香川大学経済論叢
第90巻 第1号 2017年6月　173－207

研究ノート

データ分析へのExcel VBAの活用(1)

大野拓行

はじめに

Excel VBAはMicrosoft社のExcelに標準搭載されているプログラム言語VBA（Visual Basic for Applications）である。Excelはビジネスをはじめ，日常のさまざまな場面で，広範に利用されている。また，Excelの作業を効率化するために，マクロ（VBAで記述されたプログラム）やExcel VBA関連の書籍は多数出版されており，ネット上にも膨大な情報が蓄積されている。データ分析に限っても，単純な集計作業から統計分析にいたるまでExcelは活用されており，データ分析を目的とした多くのマクロも蓄積されている。

筆者自身，Excel VBAに精髄しているわけではないが，情報を整理しながら，Excel VBAの特質を活かしたデータ分析について考察し，それを教育・研究に活かしていきたい。このように考えるのは，大学教育，特に文系の大学教育，においては，情報リテラシーとしてExcelの知識があるにもかかわらず，一歩進んだExcel VBAについては，十分に活用されていないと思われることも一因になっている。

「データ分析」への活用という観点から，Excel VBAの特徴をみていくわけであるが，先ず，本稿では，オブジェクト，プロパティ，メソッドから成るVBAの構文の特徴をみた上で，Range型変数，バリアント型変数の特性を活かした，ワークシート上のデータの操作について考えていく[(1)]。

（1） 本稿中のマクロはExcel 2010，2013，2016の各バージョンで動作確認を行っている。

Ⅰ．Excel VBA の概要

1．マクロの記録・実行，マクロの構成，VBE

VBA の特徴として，先ず，Visual Basic for Applications の名称が示すように Microsoft 社のアプリケーション（Excel，Word，Access，PowerPoint など）に結びつけられたプログラム言語であることが挙げられる。VBA によるプログラミングにおいて，始めからコードを記述していくのではなく，各アプリケーションのメニューにある「マクロの記録」機能を利用して VBA のコードを生成し，生成されたコードを VBE（Visual Basic Editor）上で修正して目的とするマクロを完成させていくことが広範に行われていることからもそのことが分かる。

マクロの記録・実行，マクロの構成，VBE などに関しては，標準的な解説書で説明されているので説明は省略するが，付表の形で簡単に要約しておくことは，Excel VBA のデータ分析への活用を考察する際にも有益であろうと考え，付表 1 を作成した。(2)

また，前述したように，Excel VBA は解説書，関連ウェブサイトに膨大な情報があり，時にはどの情報が正しいのか迷うケースもある。そのような時には MSDN（Microsoft Developer Network）を利用するのも一つの方法である。(3)

2．VBA のオブジェクト式(4)について

VBA(5)がプログラミング初心者にとって学びやすい言語である理由は，前述した，Microsoft 社のアプリケーションに結びつけられたプログラム言語であることと共に，その文法構造が比較的簡単なことによると考えられる。

（2） 大村（2011），門脇（2016）などの内容を参考にして，筆者がまとめたものである。解説書では，プロシージャ，モジュール，プロジェクトなどにも言及しているが，それらについては必要に応じて考察していきたい。また，VBE 自体，高機能なソフトウェアで，その機能を一覧にすることは難しくはあるが，本稿では筆者が使用している機能を中心にまとめた。ただし，「自動構文チェック」は，修正の途中でも頻繁にエラー表示をするので使用しない方がよいのかも知れない。

（3） ただし，MSDN 自体，膨大な情報を有しており，複雑な階層構造になっている。検索エンジンにおいて，知りたい事項を“Excel VBA”，“MSDN”と共にキーワードにして検索することを推奨しておきたい。例）“Range プロパティ Excel VBA MSDN”

VBAにはループ処理や条件分岐のほかに，オブジェクト式と呼ばれるマクロの中心となる一群の構文があり，それがVBAの大きな特徴となっている。

オブジェクト式構文は3つに分類できる。[6]

(1)　プロパティの値を設定する：[Object].[Property]＝[Property]の値

(2)　プロパティの値を取得する：[Object].[Property]

(3)　オブジェクトを操作する：[Object].[Method]

オブジェクト式構文は最初に命令の対象となるオブジェクトObject（Excelで言えば，セルやワークシートなど）を記述し，その後に，プロパティProperty，メソッドMethodをピリオド（.）で繋いで記述する。以下，上記の3つの構文のそれぞれについてみていく。

(1)　プロパティの値を設定する：[Object].[Property]＝[Property]の値

例①　Range("A1").Value＝3

この例は，A1セルの値（プロパティ）を3に設定する構文である。

プロパティはオブジェクトの属性であり，セルで言えば，セルの値，セルの色などである。この例では，Range("A1")がオブジェクトで，Valueがそのプロパティのように見える。[7]しかし，正確には，Rangeもプロパティであり，対象となるオブジェクトは「オブジェクトへの暗黙の参照」[8]により省略されている。上位オブジェクトを含めて記述すれば，

Application.ActiveWorkbook.ActiveSheet.Range("A1").Value＝3　　(a)

（4）「オブジェクト式」という言葉は，解説書ではあまり見かけない用語であるが，Excel VBA関連のウェブサイト，例えば，Office TANAKA（http://officetanaka.net/），エクセルの神髄（http://excel-ubara.com/），インストラクターのネタ帳（http://www.relief.jp/itnote/）などで使用されており，VBAの特徴を理解するのに有用なのでここでも使用することにする。

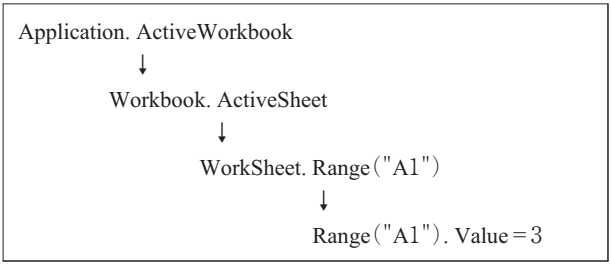
（5）「VBA」と「Excel VBA」の用語の使い分けも難しい。本稿ではVBAに一般的な事項については「VBA」の用語を使い，Excel VBAにしか存在しない事項，例えばWorksheetオブジェクトなど，の記述の際は「Excel VBA」の用語を使用したいと思う。

（6）　本稿では，単純化のため3つにまとめたが，その変形については，3つの基本構文をみていく中で説明していくことにしたい。

（7）　Range("A1")をオブジェクトと考えて進んで，後に，再考するのも一つの方法である。

となるのだが，「オブジェクトへの暗黙の参照」により，下線部分が省略されているのである。⑴の構文においては，オブジェクトは［Object］としか記述されていないが，階層構造になっていることを理解することが重要である。Application は Excel そのものを示すオブジェクトであり，⑷の構文は，Excel において，そのプロパティの１つである ActiveWorkbook プロパティが Workbook オブジェクトを返し，そのプロパティの１つである ActiveSheet プロパティが WorkSheet オブジェクトを返し，そのプロパティの１つである Range プロパティが Range オブジェクトを返し，そのプロパティの１つである Value の値を３と設定していることになる（図１）。

図１　オブジェクトの階層構造のイメージ



プロパティには戻り値があり，戻り値がオブジェクトの場合には，構文をそこで終えることはできない点に留意しておきたい。

⑵　プロパティの値を取得する：［Object］.［Property］

例②　x＝Range("A1"). Value

（８）「Application オブジェクトは，Excel の最上位オブジェクトです。Application オブジェクトを使用すると，アプリケーションレベルのプロパティを設定し，アプリケーションレベルのメソッドを実行することができます。Application オブジェクトのプロパティおよびメソッドを操作する場合，Application オブジェクトは既定で使用可能になっています。これはオブジェクトへの暗黙の参照として知られています。」MSDN から引用（一部改変）

この例は，A1セルの値を変数xに代入する構文である。

この例からもわかるように，VBAではプロパティから値を取得した場合は，必ずその値を何かに使わなければならない[(9)]。この例のように変数に代入して利用する，または関数などの引数として利用するのが一般的である。

以上，2つの構文から，プロパティを使用する際には次のことに留意する必要があると考えられる。

1）Excelの最上位オブジェクトであるApplicationは別として，それより下位のオブジェクトは上位オブジェクトの（オブジェクトを返す）プロパティと考えられること。

