○ AI「scikit-learn」による FX テクニカル分析の試み　［第１部］

副題；　scikit-learn による「予測と結果」の対比例　　amenbo the 3rd
分類用テキスト名；　『 アメンボ式 AI 活用法［第１部］ 』　　2019.11.30

前書き；

本シリーズは、FX（USDJPY；ドル/円）のテクニカル分析による予測に AI（scikit-learn）を使用した場合のテスト結果を報告するものです。（現状はプロトタイプ版です）

> 本稿では、為替（FX）チャートから生成済みの DataFrame（特徴量とラベル）と Python プログラムを使って機械学習・予測を実施した例を解説します。

読者は、下記２つの添付資料を使って実際に AI プログラムを体験することが出来ます。
①サンプルデータ（DataFrame に纏めた特徴量とラベル）　　；生成済み品を添付
②サンプルソフト（Python の「scikit-learn」による機械学習）；ソースコードを添付

※本稿では、Python、scikit-learn、および MT4（MQL4）の技術的な詳細解説はいたしません。

○『 AI「scikit-learn」による FX テクニカル分析の試み 』は現時点では下記の『２部作』ですが、今後も増やしていく予定です。（「第１部」⇒「第２部」の順にお読みください）

第１部［副題］scikit-learn による「予測と結果」の対比例　・・**本稿**（無料版）
第２部［副題］特徴量（説明変数）の選定と DataFrame 生成・・別稿（有料版）

本稿目次；

開発環境；

- 開発環境ソフト；　Spyder3（IPython）32bit 版を使います
- 機械学習用ライブラリ; scikit-learn を使います
- AI アルゴリズム；　RandomForest による「分類」を使います
- 解析対象「為替ﾃﾞｰﾀ」; USDJPY　５分足（OANDA デモ版 MT4 の為替データ）

開発環境ソフトには 32bit 版を使用します、理由は DLL 経由で情報の授受を行う予定の相手である MT4 が 32bit 版アプリのためです。（DLL を使うためには bit 統一が必要です）

※ scikit-learn は David Cournapeau 氏が開発した、Python 用のオープンソース機械学習ライブラリです。
※ MT4（MetaTrader4）は MetaQuotes Software Corp.が開発したソフトウエアです。
※ OANDA は OANDA Corporation. の登録商標です。

開発環境の補足；

現在メジャーである「Jupyter Notebook」では無く、同じ Anaconda 系の「Spyder3（IPython)」を使っている理由は単純に筆者の好みに過ぎません。
両者の相違はコード表記の「インライン表示宣言」の有無ぐらいであり、基本的に同じコードが 2 つの開発環境上で実行可能です。

筆者の立ち位置、および見解；

筆者は、これまで MT4(MQL4）をメインに扱ってきました、
筆者の立ち位置は、AI（機械学習など）は MT4(MQL4）の機能拡張のためのツールです。
また、Python や scikit-learn については、素人ですので、厳密性などは考慮していない部分が多々ありますのでご了解ください。（Python コードはメモだらけの件を、ご容赦）

AI では、予測や判定の「因果関係」が外部から観ると判らない場合があります、
そこで予測や判定に有効な「説明変数（特徴量)」のスクリーニングの際は、MT4(MQL4)など、因果関係が明白な手法との併用を推奨します。（DeepLearning の場合は除く）

用語の混乱について；　（恥ずかしながら用語統一が出来ていません）

どの「用語」が一般的なのか不明な場合が多々ありましたので、同一内容を表す「用語」でも、その時々の解説内容に最も適していると判断した「用語」を使いました。
例；

- 学習データ、教師データ、訓練データ
- 推論、予測
- 試験用データ、テストデータ
- 教師ラベル、テストラベル、
- 目的変数、ラベル、ラベルデータ、
- 特徴量、説明変数、パラメータ

幾つかの感想；

- MT4(MQL4)では EA を作る際、詳細にパラメータ設定を行う必要がありますが、他方、AI では基本条件（特徴量など）だけを指定すれば、詳細な設定値は勝手に「学習」して設定してくれるのは「さすが」と思いました。
- ただ残念なのは「Python、AI」には、MT4 に備え付けられている様な、パラメータ（特徴量）が有効であるか否かのスクリーニング機能がない事です。
  具体的に言うと MT4 にはパラメータの最適化機能があり、これを使うと対象とするパラメータがトレードに有効か否かを判断することが出来ます。
  （もし、AI 用に使えるツールが存在するならば、是非教えてください）

◎筆者の間違い、理解不足、思い違いを見つけましたら、ご指摘ください。
（拙い・汚い Python コード記述が多々あります件、未整理の状態を含めご容赦ください）
※筆者にとって、後から読んでも判り易いようにメモとして残した部分が多いです。

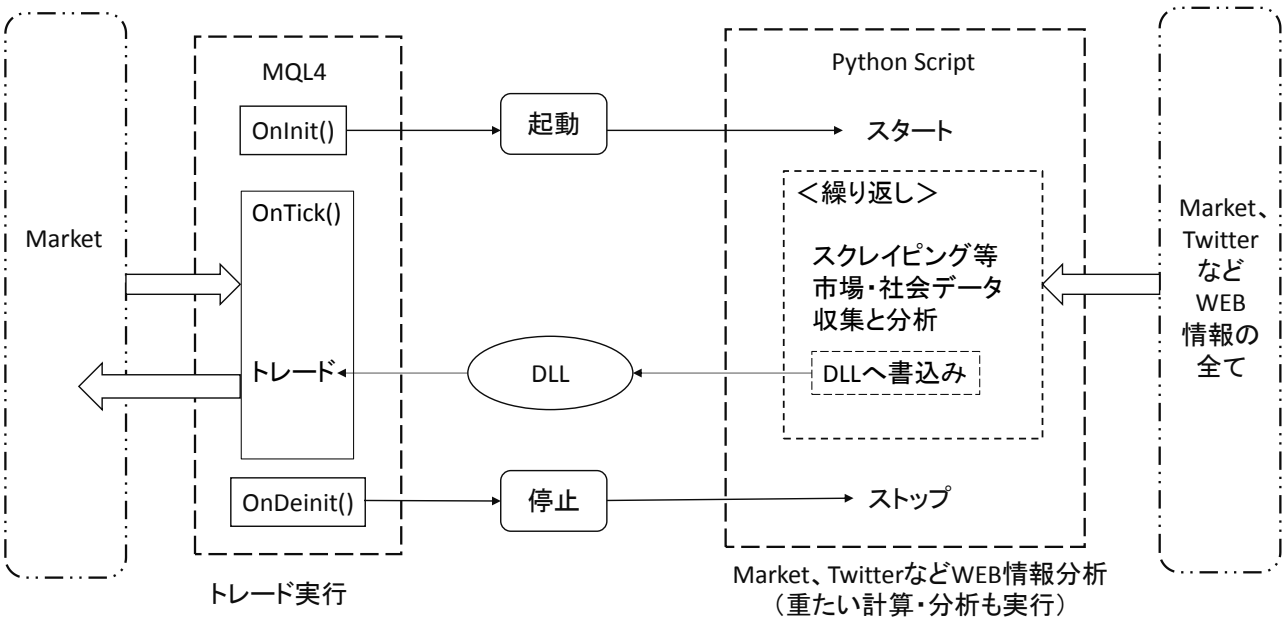
１．このシリーズ（２部作）で実現すること

※本稿の内容と直接には関係ないのですが、AI を MT4(MQL4)の機能拡張に使うための検討に、すでに余りにも時間がかり過ぎており、ときおり本来の目標と目的を見失いそうになるので、ここに改めて確認しておきました。

（１）筆者の「本来の目標」と「本稿の報告内容」との関係

※本来の目標は、AI 側から MT4（MQL4）側に「勝てそうな売買タイミング」を知らせることです。概要は下記のブロック図（過去の投稿原稿より抜粋）を参照頂ければ、一目瞭然と思います。

目標ブロック構成；



○ただ、実用上での大きな課題は、為替の変動を予測するための各種パラメータ、例えば Twitter 上などで特定の「文言」が頻発しても、その影響がマーケット上に現れるまでの時間遅れ（タイムラグ）は一定ではなく、「即座」に反映される場合（題材）から、「半月ぐらい」かかる場合（題材）まで、まちまちであることが経験則で（定性的に）判っています。

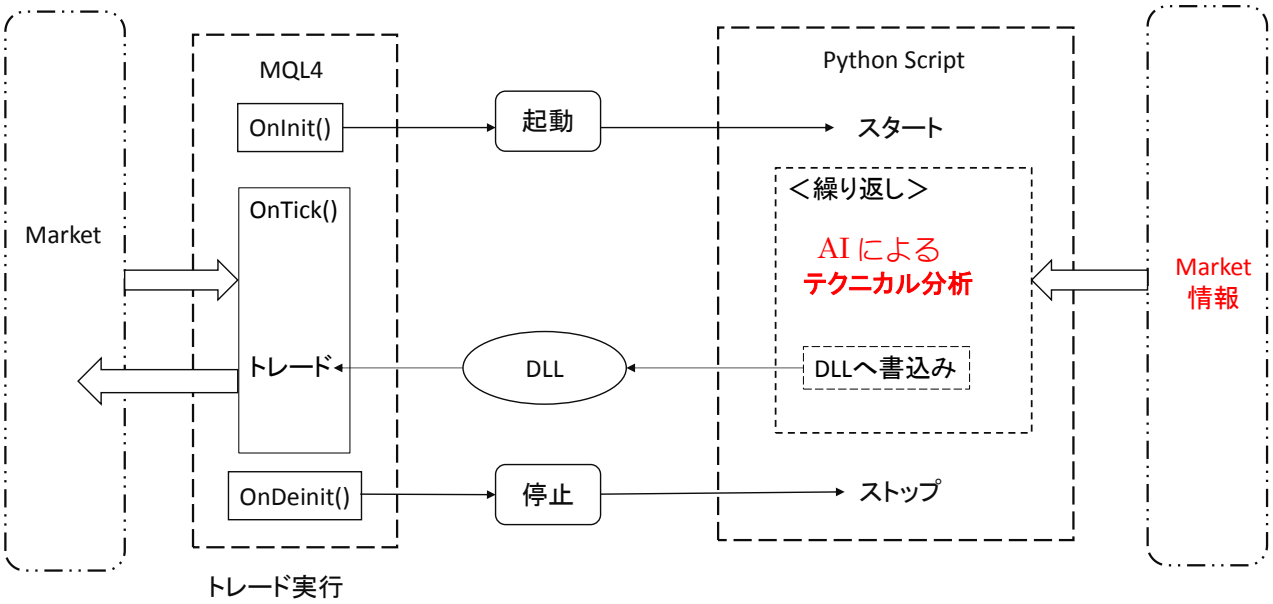
つまりシステムを構築して実際に稼働する前段階として、各パラメータによる効果の「時間遅れ、およびその影響力」などを数値化し、データ化することが必要になります。

・・正直言いますと、筆者には この測定方法の確立目途がまだ立っていません。

（　プロの世界では「たぶん確立済み」なのでしょうが！？　）

○そこで、今回は直ぐに試せる「テクニカル分析」を AI で実行してみて、その効果を確認することにしました。MT4(MQL4)で充分実行可能なことなのですが、敢て試してみることにしたのですが！、**これが厄介** でした。

今回（本稿）の目標をブロック上に表すと「朱書」の部分になります；



参考；DLL を経由して AI 側から MT4(MQL4)に情報（データ）を渡す具体的な方法は、
筆者のアーカイブ・サイト
[http://mql4.s1002.xrea.com/ai_mql4/] の記事「MQL4 と外部アプリの連携」を参照ください。

MT4(MQL4)の使い方については、筆者のもう一つのアーカイブ・サイト
[http://www.green.dti.ne.jp/sdimension/mql/] も参考になります。

上記の 2 サイトには何れも筆者が MT4 と AI を調査した内容を掲載しています。

（2）「目的変数」と「説明変数」の設定；
※本稿での「目的変数」は、FX 取引に使用する「MT4 での自動売買プログラム（EA）作成」、あるいは「IFO 注文」に活用できる内容（情報）であることを目標としています。
・本稿では「利益確定、損切」の各レベルを設定して、新規注文「売り、又は買い」を出した時に「勝てそうか！？否か」を示す「フラグ」を「目的変数」とします。
※「説明変数」はテクニカル分析で使う各種指標（インディケータ）をイメージしてください。

（3）AI アルゴリズムは、「RandomForest による分類」を使います；
※予測（推論）と言っても、目的変数は「FX（為替）の予測値」ではなく、「勝てるか否か」ですので、回帰ではなく「分類」を使います。
※当初、識別アルゴリズムでは有名な「サポートベクタマシン（SVM）」で分類を行おうとトライしたのですが、後述する問題に遭遇し「RandomForest」に切り替えました。
（後述＝「8.（4）＜参考＞アルゴリズムに「SVC(SVM)」を使用した場合の結果について」）

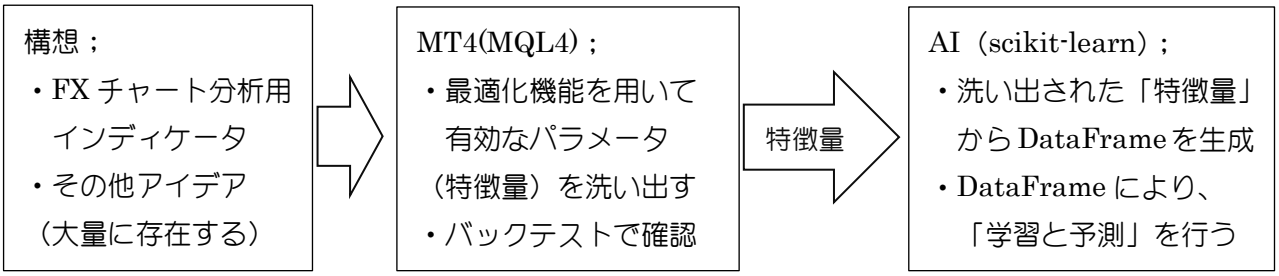
（4）対象とした「FX（為替；USD JPY）データ」
※OANDA の MT4 デモ・トレードからエクスポートしたものを使っています。

2．筆者の「AI テクニカル分析」方法論

※本稿で筆者が採用した「AI を FX テクニカル分析に応用する方法」は、簡単に表現するなら下記のブロック図で示すことができます。

・つまり、MT4(MQL4)は「特徴量」の洗い出しに使い、AI(scikit-learn)側では洗い出された特徴量から DataFrame（含、ラベル）の生成と「学習・推論（つまり予測）」を行います。

