ノートブックプログラミングにおける手戻りの調査と分析

中丸 智貴[1]，佐藤 重幸[2]

[1] 東京大学大学院 総合文化研究科
nakamaru@graco.c.u-tokyo.ac.jp

[2] 東京大学大学院 情報理工学系研究科
sato.shigeyuki@mi.u-tokyo.ac.jp

概 要　データサイエンスのプログラミングでは，仕様の詳細がデータに基づいて探索的に決定されるため，手戻りが必然的に発生する．JupyterLab などのノートブック環境は，そのような手戻りを手軽に行えるプログラミング環境として人気が高い．一方，ノートブック環境では，インタプリタの実行状態が破壊的に更新されるため，手戻りの際に間違いを犯しやすく，その生産性の低さが大きな批判の的となっている．このような批判に適切に対処し，ノートブック環境下でのプログラミング（ノートブックプログラミング）の生産性を改善するには，現実の探索的過程においてどのような手戻りが起きているかを知ることが重要である．しかし現状，手戻りの実態が十分理解されているとは言い難い．そこで本研究では，Kaggle 上にあるノートブックを対象として，その改訂過程で生じた手戻りを定量的に分析する．更に，その改訂過程をノートブック環境で安全に実行する際のオーバーヘッドも実験的に評価する．本研究の結果は，ノートブック環境上のでセル単位の差分チェックポイントが，手戻りを伴うプログラミングにおいて効果的であることを示した．

1　はじめに

対話的プログラミング・データ可視化・文書化をまとめて行えるノートブックの人気は留まるところを知らず，その利用者は爆発的に増え続けている[1] [9]．プログラミング環境としてのノートブックの特徴は，コード片をセルとしてまとめ，任意の位置に配置でき，任意の順序で実行できることである．ノートブック環境では，セルを実行する度にインタプリタの実行状態が更新されるため，セルの実行結果を見ながら，段階的にコードを追記・編集できる．この特徴がコーディングとデータ分析を探索的に反復するデータサイエンスのプログラミングに適しており，今やノートブックはデータサイエンスにおけるデファクトスタンダードとなっている[2] [3] [11].

一方，特にソフトウェア工学の観点では，ノートブックは批判の的となっている [5]．任意の位置のセルを任意の順番で実行でき，実行状態が破壊的に更新されるということは，ノートブックの字面から，現在の実行状態に関する推論（reasoning）が酷く困難になる．プログラマが実行状態を誤解しやすくなることは，コードにバグを埋め込みやすくなるだけでなく，実行結果の解釈を誤り易くなることに繋がる．更に，ソフトウェア工学上の悪癖の温床となることも知られており，具体的には，不必要なセルが散らばっていたり [6]，スタイル面で低品質なコードであったり [4,19]，実行結果の再現性が失われていたり [12,13] する．このようなノートブックの負の側面は，データサイエンティストの間でも認識されてきている[4] [1,2] ものの，データサイエンスプログラミングにおける探索的過程との親和性のために，依然としてデファクトスタンダードの地位にある．

[1] https://github.com/parente/nbestimate
[2] https://analyticsindiamag.com/why-jupyter-notebooks-are-so-popular-among-data-scientists/
[3] https://www.datacamp.com/blog/the-past-present-and-future-of-the-data-science-notebook/
[4] https://towardsdatascience.com/why-data-scientists-should-use-jupyter-notebooks-with-moderation-808900a69eff

本研究では，現状のノートブックプログラミングを理解する重要な手掛かりとして，手戻り（backtracking）[18,21] に注目する．手戻りとは，古典的には，ウォーターフォールモデルにおける開発フェーズの巻戻りと再実行を意味する [18] が，近年ではコーディングレベルでの巻戻り [21] も含意する．このような手戻りは，データサイエンスプログラミングにおける探索的過程では，頻発することが知られている [7]．そして，この手戻りが，ノートブック環境の功罪の中心にある．

ノートブック環境では，実行済みのセルを編集したり，新たにセルを追加した後に，セルを選択的に再実行できる．つまり，複数セルで段階的に記述されたタスクの実行結果を見ながら，その途中段階をピンポイントに改変して再実行するという手戻りが，簡単に素早くできる．これは，データ可視化の結果を踏まえて，前処理・機械学習モデル・ハイパーパラメータを探索するデータサイエンティストにとっては，とても快適なことである．

一方，コードレベルで手戻りが生じても，実行状態が巻き戻っているわけではなく，破壊的に更新されていく．そのため，一旦手戻りを行なってしまうと，現在の実行状態が暗黙的になり，コードから推論することが困難になる．また，手軽だからといってピンポイントの手戻りを繰り返すと，アドホックなコード修正が全体に散らばり，全体として汚いコードになる．これは，高機能な統合開発環境や継続的インテグレーションツールの支援の下で，綺麗で高品質なコードを効率良く記述しているソフトウェアエンジニアにとっては，不愉快なことである．

ノートブック上の手戻りへの評価を通して，データサイエンスとソフトウェア工学の間でのノートブックの賛否を説明できると我々は推測している．しかし，実世界のノートブックを調査する研究は数多くある [1–4,8,12–17,19] ものの，ノートブック上の手戻りがどの程度どのように生じているかは，定量的に調べられていない．当然，プログラマの推論の障害となり，計算の再現性を毀損するような，安全ではない手戻りがどの程度生じているのかも，定量的に調べられていない．安全ではない手戻りを行ったとしても，インタプリタを再起動してノートブックを先頭から再実行すれば，コードと実行状態の不一致は解消する．しかし，その再実行の計算コスト，すなわちプログラミングを中断させるオーバーヘッドがどの程度あるのかも，不明である．

プログラミングの観点でノートブックを理解するには，ノートブック上の手戻りを単なる悪癖と片付けるのではなく，データサイエンスにおける手戻りの実態を多面的に分析することが必要不可欠である．そこで本研究では，データサイエンスの競技プラットフォームである Kaggle 上のノートブックデータに基づいて，次の 3 つのリサーチクエスチョンに沿った調査を行い，定量的分析を与える．

RQ1 ノートブックプログラミングにおける手戻りが，どの程度生じているか．

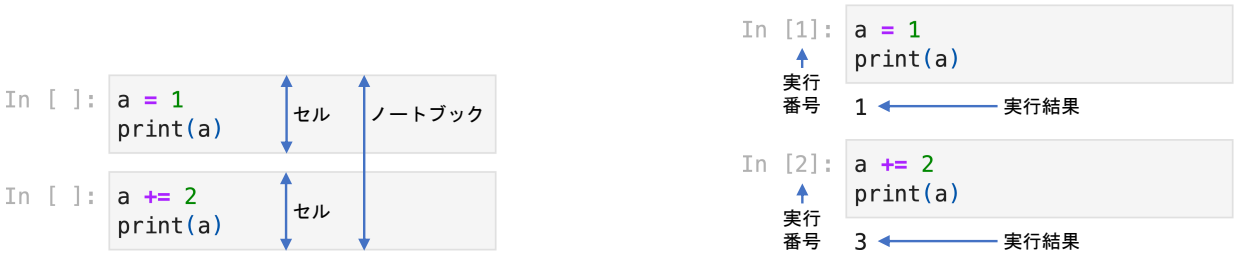
RQ2 ノートブックプログラミングにおける手戻りは，どの程度安全なのか．

RQ3 手戻りを安全に実行にする際に，どれだけオーバーヘッドが生じるか．

本研究の主な貢献は次の通りである．

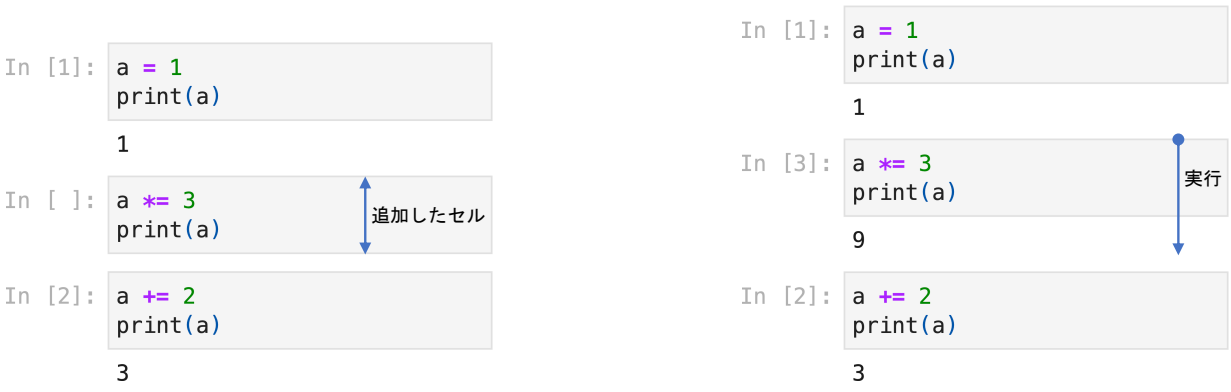
- Kaggle 上のデータを用い，ノートブックの改訂過程のデータセットを構築した（3 節）．構築したデータセットは GitHub Pages を用いて公開されている[5]．
- ノートブックプログラミングにおける手戻りの実態に関して定量的な分析を与えた（4 節）．具体的には，改訂の半数は 12 セル以上という比較的長い手戻りであること，改訂の 8.0%は安全ではない手戻りであることなどがこの分析によって明らかになった．
- ノートブックプログラミングにおける手戻りを安全に実行する際のオーバーヘッドを実験的に評価した（5 節）．この実験評価により，安全な実行によるオーバーヘッドは無視できない割合になる可能性が示された．