2）オブジェクト式の最後（例①では左辺の最後，例②では右辺の最後）はプロパティがくるが，オブジェクトを返すプロパティはオブジェクト式の最後には使用できないこと。

3）プロパティの値を取得した場合には何かに使う必要があること。

4）プロパティには戻り値があり，戻り値として何がくるか重要であること。

(3)　オブジェクトを操作する：[Object].[Method]

例③-1　Worksheets.Add

この例は新規シートを挿入する構文である[(10)]。

メソッドはオブジェクトに対する操作である。前述したように，プロパティには戻り値があり，プロパティの値を取得する場合には必ずその値を何かに使わなければならないが，メソッドには，戻り値がないメソッド[(11)]もあり，また，戻り値があっても，

（9）　Range("A1").Valueのみの構文は，コンパイルエラー「プロパティの使い方が不正です。」となる。

(10)　同じタイプのオブジェクトの集合はコレクション（コレクションオブジェクト）と呼ばれる。1つのブックに存在するすべてのワークシートはWorksheetsコレクションで表すことができる。コレクションもオブジェクトであるので，プロパティやメソッドを持つ。また，コレクションに属する個々のオブジェクトはインデックス番号あるいは名前で参照できる。例）Worksheets(1)，Worksheets("Sheet1")など。

(11)　戻り値のないメソッドの例として，ワークシートを削除する構文，Worksheets(1).Deleteやワークシートを選択する構文，Worksheets(1).Selectなどが挙げられる。

その戻り値を使用しないことが許される。例③-1 における Add メソッドは Worksheet オブジェクトを返すが，この例では，それを無視している。また，メソッドの中には，引数を使ってオブジェクトに対する操作を細かく指示できるものもある。

例③-2）WorkSheets. Add After：＝Worksheets(2)

例③-2 は 2 番目のシートの後ろに新規シートを挿入する構文である。例③-2 は Add メソッドの戻り値を利用しない場合の記述例であるが，メソッドの戻り値を利用する場合は，引数を括弧で括る必要がある。例えば，例③-2 で挿入した新規シートへの参照を Worksheet オブジェト NWs に代入したい場合には，

Set NWs＝Worksheets. Add(After：＝Worksheets(2))

と引数を括弧で括らなければコンパイルエラーとなる。(12)

3．Excel VBA の主なプロパティ，メソッド

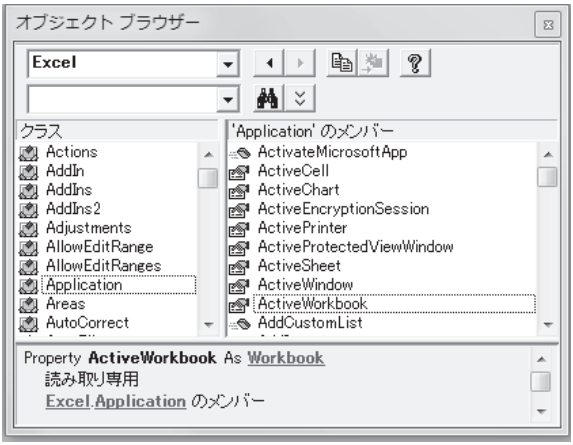
Excel VBA におけるプロパティ，メソッドは VBE 画面からオブジェクトブラウザーを起動（F2 キーを押下）して調べることが可能である。図 2 は Application オブジェクト（Excel）の ActiveWorkbook プロパティを調べたものである。ブラウザーの下欄に

Property ActiveWorkbook As Workbook
　読み取り専用

とあるので，ActiveWorkbook プロパティは読み取り専用（取得専用）であり，Workbook オブジェクトを返すことがわかる。Excel VBA には多数のプロパティ，メソッドが存在しており，どの解説書においても，その著者が重要だと思うプロパティ，メソッドについて解説がされている。(13) 本稿では，次の観点から Excel VBA の主なプロパティ，メソッドをまとめていく。(14)

(12) Set ステートメントについては後述する。ここでは，Add（…）の理解が重要である。戻り値を利用する場合には引数を括弧で括る必要があるのは，よく利用される VBA 関数である MsgBox 関数などでも同様である。

図２　オブジェクトブラウザーの例示



1）Excel でのデータ分析の中心はセル操作なので，プロパティ，メソッドをまとめていく際にも，Application（Excel 本体）－Workbook－Worksheet－Range（セル領域）を軸に考える。

2）プロパティについては，戻り値が何なのか，また取得専用かどうか，を調べる。メソッドについては，戻り値があるかを調べる。

3）入門的な解説書で挙げられているプロパティ，メソッドはできるだけ含める。(15)

まず，Application オブジェクトの主なプロパティ，メソッドを付表２にまとめた。(16)

・前述した「オブジェクトへの暗黙の参照」により，付表２に記載されているプロパティでは Application オブジェクトは省略できる。

---

(13)　基本的な構文を習得した後は，「マクロの記録」機能，オブジェクトブラウザー，MSDN（Microsoft Developer Network）や VBA 関連のウェブサイトから必要な情報を得られるのも VBA の強みである。

(14)　それぞれのプロパティ，メソッドの使用方法については，解説書や Excel VBA 関連のウェブサイトに詳しいので省略する。MSDN の活用を推奨する。

(15)　ここで参考にしたのは大村（2011），門脇（2016）などである。

(16)　なお，付表に記載したプロパティ，メソッドの説明は MSDN からの引用であるが筆者が一部改変している。

・ただし，InputBox メソッドは Application オブジェクトを明記して，Application. InputBox とする必要がある。(17)

・WorksheetFunction プロパティは Excel VBA で Excel のワークシート関数を使用するためのものである。(18)

Range プロパティ，Cells プロパティはセル操作で中心的な役割を担うものであり，InputBox メソッドは，データ分析を行う際，ワークシート上のデータの取得に欠かせないものであるので，次節で詳しくみていく。(19)

付表２における戻り値の欄を見ると，この中で最も上位のオブジェクトは Workbooks コレクション（コレクションオブジェクト）であるが，Workbooks コレクション，Workbook オブジェクトの使用頻度はさし当たり低いと判断して省略し，次は Worksheets コレクションの主なプロパティ，メソッドをみてみた（付表３）。付表３で留意しておきたいことは，新規ワークシート挿入のための Add メソッドが Worksheet オブジェクトのプロパティではなく，Worksheets コレクションのプロパティである点である。

Worksheet オブジェクトの主なプロパティ，メソッド（付表４）に挙げたプロパティ，メソッドは Excel の基本操作に対応しているもので，追加的な説明は必要ないであろう。

セル操作で中心的な役割を担う Range オブジェクトの主なプロパティ，メソッドをまとめたものが付表５である。表に挙げた，プロパティ，メソッドの多くは MSDN の説明文でその働きは分かるし，Excel VBA 解説書の多くが Range オブジェクトの

(17) Application オブジェクトを省略すると，VBA 関数の InputBox 関数が呼び出される。InputBox メソッドと InputBox 関数の差異についても後述（第Ⅱ節）する。

(18) 例えば，ワークシート関数 Sum を使用してセル範囲 A1：A100 の合計を B1 セルに代入するコードは Range("B1"). Value = WorksheetFunction. Sum(Range("A1：A100")) となる。

(19) 付表２において，ActiveSheet プロパティの戻り値が Object となっているのは，Activesheet が，ワークシート（Worksheet オブジェクト）のケースもあり，グラフシート（Chart オブジェクト）のケースもあることを反映している。Chart オブジェクトの集まりが Charts コレクションであり，Charts コレクションと Worksheets コレクションの集まりが Sheets コレクションとなっている。Sheets コレクションはブックにあるすべてのシートの集まりである。また，InputBox メソッドの戻り値の Variant については次項「VBA における主な変数型」を参照。

利用法を中心に記述しているので，本稿では説明を省略する。ただ，データ分析において有用な Offset，Resize の 2 つのプロパティについては，次節でみることにしたい。

この項の最後に，データ分析には直接的には関係しないが，Excel での作業でよく利用される，罫線についてまとめたのが付表 6 である。ここでは，ワークシート上のセル範囲に罫線を設定する時，マクロ記録で生成されるコードは膨大なものになるが，Borders コレクションを使用すれば一行で記述可能となることに留意しておきたい。[20]

