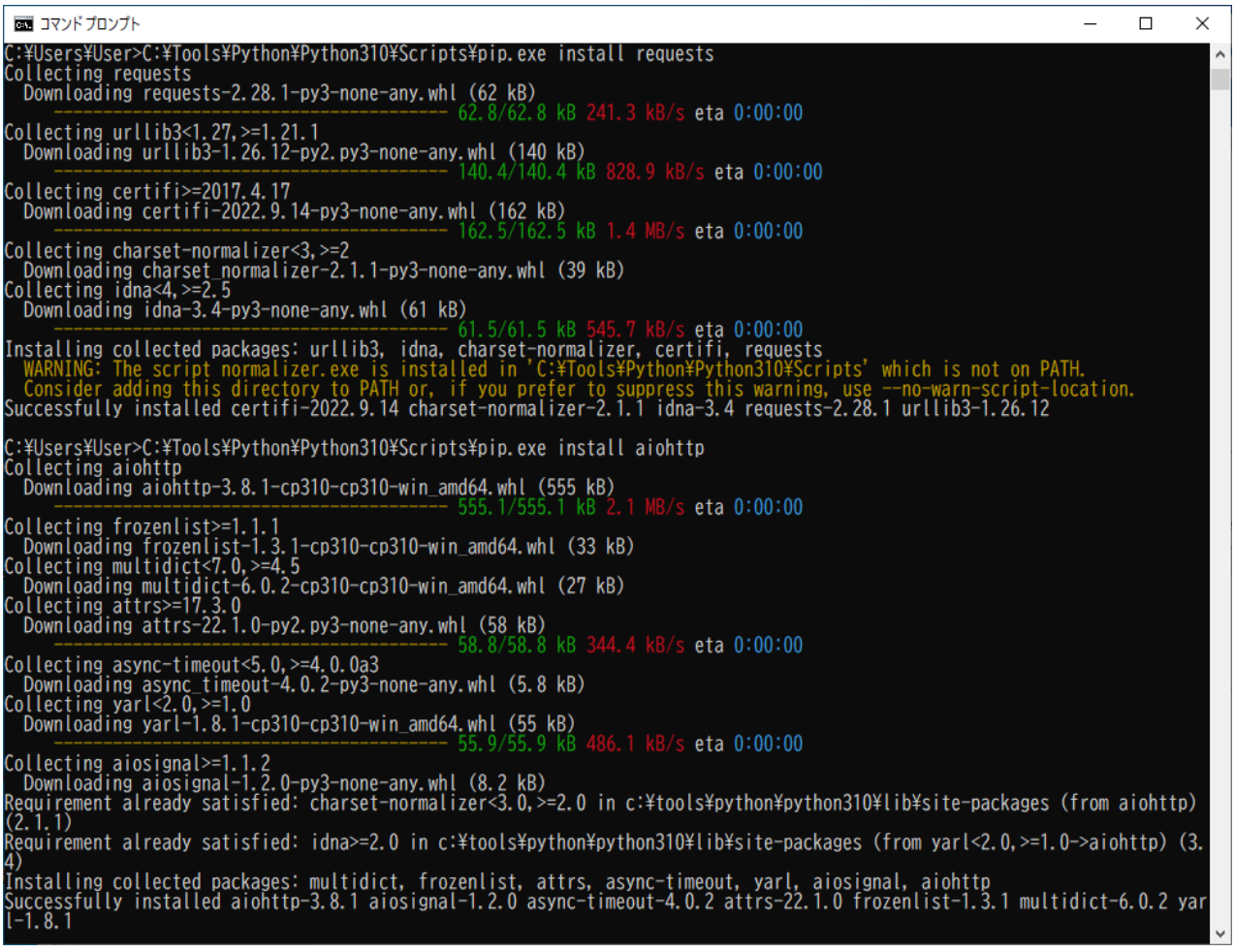
以下のライブラリは利用者が import して使うことは無く、petl が必要に応じて（使う機能によって）呼び出すのでインストールだけしておきます。

