Vol.2022-SE-210 No.27
2022/3/12

Kaggle 熟練度に着目したデータ分析プログラム実装におけるソースコード再利用方法の探索的分析

池上 綾乃[1,a)]　佐藤 郁弥[1]　Ani Hovhannisyan[1]　石尾 隆[1]　松本 健一[1]

概要：データ分析技術は情報社会における重要な技術のひとつである．世界規模のデータ分析コミュニティである Kaggle では，データ分析技術を競うコンペティションの実施や，データ分析に使用したプログラムの公開，共有が行われている．データ分析プログラムには，プログラムの再利用がコピー＆ペーストで行われるという特徴がある．一方で，類似するプログラム片を複数箇所に記述すると，保守性の低下を招くと言われており，これを防ぐためには関数の定義やライブラリの活用が必要である．しかし，データ分析プログラムの実装において，プログラムの再利用は容易だが，保守作業が困難であるという報告があり，保守性を保ちながらプログラムを再利用するのは難しいと考えられる．本研究では，データ分析の熟練者はプログラムを適切に再利用しているという仮説に基づいて，Kaggle で定義されている熟練度の異なる作者が作成したプログラムに対して，類似するプログラム片の割合やライブラリの利用方法を比較する分析を行った．その結果，熟練度が高い作者ほど関数を多く定義し，類似するプログラム片の割合が少ない傾向にあることがわかった．また，使用されるライブラリの種類に大きな差異は見られなかった．これらの結果から，プログラム片の再利用の観点でデータ分析の初学者が熟練者に近づくには，多くのライブラリを学習することより，類似する処理がある場合には自作関数を定義するような工夫が重要であると考えられる．

1. はじめに

2010 年代頃から起こった機械学習ブームから，データサイエンスの重要性が高まっている．このような背景から，世界規模のデータ分析コミュニティである Kaggle[1] では，データ分析技術を競うコンペティションの実施や，コミュニティに参加しているユーザが実装したプログラムの公開，共有が行われている．データ分析に用いられるプログラミング言語の中で特に人気があるのは Python [2] である．Python は機械学習や統計，グラフ描画などの機能に関するライブラリが豊富であり，プログラミング初心者でも学習しやすい言語であると言われていることから，近年人気があるプログラミング言語である [3].

Kaggle 上でデータ分析プログラムを実装する際に用いられる統合開発環境（IDE）が Jupyter Notebook である．Jupyter Notebook が広く利用されていることから，Notebook の利用実態に関する分析 [4], [5], [6] が行われている．これらの研究では，Notebook ファイルに記述された Python プログラムはアプリケーション開発のために記述されるプログラムよりも比較的短く, 単純な構造で実装されている一方で、Notebook に出力された結果と上から実行した結果が一致しないことがあるという再現性の低さがも指摘されている．しかし，Notebook ファイルにはソースコードとグラフやテキストなどを一つのファイルにまとめることができるという特徴から，ドキュメンテーションの側面も持っているため，Notebook の質を保つことは重要である．

Jupyter Notebook を利用する上で重要かつ困難な作業の一つが，ソフトウェアの振る舞いを変更せず，内部の構造を改善するリファクタリングである [4], [7]．ソースコードの品質が低下する要因の一つとして，コードクローンが挙げられる [7]．コードクローンとは，類似もしくは完全一致するソースコード片のことを指す [8]．コードクローンは開発者がソースコードを再利用する際にコピー& ペーストしたり，同様の処理を繰り返し記述したりすることによって発生する．実際に，Jupyter Notebook 中のコードクローンに関する既存研究では，StackOverflow などの外部ソースからのコピー& ペーストによってコードクローンが発生することが報告されている [6], [9]．コードクローンを解消する方法には関数抽出があり，類似する処理を関数として記述すればよいことは知られているが，熟練したデー

[1] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology, Ikoma, Nara 630–0192, Japan
[a)] ikegami.ayano.hs9@is.naist.jp

図 1 熟練者によるソースコードの再利用例

タ分析者が実際にそのような方法を適用しているかどうかは明らかになっていない．

本研究の目的は，データ分析の熟練度の差でソースコードの再利用方法に関する特徴が異なるのかを明らかにすることである．図 1 は，Kaggle のある熟練者が実際に使っていた関数の利用例である[*1]．図の左側に示したコード片は複数のグラフを作成するために関数であり，Notebook 中では計 5 回呼び出され，図の右側に示したような異なるグラフを描画している．このような事例が存在することから，熟練者は非熟練者と比べ，類似するプログラムを複数記述するのではなく，関数を定義してプログラムの再利用をすることが多いという仮説を立てた．熟練度ごとにプログラムの特徴が異なることが明らかになれば，非熟練者が熟練者に近づくための指針を提言することが可能である．

