

株式分析チュートリアル

<https://github.com/JapanExchangeGroup/J-Quants>

2021-01-29

目次

まえがき	1
1. はじめに	2
2. 財務諸表で株価の先行きを予測しよう	3
2.1. 概要	3
2.2. 実行環境および必要なライブラリ	6
2.3. データセットの説明	8
2.4. データセットの可視化	17
2.5. データセットの前処理	27
2.6. 特徴量の生成	30
2.7. バックテスト用のテストデータ作成	33
2.8. モデルの構築	34
2.9. モデルの推論	40
2.10. 予測結果に対する分析の道筋	42
2.11. モデルの評価	46
2.12. モデルの提出	51
3. ニュースでポートフォリオを構築しよう	63
3.1. 概要	63
4. tips集	64
4.1. 金融用語集	64
4.2. 東証マネ部	65
4.3. 可視化テクニック	65
4.4. 簡単な正規化の例	66
4.5. 交差検証	66
4.6. Colaboratory	67
4.7. 参考になるコンペ	68
4.8. 参考になる書籍	68
4.9. ファンダメンタル分析の活用方法	69
4.10. テクニカル分析の活用方法	71
4.11. ファクター分析の活用方法	71
4.12. 複数個のモデルの出力をアンサンブルするアプローチ	72
4.13. プライベート期間の性能の向上のために考慮すべきこと	73
4.14. 本コンペでは利用できないが、モデルを将来的に発展させるために検討する価値のある外部データ	73
4.15. モデルの再学習の運用について	74
5. J-QuantsAPI	76
5.1. 概要	76
5.2. APIの利用	76
5.3. 必要なパッケージのインポート	76
5.4. Refresh API	76
5.5. 共通で使用するメソッド	77
5.6. Stock List API	79
5.7. Price API	79
5.8. Stock Fin API	80
5.9. Stock Labels	81
5.10. News API	82
5.11. TDnet API	83
6. 参考文献	84
7. ライセンス	85

まえがき

- 更新履歴

2021-01-29: チュートリアルリリース

1. はじめに

証券市場では、長年、様々なデータや数学的手法を用いてさまざまな市場を分析したり、金融商品の組成や投資戦略の立案が行われてきました。以前はこのような分析を行うことができるのは、金融機関や機関投資家と呼ばれる大手の投資家に限られてきました。しかし近年では、個人の方にも、ITやデータを活用した金融市場の分析や取引が拡大しています。

日本取引所グループは、証券分野におけるデータ活用や人工知能の活用を発展させたいと考えています。日本においてもさまざまなデータの活用やデータサイエンティストの育成が推進されていますが、金融分野に特化したチュートリアルの作成やコンペティションはあまり行われていませんでした。本チュートリアルを学ぶことで、データサイエンスを活用した株価予測を行う際に、最低限必要な知識や実践方法を学ぶことができます。

本コンペティションは幅広い方にご参加いただけることを期待していますが、プログラミングの経験があり確率・統計の基礎など勉強された学生の方や他分野でのデータ分析経験をお持ちの社会人の方、金融分野での知見はあるがデータサイエンスについてこれから勉強をされたいと考えている社会人の方には特に楽しんでいただける内容となっています。

本コンペティションを通じて金融データやデータ分析について理解を深めていただき、ポートフォリオ分析や資産運用に活用いただきたいと考えています。また、データサイエンスを学ぶ学生の方々にとって、金融データを用いたデータ活用や人工知能活用に関する研究にも興味を持っていただきたいと考えています。

ハンズオンで使うソースコード類は以下のウェブページで公開しています。

チュートリアルドキュメント

<https://jpx-jquants-guide.com>

Jupyterノートブック

[https://github.com/
JapanExchangeGroup/J-Quants-
Tutorial/tree/main/handson/](https://github.com/JapanExchangeGroup/J-Quants-Tutorial/tree/main/handson/)

2. 財務諸表で株価の先行きを予測しよう

2.1. 概要

本節では、本コンペティションにおける問題の概要及び本コンペティションに参加することで得られる知見について説明します。

2.1.1. 本コンペティションの趣旨

データ分析や株式取引には興味はあるけど、きっかけがないという方に、投資にまつわるデータ・環境を提供し、個人投資家の皆様によるデータ利活用の可能性を試してもらいたいという想いでこのコンペは設計されております。

本コンペティションが、皆様にとって、新しいアイデアや学習意欲に繋がり、株式投資の面白さが発見できることを期待しております。

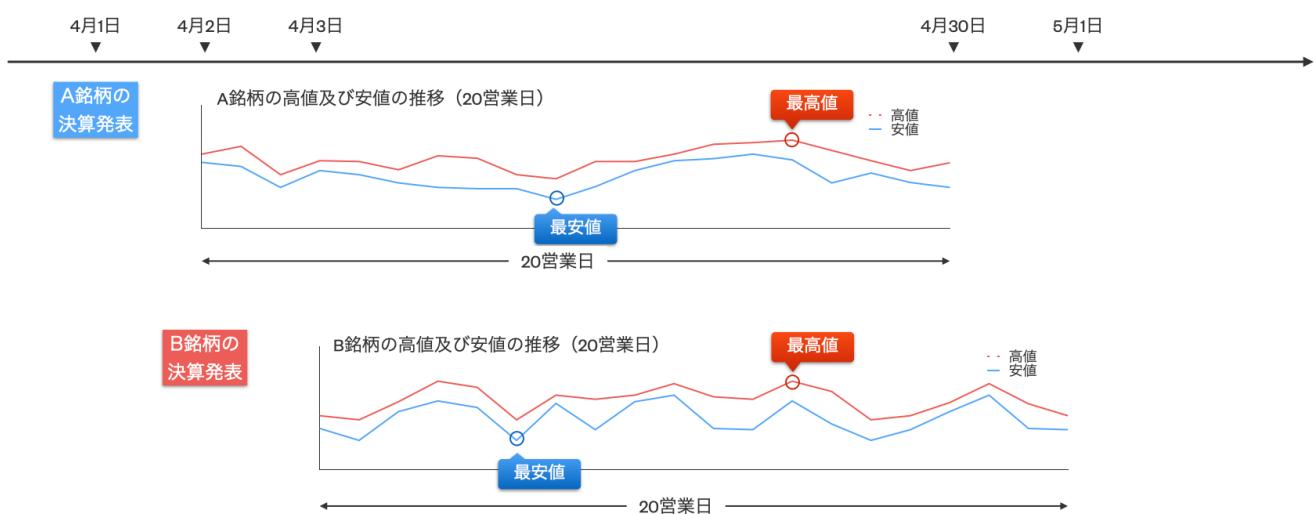
2.1.2. 課題概要

本コンペティションで取り組んでいただく課題は以下のとおりです。

課題

上場企業は1年を4期に分けて3カ月に一度、決算を発表しています。本コンペティションでは、各上場企業が決算情報を発表した後の20営業日の間における当該企業の最高値および最安値の予測について取り組んでいただきます。予測にあたっては、銘柄情報・株価情報・ファンダメンタル情報を用います。データセットの詳細については2.3をご参照ください。

課題イメージ



2. 財務諸表で株価の先行きを予測しよう

データ概要

ファイル名	説明
stock_list	各銘柄の情報が記録されたデータ
stock_price	各銘柄の株価情報(始値・高値・安値・終値等)が記録されたデータ
stock_fin	各銘柄のファンダメンタル情報(決算数値データや配当データ等)が記録されたデータ
stock_fin_price	データが扱いやすいようにstock_priceおよびstock_finをマージしたデータ
stock_labels	本コンペティションで学習に用いるラベル(目的変数)が記録されたデータ

提供データについては、2016年1月初から2020年12月末をcsvファイル形式、2021年1月初からのデータについては、本コンペティション専用のAPIにて提供いたします。APIによるデータ取得につきましては、5章をご参照ください。

スケジュール

日時	内容
2021年1月29日(金)	コンペティション開始
2021年3月28日(日)	モデル提出締切
2021年3月29日(月)～6月11日(金)	モデル評価期間
2021年7月頃	入賞者の決定

2.1.3. 本コンペティションに参加することで得られる知見

本コンペティションに参加することで、株価や企業業績の推移などの時系列データの解析手法や、さまざまな金融データを用いて市場の動向やリスクの分析についての知見が得られることを期待しています。

2.1.4. 評価方法

本コンペティションでは、モデルの予測と真の値（20営業日以内に発生する最高値及び最安値）との順位相関係数による定量評価方法を採用します。高値と安値の両方の順位相関係数を計算し、その平均で評価をするものとします。内部的にはpandasライブラリを利用し、`corr()` メソッドの引数 `method` に `spearman` を指定して計算しています。

$$\rho = 1 - \frac{6 \sum d^2}{n(n^2 - 1)}$$

d = 対応するXとYの値の順位の差

n = 値のペアの数

dのXとYは、

X = 該当期間の決算日に対して出力されたモデルのスコア

Y = 20営業日以内に発生した高値、もしくは安値

(式は一部英語Wikipediaスピアマンの順位相関係数より引用

https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient)

順位相関係数を採用する理由としては、以下の説明にもあるとおり、金融商品の価格変動の変化率の分布は必ずしも正規分布になるとは限りません。そのため、本コンペティションでは、特定の分布を仮定しない順位相関係数を採用しています。

一定間隔刻みで集計した騰落率の度数（頻度）分布が、騰落率の平均値を中心軸として左右対称の釣り鐘型の形状になる分布（正規分布=Normal Distribution）では、「平均値±標準偏差」の範囲に全データの約7割が収まるという確率的な特性を持ちます。ただし、金融商品の価格変動が厳密な意味での正規分布に従うことは実際上ほとんどありません。このため、「平均±標準偏差の範囲に騰落率の約7割が収まる」という考え方は理論的な目安に過ぎなく、発生確率は低いものの標準偏差を大幅に超す価格変動も起こります。こうした価格変動のリスクをテールリスクと呼び、とくに、金融市場の混乱期には分布がマイナス方向に偏るケースや、裾が極端に広く厚い"ファット・テール"という現象が確認できます。

(野村証券証券用語解説集より引用 <https://www.nomura.co.jp/terms/japan/hi/A02397.html>)

本コンペにおいても、例えば新型コロナウイルス感染症(COVID-19)のような外部影響を受け、マーケットの変化率の分布が歪む期間が存在すると想定されます。したがって、順位相関係数は相関係数と比較して特定の分布を仮定しないことから、本コンペティションにおいては、より適した評価方法であると考えられます。

順位相関係数は最高値と最安値の両方に対して個別に計算したうえ、以下の式を用いて結合スコアを最終スコアとします。

$$\text{score} = (P_{\text{high}} - 1)^2 + (P_{\text{low}} - 1)^2$$

P_high : 最高値の順位相関係数

P_low : 最安値の順位相関係数

2.2. 実行環境および必要なライブラリ

2.2.1. 実行環境

本チュートリアルの実行環境は、本コンペティションで提出するモデルの実行環境と同一環境とするために以下のpython環境を用います。環境構築方法について、詳しくは [SIGNATE: Runtime 投稿方法: ローカル開発環境の構築方法は？](#) をご参照ください。

anaconda3-2019.03

2.2.2. 必要なライブラリのインストール

本チュートリアル内では、上記の実行環境に含まれていないライブラリを使用するため、以下のコマンドを使用して個別にインストールします。

```
# shap用にg++とgccをインストールします
apt-get update
apt-get install -y --no-install-recommends g++ gcc

# 必要なライブラリをインストールします
pip install shap==0.37.0 slicer==0.0.3 xgboost==1.3.0.post0
```

2.2.3. ライブラリの読み込み

本チュートリアルでは、下記のライブラリのインポートを行います。

```

import os
import pickle
import sys
import warnings
from glob import glob

import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import shap
import xgboost
from scipy.stats import spearmanr
from sklearn.ensemble import (
    ExtraTreesRegressor,
    GradientBoostingRegressor,
    RandomForestRegressor,
)
from sklearn.metrics import accuracy_score, mean_squared_error
from tqdm.auto import tqdm

# 表示用の設定を変更します
%matplotlib inline
pd.options.display.max_rows = 100
pd.options.display.max_columns = 100
pd.options.display.width = 120

```

2.2.4. ライブライ解説

ライブラリ名	目的	公式ドキュメント	入門解説 q
pandas	データの処理	pandas documentation	Qiita:データ分析で頻出のPandas基本操作
numpy	データの処理	NumPy Tutorials	Qiita:numpyの使い方
glob	ファイルの検知	glob – Unix style pathname pattern expansion	Qiita:【備忘録】globの使い方
tqdm	計算の進捗確認	tqdm	Qiita:tqdmでプログレスバーを表示させる
sklearn	機械学習モデルを作成	https://scikit-learn.org/stable/tutorial/index.html	Qiita:scikit-learn から学ぶ機械学習の手法の概要
matplotlib	データの可視化	matplotlib tutorials	Qiita:早く知っておきたかったmatplotlibの基礎知識、あるいは見た目の調整が渉るArtistの話

2. 財務諸表で株価の先行きを予測しよう

ライブラリ名	目的	公式ドキュメント	入門解説 q
scipy	統計用のライブラリ	SciPy Tutorial	千葉大: コンピュータ処理ドキュメント 11. scipyの基本と応用
seaborn	データの可視化	User guide and tutorial	Qiita: pythonで美しいグラフ描画 -seabornを使えばデータ分析と可視化が捲るその1
shap	SHAP分析	Welcome to the SHAP Documentation	Shapを用いた機械学習モデルの解釈説明
xgboost	機械学習モデル	XGBoost Documentation	XGBoost論文を丁寧に解説する(1)

2.2.5. 実行環境の確認

pythonのバージョンが3.7.3であることを確認します。

```
print(sys.version)
```

出力

```
3.7.3 (default, Mar 27 2019, 22:11:17)
[GCC 7.3.0]
```

2.3. データセットの説明

提供されるデータは以下の5種類です。

ファイル名	説明
stock_list	各銘柄の情報が記録されたデータ
stock_price	各銘柄の株価情報(始値・高値・安値・終値等)が記録されたデータ
stock_fin	各銘柄のファンダメンタル情報(決算数値データや配当データ等)が記録されたデータ
stock_fin_price	データが扱いやすいようにstock_priceおよびstock_finをマージしたデータ
stock_labels	本コンペティションで学習に用いるラベル(目的変数)が記録されたデータ

2.3.1. 銘柄情報: stock_list

stock_listは、銘柄の名前や業種区分などの基本情報が含まれています。発行済株式数は、会社が発行することをあらかじめ定款に定めている株式数（授権株式数）のうち、会社が既に発行した株式数のことです。発行済株式数と株価とかけ合わせて時価総額を計算することができます。時価総額は企業価値を評価する際に用いられる重要な指標です。業種区分情報は、マーケットにおける業種別の平均などを計算する時に役立つ情報です。33業種は証券コード協議会が定めており、17業種はTOPIX-17シリーズとして「投資利便性を考慮して17業種に再編したもの」(JPX東証33業種別株価指数・TOPIX-17シリーズファクトシートより引用 <https://www.jpx.co.jp/markets/indices/line-up/index.html>) です。

「業種」(JPX用語集より引用 <https://www.jpx.co.jp/glossary/ka/112.html>)

変数名	説明	型	例
prediction_target	予測対象銘柄	bool	True
Effective Date	銘柄情報の基準日	int64	20201030
Local Code	株式銘柄コード	int64	1301
Name (English)	銘柄名	object	KYOKUYO CO.,LTD.
Section/Products	市場・商品区分	object	First Section (Domestic)
33 Sector(Code)	銘柄の33業種区分(コード)	int64	50
33 Sector(name)	銘柄の33業種区分(名前)	object	Fishery, Agriculture and Forestry
17 Sector(Code)	銘柄の17業種区分(コード)	int64	1
17 Sector(name)	銘柄の17業種区分(名前)	object	FOODS
Size Code (New Index Series)	TOPIXニューインデックスシリーズ規模区分(コード)	object	7
Size (New Index Series)	TOPIXニューインデックスシリーズ規模区分	object	TOPIX Small 2
IssuedShareEquityQuote AccountingStandard	会計基準 単独:NonConsolidated、連結国内:ConsolidatedJP、連結SEC:ConsolidatedUS、連結IFRS:ConsolidatedIFRS	object	Consolidated JP
IssuedShareEquityQuote ModifyDate	更新日	object	2020/11/06