（例）Range("A2:C4"). Borders. LineStyle = xlContinuous
セル範囲 A2:C4 に罫線が引かれる。

なお，セル範囲に外枠罫線を引くには，付表 4 にある BorderAround メソッドを利用する。

（例）Range（"A2:C4"). BorderAround Weight: = xlMedium
セル範囲 A2:C4 に外枠太罫線が引かれる。

4．VBA における主な変数型

表 1 は MSDN（Microsoft Developer Network）の「データ型の概要」から，データ分析において，使用頻度が高いと思われる変数型をまとめたものである。[21]

マクロ作成においては，変数名のスペルミスを防ぐ意味もあり，Dim ステートメントにより，マクロで使用する変数を明示的に宣言するのが一般的である。また，モジュールの先頭に Option Explicit を記述しておくと，Dim 文で宣言した変数以外の用語を使用するとエラーになるようにできる。[22]

---

(20) 田中（2016）を参照。
(21) 土屋（2016），大村（2011）の記述内容も参考にした。型宣言文字の使用例は Sample 2（図 4 ）を参照。
(22)「付表 1 マクロの記録・実行・保存，マクロの構成，VBE」を参照。

表１　VBA における主な変数型

データ型	指定方法	型宣言文字	範　　囲
バイト型	Byte		0 ～255
ブール型	Boolean		True または False
整数型	Integer	%	－32, 768～32, 767
長整数型	Long	&	－2, 147, 483, 648～2, 147, 483, 647
単精度浮動小数点型	Single	!	負の値の場合は－3. 402823Ｅ38～－1. 401298Ｅ－45, 正の値の場合は 1. 401298Ｅ－45～3. 402823Ｅ38
倍精度浮動小数点型	Double	#	負の値の場合は －1. 79769313486231Ｅ308～－4. 94065645841247Ｅ－324, 正の値の場合は 4. 94065645841247Ｅ－324～1. 79769313486232Ｅ308
通貨型	Currency	@	－922, 337, 203, 685, 477. 5808～922, 337, 203, 685, 477. 5807
日付型	Date		100 年 1 月 1 日～9999 年 12 月 31 日
文字列型	String	$	0 ～約 20 億
バリアント型	Variant		固定長の文字列型を除くすべてのデータ型
オブジェクト型	Object		任意の Object 参照

（構文）Dim 変数名 As データ型

（例）　Dim i As Integer　←変数 i を整数型として使用することを宣言[23]

VBA に特徴的な変数型はオブジェクト型とバリアント型である。

（オブジェクト型変数）

オブジェクト型変数（特に Range 型変数）はデータ分析において多用される変数

(23)　カンマで区切ることにより，一行で複数の変数を宣言すること可能であるが，注意が必要である。例えば，変数 i，変数 j を整数型として宣言したい場合

Dim i, j As Integer

としてもエラーは表示されないので，変数 i，変数 j とも整数型として宣言できているように見えるが，このステートメントでは変数 j のみ整数型となり，変数 i はバリアント型として宣言したことになっている。２変数ともに整数型としたい場合には

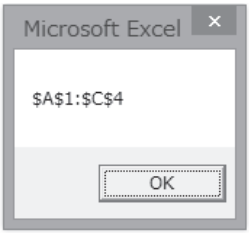
Dim i As Integer, j As Integer

と記述する必要がある。これは，Dim ステートメントの構文において［As データ型］の部分は省略可能で，データ型が省略された変数はバリアント型として宣言されたことになるからである。

型なので，節を改めてみていくことにし，ここでは，簡単に使用方法を示しておくことにする。マクロ Sample1 は，２行目で Range 型（オブジェクト型）変数 r の使用を明示的に宣言し，４行目でセル範囲 A1：C4 への参照を変数 r に代入し，５行目で変数 r が参照しているセル範囲のアドレスを表示させている。ここで，留意しておくことは次の２点である。

図３　Range 型変数の使用例

```
1  Sub Sample1()
2      Dim r As Range
3
4      Set r = Range("A1:C4")
5      MsgBox r.Address
6  End Sub
```



１）Range 型変数に入っているのは，セルの内容ではなく，セル範囲への参照であること。

２）オブジェクトへの参照をオブジェクト型変数に代入する際は Set ステートメントを使用すること。

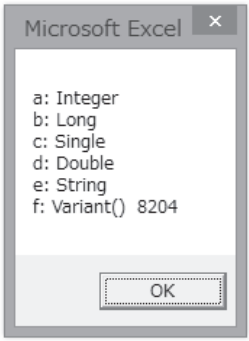
（バリアント型）

表１にあるようにバリアント型変数には，ほぼ全てのデータ型を格納できる[24]。ここで留意しておきたい点は，Office VBA 言語リファレンス（MSDN）にある「一般に，数値 Variant データは，元のデータ型で Variant に保持されます。たとえば，Integer を Variant に代入した場合，以降の操作では Variant は Integer として扱われます。」の記述内容である。バリアント型は便利な型であるからこそ，バリアント型変数を使用した場合の実質的な型が何になるかは重要である。

(24)　Office 2013 以降，ユーザー定義型もバリアント型変数に格納できる。「Office VBA 言語リファレンス」（MSDN）を参照。

図 4　バリアント型変数の特性

```
Sub Sample2()
    Dim a, b, c, d, e, f:   'Variant 型として使用
'数値型
    a = 1:  '整数型
    b = 1&:  '&は Long 型の型宣言文字
    c = 1!:  '!は Single 型の型宣言文字
    d = 1#:  '#は Double 型の型宣言文字
'文字型
    e = "Test"
'Range 型
    f = Range("A1:C4")
'結果
    MsgBox "a: " & TypeName(a) & vbCrLf & _
        "b: " & TypeName(b) & vbCrLf & _
        "c: " & TypeName(c) & vbCrLf & _
        "d: " & TypeName(d) & vbCrLf & _
        "e: " & TypeName(e) & vbCrLf & _
        "f: " & TypeName(f) & " " & VarType(f)
End Sub
```



このことを検証するために作成したマクロが Sample2 である。Sample2 では数値型データに加えて，文字型と Range 型のデータについても，バリアント型変数に代入した時の型の変化をみてみた。これにより次の 3 点がわかる。

1）数値型データについては，「Office VBA 言語リファレンス」の記述どおりに型が変化する。

2）文字型データについても，入力されたデータの型に変化する。

3）Range 型データについては，型は変化せず，バリアント型のままである。VarType 関数の値 8204 は Variant 型の配列（12＋8192）であることを示している。[25]

数値型データや文字型データに対する特性は，MSDN において「VarType 関数または TypeName 関数を使用して Variant 内のデータを処理する方法を決定できます。」と記述されているようにマクロ作成時に威力を発揮する。しかしながら，Sample2 でみるように Range 型変数については型が変化しないことに留意が必要である。[26]

(25)　Variant() の表記がこのことを表す。MSDN を参照。
(26)　後述する InputBox メソッドについての考察を参照。

Ⅱ．データ分析の手順から見るExcel VBAの特徴

Excelを用いたデータ分析ではワークシートにあるデータについて，何らかの操作を行い，その結果をワークシートに表示するケースが多い。データの操作については分析内容に依存するので次稿以降で考察することとし，本稿では，１.ワークシートにあるデータの取得，２.分析結果のワークシートへの表示，と３.分析の枠組み，の考察を通してExcel VBAの特徴をみていく。

１．ワークシートにあるデータの取得についての考察

ワークシート上のデータの位置（セル番地）が定まっているケースでは，直接Rangeプロパティ（あるいはCellsプロパティ）を利用すれば，データを取得できる。しかし，データの位置が定まっておらず，マクロ実行時に，ワークシート上のデータを指定（選択）したいケースの方がより一般的である。このようなケースではInputBoxメソッドが利用される。Excel VBAを活用したデータ分析のおいても，InputBoxメソッドがデータ取得の中心的な役割を担うので，ここで，InputBoxメソッドを通して，Excel VBAの特徴をみていくことにする。

InputBoxメソッドは，Application（Excel）オブジェクトのメソッドである[27]ことから，VBA関数であるInputBox関数の機能を，Excelにおける利用を考慮して，拡張したものと捉えることができる。拡張された機能は，以下の２点であるといえる。