そこで，この仮説を検証するために，非熟練者と熟練者のソースコード再利用の特徴を比較した．本研究では，熟練度による比較を行うため，Kaggle からデータ分析プログラムを取得し, コードクローンと使用されているライブラリ・関数の違いを分析する調査課題を設定した．

- RQ1. 熟練度によるコードクローンの特徴分析
- RQ2. 熟練度による自作関数の特徴分析

この分析により，データ分析に特化した Python プログラムの再利用・メンテナンス支援等に向けた有用な知見を得ることを目指す．

2. 背景

本節では Jupyter Notebook およびコードクローンについて解説する．

2.1 Jupyter Notebook

Jupyter Notebook とは，対話的にプログラムを実行できる IDE である．複数の言語に対応しており，Python や R などでプログラムを実装することができる．Jupyter Notebook にはセルという単位があり，セルごとにプログラムや，Markdown のようなテキストデータ，数式などを記述することができる．ファイルの内部表現には JSON 形式が用いられており，セルの種類やその中身が保存されている．

近年，Jupyter Notebook に関する実証的研究が行われている．Pimentel ら [5] は，Jupyter Notebook に関する実証的研究が十分に行われていないという背景から，264, 023 個の Github リポジトリに含まれる 1, 159, 166 個の Jupyter Notebook を分析した．この研究では Notebook に書かれたプログラムを実行しなおしても，Notebook に保存された実行結果が得られないことがあるという再現性の低さを指摘したほか，よく出現する Python の構文の分布から，Notebook で実装されるプログラムは従来の Python プログラムよりも比較的簡単な文法かつ短く記述されていると報告した．Chattopadhyay ら [4] は，Jupyter Notebook のユーザに利用する上で重要もしくは，難しいと感じる作業に関する調査を行った．調査の結果，重要かつ困難な作業として，バージョン管理，リファクタリング，長時間実行するジョブの管理，簡単にデプロイすることが挙げられている．特に，リファクタリングにおいては，Notebook 内で利用できるコーディング支援はほとんど存在しないと報告している．

[*1] https://www.kaggle.com/daishu/covid-19-knowledge-graph

これらの既存研究では，Jupyter Notebook に関する基礎分析やユーザへの質問票による調査が行われている．本研究では，既存研究で明らかになったプログラムの質が低いことに対し，熟練者が非熟練者と比べてどのように対応しているのかについて分析する．

2.2 コードクローン

コードクローンとは，類似するコード片の組のことである [7]．コードクローンを発生させる操作として，コピー& ペースト，リポジトリのフォーク，ソフトウェアの設計やロジックの再利用が挙げられるが，特にコピー& ペーストによる発生が多いと言われている [10]．コードクローンはコードを再利用する際の手間がかからない一方で，コピー元に不具合やバグが含まれていた場合，それらを広めてしまう恐れがある．したがって，複数利用されるようなコード片は共通の関数やモジュールとしてまとめることによって，将来のコード再利用のコストを下げることができる．

また，Roy らによってコードクローンの分類が 4 種類定義されている [8]．

- Type 1: コメント，空行以外が一致するコード片の組
- Type 2: コードの構造（構文木）が同一だが，識別子，型，文字列などの要素が異なるコード片の組
- Type 3: Type2 のコードクローンにおける違いに加え，文を追加・削除・修正等されたコード片の組
- Type 4: コード構造は異なるが，同じ振る舞いをするコード片の組

Type 1, Type 2 のコードクローンの定義は明確であるが，Type 3 のクローンの定義は文献によって異なる．Type 3 のコードクローンを検出する場合，コード片のテキストの類似度が用いられる [11]．類似度の閾値を設定し，閾値を超えた類似度を持つコード片の組をコードクローンとして検出する．Type 1,2 のコードクローンは構文が同一であるという制約がある一方で，Type 3 のコードクローンでは構文が同一でなくとも類似度が閾値以上であればコードクローンとみなす種類である．また，Type 4 のコードクローンはテキストの類似度が高くなくとも，同じ振る舞いをするコード片の組み合わせであればコードクローンと見なす分類である．しかし，Type 4 のコードクローンの検出は困難であることから，本研究では意味的なコードクローンの分類として用いられる Type 3 のコードクローンに着目した．