2. 財務諸表で株価の先行きを予測しよう

変数名	説明	型	例
IssuedShareEquityQuote	発行済株式数	int64	10928283
IssuedShare			

メモリ使用量: 380.3+ KB

(JPX東証上場銘柄一覧より引用 <https://www.jpx.co.jp/markets/statistics-equities/misc/01.html>)
 (Quick xignite API Market Data API Catalogより引用 <https://www.marketdata-cloud.quick-co.jp/Products/>)

prediction	target	Effective Date	Local Code	Name (English)	Section/Products	33 Sector(Code)	33 Sector(name)	17 Sector(Code)	17 Sector(name)	Size Code (New Index Series)	Size (New Index Series)	IssuedShareEquityQuote	AccountingStandard	IssuedShareEquityQuote	ModifyDate	IssuedShareEquityQuote	IssuedShare
0	True	20201030	1301	KYOKUYO CO.,LTD.	First Section (Domestic)	50	Fishery, Agriculture and Forestry	1	FOODS	7	TOPX Small 2	ConsolidatedJP	2020/11/06	10928283			
1	True	20201030	1332	Nippon Suisan Kaihatsu,Ltd.	First Section (Domestic)	50	Fishery, Agriculture and Forestry	1	FOODS	4	TOPX Mid400	ConsolidatedJP	2020/11/05	312430277			
2	True	20201030	1333	Marubu Nihon Corporation	First Section (Domestic)	50	Fishery, Agriculture and Forestry	1	FOODS	4	TOPX Mid400	ConsolidatedJP	2020/11/02	52656910			
3	True	20201030	1352	HOKUSUI CORPORATION	First Section (Domestic)	6030	Wholesale Trade	13	COMMERCIAL & WHOLESALE TRADE	7	TOPX Small 2	ConsolidatedJP	2020/10/30	8379000			
4	True	20201030	1375	YUKIGUNI MAITAKE CO.,LTD.	First Section (Domestic)	50	Fishery, Agriculture and Forestry	1	FOODS	7	TOPX Small 2	ConsolidatedFRS	2020/11/05	39850000			

2.3.2. 株価情報 : stock_price

stock_priceには各銘柄の各日付の始値や終値などの株価情報が記録されています。テクニカル分析などで終値ベースの分析を実施する場合は、ExchangeOfficialCloseを利用します。

ここでいうテクニカル分析というのは、マーケットデータから計算される指標に基づいた分析のことです。また、終値ベースの分析とは、マーケットデータの中でも、終値のみを用いた分析を表しています。

株価情報は、「株式分割」や「株式併合」が発生した際に生じる株価の変動を、株式数の変化率に応じて調整されています。特徴量の定義によっては、その日付時点で実際に取引された株価や出来高を取得したい場合がありますが、その場合は累積調整係数を使用して

[調整前株価] = [調整済株価] * [累積調整係数] 及び [調整前出来高] = [調整済出来高] / [累積調整係数]

という計算で算出可能です。

「株式分割」(JPX用語集より引用 <https://www.jpx.co.jp/glossary/ka/81.html>)

「株式併合」(JPX用語集より引用 <https://www.jpx.co.jp/glossary/ka/83.html>)

ただし、これらの特徴量をモデルに使用する場合には注意が必要です。

履歴データの累積調整係数は過去のある日時点では知り得ない未来の情報を含んでいることに注意する必要があります。具体的には、過去のある日時点の累積調整係数が2である場合、その日以降に1:2の株式分割が発生していることがわかります。

一般に株式分割は流動性向上を期待できるポジティブなイベントとみなされています。仮に、モデル学習時に累積調整係数をモデルへ入力し、モデルが累積調整係数が大きい銘柄は未来の株価が上がる傾向があるということを学習し、履歴データを使用したバックテストでは良い結果がでたとします。しかし、このモデルに最新データを入力して予測を出力した場合、その予測は期待する結果を得られない可能性があります。なぜなら、最新データの累積調整係数にはまだ発生していない未来の情報は含まれていないためです。

このように、その日付時点では取得できない未来の情報をモデルに入力することをリークといい、時系列データには累積調整係数のようにその日付時点では取得できない情報が含まれていることがあるため、リークにはとくに注意する必要があります。

変数名	説明	型	例
Local Code	銘柄コード	int64	1301
EndOfDayQuote Date	日付	object	2016/01/04
EndOfDayQuote Open	始値	float64	2800
EndOfDayQuote High	高値	float64	2820
EndOfDayQuote Low	安値	float64	2740
EndOfDayQuote Close	終値。大引け後にセットされる	float64	2750
EndOfDayQuote ExchangeOfficialClose	取引所公式終値。最終の特別気配または最終気配を含む終値	float64	2750
EndOfDayQuote Volume	売買高	float64	32000
EndOfDayQuote CumulativeAdjustmentFactor	累積調整係数	float64	0.1
EndOfDayQuote PreviousClose	前回の終値	float64	2770
EndOfDayQuote PreviousCloseDate	前回の終値が発生した日	object	2015/12/30
EndOfDayQuote PreviousExchangeOfficialClose	前回の取引所公式終値	float64	2770
EndOfDayQuote PreviousExchangeOfficialCloseDate	前回の取引所公式終値が発生した日	object	2015/12/30
EndOfDayQuote ChangeFromPreviousClose	騰落幅。前回終値と直近約定値の価格差	float64	-20
EndOfDayQuote PercentChangeFromPreviousClose	騰落率。前回終値からの直近約定値の上昇率または下落率	float64	-0.722
EndOfDayQuote VWAP	売買高加重平均価格(VWAP)	float64	2778.25

メモリ使用量: 515.8+ MB

(Quick xignite API Market Data API Catalogより引用 <https://www.marketdata-cloud.quick-co.jp/Products/>)

2. 財務諸表で株価の先行きを予測しよう

Local Code	EndOfDayQuote Date	EndOfDayQuote Open	EndOfDayQuote High	EndOfDayQuote Low	EndOfDayQuote Close	EndOfDayQuote ExchangeOfficialClose	EndOfDayQuote Volume	EndOfDayQuote CumulativeAdjustmentFactor	EndOfDayQuote PreviousClose	EndOfDayQuote PreviousCloseDate	EndOfDayQuote PreviousExchangeOfficialClose	EndOfDayQuote PreviousExchangeOfficialCloseDate
0	1301	2016/01/04	2800.0	2820.0	2740.0	2750.0	32000.0	0.1	2770.0	2015/12/30	2770.0	2015/12/30
1	1301	2016/01/05	2750.0	2750.0	2760.0	2760.0	20100.0	0.1	2760.0	2016/01/04	2760.0	2016/01/04
2	1301	2016/01/06	2760.0	2770.0	2740.0	2760.0	2760.0	0.1	2760.0	2016/01/05	2760.0	2016/01/05
3	1301	2016/01/07	2740.0	2760.0	2710.0	2710.0	31400.0	0.1	2760.0	2016/01/06	2760.0	2016/01/06
4	1301	2016/01/08	2700.0	2740.0	2690.0	2700.0	26200.0	0.1	2710.0	2016/01/07	2710.0	2016/01/07

2.3.3. ファンダメンタル情報: stock_fin

株式投資における ファンダメンタル情報 とは、対象銘柄の純資産といった財務状況や当期純利益といった業績状況を表す情報のことです。ファンダメンタル情報を用いて、各銘柄の成長性、収益性、安全性、割安度などの投資判断に活用することができます。ファンダメンタル情報を利用した解析は、さまざまな手法が考案されています。

ファンダメンタル情報のデータセットであるstock_finにおいて、いくつかの変数名は Forecast から始まっていますが、これらは各企業が来期の自社の業績・財務状況を予想したデータです。例えば、企業が来期の業績が厳しいことが予め分かっている場合には、予想として早めに開示するため、予想のデータも重要な可能性があります。

変数名	説明	型	例
base_date	日付	object	2016/01/04
Local Code	銘柄コード	int64	2753
Result_FinancialStatement AccountingStandard	会計基準 単独:NonConsolidated、連結国内 :ConsolidatedJP、連結SEC:ConsolidatedUS、連結IFRS:ConsolidatedIFRS	object	Consolidated JP
Result_FinancialStatement FiscalPeriodEnd	決算期	object	2015/12
Result_FinancialStatement ReportType	決算種別 第1四半期:Q1、中間決算:Q2、第3四半期:Q3、本決算:Annual	object	Q3
Result_FinancialStatement FiscalYear	決算年度。本決算の決算期末が属する年。	float64	2016
Result_FinancialStatement ModifyDate	更新日	object	2016/01/04
Result_FinancialStatement CompanyType	会社区分 一般事業会社:GB、銀行:BK、証券会社:SE、損保会社:IN ※上記に該当しない場合は空文字を設定してます。	object	GB
Result_FinancialStatement ChangeOfFiscalYearEnd	決算期変更フラグ 決算期変更あり:true、決算期変更なし:false	object	False
Result_FinancialStatement NetSales	売上高(単位:百万円) 会社区分によって項目名の読み替えを行います。銀行:経常収益、証券:営業収益、損保:経常収益 ※未開示の場合 空文字を設定してます。	float64	22354

変数名	説明	型	例
Result_FinancialStatement OperatingIncome	営業利益(単位:百万円) ※未開示の場合は空文字を設定します。	float64	2391
Result_FinancialStatement OrdinaryIncome	経常利益(単位:百万円) 会計基準が連結SECの場合は、項目名を「税引前利益」に読み替えます。※未開示の場合は空文字を設定します。	float64	2466
Result_FinancialStatement NetIncome	当期純利益(単位:百万円) ※未開示の場合は空文字を設定します。	float64	1645
Result_FinancialStatement TotalAssets	総資産(単位:百万円) ※未開示の場合は空文字を設定します。	float64	21251
Result_FinancialStatement NetAssets	純資産(単位:百万円) ※未開示の場合は空文字を設定します。	float64	16962
Result_FinancialStatement CashFlowsFromOperatingActivities	営業キャッシュフロー(単位:百万円) ※未開示の場合は空文字を設定します。	float64	12404
Result_FinancialStatement CashFlowsFromFinancingActivities	財務キャッシュフロー(単位:百万円) ※未開示の場合は空文字を設定します。	float64	-98
Result_FinancialStatement CashFlowsFromInvestingActivities	投資キャッシュフロー(単位:百万円) ※未開示の場合は空文字を設定します。	float64	-1307
Forecast_FinancialStatement AccountingStandard	予想: 会計基準 単独:NonConsolidated、連結国内:ConsolidatedJP、連結SEC:ConsolidatedUS、連結IFRS:ConsolidatedIFRS	object	ConsolidatedJP
Forecast_FinancialStatement FiscalPeriodEnd	来期予想情報: 決算期	object	2016/03
Forecast_FinancialStatement ReportType	来期予想情報: 決算種別 第1四半期:Q1、中間決算:Q2、第3四半期:Q3、本決算:Annual	object	Annual
Forecast_FinancialStatement FiscalYear	来期予想情報: 決算年度。本決算の決算期末が属する年。	float64	2016
Forecast_FinancialStatement ModifyDate	来期予想情報: 更新日	object	2016/01/04
Forecast_FinancialStatement CompanyType	来期予想情報: 会社区分 一般事業会社:GB、銀行:BK、証券会社:SE、損保会社:IN ※上記に該当しない場合は空文字を設定します。	object	GB

2. 財務諸表で株価の先行きを予測しよう

変数名	説明	型	例
Forecast_FinancialStatement ChangeOfFiscalYearEnd	来期予想情報: 決算期変更フラグ 決算期変更あり:true、決算期変更なし:false	object	False
Forecast_FinancialStatement NetSales	来期予想情報: 売上高(単位:百万円) 会社区分によって項目名の読み替えを行います。銀行:経常収益、証券:営業収益、損保:経常収益※未開示の場合は空文字を設定します。	float64	30500
Forecast_FinancialStatement OperatingIncome	来期予想情報: 営業利益(単位:百万円) ※未開示の場合は空文字を設定します。	float64	3110
Forecast_FinancialStatement OrdinaryIncome	来期予想情報: 経常利益(単位:百万円) 会計基準が連結SECの場合は、項目名を「税引前利益」に読み替えます。※未開示の場合は空文字を設定します。	float64	3200
Forecast_FinancialStatement NetIncome	来期予想情報: 当期純利益(単位:百万円) ※未開示の場合は空文字を設定します。	float64	2130
Result_Dividend FiscalPeriodEnd	配当情報: 決算期	object	2015/11
Result_Dividend ReportType	配当情報: 決算種別 第1四半期:Q1、中間決算:Q2、第3四半期:Q3、本決算:Annual	object	Annual
Result_Dividend FiscalYear	配当情報: 決算年度。本決算の決算期末が属する年。	float64	2015
Result_Dividend ModifyDate	配当情報: 更新日	object	2016/01/07
Result_Dividend RecordDate	配当情報: 配当基準日	object	2015/11/30
Result_Dividend DividendPayableDate	配当情報: 配当支払開始日 ※予想の場合は空文字を設定します。	object	2016/02/29
Result_Dividend QuarterlyDividendPerShare	配当情報: 一株当たり四半期配当金(単位:円) ※未開示の場合は空文字を設定します。	float64	8
Result_Dividend AnnualDividendPerShare	配当情報: 一株当たり年間配当金累計(単位:円)※未開示の場合は空文字を設定します。	float64	15
Forecast_Dividend FiscalPeriodEnd	予想配当情報: 決算期	object	2016/03
Forecast_Dividend ReportType	予想配当情報: 決算種別 第1四半期:Q1、中間決算:Q2、第3四半期:Q3、本決算:Annual	object	Annual

変数名	説明	型	例
Forecast_DividendFiscalYear	予想配当情報: 決算年度。本決算の決算期末が属する年。	float64	2016
Forecast_DividendModifyDate	予想配当情報: 更新日	object	2016/01/04
Forecast_DividendRecordDate	予想配当情報: 配当基準日	object	2016/03/31
Forecast_DividendQuarterlyDividendPerShare	予想配当情報: 一株当たり四半期配当金(単位:円) ※未開示の場合は空文字を設定します。	float64	45
Forecast_DividendAnnualDividendPerShare	予想配当情報: 一株当たり年間配当金累計(単位:円) ※未開示の場合は空文字を設定します。	float64	90

メモリ使用量: 27.5+ MB

(Quick xignite API Market Data API Catalogより引用 <https://www.marketdata-cloud.quick-co.jp/Products/>)

2.3.4. 財務諸表+株価情報: stock fin price

`stock_fin_price`は、株価情報である<<株価情報: `stock_price`, `stock_price`>>と財務諸表である<<ファンダメンタル情報: `stock_fin`, `stock_fin`>>をデータとして扱いやすいように結合したデータです。変数名や型などについては、同じであるため記載を省略しています。

また、データのサイズが1.8GB以上と非常に大きいため、必要に応じて活用していただきたいと思います。

2.3.5. 目的變數: stock labels

`stock_labels`は予測の目的変数のデータであり、各銘柄で決算発表が行われた日の取引所公式終値から、その日の翌営業日以降N営業日間における取引所公式終値の最高値および最安値への変化率を記録したデータです。各値の計算式は、 $\left(\left[\text{基準日付の翌日以降N営業日間における高値/安値}\right] / \left[\text{基準日付の終値}\right]\right) - 1$ です。

変数名	説明	型	例
base_date	基準日付	object	2016-01-04
Local Code	銘柄コード	int64	1301

2. 財務諸表で株価の先行きを予測しよう

変数名	説明	型	例
label_date_5	データの基準日から5営業日後の日付。 label_high_5算出に使用される終値範囲の基準日	object	2016-01-12
label_high_5	データの基準日から5営業日以内の最高値への変化率	float64	0.00364
label_low_5	データの基準日から5営業日以内の最安値への変化率	float64	-0.04
label_date_10	データの基準日から10営業日後の日付。 label_high_10算出に使用される終値範囲の基準日	object	2016-01-19
label_high_10	データの基準日から10営業日以内の最高値への変化率	float64	0.00364
label_low_10	データの基準日から10営業日以内の最安値への変化率	float64	-0.05455
label_date_20	データの基準日から20営業日後の日付。 label_high_20算出に使用される終値範囲の基準日	object	2016-02-02
label_high_20	データの基準日から20営業日以内の最高値への変化率	float64	0.00364
label_low_20	データの基準日から20営業日以内の最安値への変化率	float64	-0.08364

メモリ使用量: 354.6+ MB

	base_date	Local Code	label_date_5	label_high_5	label_low_5	label_date_10	label_high_10	label_low_10	label_date_20	label_high_20	label_low_20
0	2016-01-04	1301	2016-01-12	0.01091	-0.04000	2016-01-19	0.01091	-0.05455	2016-02-02	0.01091	-0.08727
1	2016-01-05	1301	2016-01-13	0.00362	-0.04348	2016-01-20	0.00362	-0.07609	2016-02-03	0.00362	-0.09058
2	2016-01-06	1301	2016-01-14	0.00000	-0.05072	2016-01-21	0.00000	-0.08696	2016-02-04	0.00362	-0.09058

2.3.6. データセットの読み込み

本コンペティション用に提供されているデータセットをダウンロードして、ファイルを解凍した場所を定義します。

```
# データセット保存先ディレクトリ（""の中身はご自身の環境に合わせて定義してください。）
dataset_dir="/path/to"
```

データを読み込みます。なお、本チュートリアルでは `stock_fin` 及び `stock_price` を使用するため、`stock_fin_price` は読み込まずに進めます。

```
# 読み込むファイルを定義します。
inputs = {
    "stock_list": f"{dataset_dir}/stock_list.csv.gz",
    "stock_price": f"{dataset_dir}/stock_price.csv.gz",
    "stock_fin": f"{dataset_dir}/stock_fin.csv.gz",
    # 本チュートリアルでは使用しないため、コメントアウトしています。
    # "stock_fin_price": f"{dataset_dir}/stock_fin_price.csv.gz",
    "stock_labels": f"{dataset_dir}/stock_labels.csv.gz",
}

# ファイルを読み込みます
dfs = {}
for k, v in inputs.items():
    print(k)
    dfs[k] = pd.read_csv(v)
```

読み込んだデータを確認します。

```
for k in inputs.keys():
    print(k)
    print(dfs[k].info())
    print(dfs[k].head(1).T)
```