１）InputBox関数の戻り値は文字型であるが，InputBoxメソッドでは引数Typeによって，戻り値の型を指定できる。特に，Type：＝8とするとセル参照が可能となる。

２）セル参照が可能になることにも関連するのだが，InputBoxメソッドではダイアログボックスを閉じることなく，セル領域の選択などのExcelの操作が可能となる。

(27)「付表２　Applicationオブジェクトの主なプロパテイ，メソッド」を参照。

Excel VBA リファレンス（MSDN）によれば，InputBox メソッドの書式は，

Application. InputBox（引数１，引数２，…，引数８）

である。引数は８つあり，最初の引数 Prompt は必須である。それ以降の引数は省略可能であるが，Title と Type は使用されるケースが多いので，本稿では，書式を

Application. InputBox（Prompt, Title, ..., Type）

と省略した形でみていくことにする。引数 Type に指定できる値は７つあるが，本稿では，そのなかでよく利用される３つを表に挙げた。

表２　InputBox メソッドの引数と引数 Type に指定できる値

引　数		データ型	説　　明
Prompt	必須	String	ダイアログボックスに表示するメッセージを指定する。
Title	省略可能	Variant	ダイアログボックスのタイトルを指定する。この引数を省略すると，既定値の“入力”がタイトルバーに表示される。
Type	省略可能	Variant	返されるデータの型を指定する。この引数を省略すると，ダイアログボックスは文字列（テキスト）を返す。

値	意味
1	数値
2	文字列
8	セル参照

（出所）Excel VBA リファレンス（MSDN）。一部改変

図５は引数 Type を８（セル参照）とした InputBox メソッドの利用例である。

Sample3 では，10 行目で InputBox メソッドが実行されると，入力を促すダイアログボックスが表示される。ここで，直接，ダイアログボックスの入力欄にセル範囲を入力するか，ワークシート上のセル範囲をマウスで選択して［OK］ボタンを押下すると，InputBox メソッドにより取得されたセル範囲への参照が変数 r に代入される。なお，引数 Type で入力された型以外の方のデータが入力されると，再入力を求められる。(28)

(28)　入力欄が空欄のまま［OK］ボタンを押下すると，Type：＝1，Type：＝８では再入力を求められるが，Type：＝２（文字列）では長さ０の文字列が取得される。

ダイアログボックスで指定するセル範囲はアクティブシートに限定されないが，ダイアログボックスが表示されている状態では新規シートの挿入はできないことに注意が必要である。

図 5　InputBox メソッドの使用例

```
 1  Sub Sample3()
 2      Dim r As Range
 3
 4      Dim myTitle As String, myMsg As String
 5      Dim c As Range
 6      Dim i As Integer
 7
 8      myTitle = "Test"
 9      myMsg = "セル範囲を入力してください。"
10      Set r = Application.InputBox(myMsg, myTitle, Type:=8)
11
12      i = 1
13      For Each c In r
14          c.Value = i
15          i = i + 1
16      Next c
17  End Sub
```

また，Range 型変数に代入するセル参照は連続したものである必要はない。Sample3 において，セル参照を A1：C3, E1：E2 としたケースの実行結果が図 6 である。

図 6　Sample3() の実行例

	A	B	C	D	E
1	1	2	3		10
2	4	5	6		11
3	7	8	9		
4					

変数 r にはセル範囲［A1:C3, E1:E2］への参照が入り，Sample3 の 12～16 行において，変数 r で参照される領域に，1 からの連続数値を代入している。この例に示したように，要素を示す index が不連続なオブジェクトに対してループ処理を行う場合，For Each ～ Next ステートメントが威力を発揮する。

（キャンセル処理について）

InputBox メソッドは，簡単な命令でマクロの実行時にユーザーが入力するデータを取得できる手段であるが，キャンセル（キャンセルボタン，あるいは［×］ボタンが押下された場合）の処理を考慮しておく必要がある。引数 Type の値によって処理の仕方が異なってくるので注意が必要である。

⑴ Type：＝1（数値），Type：＝2（文字列）の場合

InputBox メソッドの戻り値が Variant 型であること[29]と，Excel VBA リファレンス（MSDN）の記述，「ダイアログボックスには，［OK］ボタンと［キャンセル］ボタンが表示されます。［OK］をクリックすると，InputBox はダイアログボックスに入力された値を返します。［キャンセル］をクリックすると，InputBox は False を返します。」の２点から，次のようなコードが考えられる。

図７ キャンセル処理①

```
1  Sub Sample4()
2      Dim a As Variant
3                                          1 or 2
4      a = Application.InputBox("Test", Type:=  )
5      If TypeName(a) = "Boolean" Then Exit Sub
6      MsgBox TypeName(a)
7
8  End Sub
```

５行目で，取得された値の型が“Boolean”であれば，マクロを終了する[30]。また，前節の「４. VBA における主な変数型」でみたように，８行目で，Type：＝1（数値）のケースではバリアント変数 a の型が Double に，Type：＝2（文字列）のケースでは String に変化することが確認できる。

（29）「付表２ Application オブジェクトの主なプロパティ，メソッド」を参照。

（30） 条件判断の部分を a＝False としていないのは，Type：＝1 のケースでは，False＝0 なので，数値の入力と区別できない点を考慮したことによる。

(2)　Type：＝8（セル参照）の場合

セル参照のケースにおいても Sample4 の 4 行の引数を Type：＝8 とすれば，キャンセル処理は正常にできるのだが，正しく，セル範囲が入力された場合には，変数 a の型は Variant のままである[31]。また，Range 型変数にセル参照を代入するには，前節でみたように，Set ステートメントを使用する必要がある。

そこで，考えられるコードは以下のようなものである。

図 8　セル参照の取得（基本型）

```
1  Sub  Sample5()
2       Dim  a  As  Range
3
4       Set   a = Application. InputBox("Test", Type：＝8)
5       If  TypeName(a) = "Boolean"  Then  Exit  Sub
6       MsgBox   TypeName(a)
7
8  End   Sub
```

変数 a を Range 型として宣言し，4 行目で Input メソッドからの戻り値（セル参照）を変数 a に代入する。これで，セル範囲が入力された場合には，変数 a にセル参照が入るのであるが，ダイアログボックスでキャンセルが押下された場合には，実行時に，4 行目でエラーとなる。このことから，Type：＝8（セル参照）の場合には，正しくセル範囲が入力されたケースとキャンセルが選択されたケースの両方に対応するコードを考える必要があることがわかる。

【方法 1】キャンセルをエラートラップで処理する。

解説書や VBA 関連のウェブサイトで，よく紹介されている方法である[32]。図 9 はその例示である。

(31)　Sample2（図 4 ）を参照。
(32)　例えば，大村（2011）の 284～285 頁や Moug モーグ（http://www.moug.net/）の「マクロ実行中にセルを選択する（即効テクニック）」など。

図９　キャンセル処理②－１

```
1  Sub Sample6()
2      Dim a As Range
3
4      On Error Resume Next
5      Set a = Application.InputBox("Test", Type:=8)
6      On Error GoTo 0
7      If TypeName(a) = "Boolean" Then Exit Sub
8
9  End Sub
```

４行目に On Error Resume Next ステートメントを記述することにより，キャンセルが押下されて５行目でエラーが発生しても，エラーを無視して実行が継続される。(33)

【方法２】Array 関数の特性を利用する。(34)

この方法は，【方法１】のように広く利用されているとは言えないが，VBA の特性を知る上で興味深い。前述したように，バリアント型変数にセル参照を代入しても，変数の型は Variant のままである（図４を参照）。しかし，VBA の Array 関数を使用して，セル参照をバリアント型変数に代入すると，型は Range 型に変化する。

この特性をみたのが，次の Sample7 である。４行目でバリアント型変数 a に文字型（“Test”），セル参照（Range（"A1：C 10"））, Boolean 型（False）の３つの要素を持った配列を代入して，(35) ６～８行目で各要素の型をみた。a(1) が Range 型として認識されているのがわかる。(36)

(33)　６行目の On Error GoTo ０ステートメントは On Error Resume Next ステートメントを無効にするもので，通常，この２つのステートメントは対で使用される。