コードクローンは一般的に，ソフトウェアの保守性を低下させる要因の一つと言われている [7]．コードクローンにバグや脆弱性が含まれる場合に，一つ一つ探し出して修正する必要があるためである．Jupyter Notebook 中のコードクローンに関する研究では，Koenzen ら [6] は，Jupyter Notebook 中のコードクローンが含まれる量を分析し，Notebook 中の平均 7.6% のセルにコードクローンが含まれていることを明らかにした．さらに，Källén ら [9] は，70% 以上のセルは他のセルからのコードクローンであることや，50% の Notebook にはコードクローンでないセルが存在しなかったと報告している．

既存研究により，Jupyter Notebook ではコードクローンが起こりやすいことが明らかになっている．しかし，熟練度に着目したソースコードの再利用方法の差を分析する研究はこれまでに行われていない．熟練者がコードクローンを発生させることなくソースコードの再利用する方法を実践している場合，非熟練者への教育やデータ分析プログラムのメンテナンス支援につながる知見が得られる可能性がある．

3. データ収集

本節では本研究で利用したデータセットについて説明する．本研究では，Kaggle で公開されている Notebook を収集したデータセットである KGTorrent[12] を利用する．既存研究は GitHub[13] 上で公開された Notebook を分析しているが． Kaggle に公開されている Notebook はデータ分析というドメインに特化しているほか，ユーザが参加したコンペティションなどの成績によって定められる階級制度があり，本研究で着目する熟練度ごとに比較することができるという利点があるため，KGTorrent をデータセットとして利用した．

KGTorrent は 2020 年 10 月 27 日時点で公開されている 248,761 個の Python で記述された Notebook と，Notebook に付随するメタデータを記録した関係データベースから構成される．メタデータには，ユーザの基本情報や，Notebook に付与されたタグ，Notebook 内で使用された入力データセットなどの情報がある．本研究では，ユーザの熟練度として，User エンティティ中の *Performance Tier* という属性を用いた[*2]．

3.1 Kaggle ランキングシステム

Kaggle には，ユーザが実績に応じたメダルや階級を獲得するシステムがある[*3]．*Performance Tier* 属性には 6 種類の階級があり，低い順に “Novice”, “Contributor”, “Expert”, “Master”, “Grand Master”, “Staff” が定義されている．それぞれの階級に到達する目安として，以下のように記述されている[*4]．

[*2] https://zenodo.org/record/4468523#.Ye09I1hBzmF
[*3] https://www.kaggle.com/progression (2021 年 11 月 10 日閲覧．)
[*4] https://www.kaggle.com/getting-started/44919 (2022 年 1 月 3 日 閲覧．)

Novice：Kaggle に登録した際に自動で割り当てられる階級．

Contributor：プロフィールを完成させたり，Notebook を投稿したりすることで昇格できる階級．

Expert：いくつかの銅メダルを獲得することで昇格できる階級．

Master：銀メダルや金メダルなどを獲得することで昇格できる階級．階級が上位のユーザのみが挑戦できるコンペティションに挑戦できるようになる．

Grand Master：複数の金メダルを獲得，コミュニティに活発に参加することで昇格できる階級．Kaggle 内で最も高い階級である．

Staff：Kaggle を運営する職員であることを示す階級．Kaggle の職員はチュートリアルのような Notebook を投稿することがある．

Kaggle には，Competition, Dataset, Notebook, Discussion の 4 種類の分野で上記の階級が適用されるが，KGTorrent 中の Performance Tier 属性には，一番高い階級のみが記録されている．例えば，あるユーザが Competition で Expert， Dataset で Master， Notebook で Expert, Discussion で Grand Master の階級である場合，KGTorrent では，Grand Master として記録される．Expert 以上の階級に昇格することが困難なシステム設計であることから，本研究では KGTorrent 中の *Performance Tier* 属性を経験度として利用する．

3.2 前処理

RQ1，RQ2 に共通して，以下の前処理を実施した．

(1) **Notebook とメタデータの紐付け．**
Notebook のファイル名は，Users エンティティに含まれる *UserName* 属性と，Kernels エンティティに含まれる *CurrentUrlSlug* 属性を用いて，“$UserName_$CurrentUrlSlug” という形式で与えられている [12]．したがって，Users エンティティの *Id* 属性と Kernels エンティティの *AuthorUserId* 属性で結合し，Notebook の名前と *Performance Tier* 属性を紐付けた．KGTorrent を作成した論文で述べられている通り，ファイル名の同定ができなかった一部の Notebook に関してはこの工程でメタデータの紐付けに失敗した，これらの Notebook は本研究の分析の対象外とした．