2.4. データセットの可視化

データセットの各項目の特徴を把握することは、モデルを作成する上で重要な要素の1つです。一般にデータの特徴を把握するためには、各項目の意味を把握し、値の平均や標準偏差などの基本統計量を確認します。可視化もそういった特徴把握の手法の1つで、データをグラフなどで表現することで特性を直感的に理解できるようになります。

Chapter 2.3 データセットの説明では、本コンペで用いるデータセットについて説明しました。ここではそれらのデータを、`matplotlib` と `seaborn` を用いて可視化します。財務諸表、株価、移動平均、価格変化率、ヒストリカル・ボラティリティを個別で見て、最後に1つのグラフとしてまとめて可視化します。

2.4.1. 財務諸表

ファンダメンタル情報は項目が多いため、今回は、売上高、営業利益、純利益、純資産及びその決算期の間の関係について可視化します。サンプルとして、銘柄コード9984の「ソフトバンクグループ」を可視化します。

2. 財務諸表で株価の先行きを予測しよう

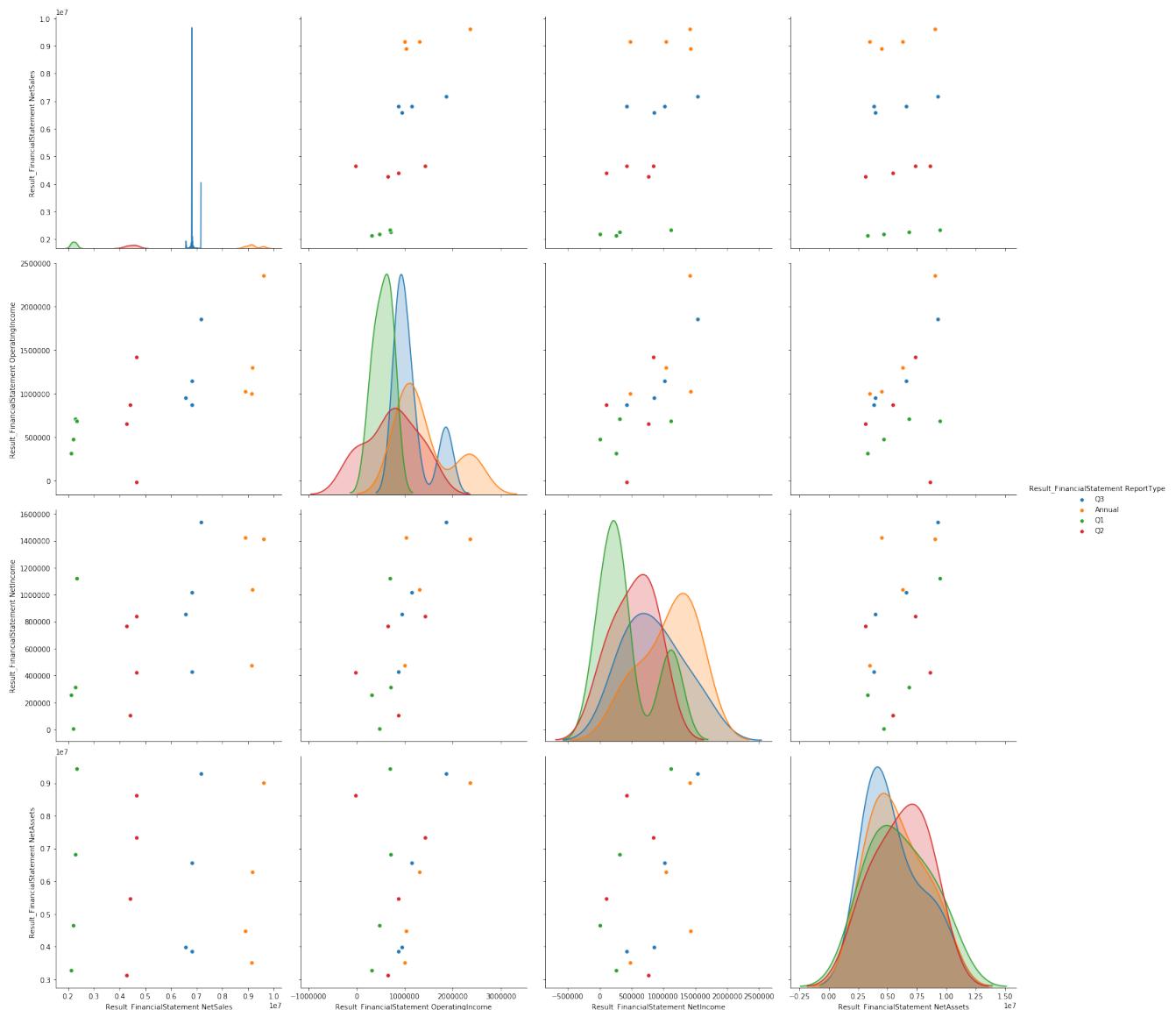
```
# stock_finの読み込み
fin = dfs["stock_fin"].copy()

# 銘柄コード9984にデータを絞る
code = 9984
fin_data = fin[fin["Local Code"] == code].copy()

# 日付列をpd.Timestamp型に変換してindexに設定
fin_data["datetime"] = pd.to_datetime(fin_data["base_date"])
fin_data.set_index("datetime", inplace=True)
# 2019年までの値を表示
fin_data = fin_data[: "2019"]

# プロット対象を定義
columns = [
    "Result_FinancialStatement NetSales", # 売上高
    "Result_FinancialStatement OperatingIncome", # 営業利益
    "Result_FinancialStatement NetIncome", # 純利益
    "Result_FinancialStatement NetAssets", # 純資産
    "Result_FinancialStatement ReportType" # 決算期
]

# プロット
sns.pairplot(fin_data[columns], hue="Result_FinancialStatement ReportType", height=5)
```



上記のプロットについて説明しますと、各色（緑、赤、青、オレンジ）はそれぞれQ1,Q2,Q3,Annualにおける決算の値に対応しており、対角に並んでいるプロットは各軸の特徴量の分布を表しています。また、その他のプロットは、各軸2つの変数の散布図を表しています。

例えば、2行1列目のグラフを見ると、横軸が売上高、縦軸が営業利益になっています。高い売上高は高い営業利益に繋がっています。また、決算期がQ1からQ3,本決算に至るまでに基本的に右肩上がりであることが分かります。このことから財務データの純売上高や営業利益などの変数は、各決算期ごとの値ではなく、各決算期を積み上げ式で記録されていると推測できます。

複数銘柄のファンダメンタル情報の比較

2. 財務諸表で株価の先行きを予測しよう

```
# stock_finの読み込み
fin = dfs["stock_fin"].copy()

# 銘柄コード9984と9983を比較する
codes = [9984, 9983]

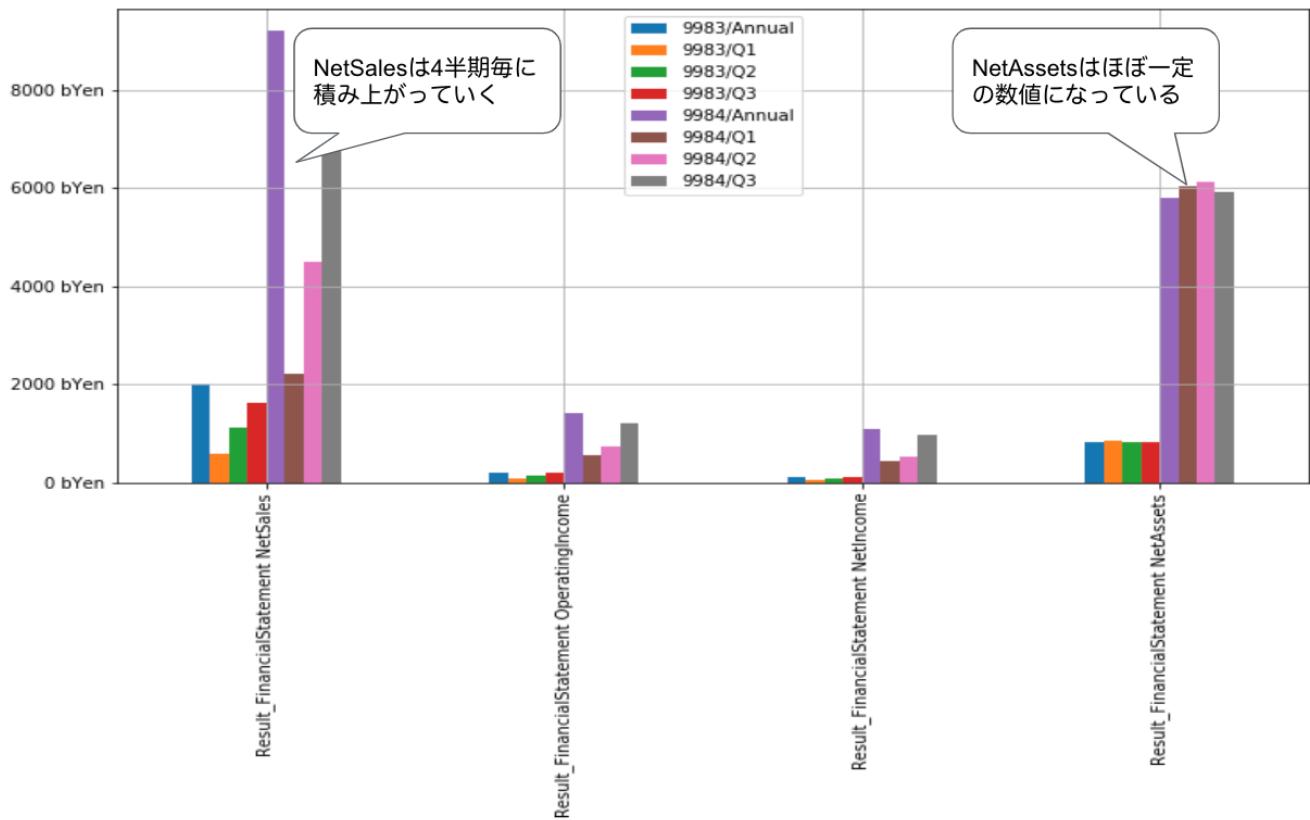
multi_df = dict()

# プロット対象を定義
columns = [
    "Result_FinancialStatement NetSales", # 売上高
    "Result_FinancialStatement OperatingIncome", # 営業利益
    "Result_FinancialStatement NetIncome", # 純利益
    "Result_FinancialStatement NetAssets", # 純資産
    "Result_FinancialStatement ReportType" # 決算期
]

# 比較対象の銘柄コード毎に処理
for code in codes:
    # 特定の銘柄コードに絞り込み
    fin_data = fin[fin["Local Code"] == code].copy()
    # 日付列をpd.Timestamp型に変換してindexに設定
    fin_data.loc[:, "datetime"] = pd.to_datetime(fin_data["base_date"])
    fin_data.set_index("datetime", inplace=True)
    # 2019年までの値を表示
    fin_data = fin_data[:"2019"]
    # 重複を排除
    fin_data.drop_duplicates(
        subset=[

            "Local Code",
            "Result_FinancialStatement FiscalYear",
            "Result_FinancialStatement ReportType"
        ],
        keep="last", inplace=True)
    # プロット対象のカラムを取得
    _fin_data = fin_data[columns]
    # 決算期毎の平均を取得
    multi_df[code] = _fin_data.groupby("Result_FinancialStatement ReportType").mean()

# 銘柄毎に処理していたものを結合
multi_df = pd.concat(multi_df)
# 凡例を調整
multi_df.set_index(multi_df.index.map(lambda t: f"{t[0]}/{t[1]}"), inplace=True)
# プロット
ax = multi_df.T.plot(kind="bar", figsize=(12, 6), grid=True)
# Y軸のラベルを調整
ax.get_yaxis().set_major_formatter(matplotlib.ticker.FuncFormatter(lambda x, p: "{} bYen".format(int(x / 1_000))))
```



画像の中にコメントを記載しておりますが、NetSales（売上高）とNetAssets（純資産）の特性の違いがわかります。NetAssetsは一定の数値になっており、決算期の影響をあまり受けていないことがわかります。一方、NetAssetsはQ1からAnnualにかけて数値が積み上がっており、Q1から決算期が進むごとに大きくなる特性を持つことがあります。

2.4.2. 株価

ここでは、サンプルとして銘柄コード9984の「ソフトバンクグループ」の終値の動きを可視化します。

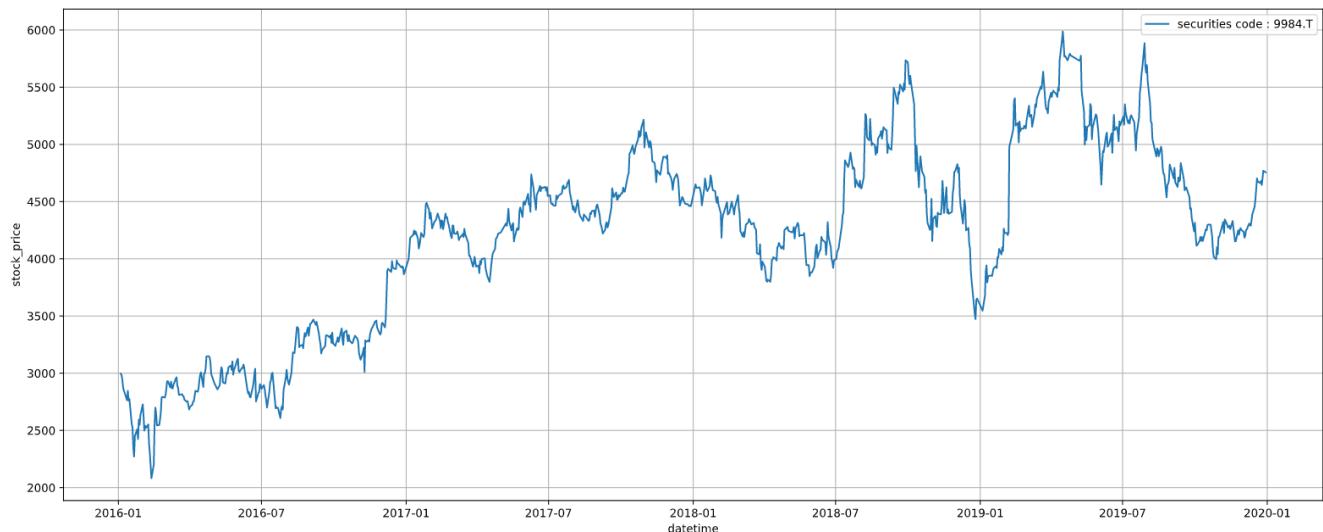
2. 財務諸表で株価の先行きを予測しよう

```
# stock_priceの読み込み
price = dfs["stock_price"].copy()

# 特定の銘柄コードに絞り込み
code = 9984
price_data = price[price["Local Code"] == code].copy()
# 日付列をpd.Timestamp型に変換してindexに設定
price_data["datetime"] = pd.to_datetime(price_data["EndOfDayQuote Date"])
price_data.set_index("datetime", inplace=True)
# 2019年までの値を表示
price_data = price_data[: "2019"]

# プロット
fig, ax = plt.subplots(figsize=(20, 8))

ax.plot(price_data["EndOfDayQuote ExchangeOfficialClose"], label=f"securities code : {code}.T")
ax.set_ylabel("stock_price")
ax.set_xlabel("datetime")
ax.grid(True)
ax.legend()
```



2.4.3. 移動平均

ここでは移動平均をプロットします。移動平均にもさまざまな種類がありますが、ここでは単純移動平均線を用います。単純移動平均線というのは、例えば、5日線であれば、直近5営業日の価格の平均値です。これを1つずつ期間をスライドしながら計算したものになります。

```

# stock_priceの読み込み
price = dfs["stock_price"].copy()

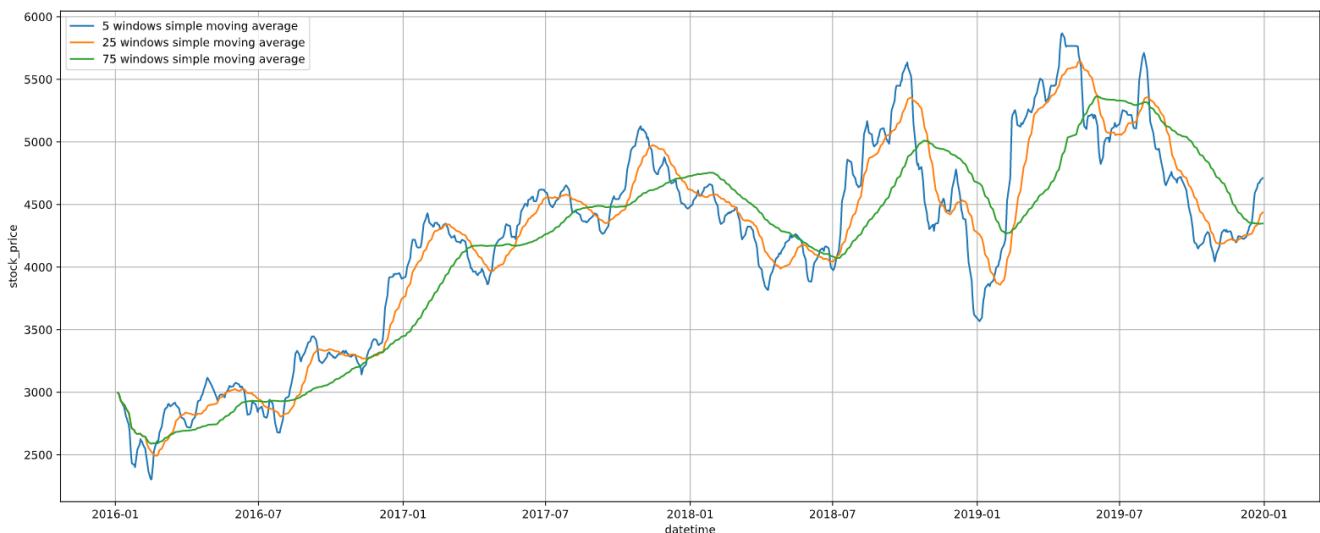
# 特定の銘柄コードに絞り込み
code = 9984
price_data = price[price["Local Code"] == code].copy()
# 日付列をpd.Timestamp型に変換してindexに設定
price_data["datetime"] = pd.to_datetime(price_data["EndOfDayQuote Date"])
price_data.set_index("datetime", inplace=True)
# 2019年までの値を表示
price_data = price_data[: "2019"]

# 5日、25日、75日の移動平均を算出
periods = [5, 25, 75]
cols = []
for period in periods:
    col = "{} windows simple moving average".format(period)
    price_data[col] = price_data["EndOfDayQuote ExchangeOfficialClose"].rolling(period,
min_periods=1).mean()
    cols.append(col)

# プロット
fig, ax = plt.subplots(figsize=(20, 8))

for col in cols:
    ax.plot(price_data[col], label=col)
ax.set_ylabel("stock_price")
ax.set_xlabel("datetime")
ax.grid(True)
ax.legend()