[5] https://mvnb-research.github.io/ppl2023/ppl2023.zip



(a) ノートブック　　(b) セルの実行

図 1: JupyterLab で作成したノートブックとセルの実行



(a) セルの追加　　(b) 追加セルの実行

図 2: セルの追加と実行

2　ノートブックプログラミング

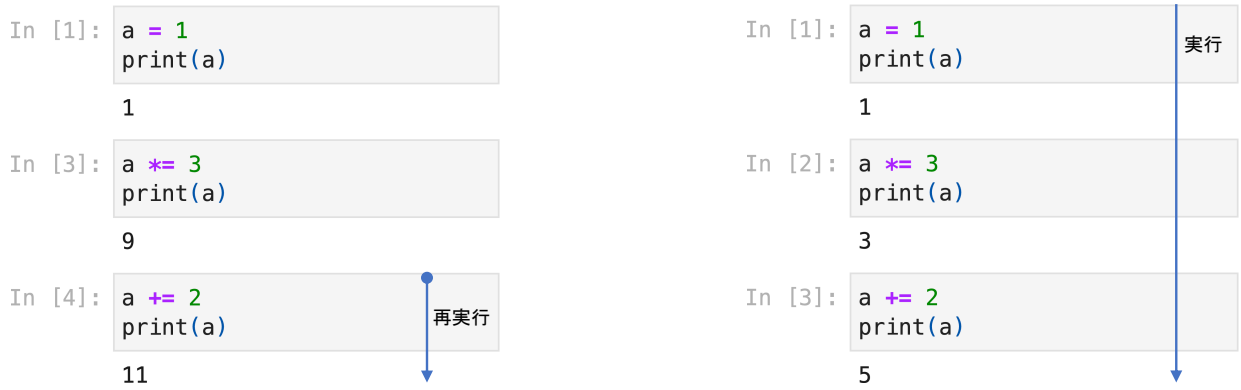
2.1　ノートブック環境

図 1a は，広く使われているノートブック環境である JupyterLab を用いて作成したノートブックである．ノートブックは，セルと呼ばれる，コード片を含む要素で構成される．例えば図 1a は，2 つのセルから成るノートブックである．各セルに記述されているのは Python のコード片である．

ノートブック環境では，セルを実行することができる．セルの実行は，ノートブック環境によって起動されたインタプリタによって行われる．図 1b は，図 1a の各セルを上から順に実行した後のノートブックである．セルの実行を行うと，その実行結果がセルの下部に表示され，実行番号がセルの左側に付与される．セルの実行では，起動中のインタプリタが継続して利用される．そのため，第 2 セルの実行直前の `a` の値は 1 であり，第 2 セルの実行結果は $1 + 2 = 3$ となる．

ノートブック環境下でのプログラミング（ノートブックプログラミング）は，セルを追加，編集，実行することで進められる．これらのセル操作には一切の制限がなく，任意の位置に新たなセルを追加し，任意のセルを編集及び実行することができる．例えば，第 1 セルの直後に新たなセルを追加し（図 2a），追加したセルのみを実行する（図 2b）といった操作が可能である．このようなセル操作を行った場合でも，起動中のインタプリタが継続してセル実行に利用される．そのため，図 2b の第 2 セル実行直前の `a` の値は 3 であり，図 2b の第 2 セルの実行結果は $3 \times 3 = 9$ となる．また，実行番号も（ノートブック上でのセルの配置とは関係なく）実行順序通りに採番される．

セルの実行は，既に実行したセルに対しても行える．図 3a は，図 2b の末尾セルを再実行した結果得られるノートブックである．この再実行をする直前の `a` の値は 9 であるため，末尾セルの再実行によって得られる結果は $9 + 2 = 11$ である．図 3a のように破壊的変更が積み重なり，コードから実行状態に関する推論が困難になった状況を解消する手段として，インタプリタの再起動がある．



(a) 末尾セルの再実行　　(b) インタプリタ再起動後の実行

図 3: セルの再実行とインタプリタの再起動

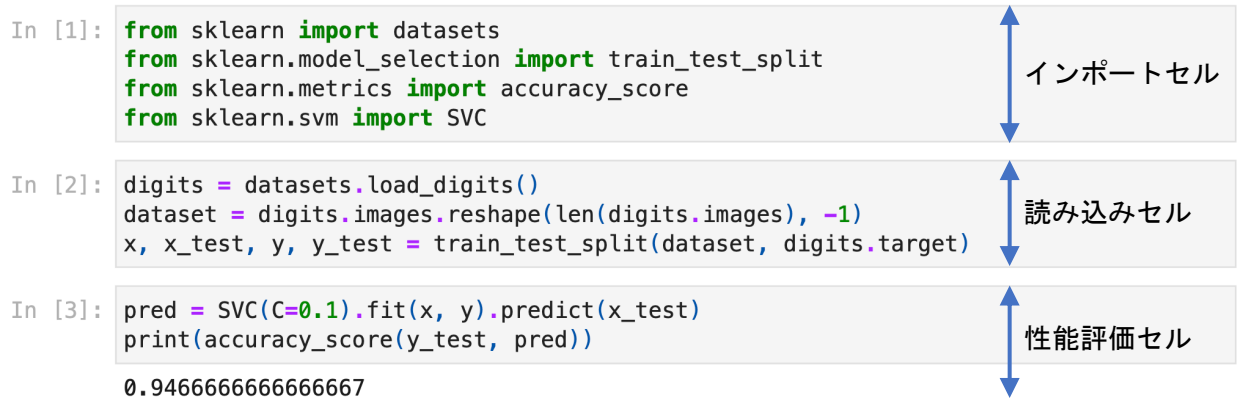


図 4: 手書き数字画像の分類に取り組むノートブック

図 3b は，インタプリタを再起動した後，上から順にセルを実行した結果得られるノートブックである．再起動によってこれまでの全てのセル実行は破棄されるため，コードは一切変更していないにも関わらず，図 3a と図 3b の実行結果は異なっている．また，インタプリタの再起動後には実行番号が 1 から順に採番されるようになる．

本節では JupyterLab で作成したノートブックを例に説明を行ったが，JupyterLab の前身である Jupyter Notebook[6] や Google Colaboratory[7] などの他のノートブック環境でも，上述したような任意のセル操作が可能であり，セルの実行方法も本節で紹介した通りとなっている．

2.2　ノートブックプログラミングにおける手戻り

ノートブックプログラミングにおける手戻りの手軽さと危険性を示すための例として，図 4 のノートブックを考える．図 4 は，scikit-learn という機械学習のための Python ライブラリを用い，手書き数字画像の分類に取り組んでいるノートブックである．図 4 には，ライブラリをインポートするセル（インポートセル），手書き画像データの読み込みと分割を行うセル（読み込みセル），Support Vector Classifier（SVC）の訓練と分類性能評価を行うセル（性能評価セル）の 3 つのセルがある．インポートセルと読み込みセルには表示する実行結果が存在しないため，図にはコードのみが表示されている．

