自身で設定したゴール

　データサイエンスにおいて実用に足るRuby環境を作ること (実用ユースケースとしては初等統計解析を行うワークフローを構築) 。

当初の予定から変更点

- numpy相当のgemの必要性の把握とその利用啓蒙実用例を加えた
- daruの改善ではなく設計の異なる別gem(暫定名sake)によるnumo/narray wrapperとしてのdata frameの模索

達成した項目，機能

- 環境構築インフラ整備とそのドキュメント化
- 再現性を保証した啓蒙実用例 (Jupyter notebook)

未達成項目

　まず、中間報告書の「調査から浮かんだDaruに必要な改善点」に挙げた問題点の内、

- DataFrame#[]メソッドの挙動がわかりづらい
- Daru::Indexの比較演算子メソッドがない

の２つ以外、すべての問題点は未解決である。これは、安易な機能追加は、daruのソースコードの肥大化を加速させると判断したからである。後にも述べるが、現状のdaruのソースコードは長く、読みづらいものとなっている。これは、クリーンなソースコードを保つことよりも、迅速な機能追加が優先された結果であると考えられる。中間報告書に述べた機能の追加は、大規模なリファクタリングの後に行なうのが適切であると考える。
　また、中間報告書の「下半期の計画について」に挙げた、「Daruの内部データへのnumo-narray導入の試行」についても、上記と同様の理由から見送った。代わりに「daruの改善」ではなくスクラッチ実装のdataframe gem(暫定名 sake https://github.com/genya0407/sake)でnumo-narray導入の試行の実例を示した。

開発スケジュールに対しての詳しい進捗状況

IRubyへのcztopサポートの追加

- githubのrepository (https://github.com/SciRuby/iruby) 上では済んでいる。rubygems.orgでは未だリリースされていない。リリースリクエストは https://github.com/SciRuby/iruby/issues/114 に送っているが反応が無い。
- 利用方法のdocumentは英日両方で作成し英文の更新は https://github.com/SciRuby/iruby/pull/120 でリクエストしているもののまだマージされていない。日本語文は

https://github.com/sciruby-jp/ruby-datascience-examples/blob/master/README.md に記述。

daru_plotlyの改良

daru_plotlyは、rbplotlyとdaruをつなげるgemである。以前はdaru_plotting_plotlyという名前だった。中間報告から最終報告までの間は、主にdaru_plotlyについて改善活動を行った。進捗は以下のとおりである。

まず、daru_plotlyのplotメソッドの使い方を変更した。以前は、df.plot(options)のような形式で使用されるようになっていたが、現在では、plot(df, options)のように使用されるようになっている。これは、サードパーティのグラフライブラリを読み込む度に、daru本体のコードをオープンクラスによって書き換えるのは複雑な手法であるし、複数のグラフライブラリを読み込んだときの挙動に不安があると判断したからである。

次に、daru_plotlyの使用例を増やした。具体的には、プロットする元データを変更したことと、heatmapを追加したことの二点である。

Windows・MacでのJupyter Notebook (IRuby kernel) の動作保証とそのドキュメント

- 完了
  https://github.com/sciruby-jp/ruby-datascience-examples/blob/master/README.md に記述

「Foundations for Analytics with Python」改め「Python Data Science Handbook」のRuby版のNotebook作成

- データサイエンスに必須のデータフレームgem daruの改善を主眼に置いている。
- データフレーム機能調査には
  https://github.com/jakevdp/PythonDataScienceHandbook の2,3章を参考にdaruと pandasの比較を行った。
- https://github.com/sciruby-jp/ruby-datascience-examples にまとめている。(ただし著作権を考慮した調整を5月19日までに行う。)

statsample (統計分析用gem) 動作検証

- wine quality datasetを用いた「Foundations for Analytics with Python」の初等統計解析(
  https://github.com/cbrownley/foundations-for-analytics-with-python/blob/master/statistics/wine_quality.py)を元に統計分析gem statsampleを用いRubyで再現したnotebookを
  https://github.com/sciruby-jp/ruby-datascience-examples/blob/master/Statistics/wine_quality.ipynb にまとめた。

今後の課題

- プロジェクト開始当初は既存SciRuby資源の活用，改善可能性を期待していたがIRuby以外(daruやstatsampleなど)は複雑度が高く継続保守・活用が非常に難しいと感じている

本projectの方針(Rubyでのデータサイエンスgem群の実装)と言語間ブリッジを活用する方針(matplot.rb)の比較

それぞれ下記のpros,consがある。

本projectの方針pros,cons

pros

- prosではないが，この方針を捨てるとRuby言語の活用の一つの大きな用途が閉ざされ長期的観点で大きな損失となる可能性がある。
- データサイエンス環境基盤の底上げ (実例，ドキュメント，今後の問題点の整理)。

cons

- 機能が不十分。まだ実用レベルにない。

言語間ブリッジを活用する方針(matplot.rb)のpros, cons

pros

- 機能が十分。
- 短期的な策として有効。
- Pythonのパッケージはメジャーosでのバイナリパッケージのサポートが進んでおり環境構築も比較的容易 (そもそも前者の方針でもJupyterの利用は必須と考えている)。

cons

- 依存パッケージの増加に伴うプログラム構成の複雑化。
- prosの部分で比較的と言ったものの科学技術計算系ユーザーは凝った解析・開発環境を避ける傾向が強いことを意識する必要がある。
- 言語をまたぐワークフローをいかに浸透させ得るか (Python, Rにおいても同様の取り組みはあるが本流とはなり得ていない)。

daruを中心としたコードの肥大化に関して

今回の調査を通して、現状のdaruでも、ある程度のデータの解析・可視化はできるということがわかった。では、daruの次の課題はなにか、あるいは、より質の高い科学技術計算の環境を構築するためには何をするべきか、ということについて述べる。