① fsspec：リモートファイルシステムの操作を行う（オープンデータの取得を行います）

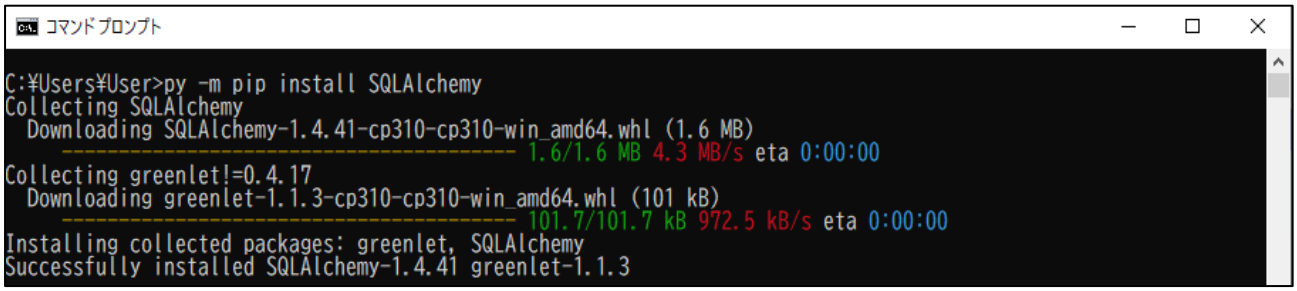


② requests：http プロトコルの API

③ aiohttp：非同期 HTTP クライアント/サーバーのライブラリ



④ SQLAlchemy：テーブルを自動で新規作成（create table）する場合等に使います



Extract(抽出)Transform(変換)Load(書出)（Python、petl）

3. petl の使い方

petl は Python の拡張ライブラリで、以下の手順で操作します。
① 各種のソースから petl のテーブルにデータを取り込む（抽出）
② petl のツールを使ってテーブルの結合や切り出し、データ加工を行う（変換）
③ petl のテーブルから残したいデータは、DB やファイル等に保存する（書出）

3.1. 抽出

DB や Excel、xml/html、JSON 他、各種の形式のファイル[2]が扱えます。また、リモートデータを直接テーブルに取り込むことができます。

＜例＞

・ リモートソースを https を使って petl のテーブルに読み込む例
[結果] newconf という名前のテーブルに url のデータが全件読み込まれます

```
import petl

# 厚生労働省（新型コロナウイルス感染症について）オープンデータ
# ○新規陽性者数の推移（日別）
newconf = petl.fromcsv('https://covid19.mhlw.go.jp/public/opendata/newly_confirmed_cases_daily.csv'
        , encoding='utf_8_sig')
```

※ URL に書かれているプロトコル(https)を変えるだけで接続方法を変えることができ、ローカルディスクのカレントディレクトリにあるファイルを読込む場合は以下のようにします

```
newconf = petl.fromcsv('newly_confirmed_cases_daily.csv', encoding='utf_8_sig')
```

※ encoding='utf_8_sig' は、ソースの文字コードが UTF-8 BOM 付きで作られていることを示します

3.2. 変換

目的のデータを得するためにソースから取得したデータを加工（計算、形式変換、その他）したり、複数のソースから取得したデータを結合することができます。

＜例 1 ＞ 列の切り出し：petl.cut

・ テーブルから列を切り出して新しいテーブルが作られます

```
newconf2 = petl.cut(newconf, 'Date', 'ALL')
```

```
[newconf テーブルの内容]
Date,ALL,Hokkaido,Aomori,Iwate,Miyagi,Akita,Yamagata,Fukushima,Ibaraki,Tochigi,…後略
2020/1/16,1,0,0,0,0,0,0,0,0,0,…後略
2020/1/17,0,0,0,0,0,0,0,0,0,0,…後略
…

[できた newconf2 テーブルの内容]
Date,ALL
2020/1/16,1
2020/1/17,0
…
```