(2) **Notebook の抽出．**
プログラムの品質が低いと考えられる Notebook を分析の対象から外すため，*Performance Tier* 属性 と Kernels エンティティに含まれる *TotalVotes* 属性を利用した．3.1 節で述べた通り，ユーザには熟練度によって階級が付与される．Novice は Kaggle にユーザ登録した際に機械的に付与される階級であること，Staff は他の階級と比べて特殊な位置付けであることから，本研究では Novice と Staff が作成した Notebook は分析の対象外とした．また，本研究ではデータ分析の能力に関わりなく昇格できる Contributor であるユーザを非熟練者，メダルを獲得しなければ昇格できない Expert 以上の階級を持つユーザを熟練者と定義する．Kaggle には，ユーザが Notebook を評価する Vote という機能がある．本研究では，Notebook に付与された Vote の総数を Notebook の品質を示す指標とみなし，*TotalVotes* が 1 以上の Notebook のみを抽出した．*TotalVotes* の分布を確認したところ，半数の Notebook はこの数値が 0 であったため，閾値に 1 を採用した．

(3) **ソースコードの抽出．**
この工程では，ソースコードの特徴を分析するため，Notebook 中のソースコードが記述されているセルのみを抽出した．つまり，本研究では Markdown や Text, グラフが出力されているセルは分析の対象外としている．Notebook は JSON 形式で表現されており，ファイル内の各セルに “cell_type” という属性が存在する．この属性は “code” や “markdown” といったセルの型を保持している．そこで，“cell_type” が “code” であるセルを取得し，セル内に含まれている “source” 属性からソースコードを順に取得した，

(4) **マジックコマンドのコメントアウト．**
Notebook 中のソースコードには，マジックコマンドと呼ばれる命令文が記述されることがある．マジックコマンドは IPython という Python を対話的に実行するために拡張されたシェルで利用することのできるコマンドである．マジックコマンドは “!” や “%” から始まる文であり，標準の Python を対象とした静的解析を行う場合，文法が定義されておらず構文エラーが出力される場合がある．また，マジックコマンドはライブラリのインストールや時間計測，出力の仕方などの設定に用いられるが，分析処理そのものに影響をおよぼすコマンドは筆者の知る限りないため，本研究ではマジックコマンドは分析対象外とした．したがって，静的解析を行う際に構文エラーの発生を防ぐためにコメントアウトした．

以上の前処理を行い，17,239 人のユーザが作成した 60,003 件の Notebook を用意した．表 1 にデータセットの概要を示す．なお，LOC は Line of Code の略であり，ソースコードの行数を計測した数値である．本研究で示す

表 1　データセット中の熟練度ごとのユーザ数，Notebook 数，セル数

経験度	ユーザ数	Notebook 数	セル数	LOC の中央値 Notebook	LOC の中央値 セル
Contributor	14,403	36,998	991,065	112	3
Expert	2,278	16,543	418,202	117	3
Master	440	4,396	90,674	119	3
Grand Master	118	2,066	41,410	113	4
合計	17,239	60,003	1,541,351	-	-

LOC は，ソースコードが記述されたセルのコメントおよびソースコードの行数を計測しており，空行は除外した．

4. RQ1. 熟練度によるコードクローンの特徴分析

熟練度が高い人ほどソースコードの再利用を適切に行っているという仮説を検証するために，熟練度でコードクローンの割合に差があるかを分析する．熟練者のコードクローンの割合が低くなる傾向にある場合，データ分析に関する能力だけでなく，熟練者の方が保守性の高いソースコードを記述できる能力も備わっている傾向にあると考えられる．

4.1 実験方法

コードクローンの検出をセル単位で行った．Jupyter Notebook を対象とする場合，ユーザが独自に定義した関数が存在する Notebook はわずか 50% 程度で，クラスを定義した Notebook は 20% であると報告されている [5]．従来のコードクローンの研究では，メソッド単位でコードクローンを検出する場合があるが，Jupyter Notebook を分析対象とする場合，関数が定義されていない場合があるため，処理の単位を判断することが難しい．さらに，Jupyter Notebook を対象としたコードクローンの研究ではセル単位でコードクローン検出を行っている [6] ことから，本研究も同様にセル単位での検出を行った．ただし，本研究では Notebook の保守性を保ちつつ，コードを再利用する方法について分析するため，特にファイル内のクローンのみに着目した．したがって，本研究ではコードクローンのコピー元に関する分析はしていない．