・AI 側では「特徴量」の洗い出しは行いません、最後に予測値とラベルを比較して正解率と期待利益を算出し、この手法の有効性を確認します。



※上記の様な「MT4 と AI の分業」を考えたのは、それぞれ下記の様な特徴があるからです。

	MT4(MQL4)	AI(scikit-learn)
有効な特徴量の洗い出し機能	・標準でパラメータ（特徴量）の「最適化機能」が装備されおり、これを使って特徴量の洗い出しと有効性の確認ができます。	たぶん、未対応！！？？と、思います ・専用ツールや実施例を探しているのですが、未だ見つかりません。※1
最適化機能（学習機能）	・パラメータ自体が f = f(x,y) の様に 2 変数以上の関数の場合でも、f を最適値にする [x,y] の組合せを探し出す（探索、最適化する）ことが可能です。	・RandomForest を使った本稿の例では、回帰線を特徴量として扱ったため、変数「足数、勾配」の扱いに苦慮しました。 ・特徴量としたいﾊﾟﾗﾒｰﾀ自体が、f = f(x,y) のように 2 つの変数で指定される場合、[x,y] の内どちらか一方（例えば x）を固定して、他の変数「y」の最適値を学習させることは出来ました。（回帰線の場合で確認） ・・[x,y] の最適組合せを学習させることが可能かは、未だ試していません。
		・DeepLearning の場合は？ f = f(x,y) の様な場合に、[x,y] の最適組合せを学習させることが可能か否かも未だ調べておらず、今後の課題としています。（たぶん、可能）※2

※1、2；ご存知の方がいれば、是非ご教授ください。

3．勝敗の考え方　・・とても重要です（EA 開発者にとっては、余りにも初歩的なことですが）

多くの「AI を為替予測に使う」記事（資料）を読んで感じたのは、筆者の出る幕など無いほどの実に高尚な数学を駆使しているのに、予測対象（目的変数）が MT4(MQL4)の EA を作る為には何の参考にもならない、と言うことでした。
どういう事かと言うと、実用的な「勝敗判定」を使っていないのです。

○実用的にする方法は、実に簡単で「利確レベル、損切レベル、スプレッド」を決めて、
「勝敗判定」をするようにすれば良いだけです。

「１章」で「分類」を使う、と述べましたが、その際に注意しなければならない事があります。

○「勝てない」との予測はいくら的中しても、トレード上は全く意味がありません。
意味があるのは、「勝てる」と予測したときに「どのくらい」の確率で勝てるのか、と
その時の、トレードの「期待利益」です。

・本稿で採用した分類方式では、新規トレードに入った後（マーケット インした後)、

①勝てた場合は　⇒　「＋1」
②負けた場合は　⇒　「－1」
③期間内（300 足）に決着がつかず「損益不明」の場合は　⇒　「0」

と、ラベリング（分類）しています。(詳細は「下記の（１）」を参照)

・マーケット インには、売りから入る「売り IN」と買いから入る「買い IN」の２通りを想定しています。

（１）勝敗の判定方法（ラベルの生成）

勝敗を判定するには、すなわち「ラベル」データを生成するには、
まず「マーケット・イン」から「マーケット・アウト」するまでの過程と条件を決めます。

○マーケット・イン条件；
①「売り」、または「買い」の何方から入るか（IN するか）を決めます。
② profit（利確レベル)、loss（損切レベル)、spread（売値と買値の差）を決めます。
（通常は「profit＞loss」となる様に決めてください）
参考；「profit＜loss」となる様に決めている場合を見かけることがありますが、
いわゆる「コツコツ稼いで、ドカット負ける」戦術であり、推奨いたしません。

○マーケット・アウト条件（判定条件）；
※FX チャートは通常 Bid（売値）表示なので、「売り IN」か「買い IN」かによって、
若干ですが判定（計算）方法が異なります。

◇［買い IN（買いからマーケットに入る)］場合のイメージ

[START]＝マーケット・インした時点
[LAST]＝「利確、損切」が出来ない場合に強制的にマーケットから出るタイミング

＜損益（勝敗）の判定プロセス＞　・・USD/JPY で「買い」から入った場合で解説

例；5 分足で、［START］足は「A 点；（Bid 表示）110.00 円／ドル」とします、
profit=0.32、loss=0.20、spread=0.02、期限（強制退出）=300 足、と決めて、
さらに単純化のために「1 ドル（1 通貨)」の売買で考えます。

①「買い IN」；　A 点
［START］足で、「買い」からマーケットに入ります（買い IN)、
FX は通常、価格表示が Bid（トレーダーから見て売りの場合）なので
「買い値（Ask=Bid+spread）＝110.00+0.02＝110.02 円」です。

②「売り OUT」；　B 点
買い IN 後、［START］足～［LAST］足までの「300 足」内で、
・最初に「利確レベル（Bid=買い値+profit）＝110.02+0.32＝110.34 」に達したら
利確の［売り］でマーケットから出ます。（手仕舞い）
⇒　「32 銭（pips)」の利益
⇒　「＋1」(Win）とカウント

・最初に「損切レベル（Bid=買い値 - loss）＝110.02 - 0.20＝108.02 」に達したら
損切の［売り］でマーケットから出ます。（手仕舞い）
⇒　「20 銭（pips)」の損失
⇒　「－1」(Loss）とカウント

③［START］足～［LAST］足までの「300 足」内で、

・「利確レベル」にも「損切レベル」にも達しなかった場合は、

「300 足」目の価格（FX 値）の［売り］で、マーケットから出ます。（強制）

⇒　約定値と損益は予測できません。

ただし、「－19 銭～＋31 銭」のどこかに収まります。

⇒　「0」（Draw）とカウント

※「売り」から入った場合でも、下記イメージ図を参照して同様に判定していきます。

◇［売り IN（売りからマーケットに入る）］場合のイメージ

・・profit レベルと loss レベルが「買い IN」の場合とは逆側にあることに注意。

（２）期待利益について

※本節では、特に筆者が重要と考える期待利益（期待損益値）を本件（下記）の例で考察してみます。

利確レベル	profit=0.32
損切レベル	loss=0.20
勝率（％）	45.0 ％ ＝0.45
負率（％）	55.0 ％ ＝0.55（例えば、ドローは全て負けに含めると仮定）

期待損益値＝１トレード当たりの平均獲得値

＝（勝ったときの利益）×勝率＋（負けたときの損失）×負率

＝ 0.32×0.45＋（－0.20×0.55）＝ 0.144－0.110 ＝ 0.034

◎上記の例では「profit＞loss」としている為、「勝率＜負率」（勝率が 50％未満）にも係らず「期待損益」は『プラス』であることに注目してください。

（3）目標とする「勝率と期待利益」について・・・実は、多少「後付け」の目標

※当初、まさか ［損切（－1）の発生頻度］＞＞［利確（+1）の発生頻度］、
つまり負けを予測する頻度が圧倒的に多いとは思わず、
安易に「トレード全体の期待利益」を計算すれば良い、

と考えていたのですが！

＜＜ ・・・考え方を変えました。・・・ ＞＞

◇常にマーケット IN するのではなく、

損切［－1］との予測が出た時は、トレード（マーケット IN）せず、
利確［＋1］と予測した時のみ、トレード（マーケット IN）をする、

事に、しました。

◇従って、予測精度を測定する際は、

①予測（推論値）が利確（+1）で、その真値（ラベル）も利確（+1）である確率、
つまり利確（+1）予測に限定した場合の勝率が、「**50%**以上」であることを目標とする、（当然、トレード内での「期待利益」はプラス）

事に、しました。

雑感；

○本件の手法は、為替（FX）に限らず「株や仮想通貨」にも適用可能です。
FX や株ならば優れたツール、例えば、

・FX なら MT4(MQL4)など、
・株なら TradeStation（トレードステーション）など、

素人でも、慣れればプログラム取引が可能なツールが存在しますので、
テクニカル分析に、無理やり AI ツールや Python を使う必要も感じられないのですが、
仮想通貨（暗号通貨）ではツールがあまり揃っていない様なので、役に立つのかもしれません。

４．Python コードの構成概要（第１部、第２部）

注；本稿（第１部）では、下記に示す「コード一覧」の内、「項番 10」のみは詳細に解説しますが、「第２部」のみに添付している「コード（1～9、11)」の詳細は解説していません。

ただし項番「1～9 まで」のコードで生成された「DataFrame（特徴量とラベルデータ)」を本稿に添付しています。第６章では、これを使って項番 10 のコード「k_10_test.py」と「k2_10_test.py」により学習と推論（予測）を行う方法と実例を、やや詳細に解説いたします。　⇒「第６章；DataFrame 例と学習・推論（予測）実行例」を参照

（１）「USDJPY5_2019.01.10.csv」データ解析用コード一覧

※「項番 1～9」の詳細な「コード名称」とその内容は、「第２部（有料)」にて詳細に解説。

項番	プログラム名	処理内容
1 ｜ 8	k_01_＊＊.py ～k_08_＊＊.py	各「特徴量（説明変数)」の抽出と、 若干のスケーリング擬き処理
9	k_09_nan_drop.py	欠損データのある行を削除する
10	k_10_test.py	DataFrame を学習用と試験用に分割後、AI による学習・予測を実行し、予測結果をラベルと比較する
11	k_11_kidou.py	項番「1」から「10」までを、順番に実行する

（２）「USDJPY5_2019.05.14.csv」データ解析用コード一覧

※「項番 1～9」の詳細な「コード名称」とその内容は、「第２部（有料)」にて詳細に解説。

項番	プログラム名	処理内容
1 ｜ 8	k2_01_＊＊.py ～k2_08_＊＊.py	各「特徴量（説明変数)」の抽出と、 若干のスケーリング擬き処理
9	k2_09_nan_drop.py	欠損データのある行を削除する
10	k2_10_test.py	DataFrame を学習用と試験用に分割後、AI による学習・予測を実行し、予測結果をラベルと比較する
11	k2_11_kidou.py	項番「1」から「10」までを、順番に実行する

・「コード（k_＊＊.py、k2_＊＊.py)」は分析対象（FX データ）に係らず、構成とコード内容は略共通ですが、下記に示す若干の違いがあります。

①マーケット IN を「買い」「売り」の何方から行うか

［k_＊＊.py］では「買い」から入り　［k2_＊＊.py］では「売り」から入る

②「学習用データ」と「試験用データ」の分割比（%）・・(色々試した)

［k_＊＊.py］では「9：1」、　　［k2_＊＊.py］では「8：2」

再確認；

○本稿では、「項番 1～9」の Python コードによって FX チャートから生成される DataFrame（特徴量とラベル）がすでに在る、との前提で解説しています。

5．サンプル「DataFrame と k_10_test.py」のダウンロード

※実際に「scikit-learn」を動かして、その感覚を理解できるように本稿では 2 セットの
サンプルを準備しました。
これは「第６章；DataFrame 例と学習・推論（予測）実行例」を試すための資料一式です
（注）「k_10_test.py」と「k2_10_test.py」の各セットは一緒にダウンロードされます。

※「サンプル DataFrame と FX データ」、および「k_10_test.py、k2_10_test.py」は、
下記「(1)」に記載の URL からダウンロード可能です。（実際に試してみてください）
（注）「k_10_test.py、k2_10_test.py」は、Spyder3（IPython）での実行を前提にしています、
他の環境（例；Jupyter Notebook）で実行する場合は環境に合わせて修正が必要です。

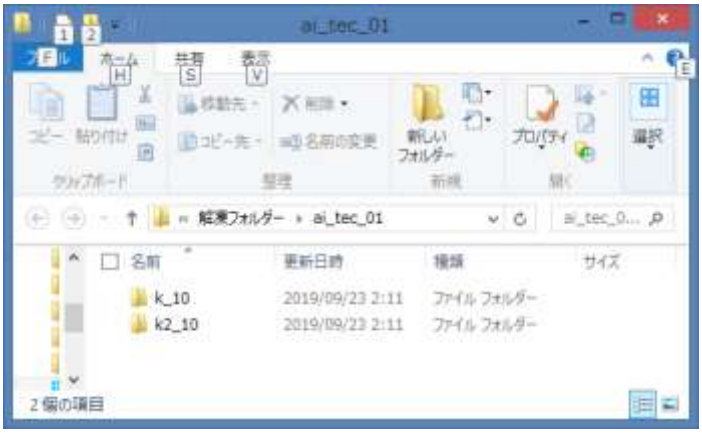
・FX データは本稿（第 1 部）では使いませんが、このデータから DataFrame を作ったのだと、
内容を確認出来る様に参考として添付しました。

（１）ダウンロードと解凍

・URL「 http://mql4.s1002.xrea.com/ai_tec/ai_tec_01.zip 」をアクセス、
（ゴゴジャンからダウンロードした場合は、同じものが本稿に同梱されていますのでアクセス不要です）

「ファイルを保存する」を選択し、［ＯＫ］をクリックすると、
「ai_tec_01.zip」がダウンロードされます。

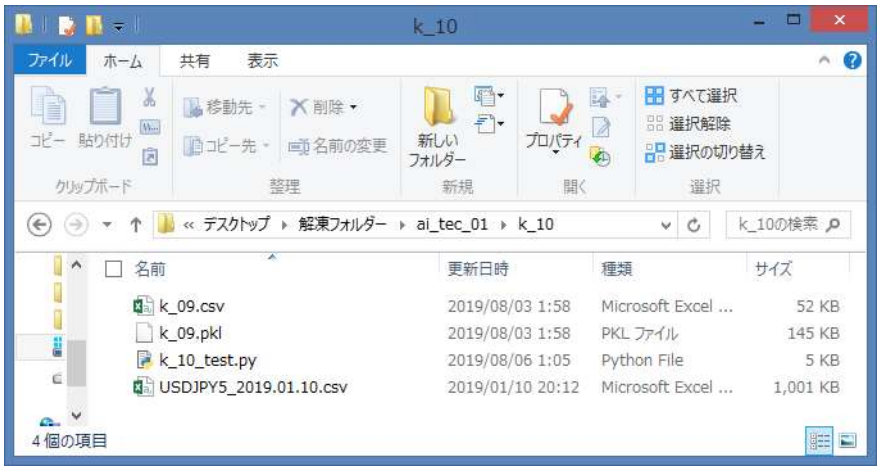
・解凍すると、「ai_tec_01」ホルダーが作成されて、その中に更に 2 つのホルダーがあります。