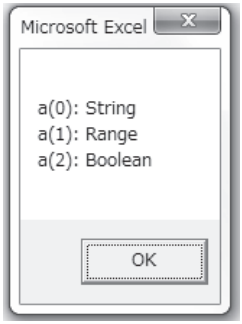
(34)　Moug モーグ（http://www.moug.net/）の「Q&A 掲示板」で教授いただいた。

(35)　Array 関数を使用する場合，明示的に VBA. を付加することによって，配列の下限を宣言する Option Base ステートメントに係わらず，下限を０とできるため，コードの安定性が増す。Moug モーグ（http://www.moug.net/）の「Q&A 掲示板」で教授いただいた。

(36)　Sample7 においては，例示のため，異なった型の要素からなる配列を挙げたが，一般的には配列における要素の型は統一する。また，異なった型の要素を扱う場合はユーザー定義型変数（構造体）を利用するのが一般的である。

図 10　Array 関数の特性

```
 1  Sub Sample7()
 2      Dim a As Variant
 3
 4      a = VBA.Array("Test", Range("A1:C10"), False)
 5
 6    MsgBox "a(0): " & TypeName(a(0)) & vbCrLf & _
 7         "a(1): " & TypeName(a(1)) & vbCrLf & _
 8         "a(2): " & TypeName(a(2))
 9
10  End Sub
```



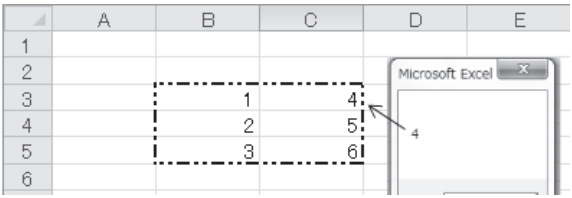
Type:＝8（セル参照）の場合に，この Array 関数の特性を利用して，キャンセル処理を含んだ Input メソッドの使用を考えたのが Sample8 である。

図 11　キャンセル処理②- 2

```
 1  Sub Sample8()
 2      Dim a As Variant
 3      Dim Data As Range
 4
 5      a = VBA.Array(Application.InputBox("Test", Type:=8))
 6      If TypeName(a(0)) = "Boolean" Then Exit Sub
 7      Set Data = a(0)
 8
 9      MsgBox Data(1, 2).Value
10
11  End Sub
```

Sample8 の 5 行目で，InputBox メソッドの戻り値がバリアント型変数 a，要素は a(0) のみ，に代入される。InputBox メソッドのダイアログボックスで，セル範囲が入力されたら，a(0) はセル参照を持つ Range 型に変化する。キャンセルが選択されたら，a(0) には Boolean 型の False が入り，6 行目でマクロが終了する。セル範囲への参照を持つ a(0) は，そのままでも，Range 型変数として扱うことが可能なのであるが，可読性を考慮して，7 行目で新たな Range 型変数 Data にセットしている。

図 12　Sample8（）の実行例



新たな Range 型変数にセル範囲をセットする利点は可読性の向上だけではない。図 11 はセル範囲 B3:C5 に数値データがあるケースで Sample8 を実行し，InputBox メソッドにより，セル範囲 B3:C5 を選択した結果である。値 4 のデータのセル番地は C3 であるが，セル範囲 B3:C5 を Range 型変数 Data にセットすることによって，Data(1, 2) と相対的な位置として扱うことができる。データ分析において，ワークシートにあるデータを，Data(行 index，列 index)として，相対的に扱える利点は大きい。

2．分析結果のワークシートへの表示についての考察

ワークシートに結果を表示する場合のセル領域の指定は，基本的には，Range プロパティ，Cells プロパティでなされ，様々なバリエーションがあるが，本稿では，基本的な例示のみを示し，話を進める。

表 3　セル指定の基本（例示）

単独セルのケース		セル範囲のケース	
Range プロパティ	Cells プロパティ	Range プロパティ	Cells プロパティ
Range("C5")	Cells(5, 3)	Range("A1:C5")	Range(Cells(1, 1), Cells(5, 3))

セルを（行 index，列 index）で指定する Cells プロパティはセル（複数）範囲の指定の場合 Range プロパティの助けを借り，しかも記述が長くなる。表 3 の例示でも，セル範囲 A1:C5 を Range プロパティでは Range("A1:C5") で済むところが，Cells プロパティを使用すると Range(Cells(1, 1), Cells(5, 3)) と冗長な記述になる。それにも拘わらず，多くのマクロでは，Cells プロパティを利用しているのだが，その理由は２つあると考えられる。

1）行 index，列 index に変数を利用することにより，ループ処理が効率化できる。

2）シート全体，あるいは特定のセル範囲を Cells で表すことができる。

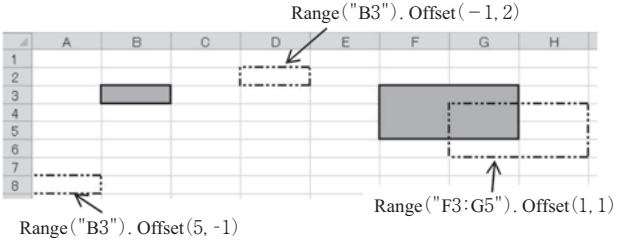
（例）アクティブシート全体のセルのクリア　Cells. Clear

記述が冗長になる欠点は Range 型変数を利用すれば改善できるのであるが，それをみる前に，セル範囲の操作で威力を発揮する Offset プロパティと Resize プロパティについてまとめておくことにする。

① Offset プロパティ

Range オブジェクトを基準とした相対位置のセル領域を参照する。戻り値は Range である。

図 13　Offset プロパティの例示

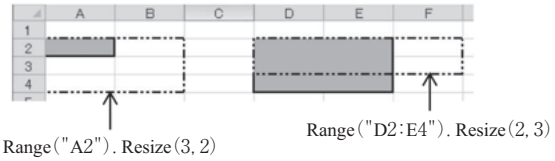


Offset プロパティを利用すれば，相対的なセル参照が可能であり，With ～ End With ステートメントと共に使用すると，読みやすいマクロを作成できる。

② Resize プロパティ

指定された範囲のサイズを変更する。戻り値は Range である。

図 14　Resize プロパティの例示



本稿では，Resize プロパティを利用して，分析結果のワークシートへの表示を考えていく。先にみたように，結果を表示するセル範囲を直接的に扱うより，セル範囲を Range 型変数にセットして相対的に扱うやり方が便利である。さらに，出力先のセル領域にデータがあった場合，そのセル領域に結果を上書きしてよいかの確認も必要である。

その手順は

1）出力領域の左上隅のセル番地を InputBox メソッドで取得する。

2）Resize プロパティでサイズを変更する。

3）出力先のセル領域にデータがあるかを調べ，存在した場合はセル領域のクリアの可否を確認をする。

1），2）はイメージとしては図 14 における左図であり，これをコードにしたのが，図 15 である。

図 15　出力先の選択の例示（コード）

```
1  myMsg = "出力先の左上隅のセルを指定して下さい"
2  a = VBA. Array(Application. InputBox(myMsg, Type:=8))
3  If TypeName(a(0)) = "Boolean" Then Exit Sub
4  Set Table = a(0). Resize(10, 3):  '例示として 10×3 の表
```

2 行目で出力領域の左上隅のセル参照を取得し，4 行目で，取得されたセルを基準にサイズを 10×3 に変更し，Range オブジェクト Table にセットしている。(37)

ワークシート上で作業をする場合，3）は常に心がける必要があるので，ここで，その機能を Function プロシージャとして作成しておくことにする。(38)

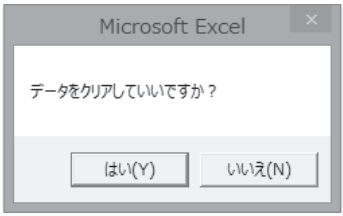
図 16 に示した Function プロシージャ F_Check1 は 1 行目に記述されているように，

(37)　4 行目を Set Table = a(0) とし，以降，Offset プロパティを利用する方法もある。

(38)「Function プロシージャは，Function~End Function で囲まれた一連のステートメントです。Function プロシージャは Sub プロシージャに似ていますが，関数は値を返すこともできます。Function プロシージャは引数を取ることができ，これらの引数は，呼び出し元のプロシージャから関数に渡されます。関数は，プロシージャ内で値をその名前に割り当てることで値を返します。」Office VBA 言語リファレンス（MSDN）から一部改変して引用。