[6] https://github.com/jupyter/notebook
[7] https://colab.research.google.com

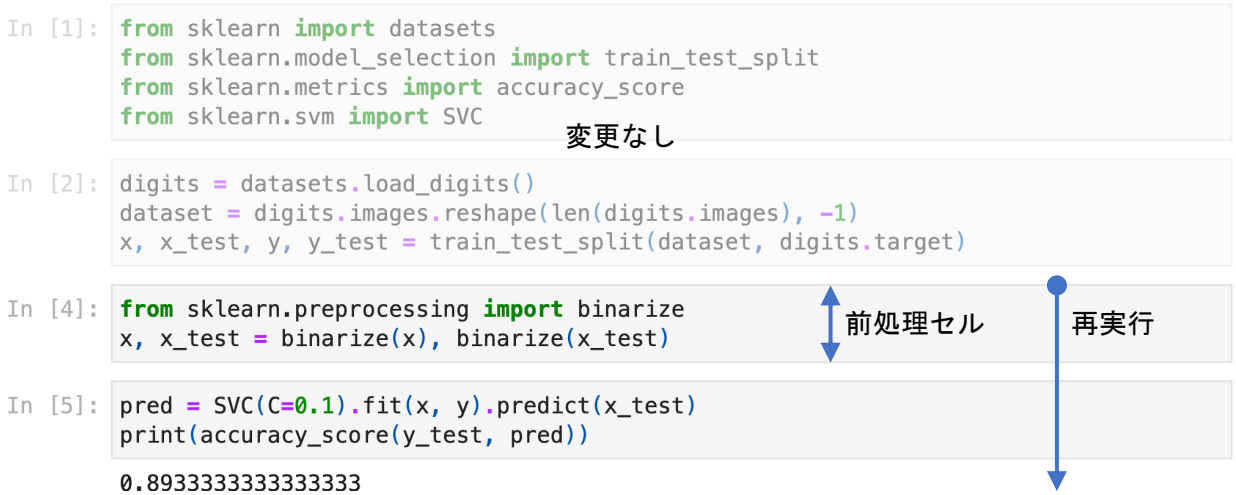


図 5: 前処理を追加し，後半セルのみ再実行を行ったノートブック

自由なセル操作が行えるノートブック環境では，手戻りを効率的かつ視覚的に分かりやすい形で行いやすい．例として，図 4 のノートブックに対して，データの二値化（binarize）処理を追加する手戻りを考える．そのような手戻りは，

1. 読み込みセルの直後に前処理セルを追加し，
2. 前処理セル以降のみを再実行する

という 2 つの操作で手軽に実現できる．図 5 は，上述の操作を行った結果得られるノートブックである．この手順は，コード編集を最小限に済ませつつ，読み込みセルの実行結果を再利用しながら（データ読み込みの再実行を回避しながら）手戻りを実現できており，作業効率が良い手順である．また，読み込みセルの直後にセルを追加することで，上から順にノートブックを読み解けるようになっており，ノートブックの可読性を維持できていることも評価できる．

しかし，末尾以外の位置へのセル追加や既存セルの編集などの手戻りは，ノートブックから誤った結論を導く危険性もある．例として，図 5 のノートブックを修正し，データの正規化（normalize）を試すことを考える．この際，

1. 前処理セルを編集して binarize を normalize に書き換え，
2. 前処理セル以降のみを再実行する

という前段落で述べたような手軽な操作手順では，正しく normalize の効果を観察することができず，誤った結論を導いてしまう．直前の手戻りで実行した前処理セルによって x と x_test は binarize 適用後のデータに書き換わっているため，この手順では，binarize と normalize の両方を適用した場合の結果を得ることになってしまうからである．このような誤解は，読み込みセルを再実行するか，インタプリタを再起動してインポートセルから全て実行し直すことで回避することができる．しかしながら，そのような操作手順はデータ読み込みの再実行を伴うため，作業効率が良いとは言えない．

上述した手戻りの危険性は定性的には明らかであるものの，そもそも手戻りが現実でどの程度発生しているのか，危険な手戻りが実際に行われているのかといった定量的な情報は，我々が知る限り存在しない．実際には手戻りの危険性が広く認識されており，現実のノートブックプログラミングでは「末尾以外の位置へのセル追加や既存セルの編集などは行わない」という習慣が存在する可能性も否定することはできない状況である．

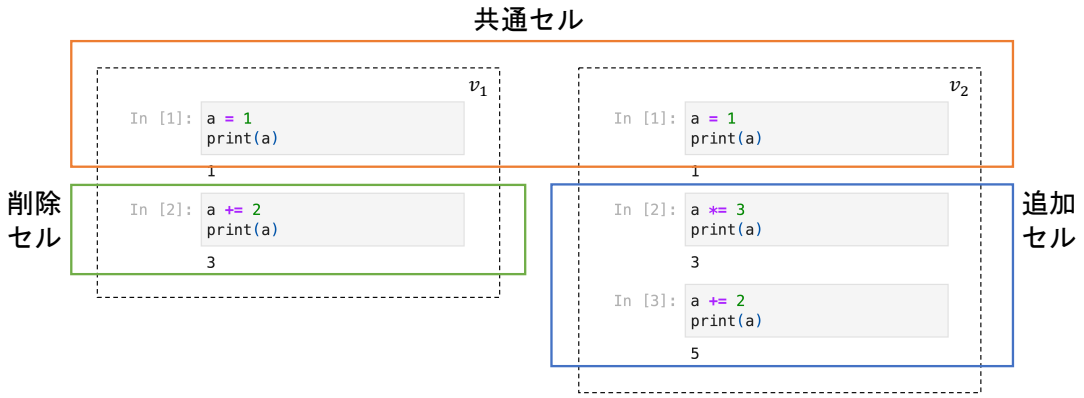


図 6: 改訂データの抽出

3　改訂データセットの構築

本研究では，現実のノートブックプログラミングにおける手戻りの実態を明らかにすることを目的とし，Kaggle から収集したノートブックの定量的な分析を行う．Kaggle はノートブック環境を提供するデータサイエンスの競技プラットフォームであり，その利用者は，与えられたデータから予測モデルをノートブック環境下で構築して予測性能を競ったり，予測モデルの構築に役立つ可視化に取り組んだノートブックを共有したりしている．Kaggle における競技には，初心者向けの “Getting Started” や本格的な競技である “Research” などの分類があり，各競技への参加はチームと呼ばれる 1 人以上の利用者集団を単位に行われる．各チームが競技のために作成するノートブックは，チーム内の利用者が望むタイミングで Kaggle 上に保存することができ，保存された全てのバージョンが Kaggle 上に残されている．保存されているノートブックには，コードだけでなく，実行結果も含まれている．また，Kaggle 上にあるノートブックには利用者による投票機能があり，優れた性能を示すノートブックや，解説が併記されたデータ可視化ノートブックが多くの票を集める傾向がある．

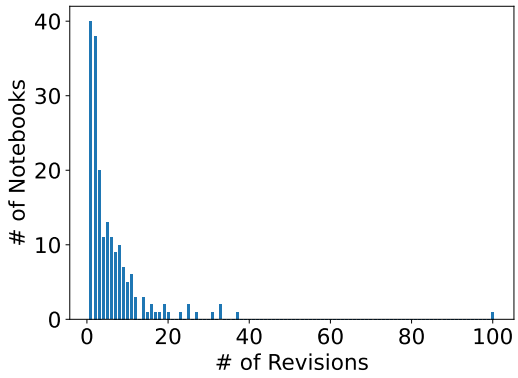
改訂データセットを構築するため，我々はまず Kaggle にあるノートブックの収集を行った．具体的には，“Research” に分類される参加チーム数上位の競技 20 個に対して作成された，得票数上位のノートブック 20 個の全バージョンのうち，(a) 記述された Python コードに構文エラーがなく，(b) 実行結果にエラーが含まれておらず，さらに (c) 実行番号が先頭から順に連番になっているもののみを収集した．(a) から (c) の条件を満たさないバージョンからは，実行可能なノートブックを復元することが困難であり，探索的過程のスナップショットとして不適当であると考えられたため，収集対象から除外した．

本研究で用いる改訂データセットの要素は，改訂前後での共通セルリスト，改訂により削除されたセルリスト，改訂により追加されたセルリストの 3 つ組である．各改訂データは，同一ノートブックの隣接バージョン v_1 と v_2 から，以下の方法により抽出した．

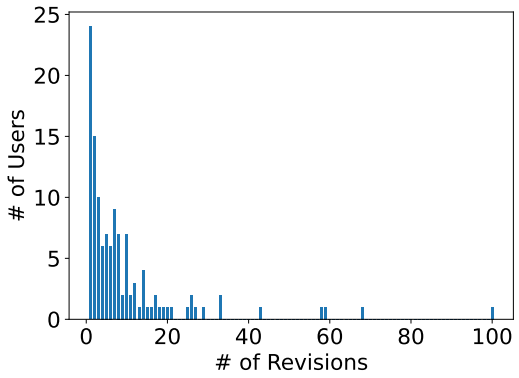
1. v_1 と v_2 に含まれる全てのコードからコメントを削除し，Python コードの整形ツール black[8] と isort[9] を用いてコードを正規化する．
2. v_1 と v_2 の先頭セルから順にコードの文字列比較を行い，共通した先頭セル群を改訂前後での共通セルリスト，v_1 の非共通セル群を改訂により削除されたセルリスト，v_2 の非共通セル群を改訂により追加されたセルリストとする．