本研究では CodeHash[11] というツールを用いて Notebook ごとのセルの組み合わせで類似度を計算し，コードクローンを検出した．なお，コードクローンを検出するために識別子の違いを無視して抽出されたトークンを 3 語単位 ($trigram$) の集合の共通要素が占める割合を算出する標準化 Jaccard 係数を類似度として採用した．式 (1) に，標準化 Jaccard 係数による類似度の定義を示す．

$$sim(c_1, c_2) = \frac{|trigrams(c_1) \cap trigrams(c_2)|}{|trigrams(c_1) \cup trigrams(c_2)|} \quad (1)$$

Svajlenko ら [14] によると，類似度を用いたコードクローン検出の場合，類似度が 0.7 以上のコードの組み合わせは強いコードクローンであると報告されている．したがって，コードクローンを検出するための類似度の閾値を 0.7 に設定した．

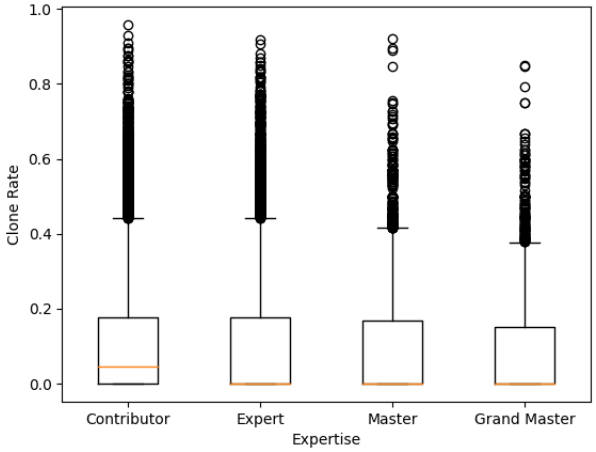

図 2　熟練度ごとの CloneRate の分布

Notebook のセル数の差を無視するため，Notebook 内のセルの総数に対するコードクローンとして検出されたセル数の割合を式 (2) にしたがって計算した．

$$CloneRate(C) = \frac{|Clones(C)|}{|C|} \quad (2)$$

式 (2) はソースコードの品質を評価する Code Clone Coverage を変形したものである [15]．ただし，元の定義では式 (2) 中の C は LOC として定義されていたが，本研究ではセル単位でコードクローンの検出を行ったため，Notebook 中のセル数を C として計算した．

検出したコードクローン内に含まれるライブラリを検出するために，Python に特化した静的解析ツールである PyCG[16] を用いて関数呼び出しを収集した．PyCG は呼び出された関数を含むライブラリを追跡できるため，ルートのライブラリを取得して集計した．

4.2 実験結果

図 2 は Notebook 内のセルを総当たりでコードクローン検出した際の CloneRate の分布である．中央値は Contributor が 4.5%，その他の階級は 0.0% である．この結果から，Contributor のユーザは他の階級のユーザよりもコー

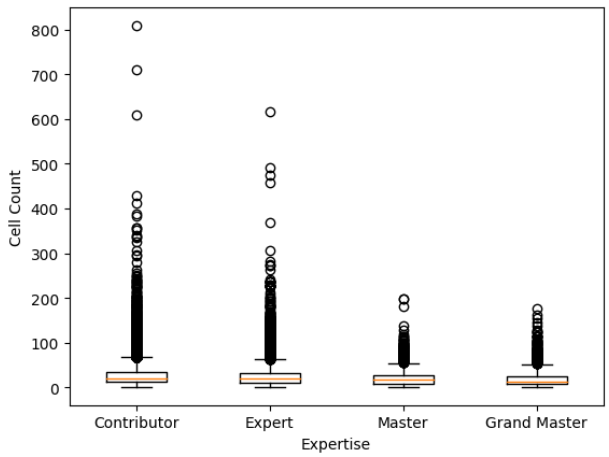

図 3 熟練度ごとの Notebook 中に含まれるセル数の分布

表 2 コードクローン内で出現したライブラリ

順位	ライブラリ名	頻度
1	matplotlib	163,327
2	builtin	162,178
3	sklearn	60,552
4	seaborn	43,525
5	numpy	40,187
6	pandas	33,251
7	plotly	20,014
8	keras	14,068
9	tensorflow	8,919
10	torch	4,129
	ライブラリ種類数	448
	コードクローン数	382,026