まず課題としてSciRubyの各gemの「計算速度の遅さ」「各gemの肥大化と複雑化」がある。特にデータサイエンスのワークフローの中核を担うdaruにそれが顕著で、根本的な書き直しが必要となる可能性がある。data frameでのfilter操作に必須のuniversal functionの機能の実装を例に挙げると，pandasがその処理をnumpyに利用しているのに対し, daruはその機能を自身で実装するだけでなくwrapする対象のarrayを切り替えるようにまでしている (https://github.com/SciRuby/daru/tree/master/lib/daru/accessors)。しかし実際はこの設計は裏目に出ている。ここでwrapの対象となっているgemはどれもnumpyほどの品質になく，その比較評価コストはdaruの改善を図るものにとって大きな負担となる。またこの肥大化と複雑化傾向は可視化処理部においても同様でNyaplot, gruffをグラフライブラリとして切り替えようとしている (https://github.com/SciRuby/daru/tree/master/lib/daru/plotting)。この傾向はdaruだけでなくdaruとの連携が想定されているSciRubyの統計計算gem (statsample) でも同様で我々がワークフローを構成するgemの改善を試みる上で大きな障壁となった。

我々はこの障壁を取り除くにはdata frameを実装するgemの再設計が必要なのではないかという疑いを拭うことができず当初予定していたリネームを中心としたリファクタリングに手を入れることができなかった。もし再設計を行わずに改善を試みる場合は下記の3つの手続きを行うことが最善策と考えている。

まずは「daruの内部データの形式を１つに絞る」ことである。現状ではArrayWrapperとNMatrixWrapperの２つからオプションで選ぶ事ができるがこれを廃止する。具体的には、ArrayWrapper（現状のデフォルト）を廃止し、NMatrixWrapperに統一する。それと並行して、NMatrixとNumo/NArrayの統一を試みる。　次に「daru中でサポートするグラフライブラリを一つに絞る，もしくはグラフライブラリのサポートに関しては別のライブラリで対応する」こと，その上で最後に「当初プロジェクトで予定していたリファクタリング」である。2つの前手続きが無ければプロジェクトのゴール達成は困難である。

numpy相当プロジェクトの継続サポートの必要性

本プロジェクトの成果としてnumo/narrayの実用例やこれをラップした暫定data frame gem (sake)を提示したがどちらもnumpy, pandasと比較して実用レベルへの改善が必要である。

拡張ライブラリ実装のハードルについて

daruにもpandasにもcorrメソッドがある。corrメソッドは、data frameの列についての相関係数行列を計算するメソッドである。しかし、両者のcorrメソッドの実行速度には雲泥の差がある。簡易的なベンチマーク（ https://github.com/ash1day/ra2016/blob/master/corration_benchmark/Benchmark.md）を実施した結果、daruのcorrメソッドの実行時間がユーザー時間で5秒程度であるのに対し、pandasでは0.0000秒となっている。

　このような実行時間の差は、pandasがネイティブ拡張ライブラリを用いていることによって生まれている。pandasのcorrメソッドはCythonで実装されており、ネイティブ拡張ライブラリとしてコンパイルされる。一方、daruのcorrメソッドはpure-Rubyで実装されている。これは、Rubyには簡単に拡張ライブラリを実装するためのCythonのような仕組みがなく、拡張ライブラリを書くためにはC言語で書く必要があり、拡張ライブラリを作成するハードルが高いからだと考えられる。この問題を解決するためには、RubyにもCython相当の拡張ライブラリを実装する手段が必要と考えられる。

高級なグラフライブラリの必要性

　pythonにはseabornというライブラリがある。これは、matplotlibのラッパーであり、単にグラフを描画するだけでなく、グラフの見た目をきれいにしてくれたり、統計解析に役立つようなことをやってくれる。例えば、
http://seaborn.pydata.org/generated/seaborn.PairGrid.html のように、非常に少ないコードで、データフレームに含まれる２つの列のすべての組み合わせについての散布図を作成してくれる。
　現状、Rubyにseabornのようなライブラリは存在しない。中間報告の段階ではこうした高級なグラフライブラリの必要性について考えていなかったが、今後必要になると考えられる。

　なお、rbplotlyはplotly.jsのラッパーなので、plotly.jsが提供する以上の機能は提供するべきではないと考える。また、daru_plotlyは、plotlyとdaruをつなげるgemなので、それ以上のことをするべきではない。seabornのPairGridのようなことをするのであれば、seabornに相当するような、データフレームから独立した高級なグラフライブラリをまず作成し、そのライブラリとデータフレームをつなぐようなgemを作成するのが良いと考える。