[2] petl で扱えるファイル https://petl.readthedocs.io/en/stable/io.html#module-petl.io.csv

Extract(抽出)Transform(変換)Load(書出)（Python、petl）

＜例２＞列名の変更：petl.rename

・列名を変更た新しいテーブルが作られます

```
newconf3 = petl.rename(newconf2, 'ALL', 'newconf')
```

[できた newconf3 テーブルの内容]

```
Date,newconf
2020/1/16,1
2020/1/17,0
（中略）
2020/5/9,108
2020/5/10,66
…
```

＜例３＞テーブルの結合：petl.outerjoin

・Date で外部結合したテーブルが作られます（内部結合が必要であれば join を使います）

```
table = petl.outerjoin(newconf3, deathsc, key='Date')
```

[deathsc テーブルの内容]

```
Date,deathsc
2020/5/9,267
2020/5/10,249
…
```

[できた table の内容]

```
Date,newconf,deathsc
2020/1/16,1,None
2020/1/17,0,None
（中略）
2020/5/9,108,267
2020/5/10,66,249
…
```

＜例４＞計算等の結果からテーブルを生成：petl.convert

・集計値を算出しながら列に設定していきます

```
sumData = 0
def calcReqcare(rec):
    global sumData  # 以降、関数の外で宣言した sumData を使う
    sumData += int(rec['(ALL) Requiring inpatient care']) # 日々の入院患者を累計
    return sumData - int(rec['(ALL) Discharged from hospital or released from treatment']) # 退院を減

mappings = dict()
mappings['Date'] = 'Date'
mappings['reqcare'] = calcReqcare  # 先行の行で宣言した関数 calcReqcare のコードオブジェクトを設定
table = petl.fieldmap(reqcare, mappings) # テーブル reqcare を基に mappings の内容でテーブルを作成
```

[reqcare の内容]

```
Date,(ALL) Requiring inpatient care,(ALL) Discharged from hospital or released from treatment,
2020/5/9,6250,8276, …後略
2020/5/10,6074,8514, …後略
```

[できた table の内容]

```
Date,reqcare
2020/5/9,-2026
2020/5/10,3810
```


3.3. 書き出し

書き出しは RDB を含む、抽出で扱える各種のファイル形式に対応しています。
RDB：SQLite に書き出すコードは以下のようになります。

＜例＞ RDB への登録：petl.todb

・ カレントディレクトリに covid19.db というファイル名でテーブル dailyTrace を作成し、table の内容をインサートします。既にテーブルが存在する場合は消して新たに作成します

```
import sqlite3
conn = sqlite3.connect('covid19.db')
petl.todb(table, conn, 'dailyTrace', create=True, drop=True, dialect='sqlite', sample=0)
```

4. 一連のコードサンプル

政府が管理しているデータのいくつかは「オープンデータ」として公開されています。以下のサンプルでは新型コロナウィルスの感染に関わる日々のデータを取り寄せて日付をキーにして統合します。
抽出で厚生労働省とデジタル庁に保管されている csv ファイルを取得します。
変換では、異なるっている日付形式や列名を合わせて必要なデータを日付をキーに統合します。
書出では、RDB と新たなテーブルを作成して変換で統合したデータを書きだします。

＜使用データの形式＞

●厚生労働省（新規陽性者、死亡者数、重傷者数）

Date,ALL,Hokkaido,Aomori,Iwate,Miyagi,Akita,Yamagata,Fukushima,Ibaraki,Tochigi,Gunma,…
2020/5/9,613,48,0,0,1,0,0,0,9,0,18,41,39,180,47,0,13,16,8,0,0,6,1,34,1,1,13,59,32,2,2,0,0,0,2,0,0,…
2020/5/10,621,51,0,0,1,0,0,0,9,0,18,42,40,180,47,0,13,16,8,0,0,6,1,34,1,1,13,59,34,2,2,0,0,0,2,0,…
（以下、日次データが続く）
※ Date 列と ALL 列を使います（以降は都道府県毎の値になっています）