```



2.4.4. 値格変化率

価格変化率は、価格がその期間でどれくらい変化したかを(%)で表現したものです。相場の勢いや方向性等を判断する際によく使われます。

2. 財務諸表で株価の先行きを予測しよう

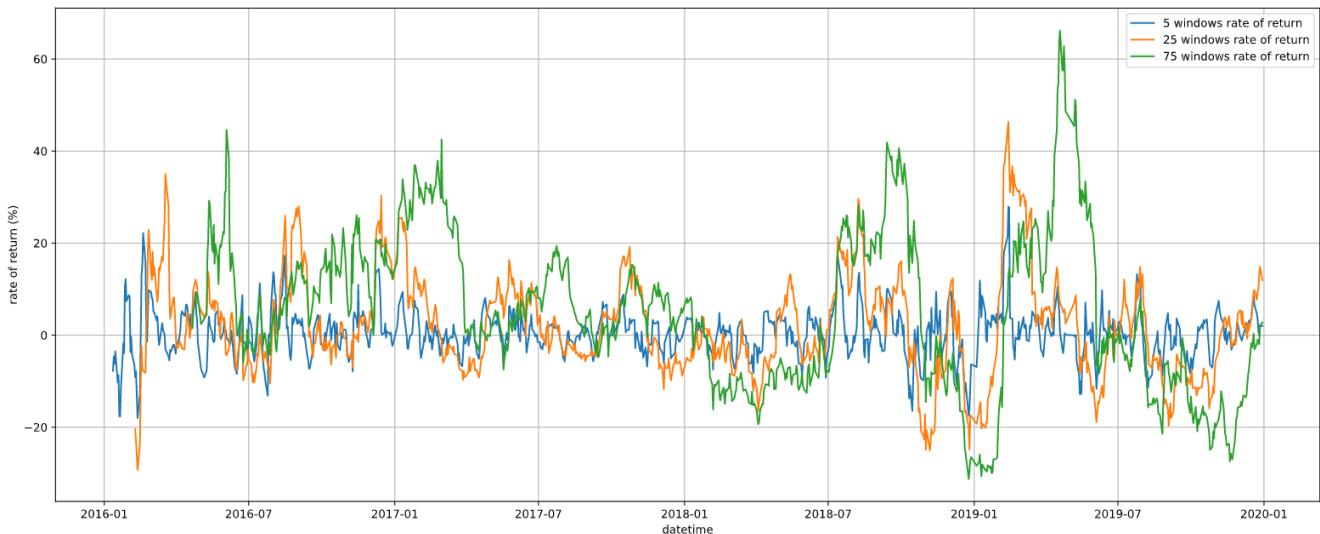
```
# stock_priceの読み込み
price = df["stock_price"].copy()

# 特定の銘柄コードに絞り込み
code = 9984
price_data = price[price["Local Code"] == code].copy()
# 日付列をpd.Timestamp型に変換してindexに設定
price_data["datetime"] = pd.to_datetime(price_data["EndOfDayQuote Date"])
price_data.set_index("datetime", inplace=True)
# 2019年までの値を表示
price_data = price_data[: "2019"]

# 5日、25日、75日の価格変化率を算出
periods = [5, 25, 75]
cols = []
for period in periods:
    col = "{} windows rate of return".format(period)
    price_data[col] = price_data["EndOfDayQuote ExchangeOfficialClose"].pct_change(period) *
100
    cols.append(col)

# プロット
fig, ax = plt.subplots(figsize=(20, 8))

for col in cols:
    ax.plot(price_data[col], label=col)
ax.set_ylabel("rate of return (%)")
ax.set_xlabel("datetime")
ax.grid(True)
ax.legend()
```



2.4.5. ヒストリカル・ボラティリティ

ここではヒストリカル・ボラティリティを計算します。ここで計算するヒストリカル・ボラティリティは、14日の対数リターンの標準偏差です。ヒストリカル・ボラティリティはリスク指標の一つで、価格がどの程度激しく変動したかを把握するために利用します。一般的にヒストリカル・ボラティリティが大きい銘柄は、小さい銘柄よりも資産として保持するリスクが相対的に高いと考えられます。

```

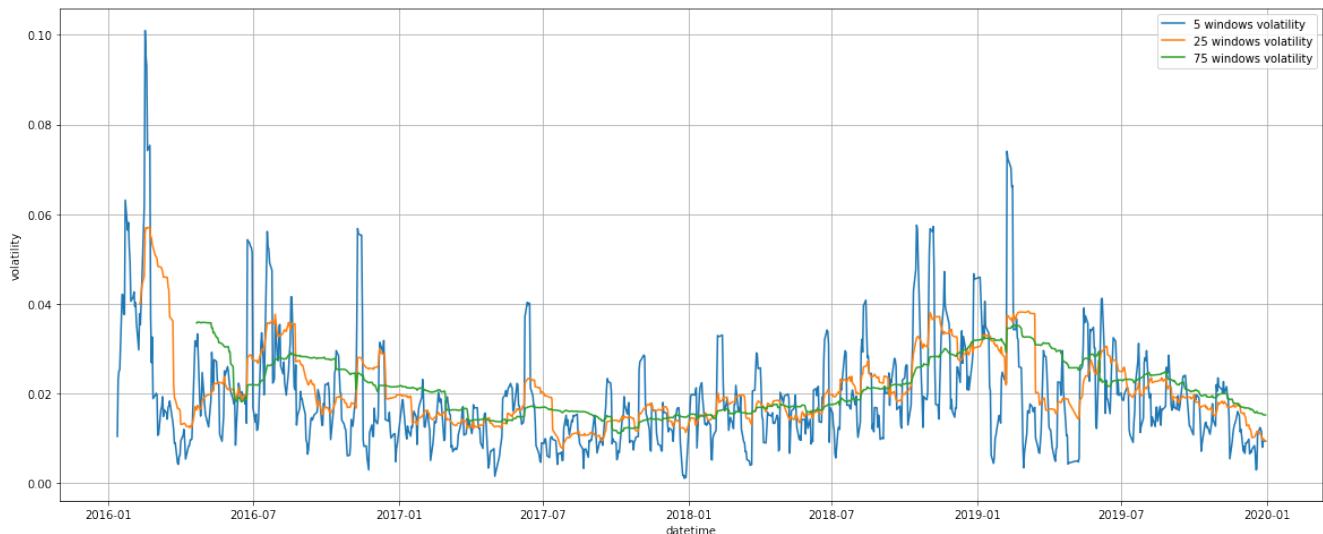
# stock_priceの読み込み
price = dfs["stock_price"].copy()

# 特定の銘柄コードに絞り込み
code = 9984
price_data = price[price["Local Code"] == code].copy()
# 日付列をpd.Timestamp型に変換してindexに設定
price_data["datetime"] = pd.to_datetime(price_data["EndOfDayQuote Date"])
price_data.set_index("datetime", inplace=True)
# 2019年までの値を表示
price_data = price_data[: "2019"]
# 5日、25日、75日のヒストリカル・ボラティリティを算出
periods = [5, 25, 75]
cols = []
for period in periods:
    col = "{} windows volatility".format(period)
    price_data[col] = np.log(price_data["EndOfDayQuote ExchangeOfficialClose"]).diff().rolling(
        period).std()
    cols.append(col)

# プロット
fig, ax = plt.subplots(figsize=(20, 8))

for col in cols:
    ax.plot(price_data[col], label=col)
ax.set_ylabel("volatility")
ax.set_xlabel("datetime")
ax.grid(True)
ax.legend()

```



2.4.6. 複数の株価データを同時にプロット

これまで可視化してきた株価に関するデータを同時にプロットすることで、それぞれの値の関連性について考察します。

2. 財務諸表で株価の先行きを予測しよう

```
# stock_priceの読み込み
price = dfs["stock_price"].copy()

# 特定の銘柄コードに絞り込み
code = 9984
price_data = price[price["Local Code"] == code].copy()
# 日付列をpd.Timestamp型に変換してindexに設定
price_data["datetime"] = pd.to_datetime(price_data["EndOfDayQuote Date"])
price_data.set_index("datetime", inplace=True)
# 2019年までの値を表示
price_data = price_data[:"2019"]
# 5日、25日、75日を対象に値を算出
periods = [5, 25, 75]
ma_cols = []
# 移動平均線
for period in periods:
    col = "{} windows simple moving average".format(period)
    price_data[col] = price_data["EndOfDayQuote ExchangeOfficialClose"].rolling(period,
min_periods=1).mean()
    ma_cols.append(col)

return_cols = []
# 価格変化率
for period in periods:
    col = "{} windows rate of return".format(period)
    price_data[col] = price_data["EndOfDayQuote ExchangeOfficialClose"].pct_change(period) *
100
    return_cols.append(col)

vol_cols = []
# ヒストリカル・ボラティリティ
for period in periods:
    col = "{} windows volatility".format(period)
    price_data[col] = np.log(price_data["EndOfDayQuote ExchangeOfficialClose"]).diff().rolling(
period).std()
    vol_cols.append(col)

# プロット
fig, ax = plt.subplots(nrows=3 , figsize=(20, 8))

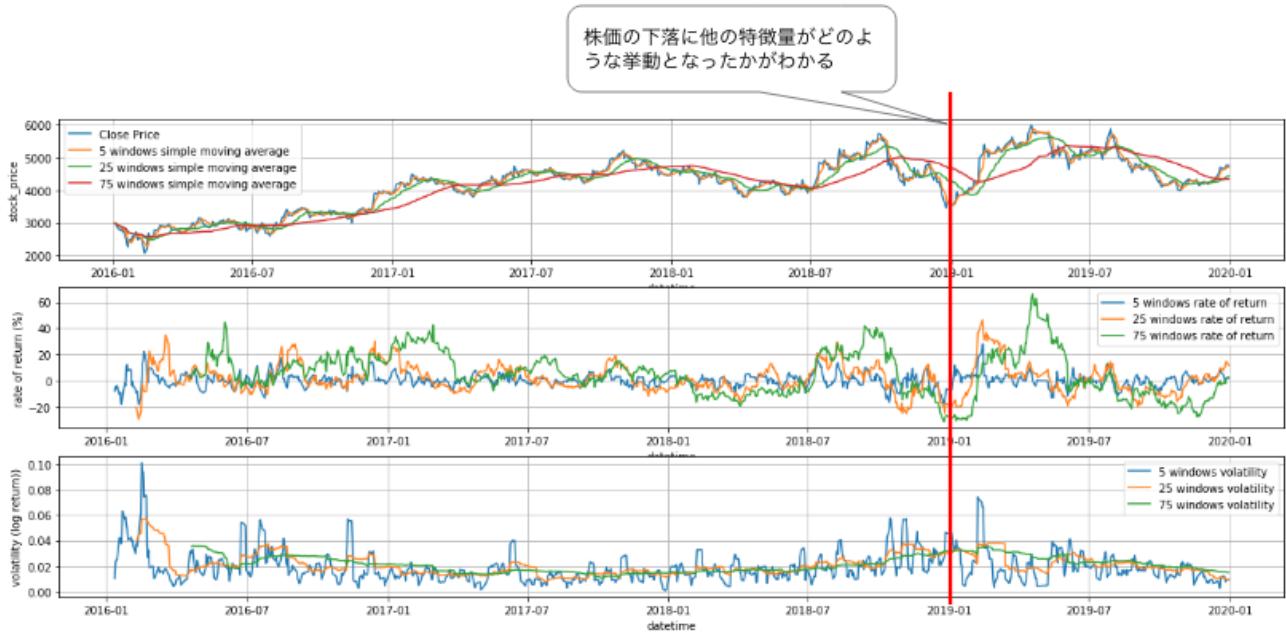
ax[0].plot(price_data["EndOfDayQuote ExchangeOfficialClose"], label="Close Price")

for col in ma_cols:
    ax[0].plot(price_data[col], label=col)

for col in return_cols:
    ax[1].plot(price_data[col], label=col)

for col in vol_cols:
    ax[2].plot(price_data[col], label=col)

ax[0].set_ylabel("stock_price")
ax[1].set_ylabel("rate of return (%)")
ax[2].set_ylabel("volatility (log return)")
for _ax in ax:
    _ax.set_xlabel("datetime")
    _ax.grid(True)
    _ax.legend()
```

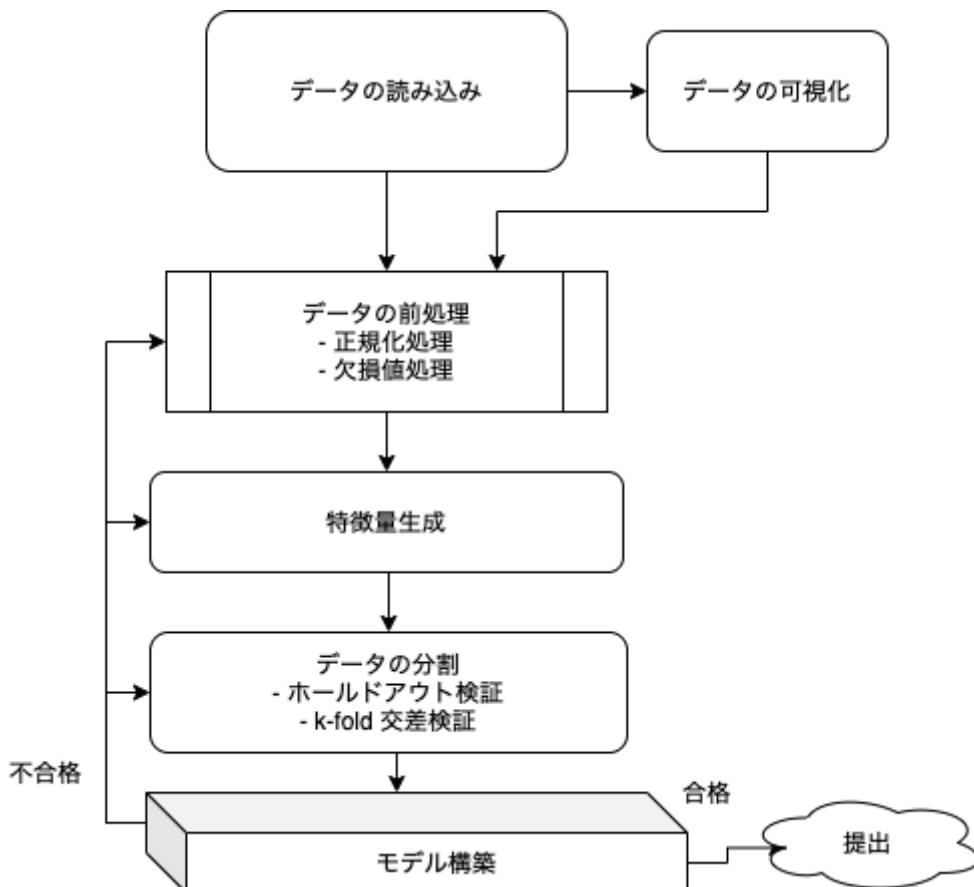


ここでは2018年末に起きた株価の下落に着目してみます。複数の特徴量を並べてプロットすると、株価に大きな変動があった時に他の特徴量にどのような影響を与えているかを観測することができます。

移動平均の特徴量は、期間が短いほど敏感に株価の下落に反応し、期間が長い特徴量ほど反応が遅れることがわかります。リターンの特徴量も下落時には同一の傾向が見て取れますが、その後高いリターンが観測されることがわかります。一方、ヒストリカル・ボラティリティの挙動をみると、下落の前にじわじわとボラティリティが上昇していることがわかります。このように一つの株価の下落を見ても、それぞれの特徴量の挙動が微妙に異なっており、複数個の特徴量をモデルに投入することで、これらの挙動のパターンを学習することができます。

2.5. データセットの前処理

ここまでデータの読み込みおよび可視化について説明してきましたが、ここからはデータの前処理やモデル構築に関して説明していきます。大まかな流れは、次の図のとおりです。