ドクローンを行う習慣があると考えられる．有意に差があるかどうかを検証するために，有意水準を 0.05 に設定した Kruskal-Wallis 検定を行った．ノンパラメトリックな検定を採用した理由は，Kolmogorov–Smirnov 検定で正規分布であるかどうかを検証したところ，図 2 に示したデータは正規分布に従っているとは言えないという結論が出たためである．この検定の帰無仮説は次の通りである．

H_{01}: 熟練度間で *Notebook* あたりの *CloneRate* の分布には差がない．

検定の結果，熟練度の間でコードクローンの割合には統計的に有意な差が見られた（$p < 0.05$）．

次に，どの熟練度の組み合わせが有意に異なるかを明らかにするために，多重比較を行った．本研究では 4 種類の熟練度に対し，総当たりで比較を行うため，12 回統計検定を適用することになる．複数回統計検定を適用すると有意水準以上の確率で棄却されやすくなるため，多重比較で適用する有意水準は Bonferroni 補正で 0.0041 に補正した．そして，Mann-Whitney の U 検定をすべての階級で総当たりに適用した．

H_{02}: 各熟練度で *Notebook* あたりの *CloneRate* の分布は差がない．

結果，Contributor と Expert を除く全ての組み合わせ（Contributor と Master,・・・，Master と Grand Master）で，統計的に有意な差が見られた ($p < 0.001$)．この結果から，Master, Grand Master は Expert, Contributor に比べて比べてコードクローンの割愛が有意に低いことが明らかになった．

図 3 に Notebook 中のソースコードが記述されたセル数の分布を示す．中央値は Contributor から順に 20, 19, 17, 14 である．この結果から，経験度が高い人ほど少ないセルを用いてソースコードを記述する傾向にあると考えられる．統計的に差があるのかを検証するために，下記の帰無仮説を設定し，Mann-Whitney の U 検定を行った．なお，前述と同様に Kolmogorov–Smirnov 検定を行ったが，正規分布とは言えないという結論が出たため，ノンパラメトリック検定を採用した．

H_{03}: すべての熟練度で *Notebook* あたりのセル数の分布は等しい．

結果，すべての組み合わせにおいて $p < 0.01$ となり，有意な差が見られた．以上の結果より，熟練者ほどコードクローンの割合が低い傾向にある．また，熟練者は長いコードを少ないセルに記述する傾向があることが明らかになった．

表 4.2 にコードクローン内で呼び出されたライブラリの出現頻度が高い 10 件を示す．なお，熟練度によって利用するライブラリに違いがあるかを調べたところ，大きな差異は見られなかったため，全データで集計した結果を提示する．Matplotlib や Plotly のようなグラフを描画するライブラリや，print 関数や sum 関数のように Python でデフォルトで提供されている組み込み関数がコードクローン中の約 4 割強で使用されていることがわかった．また，機械学習フレームワークとして著名な scikit-learn や keras, tensorflow, torch などのライブラリも利用頻度が高いことが明らかになった．

5. RQ2. 熟練度による自作関数の特徴分析

RQ1 の結果から，熟練度はコードクローンの割合が非熟練者よりも低い傾向にあることが明らかになった．そのため，コードクローンを抑えるために，熟練者は類似するソースコードを必要とする場合，自作関数を定義する傾向にあるのではないかと考えた．熟練度が自作関数を定義するの習慣に影響するのかどうかを明らかにするために，自作関数の定義数を分析する．

5.1 実験方法

熟練度ごとに自作関数が定義された頻度を集計するために，パーサジェネレータである ANTLR（ANother Tool for Language Recognition）[17] を使用した．ANTLR は

表 3　自作関数の定義が含まれる Notebook の割合

熟練度	自作関数の有無による Notebook 数 含まれる	含まれない	割合
Contributor	19,439	17,559	0.525
Expert	9,560	6,983	0.578
Master	2,859	1,537	0.650
Grand Master	1,264	802	0.612
Total	33,122	26,881	0.552

プログラミング言語の文法が定義されたファイルを読み込み，構文解析器を生成するツールである．そこで，本研究では ANTLR が公式に提供している Python3 の文法ファイル[*5] を用いて Python3 用の構文解析器を生成した．そして，関数を定義する構文を下記の条件で検出した．

- 文法ファイル中の *classdef* の子要素に含まれない *funcdef*

クラスとしての処理の再利用は，関数としての再利用方法とは異なるため本研究では除外した．ただし，クラス定義が Notebook で用いられることは少ない [5] ため，影響は小さいと考えている．