●厚生労働省（入院治療等を要する者等推移）

Date,(ALL) Requiring inpatient care,(ALL) Discharged from hospital or released from treatment,(ALL) To be confirmed,(Hokkaido) Requiring inpatient care,…
2020/5/9,6250,8276,442,462,424,0,0,17,10,0,0,0,8,79,0,0,0,16,13,56,0,36,45,0,71,78,10,0,25,29,65,…
2020/5/10,6074,8514,422,450,444,0,0,17,10,0,0,0,7,80,0,0,0,16,11,58,0,36,45,0,71,78,10,0,32,24,…
（以下、日次データが続く）
※ 2 列目の全国[(ALL)]の入院患者を集計し、3 列目の全国[(ALL)]の退院者をマイナスした値を算出

●デジタル庁（接種日別接種回数サマリー）

date,count_first_shot_general,count_second_shot_general,count_third_shot_general,count_fourth_shot_general,count_first_shot_general_wo_deceased,count_second_shot_general_wo_deceased,count_third_shot_general_wo_deceased,count_fourth_shot_general_wo_deceased
2021-04-12,4709,0,0,0,4004,0,0,0
2021-04-13,4318,0,0,0,3536,0,0,0
（以下、日次データが続く）
※ date 列の列名と形式を厚生労働省のデータと合わせて外部結合します

Extract(抽出)Transform(変換)Load(書出) (Python、petl)

```
import petl
from datetime    import datetime
#  【抽出】
# 厚生労働省（新型コロナウイルス感染症について）オープンデータ＞ https://www.mhlw.go.jp/stf/covid-19/open-data.html
# ○新規陽性者数の推移（日別）
newconf = petl.fromcsv('https://covid19.mhlw.go.jp/public/opendata/newly_confirmed_cases_daily.csv',
                       encoding='utf_8_sig')
# ○入院治療等を要する者等推移
reqcare = petl.fromcsv('https://covid19.mhlw.go.jp/public/opendata/requiring_inpatient_care_etc_daily.csv',
                       encoding='utf_8_sig')
# ○死亡者数（累積）
deathsc = petl.fromcsv('https://covid19.mhlw.go.jp/public/opendata/deaths_cumulative_daily.csv',
                       encoding='utf_8_sig')
# ○重症者数の推移
severec = petl.fromcsv('https://covid19.mhlw.go.jp/public/opendata/severe_cases_daily.csv',
                       encoding='utf_8_sig')

# デジタル庁ワクチン接種記録システム（VRS)＞ https://info.vrs.digital.go.jp/dashboard/
# 接種日別接種回数サマリー（CSV）
vaccins = petl.fromcsv('https://data.vrs.digital.go.jp/vaccination/opendata/latest/summary_by_date.csv',
                       encoding='shift_jis')
#  【変換】
def dateFormat(date, row, ffrom="%Y-%m-%d", to="%Y/%m/%d"):
    ''' 日付形式を引数で指定した ffrom から to に変更します '''
    return datetime.strftime(datetime.strptime(row.Date, ffrom), to)

def dateFormatmmdd(date, row):
    ''' 日付の月／日を 2 桁に変えます '''
    return dateFormat(date, row, ffrom="%Y/%m/%d")

def cut_rename(table, name):
    ''' Date,ALL 列を切り出して列名 ALL を name に変える。また、Date の月／日を 2 桁に変える '''
    table2 = petl.convert(table, 'Date', dateFormatmmdd, pass_row=True)
    return petl.rename(petl.cut(table2, 'Date', 'ALL'), 'ALL', name)

# 各 CSV から必要な項目を切りだして key='Date' で結合していく（重複や様式、並び順のチェックは割愛）
# outerjoin を使うと、「presorted=True」パラメータを指定しない限りレコードの並べ替えが行われます
table = cut_rename(newconf, 'newconf')                    # 新規陽性:newconf から基テーブルを作る
table = petl.outerjoin(table, cut_rename(deathsc, 'deathsc'), key='Date') # 死亡者数（累積）結合
table = petl.outerjoin(table, cut_rename(severec, 'severec'), key='Date') # 重症者数の推移　結合

# 入院治療者は 入院者の日々累計 - 退院者 の計算結果で中間テーブルを作成し、Date で結合
fmReqcare = petl.convert(reqcare, 'Date', dateFormatmmdd, pass_row=True)
sumData = 0
def calcReqcare(rec):
    global sumData
    sumData += int(rec['(ALL) Requiring inpatient care'])
    return sumData - int(rec['(ALL) Discharged from hospital or released from treatment'])

mappings = dict()
mappings['Date'] = 'Date'
mappings['reqcare'] = calcReqcare
#import inspect
#print(inspect.getsource(mappings['reqcare'])) # lambda(無名関数)の中身確認
table = petl.outerjoin(table, petl.fieldmap(fmReqcare, mappings), key='Date')

# 接種日別接種回数サマリーを左結合（感染者等のデータが無い日の接種データは破棄）
fmVaccins = petl.rename(vaccins, 'date', 'Date')
table = petl.leftjoin(table, petl.convert(fmVaccins, 'Date', dateFormat, pass_row=True), key='Date')
```



```
# データが欠損している（None）列のデータを前日のデータで埋める
tablefill = petl.filldown(table)
# 確認のため最後の 10 件について、データ埋め込み前後を比較しターミナルに表示
for x in table[len(table)-10:]:
    print(x)
    record = petl.tupleoftuples(petl.select(tablefill, f'{{Date}} == "{x[0]}"'))
    print('<same>' if len(record) > 1 and x == record[1] else record[1] if len(record) > 1 else '<NO DATA>')
    print()

#  【書出】
# 出来上がったテーブルを DB(SQLite) に格納
# 動的に格納先のテーブルを作成する「create=True」には SQLAlchemy のインストールが必要です
import sqlite3
conn = sqlite3.connect('covid19.db')
petl.todb(table, conn, 'dailyTrace', create=True, drop=True, dialect='sqlite', sample=0)

print('Covid19 Finish!')
```