上図のとおり、モデルを構築する際には、データセットをそのまま入力するのではなく、欠損値処理や正規化処理などのデータセットの前処理を実施してから入力することが一般的です。ここではデータセットの前処理について解説していきます。

2.5.1. 欠損値処理

機械学習モデルの多くは欠損値をそのまま扱うことができないため、補完するなどして対処する必要があります。欠損値補完の方法としては、平均値埋めやリストワיז法、多重代入法などが存在します。また、変数に欠損が存在するレコードを単に除外することや、欠損を多く含む変数自体を除外することも考えられます。

2.5.2. 欠損値の処理方法

本コンペティションのデータについて、実際に欠損が発生している箇所やパターンを特定するために、欠損の発生状況をプロットして確認してみます。

```

# stock_finデータを読み込む
stock_fin = dfs["stock_fin"].copy()

# 日付列をpd.Timestamp型に変換してindexに設定
stock_fin["datetime"] = pd.to_datetime(stock_fin["base_date"])
stock_fin.set_index("datetime", inplace=True)
# 2019年までの値を表示
stock_fin = stock_fin[: "2019"]

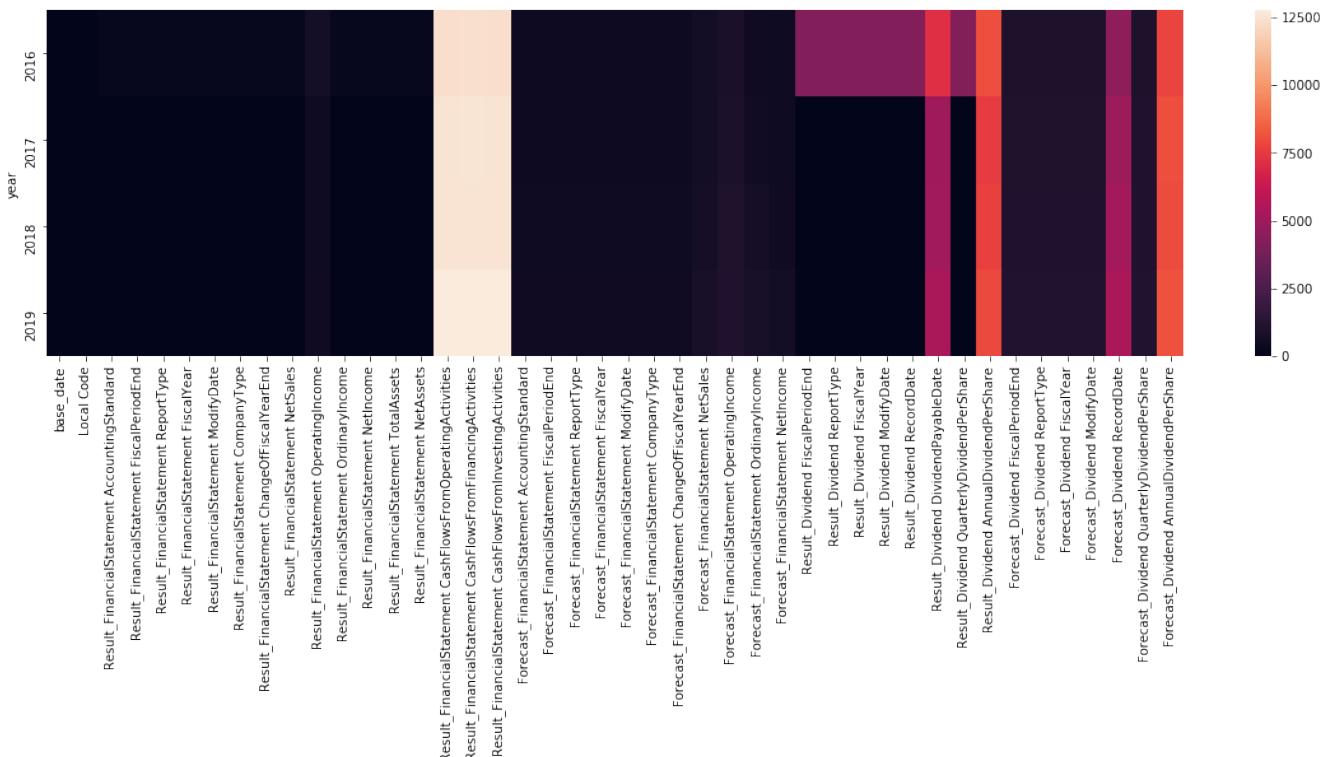
# データ数の確認
print(stock_fin.shape)

# データの欠損値数を確認
print(stock_fin.isna().sum())

# 欠損値の数を年別に集計
stock_fin = stock_fin.isna()
stock_fin["year"] = stock_fin.index.year

# データの欠損値をプロット
fig, ax = plt.subplots(figsize=(20, 5))
sns.heatmap(stock_fin.groupby("year").agg("sum"), ax=ax)

```



明るい色で示されている箇所は、欠損値が多く発生していることを表しています。本チュートリアルでは欠損値について以下の対応方針を採用します。

- ・`Result_FinancialStatement` の CashFlowsFromOperatingActivities 、
CashFlowsFromFinancingActivities 、 CashFlowsFromInvestingActivities に多くの欠損値があります。これらのカラムの値は Result_FinancialStatement ReportType が Annual の場合にのみ値が入っています。ここでは 0 を代入することで対処します。
- ・配当支払開始日を表す Result_Dividend DividendPayableDate や予想配当基準日を表す Forecast_Dividend RecordDate というカラムなどのfloat64型以外のカラムに数多く欠損が発生していること

2. 財務諸表で株価の先行きを予測しよう

が分かります。そのため、本チュートリアルではfloat64型として読み込まれているカラムのみを使用することとします。

- ・上記に記載したキャッシュフローに関するカラム以外のfloat64型として読み込まれているカラムの欠損値についても、0を代入することで対処します。ただし、変数によっては、0という数字自体に意味があるケースが有るため、気をつける必要があります。

実際の欠損値処理は、次のように変数の型(今回はnp.float64という型を選択)別に後段処理できる値(今回は0)で欠損値を埋めます。

```
# stock_finデータを読み込む
stock_fin = dfs["stock_fin"].copy()

# 銘柄コード9984にデータを絞る
code = 9984
stock_fin = stock_fin[stock_fin["Local Code"] == code]

# 日付列をpd.Timestamp型に変換してindexに設定
stock_fin["datetime"] = pd.to_datetime(stock_fin["base_date"])
stock_fin.set_index("datetime", inplace=True)

# float64型の列に絞り込み
fin_data = stock_fin.select_dtypes(include=["float64"])

# 欠損値を0でフィル
fin_data = fin_data.fillna(0)
```

2.6. 特徴量の生成

2.6.1. なぜ特徴量の設計が重要なのか

機械学習の手法にはモデルにできるだけ生に近いデータを与えてその関係性を見つける手法とドメイン知識や専門性を活かして、特徴量を設計する手法があります。

前者の手法はEnd-To-End Learningと呼ばれ、生に近いデータをモデルに与え、そのモデル自体に特徴量を発見させる手法です。音声認識などの分野で活用されています。

本チュートリアルでは、金融データに慣れ親しんでいただくためにも、特徴量の影響を細かく考察しながら汎化性能に貢献する特徴量を設計していくアプローチで、モデルを構築します。

特徴量生成は、仮説を考え、その仮説をモデルが学ぶにはどのような特徴量が必要か、ということを想像することが重要です。

本チュートリアルでは、「直近株価が上がったら、高値もより大きく変動しやすい」という仮説を立て、この仮説を基に特徴量を生成してみます。この仮説をモデルが学ぶためには直近株価が上がったことを示す特徴量が必要です。直近を仮に1ヶ月と仮定すると、20日リターンや20日移動平均乖離率などが候補になります。

また、この仮説が市場において必ずしも正しいという必要はなく、仮説を思いついたら、その仮説を学ぶことができる特徴量を想像し、実際に実験してみることが重要です。

2.6.2. 定常性を意識した特徴量設計

時系列データを扱う際には、定常性を意識して特徴量を設計することが重要です。

株価をそのまま学習させたケースと定常性がある特徴量を利用するケースについて考えてみます。

株価をそのまま学習させたケース: 例えば、モデルの訓練期間における株価が、100円～110円の範囲で動いたとします。もし、この数値をそのままモデルに投入すると、モデルは株価が100円～110円近辺で動くことを暗黙に学習します。しかし、この暗黙の仮定は実際のマーケットでは成立しておらず、テスト期間で株価が高騰すると、モデルがうまく動かないことがあります。他にも株価に特有の例としては株式分割や株式併合により株価のレンジが大きく変動する場合があります。

定常性がある特徴量を利用するケース: 例えば、20日の価格変化率を考えると、これは正規分布ではありませんが、一部のマーケットの混乱期を除けばほぼ0を中心とした正規分布に近い分布になります。特徴量は2%の上昇や4%の下落といった0を中心として時系列となっており、将来に渡っても似たような分布になることが期待でき、株価範囲に対する暗黙の仮定を学ぶ恐れがなくなります。このように将来に渡っても似たような分布を期待できる特徴量は定常性があるということができます。

定常性を意識すると、正規化処理における様々な注意点が見えてきます。たとえば、最小値と最大値を-1から1などにマッピングするMinMax正規化を株価に適用した場合、定常性を期待できるでしょうか？残念ながら株価に対してMinMax正規化を実施しても定常性を期待することはできません。株価をMinMax正規化したときにその最大値・最小値が未来に対しても適用できる保証ができないためです。このように時系列の特徴量の設計をするとき、定常性を意識しながら特徴量を設計していくことが重要です。

2.6.3. 特徴量の生成例

ここでは、特徴量の生成を `stock_price` の株価情報を利用して行います。株価情報には、価格や出来高等市場で公開されている株価の四本値(始値、高値、安値、終値)の時系列データが格納されています。本チュートリアルでは、特徴量の例として1ヶ月、2ヶ月、3ヶ月間の「終値の変化率（リターン）」、「ヒストリカル・ボラティリティ」、「移動平均線からの乖離率」を紹介します。

次のコードで具体的な計算については示しますが、定義は以下のとおりです。

特徴量の計算に使用する関数	使用する関数の説明
<code>pct_change(N)</code>	現在の観測値とN個前の観測値との変化率
<code>diff()</code>	現在の観測値と一つ前の観測値との差
<code>rolling(N)</code>	対象の観測値をN個でグループ化
<code>std()</code>	標準偏差
<code>mean()</code>	算術平均

2. 財務諸表で株価の先行きを予測しよう

```
# stock_priceデータを読み込む
price = df["stock_price"].copy()

# 銘柄コード9984にデータを絞る
code = 9984
price_data = price[price["Local Code"] == code].copy()
# 日付列をpd.Timestamp型に変換してindexに設定
price_data["datetime"] = pd.to_datetime(price_data["EndOfDayQuote Date"])
price_data.set_index("datetime", inplace=True)

# 終値のみに絞る
feats = price_data[["EndOfDayQuote ExchangeOfficialClose"]].copy()
# 終値の20営業日リターン
feats["return_1month"] = feats["EndOfDayQuote ExchangeOfficialClose"].pct_change(20)
# 終値の40営業日リターン
feats["return_2month"] = feats["EndOfDayQuote ExchangeOfficialClose"].pct_change(40)
# 終値の60営業日リターン
feats["return_3month"] = feats["EndOfDayQuote ExchangeOfficialClose"].pct_change(60)
# 終値の20営業日ボラティリティ
feats["volatility_1month"] = (
    np.log(feats["EndOfDayQuote ExchangeOfficialClose"]).diff().rolling(20).std()
)
# 終値の40営業日ボラティリティ
feats["volatility_2month"] = (
    np.log(feats["EndOfDayQuote ExchangeOfficialClose"]).diff().rolling(40).std()
)
# 終値の60営業日ボラティリティ
feats["volatility_3month"] = (
    np.log(feats["EndOfDayQuote ExchangeOfficialClose"]).diff().rolling(60).std()
)
# 終値と20営業日の単純移動平均線の乖離
feats["MA_gap_1month"] = feats["EndOfDayQuote ExchangeOfficialClose"] / (
    feats["EndOfDayQuote ExchangeOfficialClose"].rolling(20).mean()
)
# 終値と40営業日の単純移動平均線の乖離
feats["MA_gap_2month"] = feats["EndOfDayQuote ExchangeOfficialClose"] / (
    feats["EndOfDayQuote ExchangeOfficialClose"].rolling(40).mean()
)
# 終値と60営業日の単純移動平均線の乖離
feats["MA_gap_3month"] = feats["EndOfDayQuote ExchangeOfficialClose"] / (
    feats["EndOfDayQuote ExchangeOfficialClose"].rolling(60).mean()
)
# 元データのカラムを削除
feats = feats.dropna().drop(["EndOfDayQuote ExchangeOfficialClose"], axis=1)
```

2.6.4. テクニカル分析を活用した特徴量の生成

他に株価を扱うための特徴量としてRSIやストキャスティクスのようなテクニカル分析の指標などを活用することもあります。pythonの `talib` というライブラリにはたくさんのテクニカル分析が実装されているため、活用を検討する価値があります。ただし、これらに対しては、本章では取り扱わずに、第4章で詳しい解説を行います。

2.7. バックテスト用のテストデータ作成

ここでは、バックテストを行うためのデータの分割について説明します。

2.7.1. バックテストとは

バックテストとは、モデルの有効性を検証する際に、過去のデータを用いて、一定期間にどの程度のパフォーマンスが得られたかをシミュレーションすることです。モデルの有効性を検証する上で、どのようにバックテストを実施するかは重要なポイントになります。

2.7.2. ホールドアウト検証

ここでは、データセットを訓練データとテストデータに切り分けます。まずデータセットを分ける理由を説明し、次に、データセットの分け方及びコツについて解説します。

データセットを分ける理由は、モデルの汎化性能を確認し使用するモデルを決定するためです。

汎化性能とは、モデルが訓練データ以外の未知のデータに対しても機能するという能力です。汎化性能が低いモデルは、訓練データでは高い精度が得られるが、訓練データにない未知のデータについては、低い精度しか得られません。この現象を過学習と呼びます。データセットを分割せずに、そのまま全体に対して学習し、同じデータセットに対してモデルによる予測をすると、基本的には高い精度の結果を得ることができます。しかし、未知のデータに対して予測すると、予測がまったく当たらないということが起こり得ます。そのため、データセットを分割して、学習に使用していないデータをモデルの検証用として用意しておくことで、作成したモデルが過学習していないことを確認することができます。

基本的な時系列データの分割手法(ホールドアウト検証)に関して解説します。

1. 全体のデータセットを、訓練期間(TRAIN)/検証期間(VAL)/テスト期間(TEST)で分けます。
2. TRAINデータでモデルを学習させ、VALデータでモデルを評価します。これをモデルのさまざまなパラメーターで何度かを行い、一番結果が良かったパラメーターを選びます。
3. そして最後にTESTデータでモデルの予測結果を最終評価します。

本チュートリアルでは、次の期間でデータを分割します。

訓練期間	2016-01-01 - 2017-12-31
評価期間	2018-02-01 - 2018-12-01
テスト期間	2019-01-01 - 2019-12-31

※データの分割に際し、各期間に間隔（1か月）を空けている理由は、未来の情報を含ませないようにするためにです。例えば、2017年12月31日の目的変数には5営業日、10営業日、20営業日後の株価リターンの情報が入っているため、2017年12月31日のデータを使って学習したモデルは未来の情報（2018年1月のリターン）を知っていることになってしまいます。したがって、2018年1月のデータを検証データに含めてしまうとリーケが発生し、適切なモデルの評価ができなくなってしまいます。

以下のように変数を定義しておきます。

```
TRAIN_END = "2017-12-31"  
VAL_START = "2018-02-01"  
VAL_END = "2018-12-01"  
TEST_START = "2019-01-01"
```

2.7.3. その他の検証方法

- k-fold 交差検証(k-fold CV): データをまず訓練データとテストデータに分け、その後その訓練データをk個のグループに分割し、k-1個のグループに含まれるものと訓練データ、残りの1個のグループに含まれるものと評価データとすると、このような分割方法はk通り考えられます。
そこで、それぞれの分割方法したがってk回訓練と評価を行い、それぞれの試行結果の平均などを使用して、モデルやそのモデルのパラメータを評価します。なお、時系列データでは、将来の情報を含まないように注意する必要があります。交差検証する際に起き得るデータリークについては、第4章で詳しい解説を行います。

2.8. モデルの構築

ここでは、モデルの学習に用いるためのデータを準備します。

モデル作成のステップは、以下のとおりに行います。

1. 銘柄を一つ選ぶ
2. その銘柄に対して、財務データおよびマーケットデータから特徴量を作る
3. 全銘柄に対して同じことを繰り返す
4. 作成したデータを結合する
5. 全データを訓練データ、評価データ、テストデータに分ける
6. 訓練データで予測モデルを学習させる

・入力用データの作成方法

前回までのマーケットデータを用いた特徴量生成方法、および財務諸表データの欠損値処理を行った後のデータ処理について解説し、今回はそれらのデータを結合させ、モデルが学習できるフォーマットに直すことが目的です。