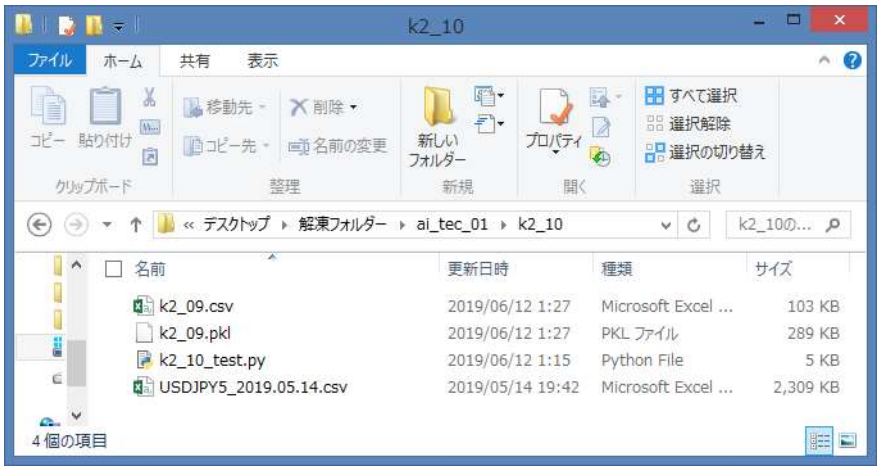


ホルダー構成;
・「k_10」
・「k2_10」

２つのホルダー中には、それぞれ下記に示すコンテンツが収納されています。

ホルダー内のスクリーン・ショット；





収納物の内容；

k_10 フォルダー

ファイル名	種　類	内　容
k_09.pkl	DataFrame	4 章 (1) の項番 1～9 により生成される特徴量とラベル
k_09.csv	csv ファイル	参考添付；上記 DataFrame 内容の（読者）確認用
k_10_test.py	Python プログラム	上記 DataFrame から学習し「勝敗」予測を行う
USDJPY5_2019.01.10.csv	FX の 5 分足データ	「k_09.pkl」を抽出した大元の為替データ

k2_10 フォルダー

ファイル名	種　類	内　容
k2_09.pkl	DataFrame	4 章（2）の項番 1～9 により生成される特徴量とラベル
k2_09.csv	csv ファイル	参考添付；上記 DataFrame 内容の（読者）確認用
k2_10_test.py	Python プログラム	上記 DataFrame から学習し「勝敗」予測を行う
USDJPY5_2019.05.14.csv	FX の 5 分足データ	「k2_09.pkl」を抽出した大元の為替データ

※「k*_09.csv」により、Python を経由せずに簡単に DataFrame 内容を確認できます。

（２）サンプルデータの保存

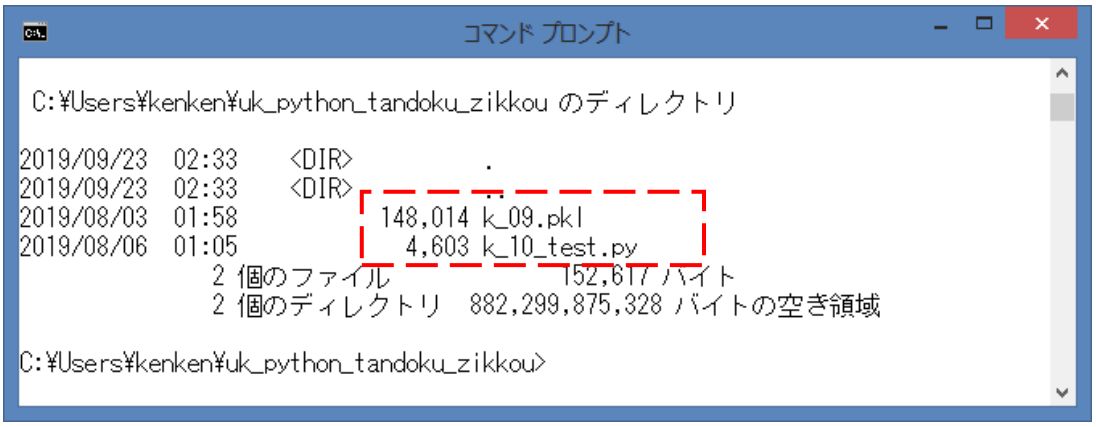
※ダウンロードしたデーター式の保存場所は、以下を参照して決めてください。

今回だけ特別に、Spyder 環境（あるいは Jupiter Notebook など）を導入しておらず、「ただ Python が使えるだけの環境しか設定していない」、と言う読者用に DOS 窓で実行する場合の保存先も下記「①」に示しました。

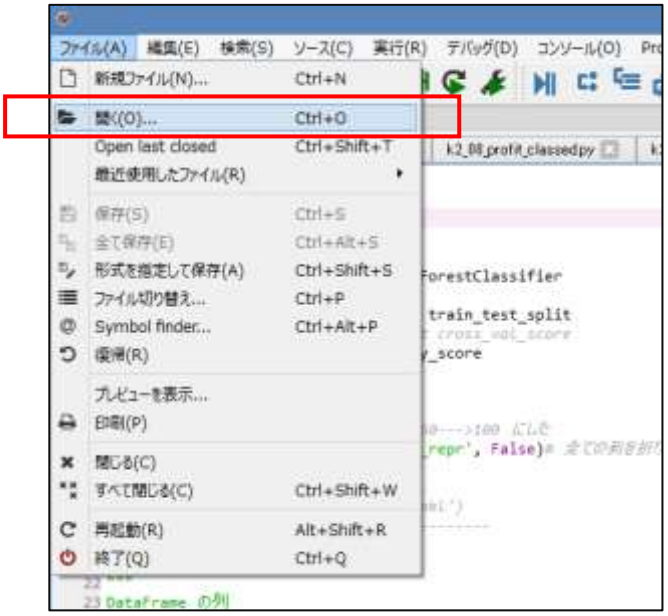
① Python のみが使え、Spyder 等をインストールしていない場合；

・Python が起動できる「フォルダー」内にデーターを保存します。
例として筆者が実行したときの例を下記に示します。

・「k_09.pkl」と「k_10_test.py」のセットは必ず同一ホルダーに保存します
（「k2_09.pkl」と「k2_10_test.py」で試す場合も同様にセットで保存します）



② Spyder 環境が使える場合；



「k_09.pkl」と「k_10_test.py」をセットで、Spyder の［ファイル］－［開く］でアクセスされる「フォルダー」内に保存します。
（Spyder を立ち上げてから確認してください）

・・「k2_09.pkl」と「k2_10_test.py」で試す場合も同様にセットで保存します。

※FX（為替）データの保存は、本稿での実行に際しては不要です。
当然ですが、別稿「第 2 部」で解説する特徴量の抽出時には必要です

６．DataFrame 例と学習・推論（予測）実行例

※「.pkl」形式ファイルは phton で開かないと可視化できず、チョット不便なので
読者用に本稿では「.csv」形式で保存したもので解説します。
全く同一の考え方なので、k2_09.pkl についての解説は省略します。

（１）DataFrame「k_09.pkl（k_09.csv)」;　データ数は 1050 セット（0～1049）です。

	s_sl	m_sl	l_sl	bl_5	bl_0	_4	_3	_2	_1	_0	_h3	_l3	_h2	_l2	_h1	_l1	p_by	p_sl
0	−44	−16	11	16	17	1	1	0	0	0	0	0	0	0	0	0	−1	1
1	−74	−40	13	14	14	0	0	0	−1	0	0	0	0	0	0	0	−1	1
2	208	52	16	14	13	−1	−1	0	1	1	0	0	1	0	0	0	−1	1
3	−168	−14	22	25	18	0	0	0	−1	0	0	0	0	0	0	0	−1	1
4	−13	84	18	31	32	1	1	0	0	0	0	0	0	0	0	0	−1	1
5	−42	90	6	20	27	1	1	1	1	1	1	1	1	0	1	0	−1	1
6	−10	−12	1	14	14	0	0	0	−1	−1	0	0	0	0	0	−1	−1	−1
7	−196	−109	0	20	16	0	0	0	0	0	0	0	0	0	0	0	−1	−1
8	192	80	−7	24	24	0	0	1	1	1	1	0	0	0	0	0	−1	−1
9	54	11	−16	26	28	1	1	1	1	1	0	0	1	0	1	0	−1	−1
10	58	19	−22	25	20	0	1	1	1	0	0	0	0	0	0	0	1	−1
11	31	153	−20	59	52	0	0	0	0	0	0	0	0	0	0	0	1	−1
12	158	−7	−17	48	54	−1	−1	−1	−1	−1	0	0	0	0	0	0	1	−1
13	59	−162	−2	35	35	0	0	0	0	−1	0	0	0	0	0	0	−1	−1
14	149	164	5	25	28	1	1	1	1	1	1	1	1	0	1	0	−1	1
15	101	23	10	37	33	−1	−1	−1	0	0	0	0	0	0	0	0	−1	−1
16	−21	−51	18	36	40	−1	−1	−1	−1	−1	0	0	0	0	0	0	−1	−1
17	39	7	21	30	29	0	0	0	0	0	0	0	0	0	0	0	−1	1
18	−75	−46	22	29	30	1	0	0	0	0	0	0	0	0	0	0	−1	1
19	−44	73	17	40	36	0	0	0	0	0	0	0	0	0	0	0	−1	−1
20	−117	−160	17	64	60	0	0	0	0	0	0	0	0	0	0	0	−1	−1

・・・・・・・・・・途中省略・・・・・・・・・・・

748	−45	2	9	10	10	1	1	1	1	1	1	0	0	0	0	0	−1	1
749	32	23	9	9	7	0	0	1	0	0	0	0	0	0	0	0	−1	−1
750	−18	32	6	15	14	1	1	1	1	0	0	0	0	0	0	0	−1	−1
751	−60	−31	1	14	14	1	0	1	1	0	1	0	0	0	0	0	−1	−1
752	36	20	−3	8	7	0	0	0	1	1	0	0	0	0	1	0	−1	−1
753	−63	−25	−4	10	9	0	0	0	−1	0	0	0	0	0	0	−1	0	−1
754	−35	−33	−3	10	10	0	0	0	0	0	0	0	0	0	0	0	0	−1
755	262	36	−5	12	11	0	0	1	1	1	0	0	1	0	1	1	0	−1
756	−3	−46	−5	15	12	0	0	0	0	0	0	0	0	0	0	0	0	−1
757	−110	−19	−2	24	23	0	0	0	0	0	0	0	0	0	0	0	0	−1
758	−141	−61	2	16	21	−1	−1	−1	−1	−1	0	−1	0	−1	−1	−1	0	−1
759	−59	−59	5	10	10	1	0	0	0	0	0	0	0	0	0	0	0	−1
760	−31	−32	6	13	12	1	0	1	0	0	0	0	0	0	0	0	0	−1
761	−11	−62	5	16	13	0	0	0	0	0	0	0	0	0	0	0	0	−1
762	45	49	3	16	16	0	0	0	0	0	0	0	0	0	0	0	−1	−1
763	−45	−66	3	16	16	−1	0	−1	−1	−1	0	0	0	0	0	0	0	−1
764	56	67	2	15	18	1	1	1	1	1	1	0	0	0	0	0	−1	0
765	63	71	4	10	12	1	1	1	1	1	1	0	1	0	1	1	−1	0
766	18	27	13	12	11	0	0	0	0	0	0	0	0	0	0	0	0	0
767	191	59	20	13	13	−1	0	0	0	0	0	0	0	0	0	0	0	0
768	−71	35	26	12	12	1	1	0	1	1	0	0	0	0	0	0	0	0
769	−74	−92	32	14	16	−1	−1	−1	−1	−1	0	−1	0	−1	0	−1	0	−1
770	−156	−61	31	17	14	0	−1	−1	−1	0	0	0	0	0	0	0	0	0
771	−49	−70	23	37	26	0	0	0	0	0	0	0	0	0	0	0	0	0
772	35	−56	12	56	52	0	0	0	0	0	0	0	0	0	0	0	0	0
773	−57	−123	10	44	47	1	1	1	0	1	0	0	0	0	0	0	0	−1

・・・・・・・・・・途中省略・・・・・・・・・・・

1032	22	-55	24	19	21	0	-1	-1	-1	-1	0	0	0	0	0	-1	-1	1
1033	-62	-153	28	19	16	0	0	0	0	0	0	0	0	0	0	0	-1	1
1034	-24	-26	18	23	23	0	0	1	0	0	0	0	0	0	0	0	-1	1
1035	9	-53	7	29	28	1	0	1	1	1	0	0	0	0	0	0	-1	1
1036	152	88	-6	27	27	1	1	1	1	1	0	0	0	0	0	0	-1	1
1037	260	163	-14	17	26	1	1	1	1	1	1	1	1	1	1	1	-1	-1
1038	144	98	-18	11	13	1	1	1	1	1	1	0	1	0	1	0	-1	-1
1039	128	-1	-15	25	18	0	0	0	0	0	0	0	0	0	0	0	1	-1
1040	-58	0	-13	37	30	0	0	0	0	0	0	0	0	0	0	0	1	-1
1041	154	-106	-10	51	50	-1	0	0	0	0	0	0	0	0	0	0	1	-1
1042	-436	-265	0	32	43	-1	-1	-1	-2	-1	-1	-2	-1	-2	-1	-2	1	-1
1043	65	-76	8	32	33	-1	-1	0	0	0	0	0	0	0	0	0	-1	1
1044	107	141	11	35	35	0	0	0	0	0	0	0	0	0	0	0	-1	1
1045	-299	12	13	38	39	1	1	0	1	0	0	0	0	0	0	0	-1	1
1046	427	-5	7	21	23	-1	0	1	1	1	0	0	0	0	1	0	-1	1
1047	-87	25	4	23	22	0	0	0	0	0	0	0	0	0	0	0	-1	-1
1048	46	-52	4	20	17	0	0	0	-1	-1	0	0	0	0	0	-1	1	-1
1049	-192	16	-5	26	26	1	1	1	0	0	0	0	0	0	0	0	-1	-1

（２）DataFrame の解説（概要のみ）

※本 DataFrame の生成方法（解説）については、「第２部（有償版）」を参照ください。

DataFrame は、下記「①、②」の２ブロック（要素）から構成されます。

① 特徴量；「USDJPY5_2019.01.10.csv」から計算される値、および「論理判定結果」

② ラベル；profit=0.32、loss=0.20、spread=0.02、期限（強制退出）=300 足として、
「３．勝敗の考え方」を適用した判定結果（利確（+1）、損切（-1）、ドロー（0））。