例えば図 6 に示した隣接バージョン v_1，v_2 から得られる改訂データは，長さ 1 の共通セルリスト，長さ 1 の削除セルリスト，長さ 2 の追加セルリストから成る改訂データである．

[8] https://github.com/psf/black
[9] https://pycqa.github.io/isort



(a) ノートブックごとの改訂数



(b) ユーザごとの改訂数

	最小値	第 1 四分位数	第 2 四分位数	第 3 四分位数	最大値
ノートブックごとの改訂数	1	2	3	8	101
ユーザごとの改訂数	1	2	5	10	101

図 7: ノートブックごと/ユーザごとの改訂数

上述の方法で得られた改訂データセットには，1226 個の改訂データが含まれている．これらの改訂データは 127 人のユーザによって作られた 193 個のノートブックから得られたものである．図 7 は，ノートブックごと及びユーザごとの改訂数を示した図である．グラフ下の表は，図に示した各分布の統計量である．図表から分かる通り，改訂データの供給源のノートブック/ユーザには偏りがあるため，4 節では適宜この影響を考慮した分析を行う．

4　ノートブックプログラミングにおける手戻りの実態

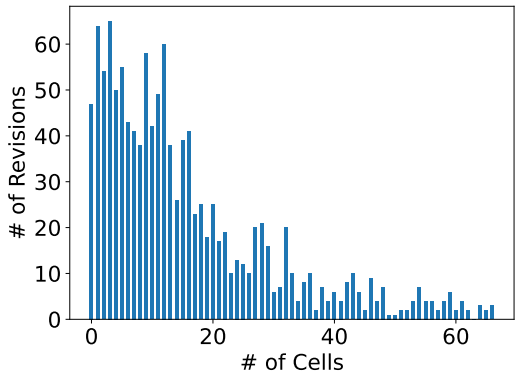
本研究では，既存セルの編集を手戻りと定義する．つまり，改訂データセットにおいては，削除セルリストが空でない改訂は手戻りとなる．この定義は，ノートブックが単なる対話的なプログラミング環境ではなく，上から順に読める/実行できる記録としても使われることを考慮した定義である．上から下への線形な読解/実行を想定する場合，上位セルの編集は，そのセル以下の全てのセル実行に潜在的に影響する．つまり，編集されたセル以下の全てのセルの実行結果は，記述されたコードと整合しない可能性がある，本質的に無効な結果となる．このようなことを考慮し，ノートブックプログラミングにおける手戻りとして上述の定義を採用した．

また，本研究では，手戻りセル数を削除セルリストの長さによって定義する．例えば，図 6 の手戻りセル数は 1 である．手戻りでない改訂も，統一的な図表化のため，手戻りセル数 0 の “手戻り” として取り扱う．

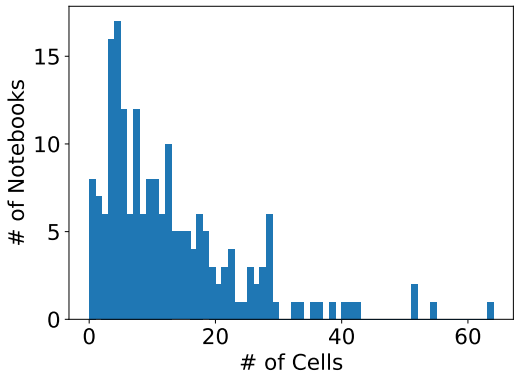
4.1　手戻りの頻度

図 8a は，改訂データセットの手戻りセル数の分布を示した図である．横軸は手戻りセル数，縦軸は改訂数である．図 8b は，各ノートブックの手戻りセル数の中央値の分布を示した図である．この図では縦軸はノートブック数である．図 8a では横軸は離散的であり，各棒の長さは棒の中央の値の個数を示す．一方図 8b では，横軸が連続的な値を取るため，各棒の長さは棒の左端から右端の範囲内の個数を示す．

図 8a と図 8b の両者から，改訂のほとんどは手戻りであることが分かる．具体的には，手戻りセル数 0 の改訂は 47 個（3.8%）のみであり，手戻りセル数中央値 1 未満のノートブック数も 8 個



(a) 手戻りセル数の分布



(b) 手戻りセル数中央値の分布

	最小値	第 1 四分位数	第 2 四分位数	第 3 四分位数	最大値
手戻りセル数	0	5	12	22	67
手戻りセル数中央値	0	4	9	17	64

図 8: 手戻りセル数/セル数中央値の分布

(4.1%) となっている．また，半数の手戻りは 12 セル以上の手戻り，ノートブックごとの中央値で見ても 9 セル以上の手戻りであり，比較的長い距離の手戻りが発生していることが観察される．

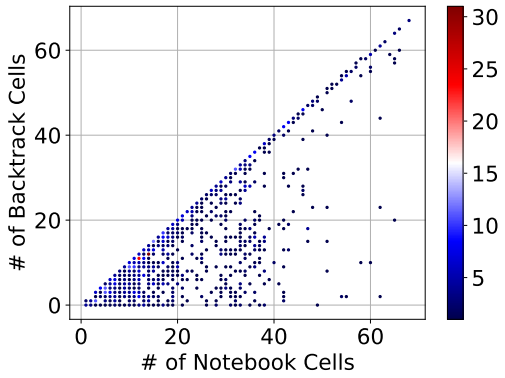
図 9a は，改訂前のノートブックセル数（共通セルリストの長さと削除セルリストの長さの合計）を横軸とし，手戻りセル数を縦軸として，両者の関係を示した図である．また，図 9b は，改訂前のノートブックセル数に対する手戻りセル数の割合を示した図である．

図 9a の下半分には点が散在しており，手戻りセル数はノートブックセル数によらず様々，つまり様々な位置への手戻りが生じていることが観察できる．また，図 9b からは，手戻りはノートブックの上部（相対値が 1 に近い範囲）に集中していることが観察される．特に手戻りセル数相対値が 1 である手戻り数は 243 個（19.8%）であり，最上部への手戻りの多くが先頭セルの編集であった．一方ノートブックの下半分を編集，つまり手戻りセル数相対値 0.5 未満となるような手戻り数は 401 個（32.7%）に留まり，手戻りの多くはノートブックの上半分に達していることが分かる．

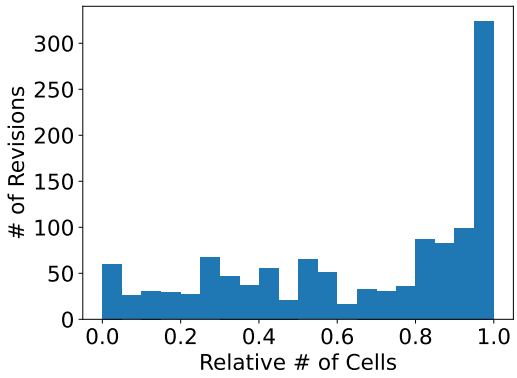
さらに，手戻り時にどのようなセル編集が行われているかを観察するため，手戻り時の非編集セル数についても調査した．ここで非編集セル数は，削除セルのうち，追加セルリストにも含まれているセルの数である．(例えば図 6 の手戻りの場合，非編集セル数は 1 となる．ノートブックの末端以外に単にセルを追加しただけである場合には，非編集セルと手戻りセル数が一致することに注意する.）図 10a は，横軸を手戻りセル数，縦軸を非編集セル数とし，それらの関係を示した図である．また，図 10b は，手戻りセル数に対する非編集セル数相対値の分布を示した図である．

図 10a と図 10b の両方から観察できる通り，1 回の手戻りにおいて編集されるのは 1 セルばかりではなく，複数セルの編集または削除がなされていることも珍しくない．特に，204 個（17%）の手戻りでは，非編集セル数相対値が 0.5 未満，つまりノートブック末尾から手戻り先までの半数以上のセルが編集または削除されていた．

RQ1 への回答　ノートブック末尾にセルを追加したり，末尾のセルを編集するという操作はあまり行われておらず，比較的長い手戻りが頻繁に行われている．また，手戻りをした後，複数のセルを同時に編集していることも少なくない．具体的には，改訂の半数は 12 セル以上の手戻り，ノートブックごとの中央値で観察しても半数は 9 セル以上の手戻りであった．また，17%の改訂では，改訂前のセルの半数以上が編集または削除されているということが明らかになった．