図 16　Function プロシージャ　F_Check1

```
 1  Function F_Check1(myTitle As String, Target As Range) As Integer
 2  '
 3  'セル範囲のデータクリアの確認
 4  '       F_Check1 = vbYes:クリア可
 5  '       F_Check1 = vbNo :クリア否
 6  '
 7     Dim k As Long: '空白セル数
 8     Dim i As Long
 9     Dim myMsg As String
10
11     F_Check1 = vbYes
12
13     Target. Worksheet. Select
14     Target. Select
15
16     k = 0
17     For i = 1 To Target. Count
18           If Target(i) = "" Then k = k + 1
19     Next i
20
21     If k < Target. Count Then
22           myMsg = "データをクリアしていいですか？"
23           F_Check1 = MsgBox(myMsg, vbYesNo, myTitle)
24     End If
25
26  End Function
```



引数 myTitle（String 型），Target（Range 型）を受取り，Integer 型の戻り値（F_Check1）を返す。4 ～ 5 行目のコメントで記述しているように，引数 Target で参照されるセル領域をクリアしてよい場合は vbYes を返し，そうでない場合は vbNo を返す。(39)

最初，11 行目で F_Check1 = vbYes としておき，出力領域にデータがあった時は，23 行目で VBA の MsgBox 関数で出力先のデータのクリアの可否を確認し，その戻り値を F_Check1 の戻り値としている。(40)

16～19 行目で出力領域（Target）での空白セルの数（k）を計算している。(41) そのほか，図 16 のコードで，VBA の特徴として，留意しておきたいのは次の 4 点である。

(39)　MsgBox 関数のメッセージボックス（図 16）で［はい(Y)］を押下した場合は定数 vbYes（値 6），［いいえ(N)］を押下した場合には定数 vbNo（値 7）が返される。

1）VBA ではオブジェクトに対して何らかの操作をする際に，そのオブジェクトを選択（アクティブに）する必要はない。例えば，１番目のワークシートがアクティブな状態で，２番目のワークシートの A1 セルにデータ（値 10）を代入する場合，手動では，２番目のワークシートをアクティブにする必要があるが，Excel VBA では，アクティブシートが，２番目のワークシート以外でも，

```
Worksheets(2). Range("A1"). value = 10
```

とすれば，目的は達成できる。

2）そう考えると，図 16 における 13～14 行目は処理においては不要なコードである。しかし，ここでは，出力領域を目で確認したいので，Select メソッドで選択している(42)。しかし，セル範囲を選択する時は，アクティブシートのセル領域しか選択できないことに注意が必要である。例えば，１番目のワークシートがアクティブな状態で，２番目のワークシートの A1 セルを選択したい場合には，コードを

```
Worksheets(2). Select
Range("A1"). Slelect
```

と２行に分ける必要がある。これを，

```
Worksheets(2). Range("A1"). Slelect
```

と記述すると，見た目には問題がないようであるが，実行時エラーとなる。1）との違いに留意が必要である。

---

(40) メソッドにおいて，戻り値を利用する場合には引数を括弧で括る必要があったのと同様に，MsgBox 関数においても戻り値を利用する場合には 23 行目にあるように引数を括弧で括る必要がある。同じ MsgBox 関数でも，図３のように，引数を利用しない場合には括弧で括る必要はない。

(41) Range オブジェクトにおける空白セルは付表５にある SpecialCells メソッドによっても取得可能であるが，SpecialCells メソッドの使用には留意すべき点が多々あるので，ここでは採用していない。SpecialCells メソッドの特性について次稿以降で考察する。

(42) Select メソッドと Activate メソッドの使い分けについては，「付表５　Range オブジェクトの主なプロパティ，メソッド」における Activate メソッドの説明を参照。

3）オブジェクト式構文では，上位オブジェクトから記述するのが一般的であるが，

Target.Worksheet

は，Rangeオブジェクトの Worksheet プロパティであり[43]，出力領域（Target）が存在するシートへの参照を取得できる。出力領域がアクティブシートにあれば，2）でみたように13行目は不要であるが，出力領域がアクティブシート以外になる可能性がある場合には必要となる。

4）先にみたようにRangeオブジェクトである Target の要素は Target（行 index，列 index）として，相対的に扱えられるのであるが，18行目にあるように Target（index）と一元配列として扱うことも可能である。その場合，データは行方向の連続の index で順序づけられる。

図17はマクロにおける Function プロシージャ F_Check1 の使用例である[44]。8行目で，InputBox メソッドを用いてセル範囲への参照を取得し，それを Range 型変数 r に

図17　Functionプロシージャ F_Check1 の使用例

```
1   Sub Sample9()
2     Dim r As Range
3     Dim myMsg As String
4     Dim myTitle As String
5
6     myTitle = "Sample9"
7     myMsg = "セル範囲を選択して下さい"
8     Set r = Application. InputBox(myMsg, myTitle, Type:=8)
9
10    If F_Check1(myTitle, r) = vbNo Then Exit Sub
11    MsgBox "選択されたセル範囲をクリアします"
12    r. Clear
13
14  End Sub
```

(43)　付表5には記載していないが，オブジェクトブラウザーで確認でき，戻り値は Worksheet オブジェクトである。

(44)　InputBox メソッドのキャンセル処理は省略している。

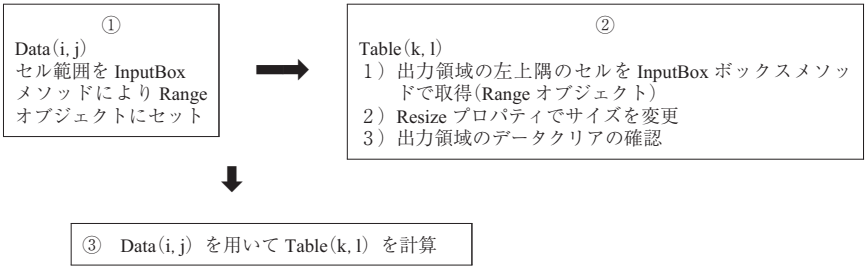
代入している。10 行目で，その r を引数として Function プロシージャ F_Check1 を呼び出し，その戻り値が vbNo であれば，なにもせずマクロを終了し，戻り値が vbYes ならメッセージを表示した後，12 行目で，セル範囲をクリアする。

ここで作成した Function プロシージャは比較的単純なものであり，使用先のマクロ（Sub プロシージャ）と同じ標準モジュールに記述している。さらに進んだプロシージャの利用については次稿以降でみていくことにする。

3．分析の枠組みとマクロの部品化

ワークシートを利用したデータ分析の枠組みは図 18 のようになると考えられる。

図 18　ワークシートを用いたデータ分析の枠組み



本節においては，図 18 における①と②について考察してきたのであるが，②の 3）の「出力領域のデータクリアの確認」が他のマクロでの利用を考慮して部品化（Function プロシージャ化）したのと同様に，①，②も部品化しておくことが望ましい。また，①と，②の 1）は同じ Function プロシージャを利用できることもわかる。

以上のことから，ワークシートを用いたデータ分析における標準モジュールの雛型として，図 19 が考えられる。

図 19 のイメージに沿って，本節で考察したマクロを付図 1 に掲げる。マクロは，一般的に，標準モジュールに記述される。付図 1 の標準モジュールには 1 つの Sub プロシージャ Sample10 と，3 つの Function プロシージャ（F_Input1，F_Table，F_Check1）が記述されている。

図 19　ワークシートを用いたデータ分析（雛型）

```
Main()
    ①  Data への参照を取得
       F_Input1() を使用
    ②  Table の準備
       F_Table() を使用
    ③  Data -> Table
       分析内容に依存
```

```
F_Input1(myMsg As String, myTitle As String) As Range
    セル参照を取得
```

```
F_Table(myTitle As String, Nr As Integer, Nc As Integer) As Range
出力領域の Range 変数の準備とデータクリアの確認
(引数) Nr:行数, Nc:
    ①  出力領域の左上隅セルの参照を取得
       F_Input1() を使用
    ②  サイズの変更と領域のセット
    ③  領域データのクリア確認
       F_Check1(myTitle As String, Target As Range) を使用
```

```
F_Check1(myTitle As String, Target As Range) As Integer
    セル範囲のクリアの確認
```