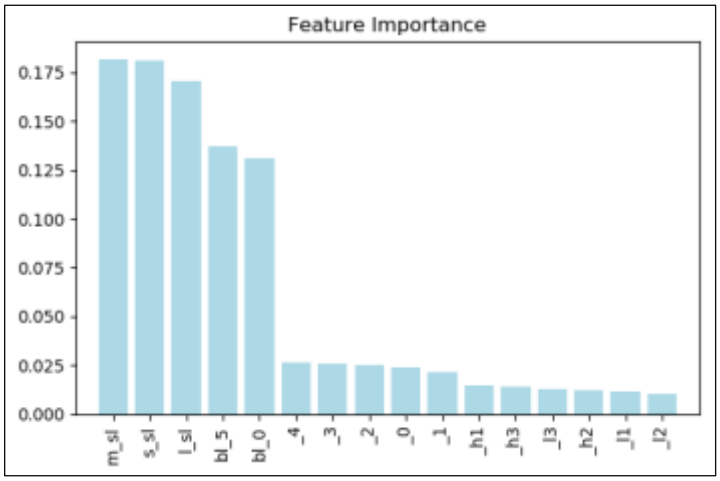
特徴量（説明変数）

ラベル；勝敗判定結果（目的変数）

	s_sl	m_sl	l_sl	bl_5	bl_0	_4	_3	_2	_1	_0	_h3	_l3	_h2	_l2	_h1	_l1	p_by	p_sl
0	-44	-16	11	16	17	1	1	0	0	0	0	0	0	0	0	0	-1	1
1	-74	-40	13	14	14	0	0	0	-1	0	0	0	0	0	0	0	-1	1
2	208	52	16	14	13	-1	-1	0	1	1	0	0	1	0	0	0	-1	1
3	-168	-14	22	25	18	0	0	0	-1	0	0	0	0	0	0	0	-1	1
4	-13	84	18	31	32	1	1	0	0	0	0	0	0	0	0	0	-1	1

p_by；買いから入った場合の勝敗判定、　　p_sl；売りから入った場合の勝敗判定

参考；勝敗判定への各特徴量の寄与度



・「s_sl、m_sl、l_sl、bl_5、bl_0」の５個の特徴量でほぼ決まることが左図から判ります。

・実は、今回やってみて初めて判ったことです。

7．テストプログラム「k_10_test.py」の解説

（1）「k_10_test.py」コードと若干の解説

・特に必要と判断した部分（コード）にのみ解説を入れました。

・読みやすいように筆者のメモ「Python コメント部分」を一部のみ「緑色」にしました。

・プロトタイプのため、コード記述が汚いこと、及び実行には不要のコード（コメント化済み）も残してあることをご了承ください。（筆者の記憶用です）

```
# -*- coding: utf-8 -*-
"""
[k_10_test.py]
@author: kenken
"""
print("")
print("------- k_10_test.py スタート----------")
print("")
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
#from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
import pandas as pd

# 表示範囲の設定
pd.options.display.max_rows = 100# 60--->100　にした
pd.set_option('display.expand_frame_repr', False)# 全ての列を折りかえさずに表示するために入れた
# 機械学習データを読込み
df=pd.read_pickle('k_09.pkl')
#df=pd.read_pickle('fx_data_target.pkl')
#----------------------------------------
print(df[0:20])
#
"""
DataFrame　の列名
s_sl,m_sl,l_sl,bl_5,bl_0,_4,_3,_2,_1,_0,_h3,_l3,_h2,_l2,_h1,_l1,p_by,p_sl
"""
#　トレーニングデータとテストデータに分離する

### 「買い｜N」用に特徴量（説明変数）を設定している
print("")
print("==========　「買い｜N」の場合で予測します　==========")
print("")
#
X_train,X_test,y_train,y_test=train_test_split(df[['s_sl','m_sl','l_sl','bl_5','bl_0',
    '_4','_3','_2','_1','_0',
    '_h3','_l3','_h2','_l2','_h1','_l1']],df['p_by'],test_size=0.1,shuffle=False,stratify=None)
#test_size=0.2 では精度が大幅に落ちる・・まだまだ改善の余地あり
"""
```

一連のインポート宣言

「DataFrame；k_09.pkl」の読込み

・これの作成方法は「別稿（第 2 部）」で詳細解説しています。

（19 頁）「解説 1-1」を参照

```
時系列データを扱う場合は、「shuffle=False,stratify=None」をセットで指定する事！
stratify は「層化抽出法」のこと
"""

"""
####　「売り ｉ Ｎ」には特徴量の再調整が必要！！
####　MT4 の最適化でも、「売り In」と「買い In」ではパラメータの最適設定値が可成り異なった
####print("==========　「売り ｉ Ｎ」の場合で予測します ==========")
X_train,X_test,y_train,y_test=train_test_split(df[['s_sl','m_sl','l_sl','bl_5','bl_0',
      '_4','_3','_2','_1','_0',
       '_h3','_l3','_h2','_l2','_h1','_l1']],df['p_sl'],test_size=0.1,shuffle=False)
"""
#
print("X_train[0:20]¥n¥t",X_train[0:20])
print("--------------")
print("y_train[0:20]¥n¥t",y_train[0:20])
print("----------------------------")
print("X_test[0:20]¥n¥t",X_test[0:20])
print("--------------")
print("y_test[:]¥n",y_test[0:20])
print("----------------------------")
##print("y_test.count(1)= ",y_test[:].count(1))
print("len(df)= ",len(df))
print("df['p_sl'].value_counts()¥n¥t",df['p_sl'].value_counts())
print("df['p_by'].value_counts()¥n¥t",df['p_by'].value_counts())
print("-----------　----------　-----------")
#print(y_test[].value_counts())
###########
#　モデルのインスタンスを生成
clf=RandomForestClassifier(n_estimators=150)#＜中心値＞
#clf=RandomForestClassifier(n_estimators=300)#やや改善
#clf=SVC()
#　fit　学習
clf.fit(X_train,y_train)
#　predict　予測
y_pred=clf.predict(X_test)
#
print("******予測結果******")
print(y_pred[:])
##print("y_test.count(1)= ",y_test.count(1))
#print("y_pred.count(1)= ",y_pred.count(1))
#print("")
#　正解率の計算
#　方法１
print("正解率；計算方法１では、{:.3f}　です。 ".format(accuracy_score(y_test, y_pred)))
print("----------------------------")
#　方法２
#print("正解率；計算方法２では、{:.3f}　です。 ".format(np.mean(y_pred==y_test)))
```

・ランダム・フォレストの
クラス分類用を使います。
・・詳しい事は理解できていない
ので、略デフォルトで使用。

学習と推論（予測）を実施

```
# 正解と予測値を DataFrame にまとめる
# 予測データと正解データ（ラベル）の比較
result = pd.DataFrame({"true":y_test,"pred":y_pred})
## ランダムの index　を「0」からに直す
result=result.reset_index()
print(result[0:20])
print("----------------------")
#
print("result['pred'].value_counts()¥n¥t",result['pred'].value_counts())
print("result['true'].value_counts()¥n¥t",result['true'].value_counts())
#print("len(df)= ",len(df))
print("len(result)= ",len(result))

count_=0
for true_ in result['true']:
    #print("true:",true_)
    if true_==1:
        count_=count_+1
    else:
        pass
#
print("true_==1 の数は、",count_)
#########----------##########
print("----------------------")
count_OK=0
count_NG=0
for pred_,true_ in zip(result['pred'],result['true']):
    if pred_==1 and true_==1:
        count_OK=count_OK+1
    elif pred_==1 and true_!=1:
        count_NG=count_NG+1
    else:
        pass
#
print("[pred==1 and true==1] の数は、",count_OK)
print("[pred==1 and true!=1] の数は、",count_NG)
print("++++++++++++++++++++++++++++++++++++++++")
#
print("")
print("------- k_10_test.py 終了----------")
```

用語不統一状態なので解説；
・「予測データ」＝「予測値」、
「正解データ」＝「ラベル」のこと
※つまり、「予測（推論）データ」と
「正解（ラベル）」とを一対一で
対比したデータフレームを作り、表示する。

「利確」数をカウント

テスト用データで予測を行った結果の内、
下記の組合せ数をカウントしています。
①「予測」が利確で、「ラベル」も利確
②「予測」は利確だが、「ラベル」は損切かドロー

上記のカウント結果を表示

特記；上記の例は、
・「買い」からマーケット IN する場合のプログラムです、
・「売り」からマーケット IN する場合には対応していません。
実は、本稿執筆時点では「売り IN」用のパラメータ設定までは
手が回らなかった！、のです。

解説 1-1；　「scikit-learn」を「時系列データ」に適用する場合に特有な内容について、など焦点を絞って解説します。

「教師データ」と「テスト・データ」への分割コード；

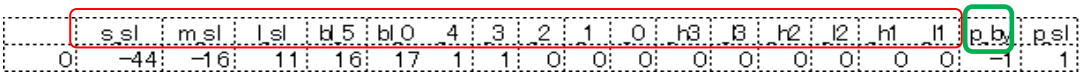
```
X_train,X_test,y_train,y_test=train_test_split(df[['s_sl','m_sl','l_sl','bl_5','bl_0',
    '_4','_3','_2','_1','_0',
    '_h3','_l3','_h2','_l2','_h1','_l1']],df['p_by'],test_size=0.1,shuffle=False,stratify=None)
```

このコードで実行している内容は下記です。

①特徴量（説明変数）として、何（どの［列］；s_sl～_l1）を設定するかの指定
②目的変数として、何（どの［列］；p_by、p_sl）を設定するかの指定
③教師データとテストデータの分割比を決定
④時系列データを扱うための指定

①特徴量（説明変数）として、何を設定するかの指定；

・DataFrame（scikit-learn による機械学習のテスト対象）から指定します。



	s_sl	m_sl	l_sl	bl_5	bl_0	_4	_3	_2	_1	_0	_h3	_l3	_h2	_l2	_h1	_l1	p_by	p_sl
0	-44	-16	11	16	17	1	1	0	0	0	0	0	0	0	0	0	-1	1

df[['s_sl','m_sl','l_sl','bl_5','bl_0','_4','_3','_2','_1','_0','_h3','_l3','_h2','_l2','_h1','_l1']　の解説

・赤枠「s_sl」～「_l1」まで全てを特徴量（説明変数）として使うと言う宣言です、
・df[['s_sl','m_sl','l_sl', '_h1']] とすれば、この指定した「s_sl、m_sl、l_sl、_h1」のみを特徴量として使うことになります。（任意に指定することが可能です）

②目的変数（ラベル）として、何を設定するかの指定・・本コードでの指定例を解説

df['p_by']　の解説

・緑枠の「p_by」を目的変数（本稿では分類予測の対象）に設定する宣言です。
その内容は「**買い**からマーケットに入った時」の結果（ラベル）です。
・一方、df['p_sl']と指定すれば「売りからマーケットに入った時」の結果が、
目的変数となります。（どちらを目的変数とするかは任意です）

③教師データとテストデータの分割

test_size=0.1

・全データのうち、学習データ（教師用データ）に 90%を、テスト用データに 10%（=0. 1）を割り当てています。

④時系列データを扱うための指定

shuffle=False、stratify=None

・時系列データを扱うので、データのシャッフル（デフォルト）はしない（False）とし同時にセットで「stratify=None」も指定します。（stratify の意味は未だ理解出来ず）

（２）「k2_10_test.py」コード

・読みやすいように筆者のメモ「Python コメント部分」を一部のみ「緑色」にしました。
※また、「k_10_test.py」と特に異なる部分のみにマーキングしました。

```
# -*- coding: utf-8 -*-
"""
[k2_10_test.py]
@author: kenken
"""
from sklearn.ensemble import RandomForestClassifier
####from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
#from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
#import numpy as np
import pandas as pd
# 表示範囲の設定
pd.options.display.max_rows = 100# 60--->100　にした
pd.set_option('display.expand_frame_repr', False)# 全ての列を折りかえさずに表示するために入れた
# 機械学習データを読込み
df=pd.read_pickle('k2_09.pkl')
#df=pd.read_pickle('fx_data_target.pkl')
#----------------------------------------
print(df[0:20])
#
"""
DataFrame　の列
s_sl,m_sl,l_sl,bl_5,bl_0,_4,_3,_2,_1,_0,_h3,_l3,_h2,_l2,_h1,_l1,p_by,p_sl
"""
#　トレーニングデータとテストデータに分離する
#　何種類かを試してみる
"""
#### 「買いＩＮ」用には特徴量（説明変数）の再調整が必要！！
X_train,X_test,y_train,y_test=train_test_split(df[['s_sl','m_sl','l_sl','bl_5','bl_0',
      '_4','_3','_2','_1','_0',
        '_h3','_l3','_h2','_l2','_h1','_l1']],df['p_sl'],test_size=0.1,shuffle=False,stratify=None)
"""
### 「売りＩＮ」用に特徴量（説明変数）を設定している
X_train,X_test,y_train,y_test=train_test_split(df[['s_sl','m_sl','l_sl','bl_5','bl_0',
      '_4','_3','_2','_1','_0',
      '_h3','_l3','_h2','_l2','_h1','_l1']],df['p_sl'],test_size=0.2,shuffle=False,stratify=None)
"""
時系列データを扱う場合は、「shuffle=False,stratify=None」をセットで指定する事！
stratify は「層化抽出法」のこと
"""
print("X_train[0:20]¥n¥t",X_train[0:20])
```