(a) ノートブックの長さと手戻りセル数の関係



(b) セル数相対値の分布

	最小値	第 1 四分位数	第 2 四分位数	第 3 四分位数	最大値
セル数相対値	0	0.38	0.76	0.96	1

図 9: 手戻りセル数相対値

4.2 手戻りの安全性

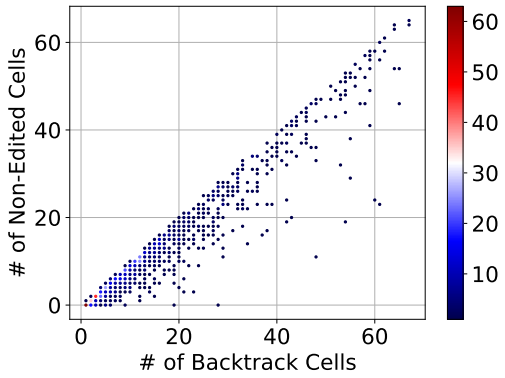
2 節で述べた通り，手戻りとその後のセル実行方法によっては，結果の誤解や実行エラーが生じる危険がある．そのような潜在的な危険がある手戻りが現実にどの程度存在しているかを調査するため，まず以下のように手戻りの安全性を定義する．

> 手戻り前のノートブックを先頭から全て実行した状態から，改訂データの削除セルリストにあるセルを削除し，追加セルリストにあるセルを追加，その後追加したセルのみを上から順に実行したとしても，手戻り後のノートブックを先頭から全て実行した場合と同等の結果を得られるならば，その手戻りは安全である．

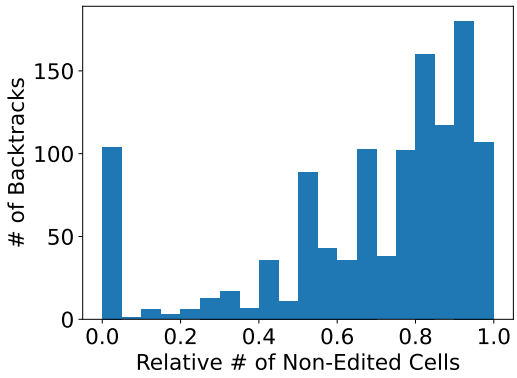
より簡潔に言えば，共通セルの再実行を回避した効率の良い実行方法でも全て再実行した場合と同等の結果が得られるならば，その手戻りは安全であるということである．但し，この定義の下で安全でないと判定される手戻りによって作られるノートブックが必ずしも誤解やエラーを生じさせるわけではないことに注意する．インタプリタの再起動を行えば結果の誤解や実行エラーが生じるのを回避できるからである．安全でない手戻りは，ある種の Smell のようなものである．

上述した安全性の定義の下，改訂データセットの全改訂を人手で静的に分析し，安全でない手戻り（効率の良い実行方法では結果が変わってしまう手戻り）の抽出と分類を行った．この人手での作業は保守的に行った．つまり，実行しないと安全か否かの判断がつかないものや，コードが複雑で判断が難しいものは全て安全であると判定し，静的に得られる情報を基に安全でないと説明できる手戻りのみを安全でないと判定した．このような人手での分析を採用したのは，実行して結果を比較する機械的判定に比べ，より安価により多くの改訂を分析できるためである．機械的な判定の実現には非常に多くの課題があるが，例えば，コードが公開されていても利用データは非公開になっている場合も多いこと，乱数の利用により結果を単純に比較するだけでは安全性の判定が困難であること，ライブラリのバージョンが明示されておらず再現実行が容易でないものも多いこと，単一バージョンの実行に数時間に要するものも少なくなく，非現実的な実験時間を要することなどが挙げられる．

表 1 は，上述の分析結果をまとめた表である．表から分かる通り，安全でない手戻りは極端に発生頻度が低いものではなく，一定程度（8.0%）存在していることが分かる．また，表の 1 行目と 2 行目や 3 行目を比較すると，安全でない手戻りは，一部の悪い習慣を持つユーザが大量に行ってい



(a) 手戻りセル数と非編集セル数の関係



(b) 非編集セル数相対値の分布

	最小値	第 1 四分位数	第 2 四分位数	第 3 四分位数	最大値
非編集セル数相対値	0	0.55	0.78	0.89	1

図 10: 非編集セル数の分布

表 1: 安全でない手戻り

	個数または人数	割合
安全でない手戻り	98	8.0%
安全でない手戻りを含むノートブック	43	22.3%
安全でない手戻りを行ったユーザ	33	26.0%

たり，出来の悪い特異なノートブックでのみ頻繁に発生しているわけでなく，多くのユーザ（およそ 4 人に 1 人）が様々なノートブック上で行っているものであることが分かる．今回の分析では保守的に分析を行ったことを考慮すると，実際にはこの表で示した以上の安全でない手戻りが発生していることが予想される．

また，発見された安全でない手戻りは以下の 5 種類に分類することができた．

二重のデータ更新 データの上書き更新が複数回行われてしまう手戻りである．具体的には図 5 を例に 2 節で述べたような操作である．特に目立ったのは，訓練データに対する前処理が二重に行われるものである．更新内容は値のスケールや形式の変換，行や列の追加など，多様であった．これに該当する手戻りは 49 個発見された．

二重のデータ分割 データ分割が二重に行われ，訓練データがコードから読み取れる以上に小さくなってしまっている手戻りである．具体的には付録の図 12a のような改訂である．二重のデータ更新であるとみなすこともできるが，特に数が多かったため独立した分類としている．このような手戻りは 13 個発見された．

二重のモデル訓練 モデルの訓練が二重に行われている手戻りである．具体的には，付録の図 12b のような改訂である．これに該当する手戻りは 9 個発見された．二重に訓練することは，学習回数を増やしたり，事前学習を行うのと同様であり，インタプリタの再起動などを行った際に結果の再現を困難にする要因となる．

実行状態の乖離 データの上書き更新がコードレベルでは消されているものの，継続して実行していた場合には実行状態が巻き戻されていない手戻りである．具体的には付録の付録の図 12c のような改訂である．このような手戻りは 15 個発見された．

変数削除によるエラー メモリ解放を促すために，しばしば読み込んだデータを `del` 文によって削除するということが行われている．この分類は，そのような削除操作により変数への参照を失い，手戻り後に期待したデータへアクセスできなくなっている手戻りである．具体的には，付録の図 12d のような改訂である．このような手戻りは 12 個発見された．

これらのうち，変数削除は実行によりエラーが発生するため比較的早期に問題が明らかになるが，それ以外はエラーが発生することなく処理が進むことも多い．そのため，適切に探索を行えていると思いながらプログラミングを進めた後，時間が経ってから結果の再現ができないことに気付く可能性が高い，非常に危険な手戻りである．

RQ2 への回答 安全でない手戻りは一般的に行われている．具体的には，保守的に数えても手戻りの 8.0%は安全ではなく，26.0%のユーザは安全でない手戻りを行なっていた．また，安全でない手戻りの多くは即座にエラーを引き起こさない，問題が発覚した段階では原因の究明が困難になっている可能性が高い非常に危険な手戻りであった．

4.3 妥当性への脅威

本節で示した手戻り頻度や安全性の傾向は，Kaggle 上の，参加チーム数の多い競技に向けて作られた，得票数の多いノートブックの分析を通して得られたものである．改訂データセットの構築にあたりノートブック中のセル数や取り組み内容による選別は行なっていないため，結果には一定程度の一般性があると考えられるが，GitHub などの別のプラットフォームからノートブックを収集した場合，別の基準で Kaggle からノートブックを収集した場合には，異なった傾向が観察される可能性がある．

また，手戻りの安全性については以下の 2 点に注意する必要がある．4.2 節で述べた通り，ノートブック提出時に全体の再実行が Kaggle 利用者の間で習慣付いている場合，本研究における安全ではない手戻りは，利用者の観点では安全である．その意味で，本研究で見つけた安全ではない手戻りと，利用者の観点で真に安全ではない手戻りには乖離がある．さらに，4.2 節で示した結果の妥当性は我々が行なった人手での抽出と分類に大きく依存する．それらの結果の妥当性を第三者によって検証可能にするため，本研究で用いた改訂データセットと抽出・分類の結果を公開している．

5 安全な実行のオーバーヘッド

インタプリタ再起動後に全てのセルを先頭から順に実行すれば，如何なる手戻りを行っていたとしても，問題は一切生じない．しかし，そのような安全性を重視した実行方法は，プログラミングを中断させるオーバーヘッドとなる．このようなオーバーヘッドの存在は定性的には指摘できるが，現実のノートブックにおいてどの程度のオーバーヘッドが存在するのかは定量的に明らかでない．本節では，収集したノートブックから選定した 3 個を用い，そのオーバーヘッドを実験的に評価する．

5.1 実験に用いるノートブック

本節の実験で用いるノートブックは以下の 3 個である．

ihelon/cassava-leaf-disease-exploratory-data-analysis Cassava Leaf Disease Classification という競技[10] に向けて，データ可視化に取り組んだノートブック[11]．この競技は，キャッサバの葉の画像から，その葉が感染している病気を予測する競技である．このノートブックでは，各病気の画像数の分布の観察，各病気のサンプル画像の表示などが行われている．以下では単に ihelon という名前でこのノートブックを参照する．