2.8.1. 特徴量の生成

ここでは、特徴量生成のコードを示しています。コードのおおまかな流れとしては、以下の3ステップに分けられます。

1. 財務データの取得及び前処理 (Chapter2.5.2と同様)
2. マーケットデータの取得及び特徴量定義 (Chapter2.6.3と同様)
3. 財務データと生成した特徴量を結合

```
def get_features_for_predict(dfs, code):
    """
    Args:
        dfs (dict) : dict of pd.DataFrame include stock_fin, stock_price
        code (int) : A local code for a listed company
    Returns:
        feature DataFrame (pd.DataFrame)
    """
    # おおまかな手順の1つ目
    # stock_finデータを読み込み
    stock_fin = dfs["stock_fin"].copy()

    # 特定の銘柄コードのデータに絞る
    fin_data = stock_fin[stock_fin["Local Code"] == code].copy()
    # 日付列をpd.Timestamp型に変換してindexに設定
    fin_data["datetime"] = pd.to_datetime(fin_data["base_date"])
    fin_data.set_index("datetime", inplace=True)
    # fin_dataのnp.float64のデータのみを取得
    fin_data = fin_data.select_dtypes(include=["float64"])
    # 欠損値処理
    fin_feats = fin_data.fillna(0)

    # おおまかな手順の2つ目
    # stock_priceデータを読み込む
    price = dfs["stock_price"].copy()
    # 特定の銘柄コードのデータに絞る
    price_data = price[price["Local Code"] == code].copy()
    # 日付列をpd.Timestamp型に変換してindexに設定
    price_data["datetime"] = pd.to_datetime(price_data["EndOfDayQuote Date"])
    price_data.set_index("datetime", inplace=True)
    # 終値のみに絞る
    feats = price_data[["EndOfDayQuote ExchangeOfficialClose"]].copy()
    # 終値の20営業日リターン
    feats["return_1month"] = feats["EndOfDayQuote ExchangeOfficialClose"].pct_change(20)
    # 終値の40営業日リターン
    feats["return_2month"] = feats["EndOfDayQuote ExchangeOfficialClose"].pct_change(40)
    # 終値の60営業日リターン
    feats["return_3month"] = feats["EndOfDayQuote ExchangeOfficialClose"].pct_change(60)
    # 終値の20営業日ボラティリティ
    feats["volatility_1month"] = (
        np.log(feats["EndOfDayQuote ExchangeOfficialClose"]).diff().rolling(20).std()
    )
    # 終値の40営業日ボラティリティ
    feats["volatility_2month"] = (
        np.log(feats["EndOfDayQuote ExchangeOfficialClose"]).diff().rolling(40).std()
    )
```

2. 財務諸表で株価の先行きを予測しよう

```
# 終値の60営業日ボラティリティ
feats["volatility_3month"] = (
    np.log(feats["EndOfDayQuote ExchangeOfficialClose"]).diff().rolling(60).std()
)
# 終値と20営業日の単純移動平均線の乖離
feats["MA_gap_1month"] = feats["EndOfDayQuote ExchangeOfficialClose"] / (
    feats["EndOfDayQuote ExchangeOfficialClose"].rolling(20).mean()
)
# 終値と40営業日の単純移動平均線の乖離
feats["MA_gap_2month"] = feats["EndOfDayQuote ExchangeOfficialClose"] / (
    feats["EndOfDayQuote ExchangeOfficialClose"].rolling(40).mean()
)
# 終値と60営業日の単純移動平均線の乖離
feats["MA_gap_3month"] = feats["EndOfDayQuote ExchangeOfficialClose"] / (
    feats["EndOfDayQuote ExchangeOfficialClose"].rolling(60).mean()
)

# おおまかな手順の3つ目
# 元データのカラムを削除
feats = feats.dropna().drop(["EndOfDayQuote ExchangeOfficialClose"], axis=1)

# 財務データの特徴量とマーケットデータの特徴量のインデックスを合わせる
feats = feats.loc[feats.index.isin(fin_feats.index)]
fin_feats = fin_feats.loc[fin_feats.index.isin(feats.index)]

# データを結合
feats = pd.concat([feats, fin_feats], axis=1).dropna()

# 欠損値処理を行います。
feats = feats.replace([np.inf, -np.inf], 0)

# 銘柄コードを設定
feats["code"] = code

return feats
```

ここで、`np.inf` を0に置換していますが、価格変化率の計算の際に発散してしまったものを、0と定義し直しています。このように、特徴量を定義する際に、特徴量変換により発散してしまった時やnanになった時の処理を、あらかじめ考慮しておくことがスムーズにデータセットを構築する上で重要です。

次にここまで処理の結果を確認します。

```
df = get_features_for_predict(dfs, 9984)
df.T
```

2.8.2. 目的変数の対応付け及び訓練データ、評価データ、テストデータの分割

次に目的変数を定義します。目的変数は、データセットの `stock_labels` 内にあり、利用する際は先ほど定義した特徴量のデータセットに対して、行（日付）を一致させる必要があります。

データセットの訓練期間、評価期間、テスト期間への分割処理も合わせて実施します。

```

def get_features_and_label(dfs, codes, feature, label):
    """
    Args:
        dfs (dict[pd.DataFrame]): loaded data
        codes (array) : target codes
        feature (pd.DataFrame): features
        labels (array) : labels which is used in prediction model
    Returns:
        train_X (pd.DataFrame): training data
        train_y (pd.DataFrame): label for train_X
        val_X (pd.DataFrame): validation data
        val_y (pd.DataFrame): label for val_X
        test_X (pd.DataFrame): test data
        test_y (pd.DataFrame): label for test_X
    """
    # 分割データ用の変数を定義
    trains_X, vals_X, tests_X = [], [], []
    trains_y, vals_y, tests_y = [], [], []

    # 銘柄コード毎に特徴量を作成
    for code in tqdm(codes):
        # 特徴量取得
        feats = feature[feature["code"] == code]

        # stock_labelsデータを読み込み
        stock_labels = dfs["stock_labels"].copy()
        # 特定の銘柄コードのデータに絞る
        stock_labels = stock_labels[stock_labels["Local Code"] == code]
        # 日付列をpd.Timestamp型に変換してindexに設定
        stock_labels["datetime"] = pd.to_datetime(stock_labels["base_date"])
        stock_labels.set_index("datetime", inplace=True)

        # 特定の目的変数に絞る
        labels = stock_labels[label]

        if feats.shape[0] > 0 and labels.shape[0] > 0:
            # 特徴量と目的変数のインデックスを合わせる
            labels = labels.loc[labels.index.isin(feats.index)]
            feats = feats.loc[feats.index.isin(labels.index)]
            labels.index = feats.index

            # データを分割（ホールドアウト法）
            _train_X = feats[: TRAIN_END].copy()
            _val_X = feats[VAL_START : VAL_END].copy()
            _test_X = feats[TEST_START :].copy()

            _train_y = labels[: TRAIN_END].copy()
            _val_y = labels[VAL_START : VAL_END].copy()
            _test_y = labels[TEST_START :].copy()

            # データを配列に格納（後ほど結合するため）
            trains_X.append(_train_X)
            vals_X.append(_val_X)
            tests_X.append(_test_X)

            trains_y.append(_train_y)
            vals_y.append(_val_y)
            tests_y.append(_test_y)

```

2. 財務諸表で株価の先行きを予測しよう

```
# 銘柄毎に作成した説明変数データを結合します。
train_X = pd.concat(trains_X)
val_X = pd.concat(vals_X)
test_X = pd.concat(tests_X)
# 銘柄毎に作成した目的変数データを結合します。
train_y = pd.concat(trains_y)
val_y = pd.concat(vals_y)
test_y = pd.concat(tests_y)

return train_X, train_y, val_X, val_y, test_X, test_y
```

次に、ここまで結果を確認します。

```
# 対象銘柄コードを定義
codes = [9984]
# 対象の目的変数を定義
label = "label_high_20"
# 特徴量を取得
feat = get_features_for_predict(dfs, codes[0])
# 特徴量と目的変数を入力し、分割データを取得
ret = get_features_and_label(dfs, codes, feat, label)
for v in ret:
    print(v.T)
```

ここまで一つの銘柄に対して処理をしてきましたが、ここからは全ての予測対象銘柄に対して上記の処理を実施するために、予測対象の銘柄コードを以下のように取得します。

```
def get_codes(dfs):
    """
    Args:
        dfs (dict[pd.DataFrame]): loaded data
    Returns:
        array: list of stock codes
    """
    stock_list = dfs["stock_list"].copy()
    # 予測対象の銘柄コードを取得
    codes = stock_list[stock_list["prediction_target"] == True][
        "Local Code"
    ].values
    return codes
```

次に、目的変数毎にデータセットを作成します。今回は全ての目的変数に同一の特徴量を使用していますが、目的変数に応じて特徴量をチューニングすることでより精度の高いモデルを作成することができます。

```

# 対象の目的変数を定義
labels = {
    "label_high_5",
    "label_high_10",
    "label_high_20",
    "label_low_5",
    "label_low_10",
    "label_low_20",
}
# 目的変数毎にデータを保存するための変数
train_X, val_X, test_X = {}, {}, {}
train_y, val_y, test_y = {}, {}, {}

# 予測対象銘柄を取得
codes = get_codes(dfs)

# 特徴量を作成
buff = []
for code in tqdm(codes):
    feat = get_features_for_predict(dfs, code)
    buff.append(feat)
feature = pd.concat(buff)

# 目的変数毎に処理
for label in tqdm(labels):
    # 特徴量と目的変数を取得
    _train_X, _train_y, _val_X, _val_y, _test_X, _test_y = get_features_and_label(dfs, codes,
feature, label)
    # 目的変数をキーとして値を保存
    train_X[label] = _train_X
    val_X[label] = _val_X
    test_X[label] = _test_X
    train_y[label] = _train_y
    val_y[label] = _val_y
    test_y[label] = _test_y

```

2.8.3. モデル学習の実行方法

データの準備が完了したので、いよいよモデルの学習を実行します。ここでは、sklearnライブラリの RandomForestRegressorモデルを使用します。モデルに設定する各種パラメータは、ここではとくに指定せずにライブラリのデフォルトパラメータを使用します。

RandomForestの回帰モデルであるRandomForestRegressorモデルを利用する理由は、予測する目的変数が連続値であるからです。RandomForestモデルは決定木をベースとするモデルであるため、以下の理由から最初に選択するモデルとして扱いやすいです。

- RandomForest内部で利用する決定木は、特徴量の大小関係のみに着目しており、値自体には意味がないので正規化処理の必要がありません
- 特徴量の重要度を取得することができ、次に実施することの道筋を立てやすい

```
# 目的変数を指定  
label = "label_high_20"  
# モデルの初期化  
pred_model = RandomForestRegressor(random_state=0)  
# モデルの学習  
pred_model.fit(train_X[label], train_y[label])
```

一方、サポートベクターマシンやニューラルネットワークを利用する際は、データの前処理における注意点が増えます。選択するモデルの特性に応じた正規化処理と特微量設計が重要です。

2.9. モデルの推論

ここでは構築したモデルから予測結果を出力し、可視化などによる分析を実施します。

2.9.1. 予測結果の出力方法

ここまで、それぞれの目的変数に対して、訓練データ、評価データ、テストデータに分割しました。ここからは、モデルの学習完了後に、テストデータを入力として予測を出力し、pandas.DataFrame形式に変換します。

```

# モデルを定義
models = {
    "rf": RandomForestRegressor,
}

# モデルを選択
model = "rf"

# 目的変数を指定
label = "label_high_20"

# 学習用データセット定義
# ファンダメンタル
fundamental_cols = dfs["stock_fin"].select_dtypes("float64").columns
fundamental_cols = fundamental_cols[fundamental_cols != "Result_Dividend DividendPayableDate"]
fundamental_cols = fundamental_cols[fundamental_cols != "Local Code"]
# 価格変化率
returns_cols = [x for x in train_X[label].columns if "return" in x]
# テクニカル
technical_cols = [x for x in train_X[label].columns if (x not in fundamental_cols) and (x != "code")]

columns = {
    "fundamental_only": fundamental_cols,
    "return_only": returns_cols,
    "technical_only": technical_cols,
    "fundamental+technical": list(fundamental_cols) + list(technical_cols),
}
# 学習用データセットを指定
col = "fundamental_only"

# 学習
pred_model = models[model](random_state=0)
pred_model.fit(train_X[label][columns[col]].values, train_y[label])

# 予測
result = {}
result[label] = pd.DataFrame(
    pred_model.predict(val_X[label][columns[col]]), columns=["predict"])
)

# 予測結果に日付と銘柄コードを追加
result[label]["datetime"] = val_X[label][columns[col]].index
result[label]["code"] = val_X[label]["code"].values

# 予測の符号を取得
result[label]["predict_dir"] = np.sign(result[label]["predict"])

# 実際の値を追加
result[label]["actual"] = val_y[label].values

```

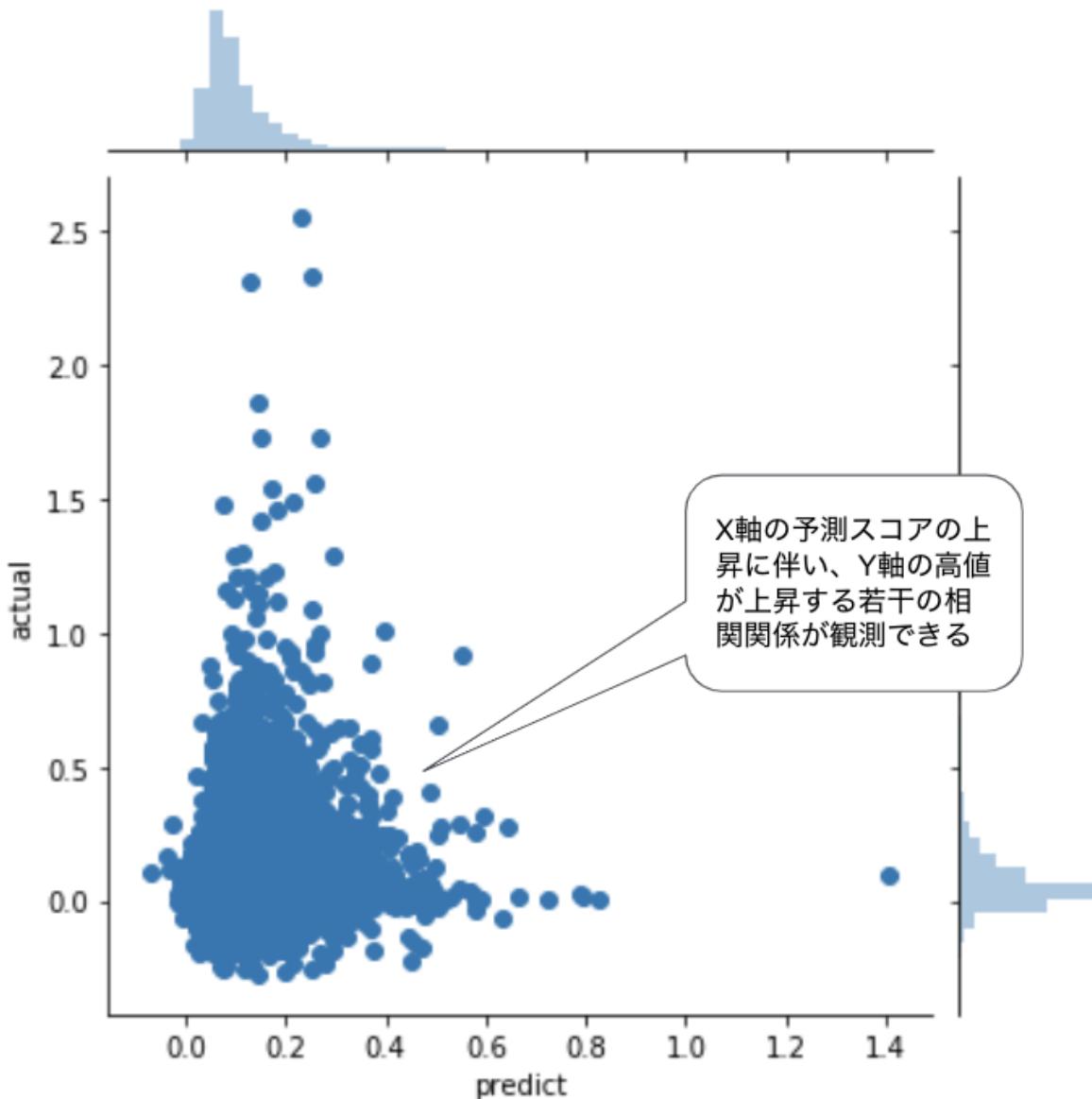
2.9.2. 予測結果の可視化方法

予測結果の確認として、実際の将来リターン(actual)と予測スコア(predict)の散布図を見ます。ここで散布図を選択する理由は、予測対象に対して予測スコアがどのような分布をとっているかを見ることが、モデルの挙動

2. 財務諸表で株価の先行きを予測しよう

を理解するわかりやすい可視化であることが挙げられます。

```
sns.jointplot(data=result[label], x="predict", y="actual")
```



この図では、横軸が予測値で、縦軸が真の値です。予測と真の値には、正の相関(0.169081)が見受けられるので、ある程度の相関関係が発生しています。一般的なデータで0.169という数字が出てもほぼ無相関に見えますが、金融データでは0.169というスコアは高い部類に入ります。このように視覚化すると、予測値と真の値の関係性を可視化できます。