「DataFrame；k2_09.pkl」の読込み
・これの作成方法は「別稿（第２部）」で詳細解説しています。

(22 頁)「解説 1-2」を参照

```
print("--------------")
print("y_train[0:20]¥n¥t",y_train[0:20])
print("-----------------------------")
print("X_test[0:20]¥n¥t",X_test[0:20])
print("--------------")
print("y_test[:]¥n",y_test[0:20])
print("-----------------------------")
##print("y_test.count(1)= ",y_test[:].count(1))
print("len(df)= ",len(df))
print("df['p_sl'].value_counts()¥n¥t",df['p_sl'].value_counts())
print("df['p_by'].value_counts()¥n¥t",df['p_by'].value_counts())
print("-----------  ----------  -----------")
#print(y_test[].value_counts())
###########
#   モデルのインスタンスを生成
clf=RandomForestClassifier(n_estimators=150)
#clf=SVC()
#   fit   学習
clf.fit(X_train,y_train)
#   predict   予測
y_pred=clf.predict(X_test)
#
print("******予測結果******")
print(y_pred[:])
##print("y_test.count(1)= ",y_test.count(1))
#print("y_pred.count(1)= ",y_pred.count(1))
#print("")
#   正解率の計算
#   方法１
print("正解率；計算方法１では、{:.3f}   です。 ".format(accuracy_score(y_test, y_pred)))
print("----------------------------")
#   方法２
#print("正解率；計算方法２では、{:.3f}   です。 ".format(np.mean(y_pred==y_test)))
# 正解と予測値を DataFrame にまとめる
# 予測データと正解データの比較
result = pd.DataFrame({"true":y_test,"pred":y_pred})
## ランダムの index   を「0」からに直す
result=result.reset_index()
print(result[0:20])
print("----------------------")
#
print("result['pred'].value_counts()¥n¥t",result['pred'].value_counts())
print("result['true'].value_counts()¥n¥t",result['true'].value_counts())
#print("len(df)= ",len(df))
print("len(result)= ",len(result))

count_=0
for true_ in result['true']:
```

```
        #print("true:",true_)
        if true_==1:
            count_=count_+1
        else:
            pass
#
print("true_==1 の数は、",count_)
#########----------##########
print("----------")
count_OK=0
count_NG=0
for pred_,true_ in zip(result['pred'],result['true']):
        #print("true:",true_)
      #if pred_==-1 and true_==-1:
        if pred_==1 and true_==1:
            count_OK=count_OK+1
        elif pred_==1 and true_!=1:
            count_NG=count_NG+1
        else:
            pass
#print("[pred=-1 and true=-1] の数は、",count_)
print("[pred==1 and true==1] の数は、",count_OK)
print("[pred==1 and true!=1] の数は、",count_NG)
print("++++++++++++++++++++++++++++++++++++++")
################################################################################
```

解説 1-2；　「解説 1-1」の場合と異なる部分を重点に解説します

「教師データ」と「テスト・データ」への分割コード；

```
X_train,X_test,y_train,y_test=train_test_split(df[['s_sl','m_sl','l_sl','bl_5','bl_0',
    '_4','_3','_2','_1','_0',
    '_h3','_l3','_h2','_l2','_h1','_l1']],df['p_sl'],test_size=0.2,shuffle=False,stratify=None)
```

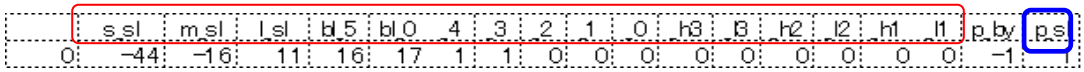
このコードで実行している内容は下記です。

①特徴量（説明変数）として、何（どの［列］; s_sl～_l1）を設定するかの指定
②目的変数として、何（どの［列］; p_by、p_sl）を設定するかの指定
③教師データとテストデータの分割比を決定
④時系列データを扱うための指定

以下、「k_10_test.py」と、一部「**青書**」部が異なりますが、その他は同じです。

①特徴量（説明変数）として、何を設定するかの指定；

・DataFrame（scikit-learn による機械学習のテスト対象）から指定します。



	s_sl	m_sl	l_sl	bl_5	bl_0	_4	_3	_2	_1	_0	_h3	_l3	_h2	_l2	_h1	_l1	p_by	p_sl
0	-44	-16	11	16	17	1	1	0	0	0	0	0	0	0	0	0	-1	-1

df[['s_sl','m_sl','l_sl','bl_5','bl_0','_4','_3','_2','_1','_0','_h3','_l3','_h2','_l2','_h1','_l1']] の解説

・赤枠「s_sl」～「_l1」まで全てを特徴量（説明変数）として使うと言う宣言です、

・df[['s_sl','m_sl','l_sl', '_h1']] とすれば、この指定した「s_sl、m_sl、l_sl、_h1」のみを特徴量として使うことになります。（任意に指定することが可能です）

②目的変数（ラベル）として、何を設定するかの指定

df['p_sl'] の解説

・青枠「p_sl」を目的変数（本稿では分類予測の対象）に設定する宣言です。
その内容は「**売り**からマーケットに入った時」の結果（ラベル）です。

・一方、df['p_by']と指定すれば「買いからマーケットに入った時」の結果が、目的変数となります。（どちらを目的変数とするかは任意です）

③教師データとテストデータの分割

test_size=0.2

・全データのうち、学習データ（教師用データ）に 80%を、テスト用データに 20%（=0. 2）を割り当てています。

④時系列データを扱うための指定

shuffle=False、stratify=None

・時系列データを扱うので、データのシャッフル（デフォルト）はしない（False）とし同時にセットで「stratify=None」も指定します。（stratify の意味は未だ理解出来ず）

補足解説；

＜時系列データを「学習データ」と「テストデータ」に分ける場合＞

・機械学習でデータを学習データとテストデータに分けるのに、
scikit learn フレームワークの train_test_split を使います。
この関数を「デフォルト設定のまま」で使うとデータの順番がシャッフルされた後に分割されます、通常の解析対象ではその方が良いのですが、『時系列データ』の場合は大問題です。

・従って、今回の FX チャート（時系列データ）を解析対象にする場合では、
「オプションで shuffle を False に設定」し、データの順番（チャートの形）を保持したままで、学習データとテストデータに分ける必要があります。

◇ さて、ここでクイズを 1 つ出します ◇

目的変数（ラベル）として、「 df['p_by']と df['p_sl'] 」の両方を選択すると、
推論（予測）結果は何を示すことになるのでしょう？？（読者が試してみて）

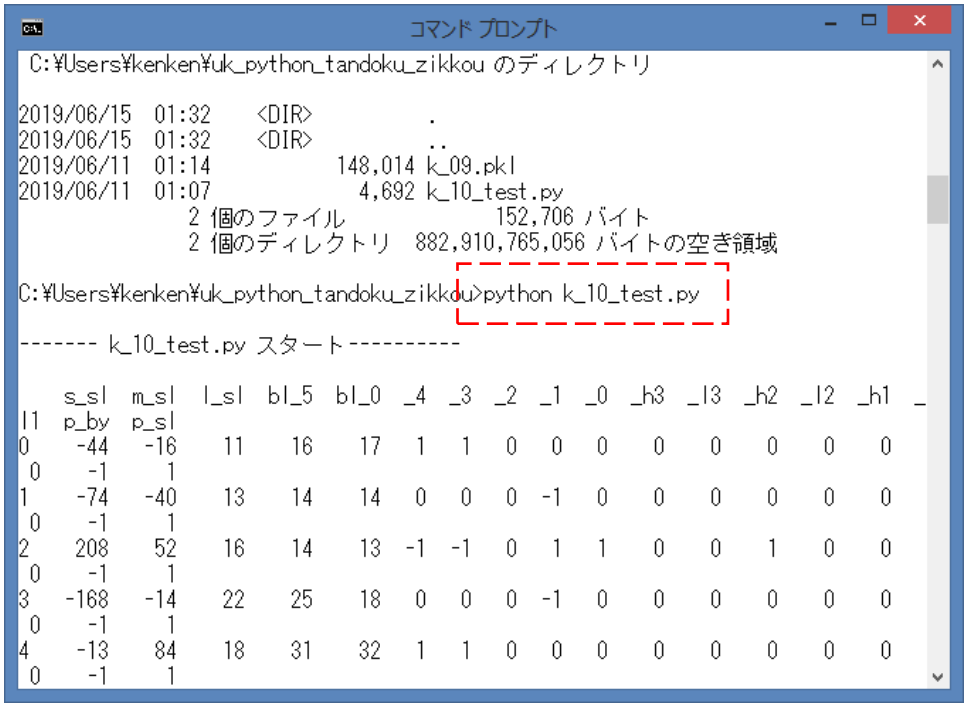
8．テストプログラム実行結果例

※今回だけ特別に、Spyder 環境（あるいは Jupiter Notebook など）が整備しておらず、
ただ Python が使えるだけの環境しか利用できない場合の実行方法も示しておきます。
本稿で扱う DataFrame は、別稿（第 2 部）の手順に従って「FX データ」から生成した
説明変数（特徴量）と目的変数（ラベル）から構成されています。

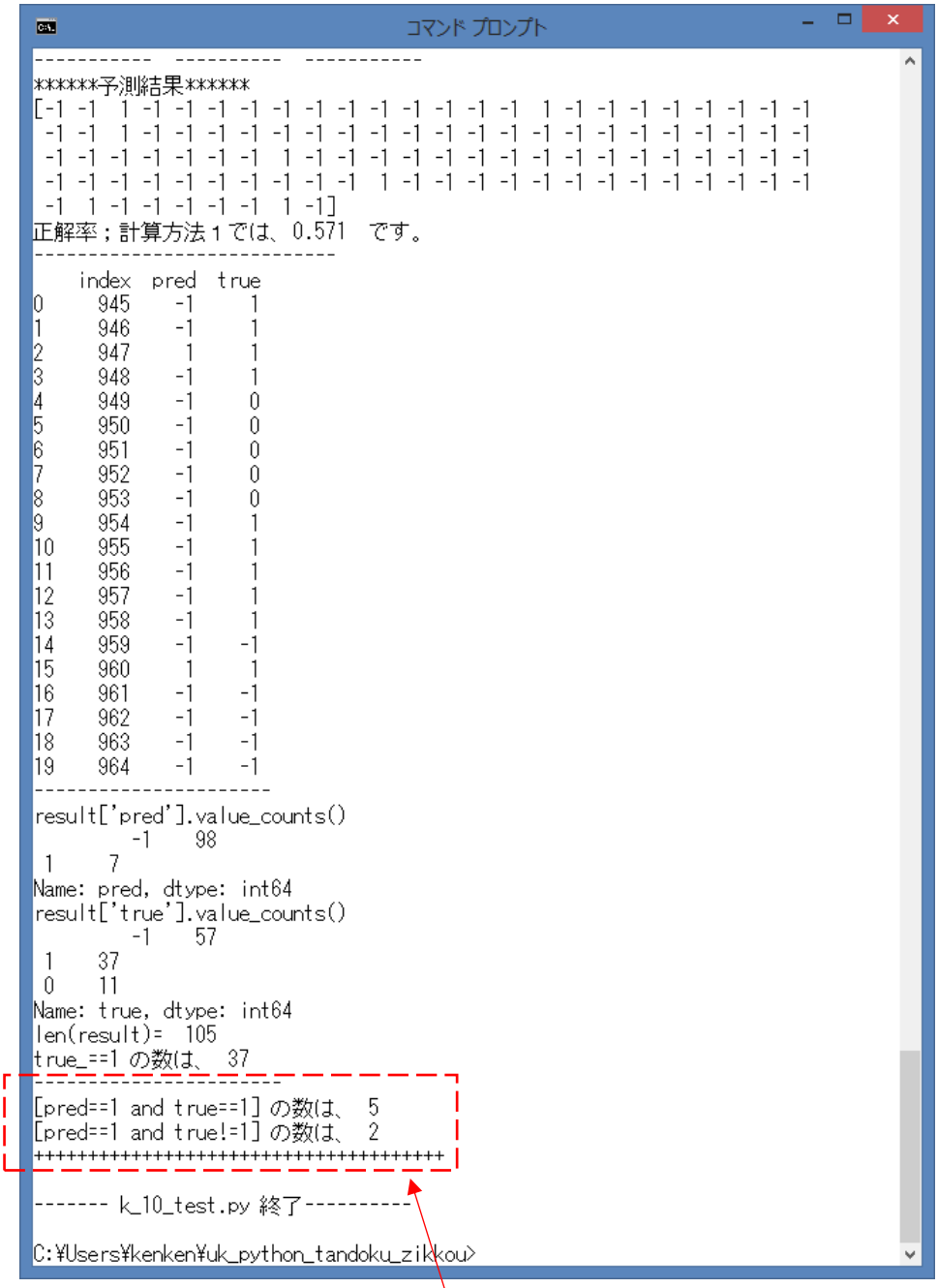
（１）Spyder または Jupiter Notebook 環境を立ち上げていない読者用の実行例

本稿では、「５．（１）」で示したコードを使用する場合で解説します。
準備品；
①「k_09.pkl」；　　DataFrame（データフレーム）
②「k_10_test.py」；　　学習・推論（勝敗予測）AI プログラム（Python）
・・① と ②は Python が実行できる環境下の同一ホルダーに入れておきます。
手順；
ステップ１；DOS 窓を開きます
ステップ２；cd コマンドで「①と②」を保存したフォルダーに入ります
ステップ３；スクリプトモードで、k_10_test.py を実行してください。
（　「Python k_10_test.py」とコマンドを入力　）

以下に筆者（アメンボ）が実行した例を示しておきます。
「Python k_10_test.py 」と入力



・・・・・　途中は省略　・・・・・



テスト結果；
・[勝敗予測が「利確」で、勝敗判定（ラベル）も「利確」]の回数＝5 回
・[勝敗予測が「利確」で、勝敗判定（ラベル）は「利確」以外]の回数＝2 回

◎ 勝敗予測が「利確」と出た時にのみ、トレードすることにすると
・勝率は「5/7＝0.71」であり、
・この時の１トレード当たりの期待利益は、
（「profit=0.32、loss=0.20」にレベル設定しているので）
0.32×（5/7）＋（-0.20）×（2/7）＝0.171 となります。