[10] `https://www.kaggle.com/competitions/cassava-leaf-disease-classification`
[11] `https://www.kaggle.com/code/ihelon/cassava-leaf-disease-exploratory-data-analysis`

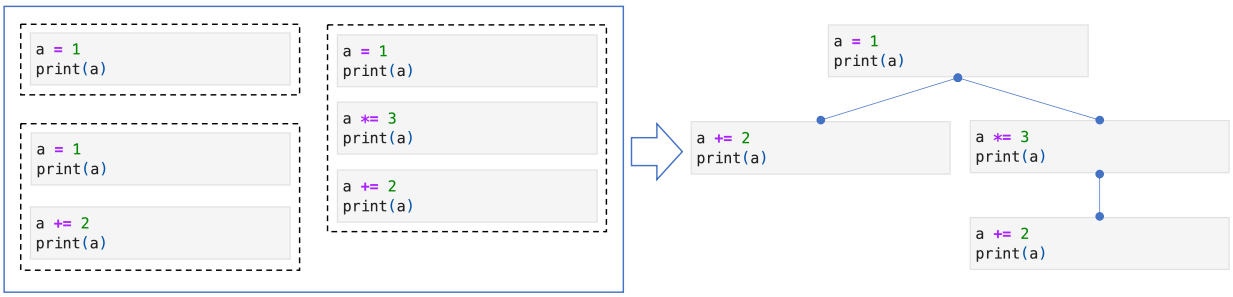


図 11: 木の構築

表 2: 構築された探索木の構造指標

ノートブック	セル数	高さ	幅	分岐数
ihelon	159	25	12	9
markpeng	91	37	4	3
paulorzp	118	13	21	12

markpeng/deepinsight-transforming-non-image-data-to-images Mechanism of Action Prediction（作用機序予測）という競技[12] で与えれているデータを用い，非画像データを画像データに変換し，畳み込みニューラルネットワークで取り扱う方法を例示しているノートブック[13]．CSV 形式のテキストデータを読み込み，可視化や値の正規化などを行った後，画像構築を行なっている．以下では markpeng という名前でこのノートブックを参照する．

paulorzp/gmean-of-low-correlation-lb-0-952x IEEE-CIS Fraud Detection という競技[14] に提出された勾配ブースティングを用いた学習結果 24 個を，Stacking という集団学習の手法で組み合わせ，組み合わせた結果の性能を評価しているノートブック[15]．IEEE-CIS Fraud Detection は，オンライン取引の記録から不正なものを識別する精度を競う競技である．以下では paulorzp という名前でこのノートブックを参照する．

これらのノートブックは，GPU を利用しておらず，最新バージョンの実行時間が 30 分以内，利用されているデータとライブラリが現在も取得可能であるものを収集したノートブック群からまず抽出し，抽出されたノートブックの中から，取り組まれている内容，実行するソフトウェア・ハードウェア上の制限，次節で述べる “探索木” の構造の多様性を考慮して選定した．

5.2 探索木の構築

上述の 3 個のノートブックのそれぞれについて，以下の方法により，探索の過程を表現する木構造（探索木）を構築した．

1. 最古のバージョンから，セルを節とし，隣接セル間に辺を持つ，直線状の木 T を構築する．
2. 最古以外の各バージョン V について，以下を繰り返す．
   (a) V から直線状の木 T' を構築する．
   (b) T の根から始まる経路のうち，T' にも存在する最長の経路 $N_{root}, \ldots, N_{tail}$ を探す．
   (c) T' の N_{tail} より下にある経路を，T の N_{tail} に対して枝として接続する．

[12] https://www.kaggle.com/competitions/lish-moa
[13] https://www.kaggle.com/code/markpeng/deepinsight-transforming-non-image-data-to-images
[14] https://www.kaggle.com/competitions/ieee-fraud-detection
[15] https://www.kaggle.com/paulorzp/gmean-of-low-correlation-lb-0-952x

表 3: 実行時間の比較

ノートブック	RERUN	DILL	FORK
ihelon	2.97×10^1 (duplicate: 41%)	3.57×10^1 (load: 6%; dump: 44%)	1.79×10^1
markpeng	2.46×10^3 (duplicate: 25%)	1.96×10^3 (load: 2%; dump: 5%)	1.84×10^3
paulorzp	1.34×10^2 (duplicate: 66%)	1.99×10^2 (load: 22%; dump: 55%)	4.36×10^1

例えば，図 11 に左側に示した 3 バージョンからなるノートブックからは，図の右側に示したような探索の木が得られる．このように探索木を構築することで，末尾へのセル追加が繰り返されるような，明らかに再実行が不要と判断できる改訂に対し，再実行コストを過大評価せずに済むようになる．例えば，手戻りの危険性を認識し，常に末尾へのセル追加しか行わないという方針をとっていたとする．このような場合に，先頭からの再実行を常に行う形で再計算コストを評価するのは不適当であるため，上述の方法で探索木を構築した．表 2 は，実験で用いるノートブックに対して構築された探索木の構造指標である．それぞれの探索木そのものは付録の図 13 に図示した．

5.3 オーバーヘッドの評価

安全な実行のオーバーヘッドを評価するため，Multiverse Notebook [10, 22] を用い，構築した探索木に沿ったセル実行を行なった．Multiverse Notebook は，各セルの実行を別々のプロセスで行うことで，各枝におけるセル実行同士の干渉を防ぎながら木構造のノートブックを実行できる環境である．各セルの実行に利用されるプロセスは，親セルのプロセスを Fork して得られるプロセスである．Fork によってプロセスを複製することで，先祖セルの状態を引き継ぎつつながらも，先祖セルの状態を破壊しないセル実行が実現されている．Fork によるプロセス複製では，メモリ空間は即座に物理的に複製されず，子プロセス側で書き込みが発生する際に物理的に複製される Copy-on-Write が採用されている．そのため，Fork によるプロセス複製と子プロセスでのセル実行は，差分チェックポイントの作成であると言える．

本実験では，まず Multiverse Notebook 上で探索木に沿った実行を行い，各セルの実行に要した時間を計測した．そして，それらのセルごとの実行時間から，探索木の根から葉までの全経路を通常のノートブック環境で実行した場合に要する時間を算出した．算出した時間と Multiverse Notebook で実行した場合の時間の差が，安全な実行のオーバーヘッドである．また，Weinman らによって提案 [20] された，差分チェックポイント手法についても実験を行なった．Weinman らの手法は，各セル実行後のインタプリタの状態を dill[16] を用いてファイルに保存しておき，手戻り時には適宜ファイルから状態を復元することで再計算を回避する手法である．

表 3 は，5 回の実験で計測または算出された時間の中央値を示した表である．RERUN 列に示したのは，探索木の全経路を通常のノートブック環境で安全に実行した場合の時間である．この列の duplicate の値は，セルの重複実行を行っている時間の割合である．また，DILL 列に示したのは，Weinman らの提案手法の場合に要する時間である．この列にある load/dump は，dill によって処理系を読み込み・保存するのに要した時間の割合である．さらに，FORK 列に示したのは，Multiverse Notebook で実行した場合の時間である．この列の結果は，実行時間の 2%の増加と引き換えに，7%から 35%の空間消費量を削減する Aggressive Tenuring [10, 22] を用いた場合の結果である．

表から分かる通り，RERUN 方式では，重複実行の時間が無視できない程度の割合，場合によっては半分以上の時間を重複実行に割いていることが分かる．また，DILL 方式では，RERUN 方式以上の時間を要したノートブックが 2 つあることも分かる．これは，取り扱うデータが巨大になれば，dill により再実行を回避するより，素直に再実行した方がオーバーヘッドが小さいということ

[16] `https://pypi.org/project/dill/`

である．実際，DILL の方が短時間であった markpeng は読み込むデータが 200MB 程度と小さい一方，他の 2 つのノートブックでは数 GB のデータ読み込みが行われていた．そして，当然ながら，いずれのノートブックでも FORK 方式が最も短時間で実行できている．

RQ3 への回答　実験を行なった範囲では，安全な実行によるオーバーヘッドは無視できない割合，最悪の場合半分以上の時間を再実行に割くことになることが判明した．本実験で実行したノートブックは 3 つのみであり，この表から一般的な結論を導くことは困難であるが，安全な実行によるオーバーヘッドは無視できない割合であること，現実のデータサイエンスプログラミングで取り扱うデータサイズでは Weinman らの手法が再実行する以上に時間がかかる可能性が高いことが暗示される．