2.10. 予測結果に対する分析の道筋

予測精度を向上させるためには、特徴量とモデルの分析を集中的に行う必要があります。特徴量の分析では、さまざまな手法がありますが、ここでは特徴量の重要度の分析とSHAPという手法を紹介します。

2.10.1. 特徴量の重要度

特徴量の重要度は、Random ForestやGradient Boostingなどのいくつかの機械学習モデルで取得でき、モデル内でどの程度それぞれの説明変数が、目的変数に対して重要であるかを判断するために参考になる指標です。

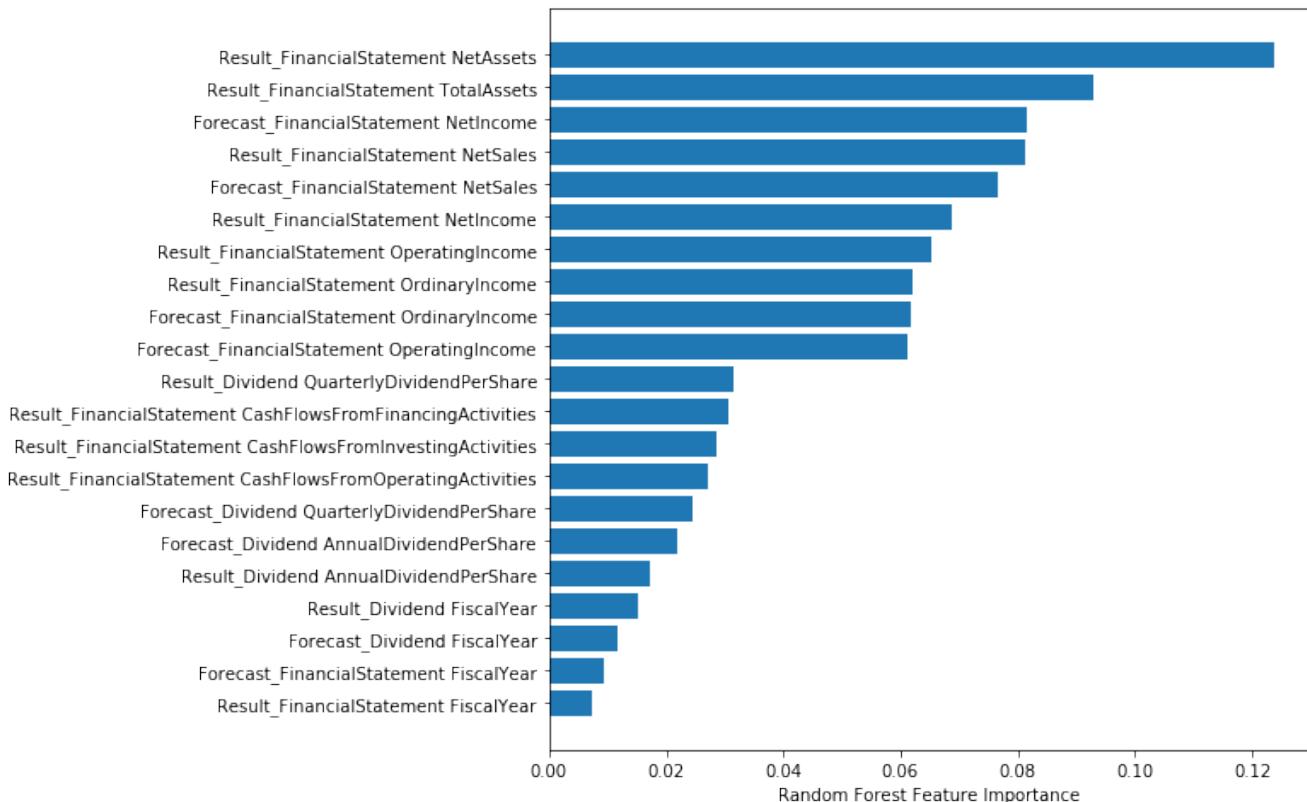
重要度が極端に低いものは、そもそも説明変数から除外したり、重要度が高いものは更に分析することで、性能の向上が期待できないか、など分析の道筋をつける上でも役に立ちます。

ここではファンダメンタル情報を用いて、モデルの訓練データ(2016年初から2017年末まで)における特徴量の重要度を調査します。

次の方法に従って、特徴量の重要度をプロットします。

```
# 学習済みモデルを指定
rf = pred_model

# 重要度順を取得
sorted_idx = rf.feature_importances_.argsort()
# プロット
fig, ax = plt.subplots(figsize=(8, 8))
ax.barh(fundamental_cols[sorted_idx], rf.feature_importances_[sorted_idx])
ax.set_xlabel("Random Forest Feature Importance")
```



上記の可視化により、一番上にある NetSales(売上高) にモデルが注目していることがわかります。売上高は会社の売上規模であり、次に登場する TotalAssets(総資産) は、流動資産や固定資産、繰延資産など、会社の全ての資産を合算したものを示す指標です。

2. 財務諸表で株価の先行きを予測しよう

この2つは会社の規模を示す代表的な指標となっています。

Random Forestモデルの内部で、この2つを利用した分岐が多数存在していることを示しており、売上規模や会社規模が重要な指標である可能性を示唆しています。

2.10.2. SHAP

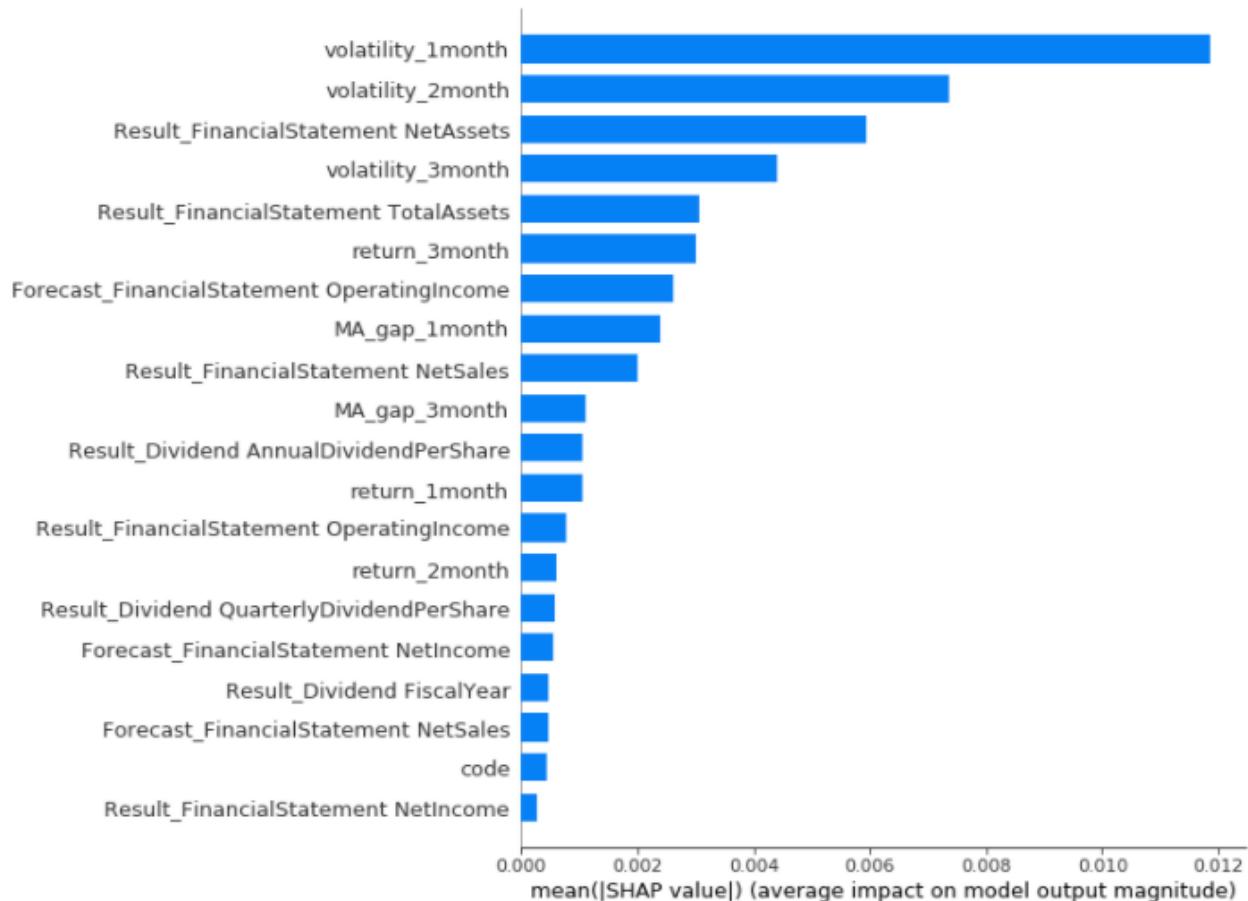
SHAPは、学習済みモデルにおいて、各特徴量がモデルの出力する予測値に与えた影響度を算出してくれるものです。

ここでは、サンプルモデルとしてXGBoostモデルを利用し、`label_high_20`という目的変数に対して、どの特徴量が学習に効果的な特徴量なのかを見てみます。

```
# モデルを定義します
sample_model = xgboost.train({"learning_rate": 0.01}, xgboost.DMatrix(train_X["label_high_20"],
label=train_y["label_high_20"]), 100)
```

次にshap値を求めます。

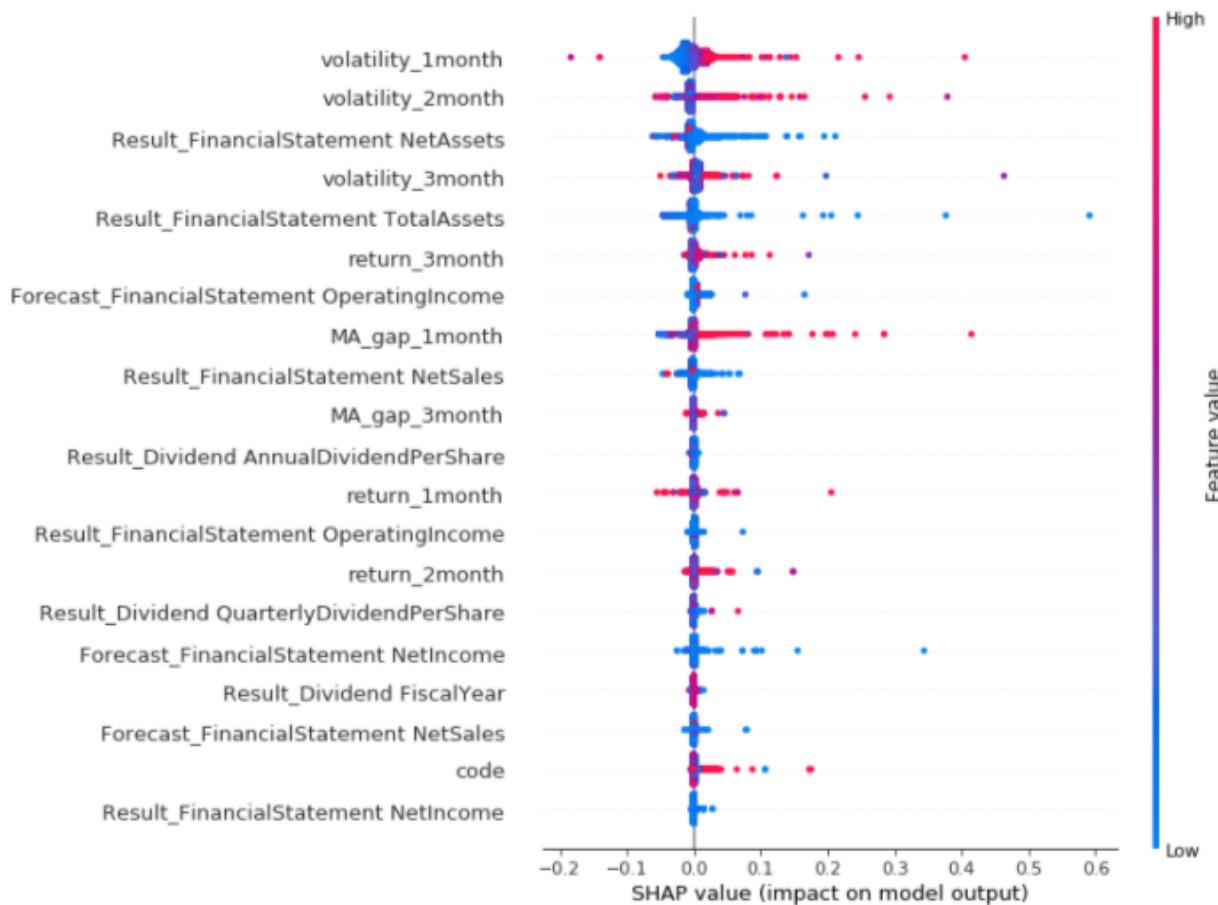
```
shap.initjs()
explainer = shap.TreeExplainer(model=sample_model, feature_perturbation='tree_path_dependent',
model_output='margin')
# SHAP値
shap_values = explainer.shap_values(X=train_X["label_high_20"])
# プロット
shap.summary_plot(shap_values, train_X["label_high_20"], plot_type="bar")
```



次にshapのsummary_plotを確認します。これは特徴量を少し変化させた時の学習のインパクトを表しています。

```
shap.summary_plot(shap_values, train_X["label_high_20"])
```

2. 財務諸表で株価の先行きを予測しよう



この図の見方ですが、上にある特徴量ほどモデルにとって重要であることを意味します。色が赤いのがその特徴量が高い時、青いのがその特徴量が低い時のSHAP値になります。図からは例えば以下のようなことが読み取れます。

- `volatility_1month` と `volatility_2month` がモデルに大きな影響を与える特徴量であることがわかります。この2つの特徴量は赤い時にプラス方向(高値が大きくなる)の影響が大きいことがわかります。これはボラティリティが上昇すると高値が高くなるということを意味するので直感に合致します。また `volatility_3month` がこの2つと比較すると影響が低いことがわかります。
- `Net Assets` が3番目にモデルに影響を与える特徴量であることがわかります。プラス方向、マイナス方向に関わらず青い色が多いので、モデルが `Net Assets` が小さい場合に活用していることがわかります。 `Total Assets` も同様の傾向が観測されます
- `MA_gap_1_month` が大きい時にプラス方向(高値が大きくなる)に影響を与えています。移動平均乖離率が大きいときは移動平均線がその時点の株価よりも上にいる期間なので、その時に高値が伸びるのは上昇トレンドが形成されている可能性が高いかもしれません。

上記のような考察を行いながら、さまざまな特徴量を考え、モデルを改善していくことが重要です。

2.11. モデルの評価

2.11.1. 複数モデルの学習および結果をまとめる

ここでは、複数モデルを用いて、予測および結果の比較を行いたいと思います。

今回はシンプルなモデルを複数用います。

モデル名	パラメーター
RandomForestRegressor	random_state = 0
ExtraTreesRegressor	random_state = 0
GradientBoostingRegressor	random_state = 0

次は学習用のデータセットも複数用意します。

学習用データセット名	説明
fundamental_only	財務諸表データのみ
return_only	価格変化率のデータのみ
technical_only	テクニカル指標のみ
fundamental+technical	財務諸表とテクニカル指標の両方

2. 財務諸表で株価の先行きを予測しよう

```
# モデルを定義
models = {
    "rf": RandomForestRegressor,
    "extraTree": ExtraTreesRegressor,
    "gbr": GradientBoostingRegressor,
}

# 学習用データセット定義
columns = {
    "fundamental_only": fundamental_cols,
    "return_only": returns_cols,
    "technical_only": technical_cols,
    "fundamental+technical": list(fundamental_cols) + list(technical_cols),
}

# 結果保存用
all_results = dict()

# モデル毎に処理
for model in tqdm(models.keys()):
    all_results[model] = dict()
    # データセット毎に処理
    for col in columns.keys():
        result = dict()
        # 目的変数毎に処理
        for label in tqdm(labels):
            if len(test_X[label][columns[col]]) > 0:
                # モデル取得
                pred_model = models[model](random_state=0)
                # 学習
                pred_model.fit(train_X[label][columns[col]].values, train_y[label])
                # 結果データ作成
                result[label] = test_X[label][["code"]].copy()
                result[label]["datetime"] = test_X[label][columns[col]].index
                # 予測
                result[label]["predict"] = pred_model.predict(test_X[label][columns[col]])
                result[label]["predict_dir"] = np.sign(result[label]["predict"])
                # 実際の結果
                result[label]["actual"] = test_y[label].values
                result[label]["actual_dir"] = np.sign(result[label]["actual"])
                result[label].dropna(inplace=True)

        all_results[model][col] = result
```

次にデータをまとめます。

```
results = []
for model in all_results.keys():
    for col in all_results[model]:
        tmp = pd.concat(all_results[model][col])
        tmp["model"] = model
        tmp["feature"] = col
        results.append(tmp)
results = pd.concat(results)
results["label"] = [x[0] for x in results.index]
results.head(5)
```

	code	datetime	predict	predict_dir	actual	actual_dir	model	feature	label
datetime									
label_high_10	2019-02-08	1301	2019-02-08	0.069097	1.0	0.07143	1.0	rf fundamental_only	label_high_10
	2019-05-13	1301	2019-05-13	0.068355	1.0	0.04379	1.0	rf fundamental_only	label_high_10
	2019-08-02	1301	2019-08-02	0.017164	1.0	0.00498	1.0	rf fundamental_only	label_high_10
	2019-11-05	1301	2019-11-05	0.027443	1.0	0.00841	1.0	rf fundamental_only	label_high_10
	2019-02-05	1332	2019-02-05	0.036817	1.0	0.02770	1.0	rf fundamental_only	label_high_10