・・USD/JPY なので、約 17 銭（約 17 pips）となります

※結果は、プログラムを実行する度に異なります。

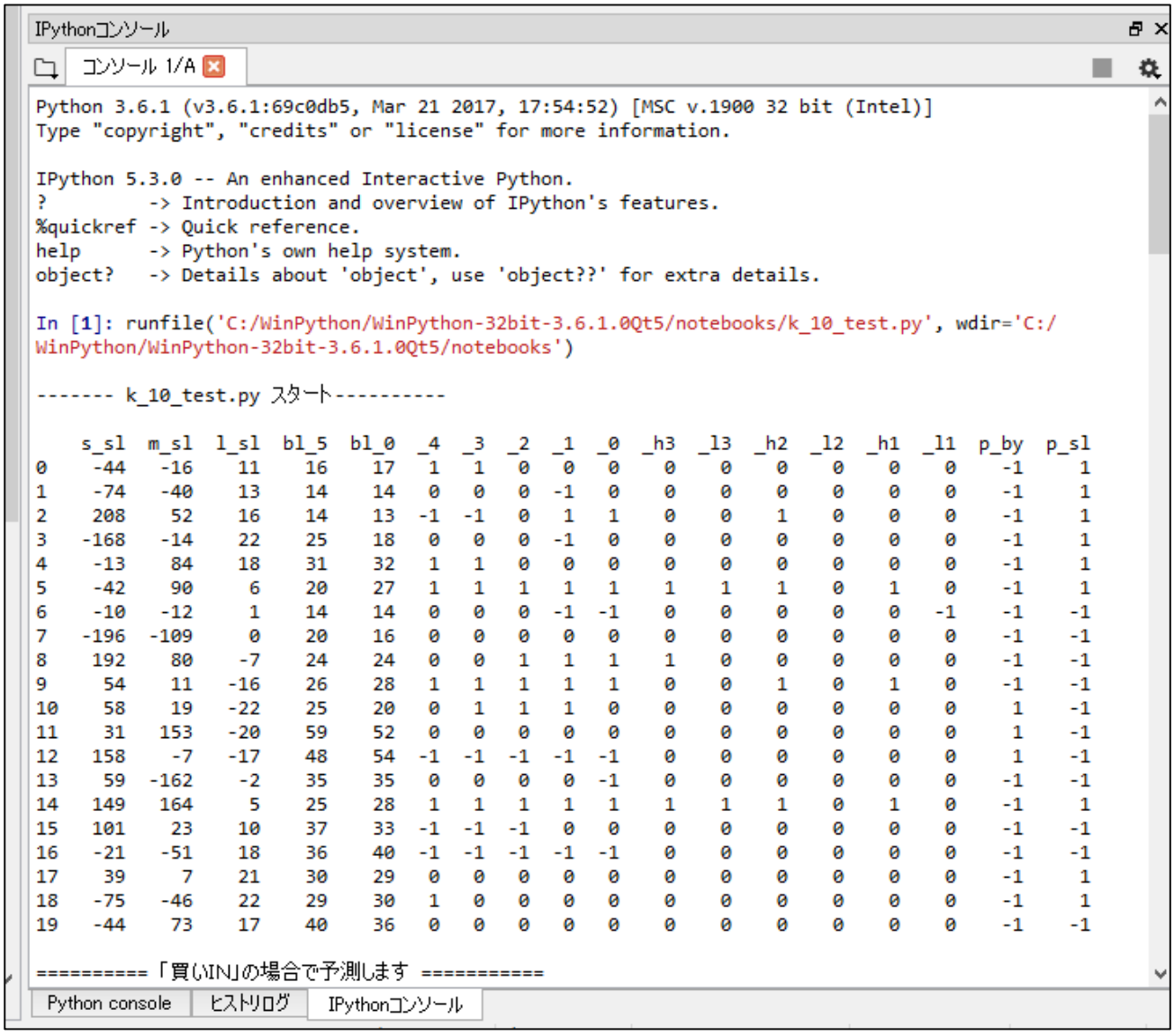
（２）Spyder 環境下での実行例

※「k_09.pkl と k_10_test.py」、および「k2_09.pkl と k2_10_test.py」は
Spyder の［ファイル］ー［開く］でアクセスできる「フォルダー」内に保存済みとします。

①「k_10_test.py」の実行例

・「USDJPY5_2019.01.10.csv」データから生成した「DataFrame；k_09.pkl」を
使用して学習と勝敗予測を行います。本節では一部のポイントのみを解説します。

＜Spyder の IPython コンソールへの出力・表示例＞



IPythonコンソール

コンソール 1/A

```
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit (Intel)]
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: runfile('C:/WinPython/WinPython-32bit-3.6.1.0Qt5/notebooks/k_10_test.py', wdir='C:/
WinPython/WinPython-32bit-3.6.1.0Qt5/notebooks')

------- k_10_test.py スタート----------

    s_sl  m_sl  l_sl  bl_5  bl_0  _4  _3  _2  _1  _0  _h3  _l3  _h2  _l2  _h1  _l1  p_by  p_sl
0    -44   -16    11    16    17   1   1   0   0   0    0    0    0    0    0    0    -1     1
1    -74   -40    13    14    14   0   0   0  -1   0    0    0    0    0    0    0    -1     1
2    208    52    16    14    13  -1  -1   0   1   1    0    0    1    0    0    0    -1     1
3   -168   -14    22    25    18   0   0   0  -1   0    0    0    0    0    0    0    -1     1
4    -13    84    18    31    32   1   1   0   0   0    0    0    0    0    0    0    -1     1
5    -42    90     6    20    27   1   1   1   1   1    1    1    1    0    1    0    -1     1
6    -10   -12     1    14    14   0   0   0  -1  -1    0    0    0    0    0   -1    -1    -1
7   -196  -109     0    20    16   0   0   0   0   0    0    0    0    0    0    0    -1    -1
8    192    80    -7    24    24   0   0   1   1   1    1    0    0    0    0    0    -1    -1
9     54    11   -16    26    28   1   1   1   1   1    0    0    1    0    1    0    -1    -1
10    58    19   -22    25    20   0   1   1   1   0    0    0    0    0    0    0     1    -1
11    31   153   -20    59    52   0   0   0   0   0    0    0    0    0    0    0     1    -1
12   158    -7   -17    48    54  -1  -1  -1  -1  -1    0    0    0    0    0    0     1    -1
13    59  -162    -2    35    35   0   0   0   0  -1    0    0    0    0    0    0    -1    -1
14   149   164     5    25    28   1   1   1   1   1    1    1    1    0    1    0    -1     1
15   101    23    10    37    33  -1  -1  -1   0   0    0    0    0    0    0    0    -1    -1
16   -21   -51    18    36    40  -1  -1  -1  -1  -1    0    0    0    0    0    0    -1    -1
17    39     7    21    30    29   0   0   0   0   0    0    0    0    0    0    0    -1     1
18   -75   -46    22    29    30   1   0   0   0   0    0    0    0    0    0    0    -1     1
19   -44    73    17    40    36   0   0   0   0   0    0    0    0    0    0    0    -1    -1

========== 「買いIN」の場合で予測します ===========
```

Python console　ヒストリログ　IPythonコンソール

＜上記「Spyder の出力・表示内容」の詳細＞

```
------- k_10_test.py スタート----------

      s_sl   m_sl   l_sl   bl_5   bl_0   _4   _3   _2   _1   _0   _h3   _l3   _h2   _l2   _h1   _l1   p_by   p_sl
0      -44    -16     11     16     17    1    1    0    0    0     0     0     0     0     0     0     -1      1
1      -74    -40     13     14     14    0    0    0   -1    0     0     0     0     0     0     0     -1      1
2      208     52     16     14     13   -1   -1    0    1    1     0     0     1     0     0     0     -1      1
3     -168    -14     22     25     18    0    0    0   -1    0     0     0     0     0     0     0     -1      1
4      -13     84     18     31     32    1    1    0    0    0     0     0     0     0     0     0     -1      1
5      -42     90      6     20     27    1    1    1    1    1     1     1     1     0     1     0     -1      1
6      -10    -12      1     14     14    0    0    0   -1   -1     0     0     0     0     0    -1     -1     -1
7     -196   -109      0     20     16    0    0    0    0    0     0     0     0     0     0     0     -1     -1
8      192     80     -7     24     24    0    0    1    1    1     1     0     0     0     0     0     -1     -1
9       54     11    -16     26     28    1    1    1    1    1     0     0     1     0     1     0     -1     -1
10      58     19    -22     25     20    0    1    1    1    0     0     0     0     0     0     0      1     -1
11      31    153    -20     59     52    0    0    0    0    0     0     0     0     0     0     0      1     -1
12     158     -7    -17     48     54   -1   -1   -1   -1   -1     0     0     0     0     0     0      1     -1
13      59   -162     -2     35     35    0    0    0    0   -1     0     0     0     0     0     0     -1     -1
14     149    164      5     25     28    1    1    1    1    1     1     1     1     0     1     0     -1      1
15     101     23     10     37     33   -1   -1   -1    0    0     0     0     0     0     0     0     -1     -1
16     -21    -51     18     36     40   -1   -1   -1   -1   -1     0     0     0     0     0     0     -1     -1
17      39      7     21     30     29    0    0    0    0    0     0     0     0     0     0     0     -1      1
18     -75    -46     22     29     30    1    0    0    0    0     0     0     0     0     0     0     -1      1
19     -44     73     17     40     36    0    0    0    0    0     0     0     0     0     0     0     -1     -1

========== 「買いＩＮ」の場合で予測します ==========

X_train[0:20]
      s_sl   m_sl   l_sl   bl_5   bl_0   _4   _3   _2   _1   _0   _h3   _l3   _h2   _l2   _h1   _l1
0      -44    -16     11     16     17    1    1    0    0    0     0     0     0     0     0     0
1      -74    -40     13     14     14    0    0    0   -1    0     0     0     0     0     0     0
2      208     52     16     14     13   -1   -1    0    1    1     0     0     1     0     0     0
3     -168    -14     22     25     18    0    0    0   -1    0     0     0     0     0     0     0
4      -13     84     18     31     32    1    1    0    0    0     0     0     0     0     0     0
5      -42     90      6     20     27    1    1    1    1    1     1     1     1     0     1     0
6      -10    -12      1     14     14    0    0    0   -1   -1     0     0     0     0     0    -1
7     -196   -109      0     20     16    0    0    0    0    0     0     0     0     0     0     0
8      192     80     -7     24     24    0    0    1    1    1     1     0     0     0     0     0
9       54     11    -16     26     28    1    1    1    1    1     0     0     1     0     1     0
10      58     19    -22     25     20    0    1    1    1    0     0     0     0     0     0     0
11      31    153    -20     59     52    0    0    0    0    0     0     0     0     0     0     0
12     158     -7    -17     48     54   -1   -1   -1   -1   -1     0     0     0     0     0     0
13      59   -162     -2     35     35    0    0    0    0   -1     0     0     0     0     0     0
14     149    164      5     25     28    1    1    1    1    1     1     1     1     0     1     0
15     101     23     10     37     33   -1   -1   -1    0    0     0     0     0     0     0     0
16     -21    -51     18     36     40   -1   -1   -1   -1   -1     0     0     0     0     0     0
17      39      7     21     30     29    0    0    0    0    0     0     0     0     0     0     0
18     -75    -46     22     29     30    1    0    0    0    0     0     0     0     0     0     0
19     -44     73     17     40     36    0    0    0    0    0     0     0     0     0     0     0
--------------
y_train[0:20]
    0   -1
1   -1
2   -1
3   -1
4   -1
5   -1
6   -1
7   -1
8   -1
9   -1
10   1
11   1
12   1
```

```
13    -1
14    -1
15    -1
16    -1
17    -1
18    -1
19    -1
Name: p_by, dtype: int64
------------------------------
X_test[0:20]
                s_sl  m_sl  l_sl  bl_5  bl_0  _4  _3  _2  _1  _0  _h3  _l3  _h2  _l2  _h1  _l1
945    -46    -96     12    18    18    0    0    0    0    0     0     0     0     0     0
946    149     20     16    15    14    0    0    0    0    1     0     0     0     0     1     0
947   -115    -25     21    12    14   -1   -1   -1   -1   -1     0    -1     0    -1     0    -1
948    -82    -21     23    16    15    0    0    0    0   -1     0     0     0     0     0     0
949      0    -72     24    17    18    0    0    0    0    1     0     0     0     0     0     0
950    -27    137     21    15    13    1    1    1    1    1     1     0     1     0     1     1
951     60     49     15    29    24    0    0    0    1    0     0     0     0     0     0     0
952     31     86     -3    30    34    1    1    1    1    1     0     0     0     0     0     0
953    -18     54    -19    16    19    1    1    1    1    1     1     0     1     1     1     0
954     10     17    -27    10     9    0    0    0    0    1     0     0     0     0     1     0
955     15     26    -31    11    11    0    1    1    0    0     0     0     0     0     0     0
956     67     38    -35    23    23    0    0    0    0    0     0     0     0     0     0     0
957     69     68    -34    21    21    1    1    1    1    1     0     0     1     0     1     0
958    192    113    -27    75    66    0    0    0    0    0     0     0     0     0     0     0
959    -67     40    -16    72    76   -1   -1   -1   -1   -1     0     0     0     0     0     0
960   -264   -465     -4    20    42   -2   -1   -2   -2   -2    -1    -2    -1    -2    -1    -2
961   -159     -3     -3    14    16    1    1    1    0   -1     0     0     0     0     0     0
962    -43     52     -7    17    14    0    0    0    0   -1     0     0     0     0     0    -1
963    125     -6     -9    17    18   -1   -1    0   -1    0     0     0     0     0     0     0
964    -11    -43    -10    19    18    0    0    0    0    0     0     0     0     0     0     0
--------------
y_test[:]
 945     1
946     1
947     1
948     1
949     0
950     0
951     0
952     0
953     0
954     1
955     1
956     1
957     1
958     1
959    -1
960     1
961    -1
962    -1
963    -1
964    -1
Name: p_by, dtype: int64
------------------------------
len(df)=  1050
df['p_sl'].value_counts()
          -1     602
 1      372
 0       76
Name: p_sl, dtype: int64
df['p_by'].value_counts()
          -1     680
 1      277
 0       93
```

「３章（１）勝敗の判定方法」を、
p_sl；「**売り**」から入った場合で FX5 分足全体に適用した結果
- 損切（-1）が、602 回
- 利確（1）が、372 回
- ドロー（0）が、76 回

「３章（１）勝敗の判定方法」を、
W；p_by；「**買い**」から入った場合で FX5 分足全体に適用した結果
- 損切（-1）が、680 回
- 利確（1）が、277 回
- ドロー（0）が、93 回

```
Name: p_by, dtype: int64
----------  ----------  ----------
******予測結果******
[-1 -1 -1 ... -1 -1 -1]
正解率：計算方法１では、0.552　です。
--------------------
      index   pred   true
0       945     -1      1
1       946     -1      1
2       947     -1      1
3       948      1      1
4       949     -1      0
5       950     -1      0
6       951     -1      0
7       952     -1      0
8       953     -1      0
9       954     -1      1
10      955     -1      1
11      956     -1      1
12      957     -1      1
13      958     -1      1
14      959     -1     -1
15      960      1      1
16      961      1     -1
17      962     -1     -1
18      963     -1     -1
19      964     -1     -1
--------------------
result['pred'].value_counts()
          -1      98
 1        6
 0        1
Name: pred, dtype: int64
result['true'].value_counts()
          -1      57
 1        37
 0        11
Name: true, dtype: int64
len(result)=   105
true_==1 の数は、 37
----------------------
[pred==1 and true==1] の数は、 4
[pred==1 and true!=1] の数は、 2
++++++++++++++++++++++++++++++++++++++++++

------- k_10_test.py 終了----------
```