6　関連研究

Kery ら [7] は，インタビュー調査を踏まえて，データサイエンスにおける現実の探索的プログラミングを分析し，特徴を整理した．探索過程において手戻りを含むことが特徴の 1 つとして挙げられているが，その具体的な様相については論じられていない．Chattopadhyay ら [1] は，データサイエンティストに対して，ノートブックを利用する際の苦痛をインタビュー調査した．その結果，被験者が重要かつ難しいと思う事柄は，コードリファクタリングとデプロイであった．

Rule ら [17] は，GitHub 上のノートブックや学術的なデータ解析のノートブックを対象に，ノートブック内のドキュメンテーションを分析した．そして，ノートブック自体は，データ解析のストーリーをめったに説明しないことを明らかにした．他に興味深いこととして，彼らが調査した GitHub 上のノートブックの 43.9%は，セルの実行順序が非線形であった．これは，手戻りによって前方のセルを編集・再実行していることを示す結果である．

Pimentel ら [12,13] は，GitHub 上のノートブックを対象に，計算結果の再現性を調査した．863,878 の有効なノートブックの内，エラーなく実行できたものが 24.11%であり，同じ結果を再現できたものが 4.03%であった．更に，少なくとも 1 つのスキップを含む実行順序が曖昧なものが 74.53%であり，実行順序が曖昧でないものの中でも 36.36%が，セル順通りではなかった．これも，手戻りが頻発していることを表す結果である．

Wang ら [19] は，Jupyter チームがまとめた有名なノートブック[17]を対象に，コードセル中の Python コードの品質を分析した．コード品質の指標として，Python 標準のコーディングスタイル（PEP8）への準拠・未使用変数の有無・deprecated ライブラリの利用の 3 点を調査した結果，36.26%のソースコード行に PEP8 検査器のエラーが生じ，40.51%のノートブックに未使用変数が含まれ，scikit-learn[18]を利用しているものの 35.05%が deprecated API を利用していることが示された．これは，ノートブックが低品質のコードを助長するという批判 [5] を裏付ける結果である．

Dong ら [3] は，GitHub 上からサンプリングしたノートブックのバージョン履歴を，手作業で定性的に分析し，ノートブックを綺麗にする活動の実態を調査した．1 番多かったのは，コードからコメントを削除することで，次に多かったのは，コードにコメントを追加することであった．そして，3 番目に多かったのは，セルの順番を変えることであった．これは，コードセルが非線形に実行された後，その実行順に応じて事後的に整列されることを示唆している．

Grotov ら [4] は，GitHub 上のノートブック中の Python コードと Python スクリプトを，構造面とスタイル面について比較して分析した．構造面として，ユーザ定義関数に大きな違いがあり，ノートブックコードは少数のユーザ定義関数を何度も用いる傾向にあり，その関数が密結合している傾向にあることが示された．スタイル面については，ノートブックコードには，現代の IDE を使っていれば回避きるようなスタイル上の問題が多数あり，コード品質が低いことが示された．こ

[17] https://github.com/jupyter/jupyter/wiki
[18] https://scikit-learn.org/stable/

れは，Wang ら [19] の結果と整合する．Grotov らは，スタイル面の問題に関して，ノートブックという媒体ではなく，反復的開発という使い方に起因するものであると結論付けた．これは，手戻りを軸としてノートブックプログラミングを分析していると本研究の立場と整合する．

Koenzen ら [8] は，Rule ら [17] のデータセットを対象として，コード複製を分析した．その結果，レポジトリ単位のコード複製比率は，指数的に分布しており，平均約 5%，最大 47.5%であった．そして，複製されたコード中のタスクとして最も多かったものはデータ可視化であり，21.35%を占めた．Ritta ら [16] は，Kaggle 上の Grandmaster[19] 競技者サンプル 3 人のノートブックを対象に，コードの複製と再利用を調査した．最もよく再利用されるコードは競技者毎に異なっていたが，3 人に共通して import は再利用されるという結果となった．

Ramasamy ら [15] は，GitHub 上のノートブックに基づいて，データサイエンスのワークフロー情報でラベル付けされたノートブックのデータセット DASWOW を構築した．そして，DASWOW に基づいて，データサイエンスのノートブックが，どんなタスクから構成され，どのように遷移するかを定量的に分析した．そして，教師あり学習で DASWOW からセルの分類器を作成した．彼らの分類器は，DASWOW サブセットのテストデータセットに対して，タスクの頻度分布や遷移確率の観点で有効な予測を与えた．

de Santana ら [2] は，GitHub 上のレポジトリと Stack Overflow 上の投稿をマイニングして，ノートブック（及びノートブック環境）のバグ分析を行った．バグの種類に 8 つの大分類を与え，バグの影響度合いを 5 つに分類し，バグの根本原因を 10 に分類した．更に，様々な背景を持つデータサイエンティストを対象に，ノートブックに関する問題をインタビュー調査を行った．その結果，必ずしも被験者はソフトウェア工学の知識を持っておらず，その知識の有無が，バグの根本原因の特定や修正に重要であることが判明した．また，被験者の 1 人は，Grus による批判 [5] と同様に，Jupyter より VSCode を使っているときの方が，良質のコードが書けると回答した．

Raghunandan ら [14] は，GitHub 上のノートブックを定性的に分類し，探索と説明のスペクトラムを与えるルーブリック（評価尺度・達成段階・スコア）を定義した．最低（最も探索的）は，データを理解するためだけのものとされ，最高（最も説明的）は，広い聴衆にデータ解析のワークフローを伝えるものとされた．GitHub 上のノートブックの時系列を分類すると，探索的段階から探索的段階に進むのが 22.6%，探索的段階から説明的段階に進むのが 15.1%，説明的段階から説明的段階に進むのが 60.1%，説明的段階から探索的段階に進むのが 2.09%という結果となった．

7　おわりに

本論文では，現実のノートブックプログラミングを理解することを目的に，Kaggle から収集したノートブックを用い，手戻りに関する定量的な分析を行なった．具体的には，手戻りの発生頻度や距離，安全性について調査（4 節）し，さらに安全な実行に要する時間的オーバーヘッドを実験的に評価（5 節）した．これらの調査と実験により，比較的長距離の手戻りが高頻度で発生していること，安全でない手戻りは珍しくないこと，安全な実行のためには大きなオーバーヘッドを要する可能性が定量的に示された．

しかしながら，5.3 節で述べた通り，実験で用いたのは 3 つのノートブックのみであるため，オーバーヘッドについて一般性のある結論を導くのは困難である．特に，今回用いたノートブックには，可視化をしながら対話的に高性能なモデル構築を目指す競争的なノートブックは含まれておらず，偏りがある状況である．量的にも質的にも実験評価を拡充させ，オーバーヘッドについてより一般的な結論を得ることが今後の重要な課題である．

[19] 2022-12-27 付の Kaggle Rankings によると，上位 0.145%の最高ランク競技者．

謝辞　本研究は科学研究費助成事業（21K21279）の助成を受けたものである．本論文の草稿に対して有益なコメントを与えてくれた森畑明昌氏に感謝する．

参考文献

[1] Souti Chattopadhyay, Ishita Prasad, Austin Z. Henley, Anita Sarma, and Titus Barik. What's wrong with computational notebooks? pain points, needs, and design opportunities. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, pp. 1–12. ACM, 2020.

[2] Taijara Loiola de Santana, Paulo Anselmo da Mota Silveira Neto, Eduardo Santana de Almeida, and Iftekhar Ahmed. Bug analysis in jupyter notebook projects: An empirical study. `https://arxiv.org/abs/2210.06893`, 2022.

[3] Helen Dong, Shurui Zhou, Jin L.C. Guo, and Christian Kästner. Splitting, renaming, removing: A study of common cleaning activities in jupyter notebooks. In *Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops*, ASEW '21, pp. 114–119. ACM, 2021.

[4] Konstantin Grotov, Sergey Titov, Vladimir Sotnikov, Yaroslav Golubev, and Timofey Bryksin. A large-scale comparison of python code in jupyter notebooks and scripts. In *Proceedings of the 19th International Conference on Mining Software Repositories*, MSR '22, pp. 353–364. ACM, 2022.

[5] Joel Grus. I don't like notebooks. JupyterCon 2018, 2018. `https://docs.google.com/presentation/d/1n2RlMdmv1p25Xy5thJUhkKGvjtV-dkAIsUXP-AL4ffI/edit?usp=sharing`.

[6] Andrew Head, Fred Hohman, Titus Barik, Steven M. Drucker, and Robert DeLine. Managing messes in computational notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, pp. 1–12. ACM, 2019.

[7] Mary Beth Kery and Brad A. Myers. Exploring exploratory programming. In *Proc. the 2017 IEEE Symposium on Visual Languages and Human-Centric Computing*, VL/HCC '17, pp. 25–29. IEEE, 2017.