5.2　実験結果

表 3 に自作関数の定義が 1 回以上検出された Notebook の割合を示す．すべての熟練度で半数以上の Notebook では自作関数の定義が含まれることが分かる．さらに，熟練度が高くなるほど，その割合が増加する傾向にある．自作関数を含む Notebook の差が統計的に有意であるかを確認するために，下記の帰無仮説に基づき，Tukey の検定を行った．

H_{04}: 自作関数を含む Notebook の割合はすべての熟練度において等しい．

はじめに，すべての熟練度の組み合わせに対し，カイ 2 乗検定を行い，母比率の差が有意であるかどうかを検証した．この結果，すべての熟練度の組み合わせにおいて，有意な差が見られた ($p < 0.01$) ため，多重比較を行った．多重比較の結果においても，熟練度のすべての組み合わせにおいて有意な差が見られる結果となった．したがって，有意に熟練度が高くなるほど，自作関数が含まれる Notebook の割合が高くなる傾向にあると結論づける．

6.　考察

RQ1, RQ2 の結果から，コードクローンの割合や自作関数の数は熟練度によって異なることが統計的に示された．さらに，図 2 から熟練度が上がるほどコードクローンの割合は小さくなる傾向にあることや，表 3 から熟練度が上がるほど自作関数を定義する傾向にあることが読み取れる．これらの結果から熟練度が上がるほど，複数箇所にソースコードをコピー& ペーストするのではなく，関数を定義して呼び出すことで再利用する傾向にあることが示唆される．したがって，ソースコードの再利用の観点から非熟練者が熟練者に近づくには，類似するソースコードがある場合に自作関数を定義するような工夫が考えられる．

[*5] https://github.com/antlr/grammars-v4/blob/master/python/python3-py/Python3.g4 (2021 年 5 月 7 日 閲覧．)

```
# Calculation of Loss and Accuracy metrics
loss, accuracy = model.evaluate(X_test, Y_test)
print('loss: ', loss, ', accuracy: ', accuracy)
```

```
# Calculation of Loss and Accuracy metrics
loss, accuracy = model.evaluate(Xn_test, Yn_test)
print('loss: ', loss, ', accuracy: ', accuracy)
```

Listing 1: 検出したコードクローン例

一方で，本研究ではコードクローンや自作関数内で呼び出されたライブラリ・関数の集計はしたが，それらの処理が具体的に何をしているのかに関する分析は行っていない．コードクローンは一般的に保守性を下げると言われているが，中には無問題なコードクローンもあると報告されている [18]．例えば，ソースコード 1 のように，機械学習モデルの評価指標を print 関数で出力するだけのコードクローンを発見した．このような事例では関数抽出を行ったとしても保守性向上には繋がらない．このように，各コードクローンが与える保守性への影響の差や，熟練度によって発生するコードクローンに違いがあるのかを分析することは今後の課題である．

7.　妥当性への脅威

本節では，本研究に含まれる妥当性への脅威について述べる．本研究には内的妥当性，外的妥当性への脅威がある．

7.1　内的妥当性

本研究に含まれる内的妥当性への脅威は 2 点挙げられる．一点目はコードクローンをセル単位で検出した点である．本研究では標準化 Jaccard 係数で類似度を計算し，セル単位でコードクローン検出を行った．したがって，コードクローンを判定するための指標やソースコードの単位を変更した場合，本研究とは異なる結果が報告される可能性がある．

二点目は PyCG を用いてライブラリ関数呼び出し群を収集した点である．本研究で示した結果は PyCG で取得できるライブラリ・関数呼び出しのみを分析しているため，データの取得に漏れがある可能性がある．しかし，PyCG は Python を対象とした静的解析ツールの中で最も適合率・再現率が高いツールであるため，現段階では最善の手段である．したがって，ライブラリ名以外の識別子から呼び出された関数も検出できるような静的解析技術が発展した

場合，本研究の分析結果と異なる結果が示される可能性がある．

7.2 外的妥当性

本研究では Kaggle 上で公開されている Notebook ファイルを分析しているため，Kaggle 上の Python で記述されたプログラムには本研究の分析結果を一般化することができると考えられるが，Kaggle 外のデータ分析プログラムには本研究の調査結果を汎化することはできない．本研究の調査を一般化するためには，Kaggle 外の Notebook に対しても同様の分析を行う必要がある．本研究では，データ分析の熟練度を測る指標として Kaggle ランキングシステムの階級を利用したが，Kaggle 外の Notebook に対して，同様の指標を用いることはできない．しかし，Kaggle は最も有名なデータ分析コンペティションであり，かつアクティブでないユーザのランキングを低下させる仕組みが存在する．したがって，データ分析に関する熟練度を測る指標として現時点でこの階級を利用することは妥当であると考える．