テスト対象部分（FX チャートの最後の 10%部分）；
「買い」から入った場合の、
「予測結果」と勝敗判定内容「W（ラベル）」を
一対一で比較した例。
・「pred」＝予測（推論）結果
・「true」＝勝敗判定内容（ラベルの値）「W」

テスト対象部分（FX チャートの最後の 10%部分）に
対して勝敗予測（推論）「pred」した結果
・損切（-1）＝98 回
・利確（1）＝6 回
・ドロー（0）＝1 回

テスト対象部分（FX チャートの最後の 10%部分）の
「３章（１）勝敗の判定方法」による生成ラベル（true）内容
・損切（-1）＝57 回
・利確（1）＝37 回
・ドロー（0）＝11 回

勝率は「(4/6）＝0.66

テスト結果；　テスト対象部分（FX チャートの最後の 10%部分）で、
・[勝敗予測が「利確」で、勝敗判定（ラベル）も「利確」]の回数＝4 回
・[勝敗予測が「利確」で、勝敗判定（ラベル）は「利確」以外]の回数＝2 回

◎勝敗予測が「利確」と出た時にのみ、トレードすることにすると
・勝率は「4/6＝0.67」であり、
・この時の１トレード当たりの期待利益は、
（「profit=0.32、loss=0.20」にレベル設定しているので）
0.32×（4/6）＋（−0.20）×（2/6）＝0.147 となります。

・・USD/JPY なので、約 15 銭（約 15 pips）となります

※結果は、プログラムを実行する度に異なります。

＜期待利益が「＊＊」の意味について＞

例えば、1 トレード当たり「0.147 円（約 15 pips)」の利益が期待できるとは、
どういうことでしょう、どれくらい稼げるのでしょう？

イメージが掴めない読者もいるかも知れないので簡単に解説することにしました。

先ず前提条件と予備知識です；

- 為替は USD/JPY（ドル／円）を対象とします
- 1 ロットは「10,000 通貨（一般的な 1 万通貨)」とします
- 「1pip=0.0001 通貨単位」（1 万分の 1）です
- レバレッジは 25 倍（日本国内 FX 会社）とします

計算例；

簡単化のために「1 ドル＝100 円」で考察します。
この場合ですと、「ドル／日本」の 1pip は「(100／10000）円＝0.01 円＝1 銭」
ですので、最近の 100 円近辺にある「ドル／円」では、
1 ドル（＝100 円）あたり、『1pip≒1 銭』と考えて
概算することができます。

さて、「0.147（＝14.7 銭；14.7pips)」の期待利益と言う事は、
1 ロット（通常は 1 万通貨）でトレードしているときは、

① 必要資金は「10,000×100 円＝1,000,000 円（100 万円)」ですが、
25 倍のレバレッジを効かせると、証拠金は「100 万円／25＝4 万円」ですみ、

② 1 トレードあたりの利益は、
0.147 円 × 10,000 通貨＝ 1,470 円 となる、と言う事です。

◎ 4 万円の元手で、1 トレードあたり「1,470 円」の利益ですから、
「まあまあ！？」ですかね。(それとも少ない！ですか？)

読者は、期待利益の目標値を「何 pips」としますか？；

◎小生にとっては「1 トレードあたり「20 pips」の利益」、これが 1 つの目標です。
・・USD/JPY では 20 銭（20pips）で計算すると、期待利益は 2,000 円ですので、
4 万円の元手（証拠金）に対して「5%」の利益となります。

（目標値は「＊pips」よりも、元手の「＊%」とした方が正解かもしれません）

②「k2_10_test.py」の実行例

・「USDJPY5_2019.05.14.csv」データから生成した「DataFrame；k2_09.pkl」を使用して学習と勝敗予測を行います。「①」の場合と同様なので結果のみの記述とします。

```
     s_sl   m_sl   l_sl   bl_5   bl_0   _4   _3   _2   _1   _0   _h3   _l3   _h2   _l2   _h1   _l1   p_by   p_sl
0     -88      9    -12     18     16    0    0    0    0    0     0     0     0     0     0     0     -1      1
1     -68    -79    -11     14     19   -1   -1   -1   -1   -1     0    -1    -1    -1    -1    -1     -1      1
2      57      6     -3      8      8    0    0    0    0    0     0     0     0     0     0     0     -1      1
3     -29      3      3      8      7    1    1    1    0    0     0     0     0     0     0     0     -1      1
4       1     10      5     16     16    1    0    0    0    0     0     0     0     0     0     0     -1      1
5     -99     29      5     18     17    1    0    0    0    0     0     0     0     0     0     0     -1      1
6      25     79      3     37     27    0    0    0    0    0     0     0     0     0     0     0     -1      1
7    -129    -28      3     37     39   -1   -1   -1   -1   -1     0     0     0     0     0    -1     -1      1
8      17    -64      4     28     30   -1   -1   -1   -1   -1     0     0     0     0     0     0     -1      1
9      79    -14      3     46     38    0    0    1    1    0     0     0     0     0     0     0     -1      1
10    -26     39     -1     50     52    1    1    1    1    1     0     0     0     0     0     0     -1      1
11    -53    130     -9     25     34    1    1    1    1    1     1     1     1     1     1     1     -1     -1
12    203    -24    -15     25     22    0   -1    0    0    0     0     0     0     0     0     0      1     -1
13    -70   -110    -12     27     27   -1   -1   -1   -1   -1    -1    -1     0    -1     0    -1      1     -1
14   -137    -25     -5     24     26   -1   -1   -1   -1   -1     0     0     0    -1     0     0     -1     -1
15   -120    -46      0     14     17   -1   -1   -1   -1   -1     0    -1     0    -1     0    -1     -1     -1
16    127     -8      3     15     13    0    0    1    1    1     0     0     1     0     0     0     -1      1
17    -14     46      4     20     20    1    1    1    1    1     0     0     0     0     0     0     -1      1
18    -45    -20      5     24     20    0    0    0    0    0     0     0     0     0     0     0     -1     -1
19     79     50     10     45     39    0   -1    0    0    0     0     0     0     0     0     0     -1     -1
X_train[0:20]
     s_sl   m_sl   l_sl   bl_5   bl_0   _4   _3   _2   _1   _0   _h3   _l3   _h2   _l2   _h1   _l1
0     -88      9    -12     18     16    0    0    0    0    0     0     0     0     0     0     0
1     -68    -79    -11     14     19   -1   -1   -1   -1   -1     0    -1    -1    -1    -1    -1
2      57      6     -3      8      8    0    0    0    0    0     0     0     0     0     0     0
3     -29      3      3      8      7    1    1    1    0    0     0     0     0     0     0     0
4       1     10      5     16     16    1    0    0    0    0     0     0     0     0     0     0
5     -99     29      5     18     17    1    0    0    0    0     0     0     0     0     0     0
6      25     79      3     37     27    0    0    0    0    0     0     0     0     0     0     0
7    -129    -28      3     37     39   -1   -1   -1   -1   -1     0     0     0     0     0    -1
8      17    -64      4     28     30   -1   -1   -1   -1   -1     0     0     0     0     0     0
9      79    -14      3     46     38    0    0    1    1    0     0     0     0     0     0     0
10    -26     39     -1     50     52    1    1    1    1    1     0     0     0     0     0     0
11    -53    130     -9     25     34    1    1    1    1    1     1     1     1     1     1     1
12    203    -24    -15     25     22    0   -1    0    0    0     0     0     0     0     0     0
13    -70   -110    -12     27     27   -1   -1   -1   -1   -1    -1    -1     0    -1     0    -1
14   -137    -25     -5     24     26   -1   -1   -1   -1   -1     0     0     0    -1     0     0
15   -120    -46      0     14     17   -1   -1   -1   -1   -1     0    -1     0    -1     0    -1
16    127     -8      3     15     13    0    0    1    1    1     0     0     1     0     0     0
17    -14     46      4     20     20    1    1    1    1    1     0     0     0     0     0     0
18    -45    -20      5     24     20    0    0    0    0    0     0     0     0     0     0     0
19     79     50     10     45     39    0   -1    0    0    0     0     0     0     0     0     0
--------------
y_train[0:20]
      0      1
1     1
2     1
3     1
4     1
5     1
6     1
7     1
8     1
9     1
10    1
11   -1
```

```
12    -1
13    -1
14    -1
15    -1
16     1
17     1
18    -1
19    -1
Name: p_sl, dtype: int64
------------------------------
X_test[0:20]
                s_sl   m_sl   l_sl   bl_5   bl_0   _4   _3   _2   _1   _0   _h3   _l3   _h2   _l2   _h1   _l1
1680    -56    -37    -51     48     50     0    0    0    0    1     0     0     0     0     0     0
1681    -64     78    -54     69     52     0    0    0    0    0     0     0     0     0     0     0
1682    193   -190    -58    103    110    -1   -1   -1   -1   -1     0     0     0     0     0     0
1683    236   -260    -55     64     79    -1   -1   -1   -1   -1    -1    -1     0    -1     0     0
1684   -278    -82    -55     32     33     0   -1   -1   -1   -1     0     0     0    -1     0    -1
1685    401     -6    -61     29     32    -1   -1    0    0    0     0     0     0     0     0     0
1686   -236   -124    -58     20     18     0   -1   -1   -1   -1    -1    -1    -1    -1    -1    -1
1687     22     10    -56     32     28     0    0    0    0    0     0     0     0     0     0     0
1688   -114     -6    -49     29     28    -1   -1   -1   -1   -1     0     0     0     0     0     0
1689     76    -34    -36     53     38     0    0    0    0    0     0     0     0     0     0     0
1690     17      0    -18     79     72    -1   -1   -1    0    0     0     0     0     0     0     0
1691    219   -195      0     63     73    -1   -1   -1   -1   -1     0    -1     0     0     0    -1
1692    -94    -55     14     36     40    -1   -1   -1   -1   -1     0     0     0    -1     0    -1
1693    -43    -68     17     20     20    -1   -1   -1   -1   -1     0     0     0     0     0     0
1694   -130    -83     15     23     22     0    0    0    0    0     0     0     0     0     0     0
1695     94     93     10     28     28     0    0    1    0    1     0     0     0     0     0     0
1696    224    -15      5     36     32     0    0    0    0    0     0     0     0     0     0     0
1697    -86   -174      1     59     54     0    0    0    0    0     0     0     0     0     0     0
1698    -34    -19     -7     44     49     1    1    1    1    1     0     0     0     0     0     0
1699    376    108    -10     16     25     1    0    1    2    2     0     0     2     0     2     1
--------------
y_test[:]
 1680    -1
1681    -1
1682    -1
1683     1
1684     1
1685     1
1686     1
1687     1
1688     1
1689     1
1690     1
1691     1
1692     1
1693     1
1694     1
1695     1
1696    -1
1697     1
1698     1
1699    -1
Name: p_sl, dtype: int64
------------------------------
len(df)=  2100
df['p_sl'].value_counts()
           -1      1135
 1         571
 0         394
Name: p_sl, dtype: int64
df['p_by'].value_counts()
           -1      1233
 1         497
```

```
 0        370
Name: p_by, dtype: int64
----------   ----------   ----------
******予測結果******
[ 1   1   1 ... -1 -1 -1]
正解率；計算方法１では、0.531   です。
--------------------------
      index   pred   true
0      1680       1      -1
1      1681       1      -1
2      1682       1      -1
3      1683      -1       1
4      1684      -1       1
5      1685      -1       1
6      1686      -1       1
7      1687      -1       1
8      1688      -1       1
9      1689      -1       1
10     1690       1       1
11     1691      -1       1
12     1692      -1       1
13     1693      -1       1
14     1694      -1       1
15     1695       1       1
16     1696      -1      -1
17     1697       1       1
18     1698      -1       1
19     1699      -1      -1
---------------------
result['pred'].value_counts()
             -1      342
 1        50
 0        28
Name: pred, dtype: int64
result['true'].value_counts()
             -1      233
 1       135
 0        52
Name: true, dtype: int64
len(result)=   420
true_==1 の数は、  135
--------------------
[pred==1 and true==1] の数は、  23
[pred==1 and true!=1] の数は、  27
++++++++++++++++++++++++++++++++++++++++++++++++++
```

テスト結果； テスト対象部分（FX チャートの最後の 20%部分）で、
・[勝敗予測が「利確」で、勝敗判定（ラベル）も「利確」] の回数＝23 回
・[勝敗予測が「利確」で、勝敗判定（ラベル）は「利確」以外] の回数＝27 回

勝敗予測が「利確」と出た時にのみ、トレードすることにすると
・この時の 1 トレード当たりの期待利益は、
（「profit=0.32、loss=0.20」にレベル設定しているので）
0.32×（23/50）＋（−0.20）×（27/50）＝0.039 となります。
・・USD/JPY なので、約 4 銭（約 4 pips）となります

◆ 勝率「(23/50) ＝0. 46」と 50%以下でも、期待利益はプラスと言う訳です。

注；こちらの例では「説明変数」の調整を余りしていないので、
改善の余地が大きいはずです。
※結果は、プログラムを実行する度に異なります。

（3）結果のまとめ

※以下は Spyder3 環境下で実行した場合の例です。

①「k_09.pkl」と「k_10_test.py」の組合せ；

特徴量（説明変数）のチューニングをかなり深く実施したので、

・「利確」を予測する精度は上がりましたが、　⇒　0.67（67%）

・「利確」の予測件数はとても少なくなっています。　⇒　6 回

言い換えると、「過学習」状態なのかもしれません。

MT4 で言うなら過適応（最適化のやり過ぎ）、つまりチャートの傾向が変わったら通用しなくなる EA に相当するのか？？とも思ったのですが、AI の場合は因果関係の把握を含め、どの様に捉えるべきかを思案中です。

②「k2_09.pkl」と「k2_10_test.py」の組合せ

こちらは特徴量（説明変数）のチューニングは浅いままです、

殆ど時間をかけませんでした。

そのせいか？、

・「利確」を予測する精度は低いのですが、　⇒　0.46（46%）

・一方、「利確」の予測件数は結構あります。　⇒　23 回

①の場合より FX データ量が「2 倍」ある事を考慮しても「利確予測件数」が多い一方で、「予測精度」は落ちる、と言う事から「2 つの項目」はトレードオフの関係にあるかも知れないと考えています。（未確認です！）

（現状では「チューニング」はかなり試行錯誤状態で行っています）

□特記；「予測とラベル」の内容を、「k_09.pkl」と「k_10_test.py」の組合せの場合で比較（見直）してみると、

	内容	出現数；回
予測（推論）	損切（－1）	98
	利確（+1）	6
	ドロー（0）	1
真値（ラベル）	損切（－1）	57
	利確（+1）	37
	ドロー（0）	11