[8] Andreas P. Koenzen, Neil A. Ernst, and Margaret-Anne D. Storey. Code duplication and reuse in jupyter notebooks. In *Proceedings of the 2020 IEEE Symposium on Visual Languages and Human-Centric Computing*, VL/HCC '20, pp. 1–9. ACM, 2020.

[9] Sam Lau, Ian Drosos, Julia M. Markel, and Philip J. Guo. The design space of computational notebooks: An analysis of 60 systems in academia and industry. In *Proceedings of the 2020 IEEE Symposium on Visual Languages and Human-Centric Computing*, VL/HCC '20, pp. 1–11. IEEE, 2020.

[10] Tomoki Nakamaru and Shigeyuki Sato. Multiverse Notebook: A notebook environment for safe and efficient exploration. In *Companion Proceedings of the 2022 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*, SPLASH Companion 2022, pp. 7–8. ACM, 2022.

[11] Jeffrey M. Perkel. Why jupyter is data scientists ' computational notebook of choice. *Nature*, Vol. 563, pp. 145–146, 2018.

[12] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. A large-scale study about quality and reproducibility of jupyter notebooks. In *Proceedings of the 2019 IEEE/ACM 16th International Conference on Mining Software Repositories*, MSR '19, pp. 507–517. IEEE, 2019.

[13] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. Understanding and improving the quality and reproducibility of Jupyter notebooks. *Empir. Softw. Eng.*, Vol. 26, No. 4, pp. 65:1–65:55, 2021.

[14] Deepthi Raghunandan, Aayushi Roy, Shenzhi Shi, Niklas Elmqvist, and Leilani Battle. Code code evolution: Understanding how people change data science notebooks over time. `https://arxiv.org/abs/2209.02851`, 2022.

[15] Dhivyabharathi Ramasamy, Cristina Sarasua, Alberto Bacchelli, and Abraham Bernstein. Workflow analysis of data science code in public GitHub repositories. *Empir. Softw. Eng.*, Vol. 28, No. 1, pp. 7:1–7:47, 2023.

[16] Natanon Ritta, Tasha Settewong, Raula Gaikovina Kula, Chaiyong Rakhitwetsagul, Thanwadee Sunetnanta, and Kenichi Matsumoto. Reusing my own code: Preliminary results for competitive coding in jupyter notebooks. In *Proceedings of the 2022 29th Asia-Pacific Software Engineering Conference*, APSEC '22, 2022.

[17] Adam Rule, Aurélien Tabard, and James D. Hollan. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pp. 1–12. ACM, 2018.

[18] Tetsuo Tamai and Akito Itou. Requirements and design change in large-scale software development: analysis from the viewpoint of process backtracking. In *Proc. the 1993 15th International Conference on Software Engineering*, ICSE '93, pp. 167–176. IEEE, 1993.

[19] Jiawei Wang, Li Li, and Andreas Zeller. Better code, better sharing: On the need of analyzing jupyter notebooks. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, ICSE-NIER '20, pp. 53–56. ACM, 2020.

[20] Nathaniel Weinman, Steven M. Drucker, Titus Barik, and Robert DeLine. Fork it: Supporting stateful alternatives in computational notebooks. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, pp. 307:1–307:12. ACM, 2021.

[21] Young Seok Yoon and Brad A. Myers. An exploratory study of backtracking strategies used by developers. In *Proc. the 2012 5th International Workshop on Co-operative and Human Aspects of Software Engineering*, CHASE '12, pp. 138–144. IEEE, 2012.

[22] 中丸智貴, 佐藤重幸. セル単位の実行状態分離を備えたノートブックプログラミング環境. 第 139 回プログラミング研究発表会, pp. 2022–1–(1):1–11, 2022.

A　付録

```
digits = datasets.load_digits()
x = digits.images.reshape(len(digits.images), -1)
```

```
x, x_test = train_test_split(x, train_size=0.5)
```

変更なし

```
digits = datasets.load_digits()
x = digits.images.reshape(len(digits.images), -1)
```

```
x, x_test = train_test_split(x, train_size=0.1)
```

(a) 二重のデータ分割

```
model = SVC(C=0.1)
```

```
digits = datasets.load_digits()
dataset = digits.images.reshape(len(digits.images), -1)
x, x_test, y, y_test = train_test_split(dataset, digits.target)
```

```
pred = model.fit(x, y).fit(x, y).predict(x_test)
```

変更なし

```
model = SVC(C=0.1)
```

```
digits = datasets.load_digits()
dataset = digits.images.reshape(len(digits.images), -1)
x, x_test, y, y_test = train_test_split(dataset, digits.target)
x, x_test = binarize(x), binarize(x_test)
```

```
pred = model.fit(x, y).fit(x, y).predict(x_test)
```

(b) 二重のモデル訓練

```
digits = datasets.load_digits()
dataset = digits.images.reshape(len(digits.images), -1)
x, x_test, y, y_test = train_test_split(dataset, digits.target)
```

```
x, x_test = binarize(x), binarize(x_test)
pred = SVC(C=0.1).fit(x, y).predict(x_test)
print(accuracy_score(y_test, pred))
```

変更なし

```
digits = datasets.load_digits()
dataset = digits.images.reshape(len(digits.images), -1)
x, x_test, y, y_test = train_test_split(dataset, digits.target)
```

```
pred = SVC(C=0.1).fit(x, y).predict(x_test)
print(accuracy_score(y_test, pred))
```

(c) 実行状態の乖離

```
digits = datasets.load_digits()
dataset = digits.images.reshape(len(digits.images), -1)
x, x_test, y, y_test = train_test_split(dataset, digits.target)
```

```
pred = SVC(C=0.1).fit(x, y).predict(x_test)
del x, x_test
```

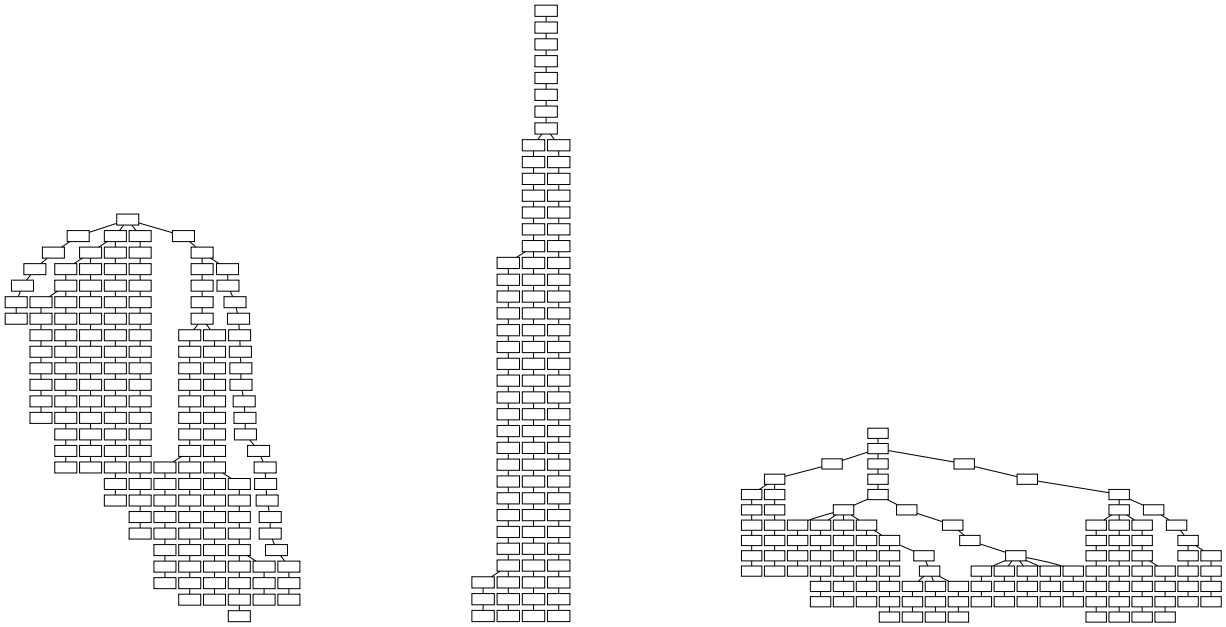
変更なし

```
digits = datasets.load_digits()
dataset = digits.images.reshape(len(digits.images), -1)
x, x_test, y, y_test = train_test_split(dataset, digits.target)
```

```
pred = SVC(C=0.01).fit(x, y).predict(x_test)
del x, x_test
```

(d) 変数削除による実行失敗

図 12: 安全でない手戻り例



(a) ihelon　(b) markpeng　(c) paulorzp

図 13: 構築された探索木