5.　実装上の留意点

抽出するデータの更新頻度やデータの量により、以下を考慮する必要があります。

① データの取得方法／タイミング

データが頻繁に更新される（例えば数分おき）場合は http/https で取得してすぐに処理しないと情報に追随できなくなりますが、そうでなければ一旦 ダウンロードしてから変換以降の処理を実行した方がシステムの負荷が減ります

② ジョブネット／スケジューリング

データが大量だったり相手側回線の都合等で取得に時間が掛かる場合は、複数のデータを平行して取得する方が全体の実行時間を減らせる可能性があります。この場合、個々のデータ取得を別ジョブにするとともに変換の開始を全てのデータの転送終了に同期させる必要があります。

”ETL”には明確な機能の定義が無いので大雑把に ETL とよばれて守備範囲が異なる関連ソフトが大量にあります。これらは主たる目的から以下の 2 種類に大別できます。

● データソースからデータを取得・加工するためのツールの組合せ …前項で取り上げた petl 等

● 変換の同期合わせを目的の一つにしているフレームワークやプラットフォーム 例えば...

Bonobo　Python 3.5+ 用の軽量の Extract-Transform-Load (ETL) フレームワーク

Apache NiFi（以下、公式サイトより）

システム間のデータの流れを自動化する。フローベースのプログラミングの概念に基づくデータフロー システムでデータ ルーティング、変換、およびシステム メディエーション ロジックの強力でスケーラブルな有向グラフをサポート

Apache Airflow（以下、公式サイトより）

バッチ指向のワークフローを開発、スケジューリング、監視するためのオープンソースプラットフォーム

Apache Kafka（以下、公式サイトより）

高性能データ パイプライン、ストリーミング分析、データ統合、およびミッション クリティカルなアプリケーションのために、何千もの企業で使用されているオープンソースの分散型イベント ストリーミング プラットフォーム

以上