となり、「損切（－1）」の予測回数が圧倒的に多いことが判ります。

実は、計算すると直ぐ判るのですが、

［予測が損切（－1）で、かつ真値（ラベル）も損切（－1）］となる

組合せはとても多いため、単純に予測数全体の正解率を求めると「高い値」になります。

◎しかし、『損切（－1）予測』の正解率がいくら高くても、

トレードをする立場から観れば全く無意味です。（少なくとも EA 制作上では無価値）

※重要なのは『利確（+1）予測』の「正解率」と、その時の「期待利益」です。

（４）＜参考＞　アルゴリズムに「SVC(SVM)」を使用した場合の結果について

・当初、SVC(SVM)を使って「利確」「損切」の分類をしようと試してみましたが、
その結果は「？？なんじゃ、コリャー！！」でした。

※Kernel SVC は、カーネルを使用する SVM（サポートベクトル・マシン）に基づく
クラス分類手法とのことです。

※「k_10_test.py」でアルゴリズムに「SVC」を使った場合の結果（初期に実施した例）を
以下に例示します。実は、筆者はこのアルゴリズムを余り理解しておらず、ブラック・ボックス
状態で使用しました。（AI に関しては、筆者は初心者レベルです）

コード；

```
・・・・・途中略・・・・・
#　モデルのインスタンスを生成
clf=SVC()
#　fit　学習
clf.fit(X_train,y_train)
#　predict　予測
y_pred=clf.predict(X_test)
・・・・・途中略・・・・
```

結果；

```
・・・・・途中略・・・・・
result['pred'].value_counts()
-1      210
Name: pred, dtype: int64
result['true'].value_counts()
-1      138
1       52
0       20
・・・・・途中略・・・・・
```

•pred は予測値
•true はラベル（真値）

予測（推論）結果;
•「損切（-1）」＝210 回
（•「利確（+1）」＝0 回　なので表示されず）
（•「ドロー（0）」＝0 回なので表示されず）

ラベル（真値）;
•「損切（-1）」＝138 回
•「利確（+1）」＝52 回
•「ドロー（0）」＝20 回

◆なんと、予測結果が全て「－１」になると言う現象が発生し、いくら色々試しても
改善出来ませんでした。
でも、全体の正解率は「138／210」（66％）となる訳で、理解に苦しみました。

調査した結果、SVC(SVM) では特に「不均衡データ」を扱う場合、つまりクラス間で
データ数が大きく偏るような対象を扱う場合には、色々と事前の調整が必須であることを
知りました。（スケーリング、カテゴリ特徴量　等）

（注）不均衡データとは、例えば上記の様に「－1」クラスのデータ数が「138 個」で、
「＋1」クラスが「52 個」と言った様に、区別すべき「各クラス」に所属するデータ数に
クラス間で大きな差（偏り）がある場合です。
この様な場合に、SVC(SVM)を単純に適用すると、予測結果の殆どはデータ数が
多い方のクラス「－1」に分類されてしまうと言った現象が発生することを知りました。

○SVC(SVM)では特徴量（説明変数）の抽出の際は「スケーリング（正規化・正則化）」が重要だ、と言うことで、色々試したものの理解不足もあり、状況は一向に改善しませんでした。

⇒ ◇そこで、アルゴリズムを「RandomForest（ランダム・フォレスト）による分類」に変えてみました

この SVC(SVM) 検討時の「スケーリング」の痕跡が、アルゴリズムをランダム・フォレストに切り替えた後の「k_01_**.py～k_08_**.py」と「k2_01_**.py～k2_08_**.py」のコードの中に残っていますが、そのまま使用しています。（コードの詳細は「第 2 部」で解説）

（注）scikit-learn で使える「ランダム・フォレスト」には、
クラス分類用（RandomForestClassifier）と回帰分析用（RandomForestRegressor ）
の 2 種類がありますが、本稿では「クラス分類用」を使用しています。

（5）＜参考＞「SVC(SVM)、Randomforest、Deeplearning」の使い分け目安；

※下記に、3 手法の特徴と使い分けを、筆者流に大雑把にまとめてみました。

（ご注意）筆者は初心者のため、誤解している内容があるかもしれず、その場合は御指摘ください。

	SVC(SVM) サポート ベクター マシン	RandomForest ランダム フィレスト	Deeplearning ディープ ラーニング
基本原理 ポイント ワード	「一対他」分類法	決定木による分類	パターン認識による分類
	境界線からのマージン最大化	YES、NO 判断の繰り返し	ノードの重みとバイアス
特徴量	人間が特徴量を抽出・指定	人間が特徴量を抽出・指定	自動的に抽出する
過学習	陥りやすい	陥りにくい	特に陥りやすい
特徴量の正規化・標準化	必要	不要	必要
ハイパーパラメータ	必須のものが幾つかある	「決定木数」以外は不要	必須のものが幾つもある
特徴量の寄与度測定	（不明）	可能	不可
データの正則化・正規化	人間が実施	自動で処理	自動で処理
必要データ数	データが多いときは避ける （学習効率が良くないため）	データが多くても問題ない	大量のラベル付けされた データが必要
不均衡データ対応	対策が複雑	対策なしで可能	対策が必須
データ数と精度	データ増加と共に精度は飽和	データ増加と共に精度は飽和	データ増加と共に精度は向上
応用	分類と回帰	分類と回帰	分類と回帰
計算負荷	クラス数に依存	中	大

＜用語の簡易解説＞

・過学習；特定の学習データに最適になるように学習し過ぎること。
過学習により未知のデータに対する誤差（汎化誤差）が逆に上がってしまう。

・正則化；極端なデータは削除する（重みを 0 にする）こと、要は「異常値」は省く。

・正規化；各入力データの平均を「0」、分散を「1」となる様に調整すること

・ハイパーパラメータ；アルゴリズムの挙動を制御するパラメータのこと、データからの学習では決まらない。（例；ニューラルネットワークの層数など、）

・不均衡データ；99%（－クラス）と 1%（＋1 クラス）のように，各クラスのデータ数に大きく差が開いている場合を示す。そのままの状態で SVM を構築すると，多くの場合、全て 「－1」 に分類されてしまうことになります。

９．本稿「第１部」と別稿「第２部」について

◎本来の開発手順（記載順）は、「特徴量とラベル」の抽出・計算から始まって、
「学習と推論（予測）」へと進むわけで、下図に示す順番に解説するのが順当なのですが、
敢て「DatFrame サンプルありき」でいきなり「学習・推論（予測）」の実行から始めました。
・理由は、結果から解説した方が読者の興味を惹けると考えたからです。（違っていたらご容赦）

本来の開発手順；

・FX 為替データから特徴量とラベルを抽出・計算し、DataFrame にまとめる。（別稿「第２部」の記載内容）	⇒	・別稿（第２部）の DataFrame を使って、「学習・推論（予測）」を実行する。（本稿「第１部」の記載内容）

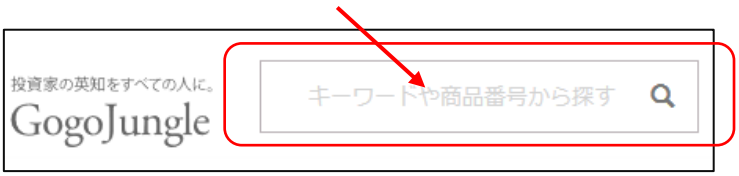
（１）別稿「第２部」は有償版と致しました（筆者が解析を続けるｲﾝｾﾝﾃｨﾌﾞとしてご了解ください）
※別稿では、本稿（第１部）で扱った「特徴量とラベル」の DataFrame を導出する
具体的な方法・過程を解説しています。
筆者は、FX（為替）チャートから有望な「特徴量」を選び出す方法として、
MT4(MQL4)を利用していますので、その例もやや詳しく解説しました。

◆別稿「第２部」（PDF 版とﾃﾞｰﾀ一式）は有料（1,600 円）でゴゴジャンにて販売しています
https://www.gogojungle.co.jp/tools
（ゴゴジャンの FX 関連「電子本・アルゴリズム」の販売サイト）

「第２部」書籍名は、　アメンボ式 AI 活用法「第２部」　　です。

商品を見つけ難い場合は、画面上部の検索窓で
「アメンボ式 AI 活用法「第２部」」と入力してください。

修正中



（２）本稿「第１部」・・無料版です
※「本稿 PDF 版とデーター式（DataFrame 等）」は下記の２か所に置いてあります。
（①と②は全く同一内容で、かつ無料品です）

①本稿 PDF　　；「http://mql4.s1002.xrea.com/ai_tec/ai_tec_01.pdf」
データー式（.zip）　；「http://mql4.s1002.xrea.com/ai_tec/ai_tec_01.zip」

②本稿 PDF とﾃﾞｰﾀ一式；「https://www.gogojungle.co.jp/tools」

殆どの読者は「①」からダウンロードしたと思いますが、「②」からも下記の名称で
入手可能です。
「　アメンボ式 AI 活用法「第１部」　」

（3）「第２部」の内容について、若干の補足

第２部では、第１部（本稿）で使用した DataFrame の作成方法を詳細に解説します。

※最後に別稿「第２部」の目次と、収録プログラムのリストを掲載しておきます。

＜第２部；目次＞

＜第２部；収録プログラム＞

「k_**.py」プログラム・セット；「USDJPY5_2019.01.10.csv」データ解析用です。

項番	プログラム名	処理内容
1	k_01_filter_slope.py	①松葉型フィルターの勾配を測定
2	k_02_bollin_wide.py	②ボリンジャー・バンドの幅を測定
3	k_03_bollin_and_chart.py	③各足とボリンジャー・バンドの位置関係を測定
4	k_04_profit_and_loss.py	④期間内（300 足）での「利確、損切、ﾄﾞﾛｰ」を判定
5	k_05_filter_classed.py	「①」を少しスケーリングしたものに変換しています
6	k_06_bollin_classed.py	「②」を少しスケーリングしたものに変換しています
7	k_07_chart_classed.py	「③」を少しスケーリングしたものに変換しています
8	k_08_profit_classed.py	「④」を少しスケーリングしたものに変換しています
9	k_09_nan_drop.py	生成された DataFrame から欠損データがある行を削除する
10	k_10_test.py	DataFrame を学習用と試験用に分割、学習後の予測結果を検証
10-2	k_10_test_and_disp.py	項番［10］の処理に加え、各特徴量の寄与度を棒グラフで表示する
11	k_11_kidou.py	項番［1］から［10］までを、順番に実行する
11-2	k_11_kidou_and_disp.py	項番［1］～［9］［10-2］の順に実行する

※「k2_**.py」プログラム・セット（「USDJPY5_2019.05.14.csv」データ解析用」
も添付していますが、基本的部分は「k_**.py」用と同じなので省略します。

１０．筆者プロフィールと、若干のヒント

※筆者は MT4、特に MQL4 による自動売買プログラムの開発に魅せられて、
その調査・研究・実績などの成果を下記のページで発表してきました。
・ここしばらくは、MQL4 の限界を打ち破るべく、AI の活用（連携）を模索しています。

＜成果発表例；いずれも無料版です＞

ホームページ１；MT4(MQL4)関係の記事
http://www.green.dti.ne.jp/sdimension/mql/
ホームページ２；AI と MT4 の連携方法について
http://mql4.s1002.xrea.com/ai_mql4/index.html

◎本稿「第１部」と別稿「第２部」は、その調査・研究の過程で試した内容の報告ですので、すぐにそのままでトレードに使えるものではなく、『プロトタイプ』であることをご了承ください。（実用的なものに仕上げられるかは、読者の努力次第です）

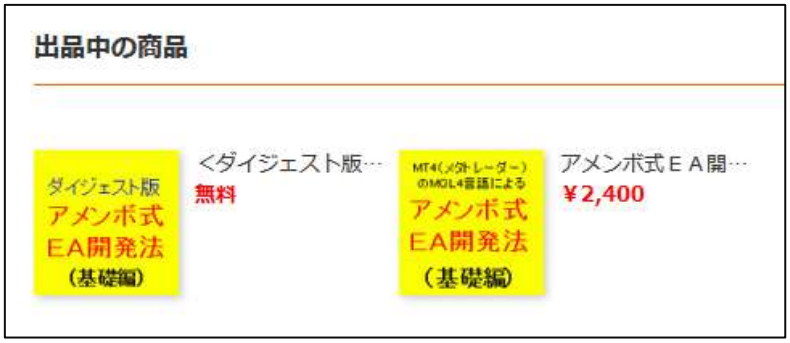
※テクニカル分析による自動売買は、MT4（MQL4）や MT5 で機能としては充分と考えます。
筆者は、テクニカル分析以外の機能を AI に求めているのですが、所期の機能を実現するには「外部情報と為替」間の「時間相関、影響度など」のデータ（経験則）を集めることに時間がかかり、現状では未だ見通しが立っていません。

そこで、時間のかかる分析作業の合間に「テクニカル分析」に AI を適用したらどうなるかを試すことにしました。しかし、これがやってみると、意外と大変（実に厄介）であるとともに、結果は結構面白いものでした。

＜宣伝＞

筆者は MT4(MQL4)関係の電子本（有料品と無料品）を「ゴゴジャン」で発行しています
電子本の販売；https://www.gogojungle.co.jp/tools
上記の検索窓で「アメンボ式 EA 開発法」と打ち込んでください、
⇒　２件がヒットします。
１つはダイジェスト版（無料品）で、もう一つの方が完全版（有料品）です。

・有料品には、MT4(MQL4)用のコード類とデータなど、実際に試すためのデータと資料の一式が含まれています。



◎いちいち、ゴゴジャンへの入口（販売サイ）から入るのが面倒な方は、
「https://www.gogojungle.co.jp/users/111402」へ直接アクセスしてください。
上記画面と同じ場所へジャンプします。

・上記の電子本（アメンボ式 EA 開発法）には、本稿で採用した「特徴量」の元になった「パラメータ」を用いて、MT4(MQL4)の EA を作成するための詳細な解説を掲載しています。
発行から若干時間が経っていますが、MT4(MQL4)は発行時点からそれほどバージョンアップがされていないので、記事の内容はそのまま使えます。

本電子本の内容は FX（為替）に限らず、仮想通貨や株などの「テクニカル分析」を行う場合にも参考になると考えます。

先ずは、無料の「ダイジェスト版」から読んで頂ければ幸いです。

追記；　最近は、スマホでも G_mail をチェックするようにしています。
何かありましたら下記へ、
アメンボの連絡先は「　amenbo.k@gmail.com　」です。

以　上