今回のまとめ

先ず，Ⅰ節において Excel VBA の特徴を概観した。一つ目の特徴として挙げられるのが，オブジェクト，プロパティ，メソッドから構成されるオブジェクト式と呼ばれる一群の構文である。特にプロパティの取り扱いについては，Application オブジェクトより下位のオブジェクトは上位オブジェクトの（オブジェクトを返す）プロパティと考えられること。また，プロパティを使用する構文においては，プロパティの戻り値が重要であることなどをみた。このことを念頭において，データ分析において利用頻度が高いと思われるプロパティ，メソッドについて一覧を作成した（付表 2 ～ 6 ）。

二つ目の特徴はオブジェクト型変数とバリアント型変数である。オブジェクト型（特に Range 型）変数はワークシートを用いたデータ分析で頻繁に利用されるものなのでⅡ節でみることとし，Ⅰ節の最後で，バリアント型変数の特性をみた。数値型データ，文字型データについては，バリアント型変数に代入すると型が入力データの変化するのが確認されたが，データ分析で頻繁に利用される Range 型については型の変化は起こらないことをみた。

Ⅱ節では，Ⅰ節での概観を元に，データ分析の手順から Excel VBA の特徴をみた。ワークシート上のデータを取得に利用される Type：＝8（セル参照）の Input メソッドの特性を調べ，次稿以降のデータ分析で利用できるコードを作成した（図 11 Sample8）。また，分析結果のワークシートへ表示について考察し，Range 型変数と Resize プロパティを利用したコードを提示した。最後に，Ⅱ節の考察のまとめとして，ワークシートを用いたデータ分析の枠組みを考え，マクロの雛形を作成した（付図１　Sample10）。

次稿においては，今回作成したマクロの雛形を拡張することを通して，さらに Excel VBA の特徴をみていくことにする。

参　考　文　献

大村あつし（2011）『Excel 2010 VBA 基礎編』技術評論社
大村あつし（2015）『Excel VBA 本格入門』技術評論社
門脇加奈子（2016）『かんたん Excel マクロ＆VBA』技術評論社
田中　亨（2016）『Excel VBA 逆引き辞典パーフェクト　第３版』翔泳社
土屋和人（2016）『Excel VBA パーフェクトマスター』秀和システム

参考ウェブサイト

インストラクターのネタ帳　http : //www.relief.jp/itnote/
エクセルの神髄　http://excel-ubara.com/
Office TANAKA　http://officetanaka.net/
Moug モーグ　http://www.moug.net/
MSDN（Microsoft Developer Network）　https://msdn.microsoft.com/ja-jp/

付表 1　マクロの記録・実行・保存，マクロの構成，VBE

準備	「開発」タブの表示	「ファイル」–「オプション」–「リボンのユーザー設定」
マクロの記録	記録	「開発」–「マクロ記録」
	記録停止	「開発」–「記録終了」

マクロの実行	「開発」タブから	「マクロ」で表示される一覧から選択，実行
		「マクロ」–「オプション」でショートカットキーに登録可能
	VBE 画面から	〈Sub/ユーザーフォームの実行〉(F5)
マクロの保存	xlsm 形式	→開くときは［コンテンツの有効化］

マクロの構成	Sub ～ End Sub	Sub マクロ名() ～ End Sub までが一つのマクロ（Sub プロシージャ）
	キーワード	「Sub」や「Range」など，マクロのためにあらかじめ用意されている単語
	ステートメント	マクロの中の個々の命令文
	コメント	アポストロフィ（'）で始まる文

VBE	(Visual Basic Editor)	
	起動	「開発」–「Visual Basic」
	Excel 画面との相互切り替え	[Alt] + [F11]
	VBE の画面構成	
	プロジェクトエクスプローラー	プロジェクトはモジュールの集合体
	標準モジュール	マクロを記述するための専用シート
	コードウインドウ	マクロの記述，修正を行う場所(標準モジュールの中身)
	マクロの記述，修正	
	（自動構文チェック）	｝ VBE 画面の「ツール」–「オプション」で設定
	自動メンバー表示	
	自動クイックヒント	
	自動データヒント	
	変数の宣言を強制する	｝ Option Explicit
	マクロの削除	
	「開発」タブから	「マクロ」で表示される一覧から削除
	VBE 画面から	コードウインドウから該当マクロ部分を削除
	標準モジュールの挿入・削除	挿入：「挿入」–「標準モジュール」
		削除：プロジェクトエクスプローラーで右クリック
	編集，エラー対策	
	オブジェクトブラウザー	[F2]
	「編集」ツールバーの表示	「ツール」を右クリックし，「編集」をチェック
	「デバッグ」ツールバーの表示	「ツール」を右クリックし，「デバッグ」をチェック
	イミディエイトウインドウの表示	「表示」–「イミディエイトウインド」([Ctrl] + [G])
	ローカルウィンドウの表示	「表示」–「ローカルウィンドウ」
	プログラムのリセット	「実行」–「リセット」

付表 2　Application オブジェクトの主なプロパティ，メソッド

オブジェクト	MSDN における説明
Application	Excel アプリケーション全体を表す

プロパティ	MSDN における説明	戻り値	取得のみ
ActiveCell	最前面に表示されている，アクティブウィンドウまたは指定されたウィンドウでのアクティブセルを表す Range オブジェクトを返す。	Range	○
ActiveSheet	作業中のブック，および指定されたウィンドウまたはブックのアクティブシートを表すオブジェクトを返す。アクティブシートが存在しないときは Nothing を返す。	Object	○
ActiveWorkbook	オブジェクトを返すプロパティ。アクティブウィンドウ内にあるブック（Workbook オブジェクト）を返す。値の取得のみ可能。ウィンドウが 1 つも開かれていないときは，Nothing を返す。	Workbook	○
Cells	作業中のワークシート上のすべてのセルを表す Range オブジェクトを返す。	Range	○
Range	セルまたはセル範囲を表す Range オブジェクトを返す。	Range	○
Workbooks	開かれているすべてのブックを表す Workbooks コレクションを返す。値の取得のみ可能。	Workbooks	○
WorksheetFunction	オブジェクトを返すプロパティ。WorksheetFunction オブジェクトを返す。値の取得のみ可能。	WorksheetFunction	○
Worksheets	Application オブジェクトでは，作業中のブックのすべてのワークシートを表す Sheets コレクションを返す。Workbook オブジェクトでは，指定されたブックのすべてのワークシートを表す Sheets コレクションを返す。値の取得のみ可能。Sheets オブジェクトの値を使用する。	Sheets	○

メソッド	MSDN における説明	戻り値
InputBox	ユーザー入力用のダイアログボックスを表示する。表示したダイアログボックスに入力された情報を返す。	Variant

付表 3　Worksheets コレクションの主なプロパティ，メソッド

コレクション	MSDN における説明
Worksheets	指定されたブックまたは作業中のブックにあるすべての Worksheet オブジェクトのコレクション。各 Worksheet オブジェクトはワークシートを表す。

プロパティ	MSDN における説明	戻り値	取得のみ
Count	コレクションに含まれるオブジェクトの数を表す長整数型（Long）の値を返す。	Long	○

メソッド	MSDN における説明	
Add	新しいワークシートを作成する。新しいワークシートがアクティブシートになる。	Worksheet

付表 4　Worksheet オブジェクトの主なプロパティ，メソッド

オブジェクト	MSDN における説明
Worksheet	ワークシートを表す。

プロパティ	MSDN における説明	戻り値	取得のみ
Cells	ワークシートのすべてのセル（現在使用されていないセルも含む）を表す Range オブジェクトを返す。	Range	○
Columns	作業中のワークシートのすべての列を表す Range オブジェクトを返す。	Range	○
Name	オブジェクト名を表す文字列型（String）の値を取得または設定する。	String	
Range	セルまたはセル範囲を表す Range オブジェクトを返す。	Range	○
Rows	指定されたワークシートのすべての行を表す Range オブジェクトを返す。値の取得のみ可能。Range オブジェクトを使用する。	Range	○
Shapes	指定されたワークシートのすべての図形を表す Shapes コレクションを返す。値の取得のみ可能。	Shapes	○