8. おわりに

本研究では，Kaggle 上での熟練度がデータ分析プログラム中のソースコード再利用方法と関連があるのかを分析し，熟練度が上がるほど，コードクローンの割合は低くなり，自作関数を定義する傾向にあることが明らかになった．

今後の展望として，どのようにソースコードの再利用方法を習得したのかを作者に質問票による調査を行うようなソースコードの再利用に関した更なる分析や，関数やコードクローンでどのような処理が実装されているのかを人手で確認するような分析が挙げられる．また，本研究ではファイル内クローンのみを対象としたが，ファイル間クローンを分析することで，非熟練者が熟練者のコードを再利用しているかどうかを明らかにできる可能性がある．これらの研究によって，データ分析における非熟練者がソースコードの保守性を考慮して実装できるように支援・教育するような場面で有用な知見になりうると考えられる．

謝辞 本研究は JSPS 科研費 No.JP20H05706, JP18H04094 の支援を受けたものである．

参考文献

[1] Kaggle Inc. Kaggle. `https://www.kaggle.com/`, January 2022. (2021 年 6 月 10 日閲覧．).
[2] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.
[3] Mike Loukides. Where programming, ops, ai, and the cloud are headed in 2021. `https://www.oreilly.com/radar/where-programming-ops-ai-and-the-cloud-are-headed-in-2021/`, January 2021. (Accessed on 2021/6/10).
[4] 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, p. 1–12. Association for Computing Machinery, 2020.
[5] Pimentel Joao Felipe, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. A large-scale study about quality and reproducibility of jupyter notebooks. In *Proceedings of MSR 2019*, Vol. 2019-May, pp. 507–517. IEEE, 2019.
[6] Andreas P. Koenzen, Neil A. Ernst, and Margaret-Anne D. Storey. Code duplication and reuse in jupyter notebooks. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 1–9, 2020.
[7] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.
[8] Chanchal Kumar Roy and James R Cordy. A survey on software clone detection research. Queen's School of Computing TR, Vol. 541, No. 115, pp. 64–68, 2007.
[9] Malin Källén and Tobias Wrigstad. Jupyter notebooks on github: Characteristics and code clones. *The Art, Science, and Engineering of Programming*, Vol. 5, , 2021.
[10] 肥後芳樹, 吉田則裕. コードクローンを対象としたリファクタリング. コンピュータ ソフトウェア, Vol. 28, No. 4, pp. 4_43–4_56, 2011.
[11] Takashi Ishio, Yusuke Sakaguchi, Kaoru Ito, and Katsuro Inoue. Source file set search for clone-and-own reuse analysis. In *Proceedings of MSR 2017*, pp. 257–268. IEEE, 2017.
[12] Luigi Quaranta, Fabio Calefato, and Filippo Lanubile. KGTorrent: A Dataset of Python Jupyter Notebooks from Kaggle. In *Proceedings of MSR 2021*. MSR2021, 2021.
[13] Inc GitHub. Github: Where the world builds software. `https://github.com/`, January 2022. (2022 年 1 月 20 日閲覧．).
[14] Jeffrey Svajlenko and Chanchal K. Roy. Bigcloneeval: A clone detection tool evaluation framework with bigclonebench. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 596–600, 2016.
[15] A. Monden, D. Nakae, T. Kamiya, S. Sato, and K. Matsumoto. Software quality analysis by code clones in industrial legacy software. In *Proceedings Eighth IEEE Symposium on Software Metrics*, pp. 87–94, 2002.
[16] Vitalis Salis, Thodoris Sotiropoulos, Panos Louridas, Diomidis Spinellis, and Dimitris Mitropoulos. PyCG: Practical Call Graph Generation in Python. In *Proceedings of the 43rd ICSE*. ICSE2021.
[17] Terence Parr. *The Definitive ANTLR Reference*. The Pragmatic Bookshelf, 2007.
[18] Cory J Kapser and Michael W Godfrey. "cloning considered harmful" considered harmful: patterns of cloning in software. *Empirical Software Engineering*, Vol. 13, No. 6, pp. 645–692, 2008.