メソッド	MSDN における説明	戻り値
Activate	指定されたシートをアクティブにする。	
Copy	シートをブック内の他の場所にコピーする。	
Delete	オブジェクトを削除する。	
Paste	クリップボードの内容をシートに貼り付ける。	
PasteSpecial	指定された形式で，クリップボードの内容をシートに貼り付ける。他のアプリケーションからデータを貼り付けるときや，あるいは特別な形式でデータを貼り付ける場合に使う。	
Select	オブジェクトを選択する。	

付表5　Rangeオブジェクトの主なプロパティ，メソッド

オブジェクト	MSDNにおける説明
Range	セル，行，列，1つ以上のセル範囲を含む選択範囲を表す。

プロパティ	MSDNにおける説明	戻り値	取得のみ
Address	コード記述時の言語で参照範囲を表す文字列型（String）の値を返す。	String	○
Borders	スタイルまたはセル範囲の罫線を表すBordersコレクションを取得する。	Borders	○
Cells	指定した範囲のセルを表すRangeオブジェクトを返す。	Range	○
Columns	指定されたセル範囲の列を表すRangeオブジェクトを返します。	Range	○
ColumnWidth	指定された対象セル範囲内のすべての列の幅を設定する。値の取得および設定が可能。バリアント型（Variant）の値を使用。	Variant	
Count	コレクションに含まれるオブジェクトの数を表す長整数型(Long)の値を返す。	Long	○
CurrentRegion	アクティブセル領域（Rangeオブジェクト）を返す。アクティブセル領域とは，空白行と空白列で囲まれたセル範囲。値の取得のみ可能。	Range	○
End	ソース範囲が含まれる領域の終端のセルを示すRangeオブジェクトを返す。End＋方向キーに相当する。値の取得のみ可能。Rangeオブジェクトを使用。	Range	○
EntireColumn	オブジェクトを返すプロパティ。指定されたセル範囲を含む1列または複数の列全体（Rangeオブジェクト）を返す。値の取得のみ可能。	Range	○
EntireRow	オブジェクトを返すプロパティ。指定されたセル範囲を含む1行または複数の行全体（Rangeオブジェクト）を返す。値の取得のみ可能。	Range	○
Font	指定されたオブジェクトに設定されているフォントを表すFontオブジェクトを返す。	Font	○
Formula	オブジェクトの数式を，A1参照形式で，マクロ言語で表すバリアント型（Variant）の値を取得または設定する。	Variant	
FormulaR1C1	指定されたオブジェクトの数式をR1C1参照形式で，コード記述時の言語で取得または設定。値の取得および設定が可能。バリアント型（Variant）の値を使用。	Variant	
Hidden	行または列が非表示かどうかを表すバリアント型（Variant）の値を取得または設定。	Variant	
Interior	指定されたオブジェクトの塗りつぶし属性を表すInteriorオブジェクトを返す。	Interior	○
Name	オブジェクトの名前を表すバリアント型（Variant）の値を取得または設定。	Variant	
NumberFormat	オブジェクトの表示形式を表すバリアント型（Variant）の値を取得または設定。	Variant	
Offset	指定された範囲からのオフセットの範囲を表すRangeオブジェクトを返す。	Range	○
Range	セルまたはセル範囲を表すRangeオブジェクトを返す。	Range	○
Resize	指定された範囲のサイズを変更する。サイズが変更されたセル範囲（Rangeオブジェクト）を返す。	Range	○
RowHeiget	指定した範囲内の先頭の行の高さをポイント単位で取得または設定。値の取得および設定が可能。バリアント型（Variant）の値を使用。	Variant	
Rows	指定されたセル範囲の行を表すRangeオブジェクトを返す。値の取得のみ可能。Rangeオブジェクトを使用。	Range	○
Value	指定されたセル範囲の値を表すバリアント型（Variant）の値を取得または設定。値の取得および設定が可能。	Variant	

メソッド	MSDN における説明	戻り値
Activate	1つのセルをアクティブにする。このセルは，選択範囲の中に含まれている必要がある。セル範囲を選択するには，Select メソッドを使用。	
AutoFill	指定された対象セル範囲内のセルに対してオートフィルを実行。	Range
Autofit	対象セル範囲の列の幅や行の高さを内容に合わせて調節。	
BorderAround	セル範囲に罫線を追加し，追加した罫線の Color，LineStyle，Weight の各プロパティを設定。バリアント型（Variant）の値を使用。	
Clear	オブジェクト全体をクリア。	
ClearCountents	選択範囲から数式と文字を削除。	
ClearFormats	オブジェクトの書式設定を削除。	
Copy	範囲を指定の範囲またはクリップボードにコピー。	
Cut	オブジェクトを切り取り，クリップボードまたは指定された位置に貼り付け。	
Delete	オブジェクトを削除。	
Insert	ワークシートまたはマクロシートの指定された範囲に，空白のセルまたはセル範囲を挿入。指定された範囲にあったセルはシフトされる。	
PasteSpecial	Range をクリップボードから指定範囲に貼り付け。	
Select	オブジェクトを選択。	
SpecialCells	指定された条件を満たしているすべてのセル（Range オブジェクト）を返す。	Range

付表 6　Range. Borders コレクションの主なプロパティ

（罫線）

コレクション	MSDN における説明
Range. Borders	Range オブジェクトの 4 つの辺の罫線を表す，4 つの Border オブジェクトのコレクション。

プロパティ	MSDN における説明	戻り値	取得のみ
LineStyle	罫線の種類を設定します。値の取得および設定が可能。	Variant	
Weight	罫線または輪郭線の太さを表す XlBorderWeight 値を取得または設定。値の取得および設定が可能。	Variant	

付図１　第Ⅱ節のまとめ

```
Sub Sample10()

    Dim Data As Range: '入力データ
    Dim Table As Range: '出力領域

    Dim myTitle As String
    Dim myMsg As String

    myTitle = "データ分析"
'入力データへの参照の取得
    myMsg = "データ範囲を選択して下さい"
    Set Data = F_Input1(myMsg, myTitle)
    If Data Is Nothing Then Exit Sub

'出力領域の準備
    Set Table = F_Table(myTitle, 10, 3): '例示として 10×3
    If Table Is Nothing Then Exit Sub

    MsgBox "入力データの範囲: " & Data.Address & vbCrLf & _
        "出力領域: " & Table.Address

End Sub
```

```
Function F_Input1(myMsg As String, myTitle As String) As Range
'
'セル範囲への参照の取得
'
    Dim a As Variant
    a = VBA.Array(Application.InputBox(myMsg, myTitle, Type:=8))
    If TypeName(a(0)) = "Boolean" Then Exit Function
    Set F_Input1 = a(0)

End Function
```

```
Function F_Table(myTitle As String, Nr As Long, Nc As Long) As Range
'
'出力領域の準備
'     Nr:出力領域の行数
'     Nc:出力領域の列数
'
    Dim r As Range
    Dim myMsg As String

s1: '出力領域の取得
    myMsg = "出力先の左上隅のセルを指定して下さい"
    Set r = F_Input1(myMsg, myTitle)
```

```
13        If r Is Nothing Then Exit Function
14        Set F_Table = r. Resize(Nr, Nc)
15
16        '出力領域にあるデータのクリア確認
17        If F_Check1(myTitle, F_Table) = vbNo Then GoTo s1
18        F_Table. Clear
19
20    End Function

1     Function F_Check1(myTitle As String, Target As Range) As Integer
2     '
3     'セル範囲のデータクリアの確認
4     '      F_Check1 = vbYes:クリア可
5     '      F_Check1 = vbNo :クリア否
6     '
7         Dim k As Long: '空白セル数
8         Dim i As Long
9         Dim myMsg As String
10
11        F_Check1 = vbYes
12
13        Target. Worksheet. Select
14        Target. Select
15
16        k = 0
17        For i = 1 To Target. Count
18            If Target(i) = "" Then k = k + 1
19        Next i
20
21        If k < Target. Count Then
22            myMsg = "データをクリアしていいですか？"
23            F_Check1 = MsgBox(myMsg, vbYesNo, myTitle)
24        End If
25
26    End Function
```