では、次に評価していきます。

2.11.2. モデルの性能を示す評価関数の紹介

まずは、今回用いる評価関数のリストを紹介します。

評価関数	説明
RMSE	二乗平均平方根
accuracy	目的変数の符号と予測した目的変数の符号の精度
spreaman_corr	スピアマンの順位相関
corr	ピアソンの相関係数
R^2 score	単回帰した時の直線と観測値のバラつき

2. 財務諸表で株価の先行きを予測しよう

```
# 結果保存用変数
all_metrics = []

# データセット毎に処理
for feature in columns:
    matrix = dict()
    # モデル毎に処理
    for model in models:
        # 目的変数毎に処理
        for label in labels:
            # 処理対象データに絞り込み
            tmp_df = results[(results["model"] == model) & (results["label"] == label) &
(results["feature"] == feature)]
            # RMSE
            rmse = np.sqrt(mean_squared_error(tmp_df["predict"], tmp_df["actual"]))
            # 精度
            accuracy = accuracy_score(tmp_df["predict_dir"], tmp_df["actual_dir"])
            # 相関係数
            corr = np.corrcoef(tmp_df["actual"], tmp_df["predict"])[0, 1]
            # 順位相関
            spearman_corr = spearmanr(tmp_df["actual"], tmp_df["predict"])[0]
            # 結果を保存
            matrix[label] = [rmse, accuracy, spearman_corr, corr, corr**2, feature, model,
tmp_df.shape[0]]
            res = pd.DataFrame.from_dict(matrix).T
            res.columns = ["RMSE", "accuracy", "spearman_corr", "corr", "R^2 score", "feature", "model",
"# of samples"]
            all_metrics.append(res)
all_metrics = pd.concat(all_metrics)
all_metrics.reset_index()
```

このままだと出力が多すぎるため、集計します。

```
numeric_cols = ["RMSE", "accuracy", "spearman_corr", "corr", "R^2 score"]
for col in numeric_cols:
    all_metrics[col] = all_metrics[col].astype(float)
# indexとデータセット毎に平均を計算
agg = all_metrics.reset_index().groupby(["index", "feature"]).agg("mean")
agg
```

index	feature	RMSE	accuracy	spearman_corr	corr	R^2 score
label_high_10	fundamental+technical	0.112768	0.771532	0.178825	0.174241	0.030556
	fundamental_only	0.112788	0.768779	0.119807	0.143822	0.021400
	return_only	0.115120	0.768758	0.094811	0.101411	0.010946
	technical_only	0.111982	0.771224	0.142846	0.152716	0.023399
label_high_20	fundamental+technical	0.150394	0.813790	0.194590	0.186714	0.034952
	fundamental_only	0.150218	0.812886	0.129481	0.161143	0.027241
	return_only	0.152848	0.813071	0.109251	0.102472	0.011087
	technical_only	0.151497	0.813729	0.170911	0.143048	0.020481
label_high_5	fundamental+technical	0.095154	0.730096	0.164183	0.147188	0.021900
	fundamental_only	0.095381	0.726168	0.112826	0.119110	0.015255
	return_only	0.097478	0.723741	0.071467	0.076763	0.006991
	technical_only	0.094335	0.729952	0.118809	0.135202	0.018336
label_low_10	fundamental+technical	0.068019	0.837688	0.226066	0.235855	0.055754
	fundamental_only	0.070085	0.835902	0.112901	0.131413	0.018303
	return_only	0.070921	0.832121	0.131674	0.138640	0.019868
	technical_only	0.068372	0.837237	0.214467	0.223859	0.050182
label_low_20	fundamental+technical	0.073584	0.865981	0.261525	0.270947	0.073505
	fundamental_only	0.076518	0.864748	0.130508	0.154717	0.025619
	return_only	0.077362	0.862941	0.153035	0.163517	0.027493
	technical_only	0.074182	0.865818	0.249398	0.261754	0.088592
label_low_5	fundamental+technical	0.063987	0.806750	0.219854	0.228396	0.052443
	fundamental_only	0.065804	0.803953	0.117532	0.133354	0.018883
	return_only	0.066844	0.797536	0.119684	0.122252	0.015445
	technical_only	0.064451	0.806215	0.201479	0.208889	0.043702

この表のテクニカル分析 (technical_only) とファンダメンタルデータ (fundamental_only) にそれぞれ着目すると、テクニカル分析のみを用いた特徴量の方が、ファンダメンタルデータよりも精度 (accuracy) という観点で若干優れていることが分かります。

また、テクニカル分析とファンダメンタルデータの両方を特徴量を用いた場合の結果に関しては、テクニカル分析よりも若干全体正解率が高くなっていますが、誤差の範囲内です。スピアマンの順位相関 (spearman_corr) に関し、テクニカル分析とファンダメンタルデータを両方用いた場合は、用いていない場合に比べて、かなり優れていることが分かります。

このように特徴量を選択しながら複数のモデルをつくり、複数の評価関数で評価を行うことで、テクニカル分析とファンダメンタルデータを組み合わせたアプローチのポテンシャルが高いことがわかります。

2.12. モデルの提出

本コンペティションでは、モデルの予測の提出方法はモデル提出方式になります。以下ではこの方式について簡単に説明しますが、詳しくは [SIGNATE: Runtime 投稿方法](#) をご参照ください。

2.12.1. Runtimeの概要

学習済モデルを投稿すると、アルゴリズム (推論プログラム) が実行され、推論時間・推論結果が出力されます。出力された推論結果は、評価関数 (既存の投稿機能) に自動で投稿されます。

— Runtime 機能でできること, [SIGNATE: Runtime 投稿方法](#)

2.12.2. 提出ファイルの作成

1. 提出ファイルのテンプレートをダウンロード

[こちら](#)からダウンロード

2. ディレクトリの構造を確認する

アップロードするディレクトリ構造は次のとおりです。

.	
└── model	必須 学習済モデルを置くディレクトリ
└── ...	
└── src	必須 Python のプログラムを置くディレクトリ
└── predictor.py	必須 最初のプログラムが呼び出すファイル
└── ...	その他のファイル（ディレクトリ作成可能）
└── requirements.txt	任意

3. 学習済みモデルの作成

以下の環境でモデルを構築してください。

- Python3 Anaconda3-2019.03 インストールガイドは次のとおりです。

一般的なケース

<https://repo.continuum.io/archive/> からバージョン2019.03をダウンロードしてください

pyenv

```
pyenv install anaconda3-2019.03
```

Docker

```
docker pull continuumio/anaconda3:2019.03
```

ここで、学習済みモデルの保存方法について説明します。学習済みモデルは pickle で保存します。保存場所は2.13.2で説明したディレクトリ構造の学習済みモデルの配置先である `model` ディレクトリになります。任意のファイル名を設定可能なので、モデルの対象としている目的変数がわかるように `my_model_{label}.pkl` という名前で保存します。以下のコードを実行して、指定した保存先に学習済みモデルが保存されていることを確認します。

```

model = [保存対象の学習済みモデル]
# モデル保存先ディレクトリのパスを定義
model_path = '../model'
label = "label_high_20"

# モデル保存先ディレクトリを作成
os.makedirs(model_path, exist_ok=True)
with open(os.path.join(model_path, f"my_model_{label}.pkl"), "wb") as f:
    # モデルをpickle形式で保存
    pickle.dump(model, f)

```

4.predictor.py を記述する

ここでは、提出する予測モデルを読み込み、当該モデルを用いて予測を出力させるコードの書き方について説明します。predictor.py ファイルには、以下のクラスおよびメソッドを作成する必要があります。

ScoringService
推論実行のためのクラスです。
以下のメソッドを実装してください。

get_model
モデルを取得するメソッドです。以下の条件があります。
- クラスメソッドであること
- 引数 model_path (str 型) を指定すること
- 正常終了時は返り値を true (bool 型) とすること

predict
推論を実行するメソッドです。以下の条件があります。
- クラスメソッドであること
- 引数 input (dict[str] 型) を指定すること
※ 詳しくはテンプレート内の、同名ファイルをご確認ください。

本コンペティションにおけるpredictメソッドの返り値の定義は以下となります。詳細は以下に記載したコードをご参照ください。

結果を以下のcsv形式の文字列として出力する。
1列目:datetimeとcodeをつなげたもの(Ex 2016-05-09-1301)
2列目:label_high_20[終値→最高値への変化率]
3列目:label_low_20[終値→最安値への変化率]
headerはなし、B列C列はfloat64

以下は、本チュートリアルで説明した内容を規約に合わせて記載したpredictor.pyです。

```

# -*- coding: utf-8 -*-
import io
import os
import pickle

import numpy as np

```

2. 財務諸表で株価の先行きを予測しよう

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from tqdm.auto import tqdm

class ScoringService(object):
    # 訓練期間終了日
    TRAIN_END = "2017-12-31"
    # 評価期間開始日
    VAL_START = "2018-02-01"
    # 評価期間終了日
    VAL_END = "2018-12-01"
    # テスト期間開始日
    TEST_START = "2019-01-01"
    # 目的変数
    TARGET_LABELS = ["label_high_20", "label_low_20"]

    # データをこの変数に読み込む
    dfs = None
    # モデルをこの変数に読み込む
    models = None
    # 対象の銘柄コードをこの変数に読み込む
    codes = None

    @classmethod
    def get_inputs(cls, dataset_dir):
        """
        Args:
            dataset_dir (str) : path to dataset directory
        Returns:
            dict[str]: path to dataset files
        """
        inputs = {
            "stock_list": f"{dataset_dir}/stock_list.csv.gz",
            "stock_price": f"{dataset_dir}/stock_price.csv.gz",
            "stock_fin": f"{dataset_dir}/stock_fin.csv.gz",
            # "stock_fin_price": f"{dataset_dir}/stock_fin_price.csv.gz",
            "stock_labels": f"{dataset_dir}/stock_labels.csv.gz",
        }
        return inputs

    @classmethod
    def get_dataset(cls, inputs):
        """
        Args:
            inputs (list[str]): path to dataset files
        Returns:
            dict[pd.DataFrame]: loaded data
        """
        if cls.dfs is None:
            cls.dfs = {}
        for k, v in inputs.items():
            cls.dfs[k] = pd.read_csv(v)
        return cls.dfs

    @classmethod
    def get_codes(cls, dfs):
        """
        Args:
```

```

dfs (dict[pd.DataFrame]): loaded data
>Returns:
    array: list of stock codes
"""

stock_list = dfs["stock_list"].copy()
# 予測対象の銘柄コードを取得
cls.codes = stock_list[stock_list["prediction_target"] == True][
    "Local Code"
].values
return cls.codes

@classmethod
def get_features_and_label(cls, dfs, codes, feature, label):
    """
Args:
    dfs (dict[pd.DataFrame]): loaded data
    codes (array) : target codes
    feature (pd.DataFrame): features
    labels (array) : labels which is used in prediction model
>Returns:
    train_X (pd.DataFrame): training data
    train_y (pd.DataFrame): label for train_X
    val_X (pd.DataFrame): validation data
    val_y (pd.DataFrame): label for val_X
    test_X (pd.DataFrame): test data
    test_y (pd.DataFrame): label for test_X
"""

# 分割データ用の変数を定義
trains_X, vals_X, tests_X = [], [], []
trains_y, vals_y, tests_y = [], [], []

# 銘柄コード毎に特徴量を作成
for code in tqdm(codes):
    # 特徴量取得
    feats = feature[feature["code"] == code]

    # stock_labelデータを読み込み
    stock_labels = dfs["stock_labels"].copy()
    # 特定の銘柄コードのデータに絞る
    stock_labels = stock_labels[stock_labels["Local Code"] == code]
    # 日付列をpd.Timestamp型に変換してindexに設定
    stock_labels["datetime"] = pd.to_datetime(stock_labels["base_date"])
    stock_labels.set_index("datetime", inplace=True)

    # 特定の目的変数に絞る
    labels = stock_labels[label]

    if feats.shape[0] > 0 and labels.shape[0] > 0:
        # 特徴量と目的変数のインデックスを合わせる
        labels = labels.loc[labels.index.isin(feats.index)]
        feats = feats.loc[feats.index.isin(labels.index)]
        labels.index = feats.index

        # データを分割
        _train_X = feats[: cls.TRAIN_END].copy()
        _val_X = feats[cls.VAL_START : cls.VAL_END].copy()
        _test_X = feats[cls.TEST_START :].copy()

        _train_y = labels[: cls.TRAIN_END].copy()

```

2. 財務諸表で株価の先行きを予測しよう

```
_val_y = labels[cls.VAL_START : cls.VAL_END].copy()
_test_y = labels[cls.TEST_START : ].copy()

# データを配列に格納（後ほど結合するため）
trains_X.append(_train_X)
vals_X.append(_val_X)
tests_X.append(_test_X)

trains_y.append(_train_y)
vals_y.append(_val_y)
tests_y.append(_test_y)

# 銘柄毎に作成した説明変数データを結合します。
train_X = pd.concat(trains_X)
val_X = pd.concat(vals_X)
test_X = pd.concat(tests_X)
# 銘柄毎に作成した目的変数データを結合します。
train_y = pd.concat(trains_y)
val_y = pd.concat(vals_y)
test_y = pd.concat(tests_y)

return train_X, train_y, val_X, val_y, test_X, test_y

@classmethod
def get_features_for_predict(cls, dfs, code):
    """
    Args:
        dfs (dict) : dict of pd.DataFrame include stock_fin, stock_price
        code (int) : A local code for a listed company
    Returns:
        feature DataFrame (pd.DataFrame)
    """
    # stock_finデータを読み込み
    stock_fin = dfs["stock_fin"].copy()

    # 特定の銘柄コードのデータに絞る
    fin_data = stock_fin[stock_fin["Local Code"] == code].copy()
    # 日付列をpd.Timestamp型に変換してindexに設定
    fin_data["datetime"] = pd.to_datetime(fin_data["base_date"])
    fin_data.set_index("datetime", inplace=True)
    # fin_dataのnp.float64のデータのみを取得
    fin_data = fin_data.select_dtypes(include=["float64"])
    # 欠損値処理
    fin_feats = fin_data.fillna(0)

    # stock_priceデータを読み込む
    price = dfs["stock_price"].copy()
    # 特定の銘柄コードのデータに絞る
    price_data = price[price["Local Code"] == code].copy()
    # 日付列をpd.Timestamp型に変換してindexに設定
    price_data["datetime"] = pd.to_datetime(price_data["EndOfDayQuote Date"])
    price_data.set_index("datetime", inplace=True)
    # 終値のみに絞る
    feats = price_data[["EndOfDayQuote ExchangeOfficialClose"]].copy()
    # 終値の20営業日リターン
    feats["return_1month"] = feats[
        "EndOfDayQuote ExchangeOfficialClose"
    ].pct_change(20)
    # 終値の40営業日リターン
    feats["return_2month"] = feats[
```

```

        "EndOfDayQuote ExchangeOfficialClose"
    ].pct_change(40)
    # 終値の60営業日リターン
    feats["return_3month"] = feats[
        "EndOfDayQuote ExchangeOfficialClose"
    ].pct_change(60)
    # 終値の20営業日ボラティリティ
    feats["volatility_1month"] = (
        np.log(feats["EndOfDayQuote ExchangeOfficialClose"]))
        .diff()
        .rolling(20)
        .std()
    )
    # 終値の40営業日ボラティリティ
    feats["volatility_2month"] = (
        np.log(feats["EndOfDayQuote ExchangeOfficialClose"]))
        .diff()
        .rolling(40)
        .std()
    )
    # 終値の60営業日ボラティリティ
    feats["volatility_3month"] = (
        np.log(feats["EndOfDayQuote ExchangeOfficialClose"]))
        .diff()
        .rolling(60)
        .std()
    )
    # 終値と20営業日の単純移動平均線の乖離
    feats["MA_gap_1month"] = feats["EndOfDayQuote ExchangeOfficialClose"] / (
        feats["EndOfDayQuote ExchangeOfficialClose"].rolling(20).mean()
    )
    # 終値と40営業日の単純移動平均線の乖離
    feats["MA_gap_2month"] = feats["EndOfDayQuote ExchangeOfficialClose"] / (
        feats["EndOfDayQuote ExchangeOfficialClose"].rolling(40).mean()
    )
    # 終値と60営業日の単純移動平均線の乖離
    feats["MA_gap_3month"] = feats["EndOfDayQuote ExchangeOfficialClose"] / (
        feats["EndOfDayQuote ExchangeOfficialClose"].rolling(60).mean()
    )
    # 元データのカラムを削除
    feats = feats.dropna().drop(["EndOfDayQuote ExchangeOfficialClose"], axis=1)

    # 財務データの特徴量とマーケットデータの特徴量のインデックスを合わせる
    feats = feats.loc[feats.index.isin(fin_feats.index)]
    fin_feats = fin_feats.loc[fin_feats.index.isin(feats.index)]

    # データを結合
    feats = pd.concat([feats, fin_feats], axis=1).dropna()

    # 欠損値処理を行います。
    feats = feats.replace([np.inf, -np.inf], 0)

    # 銘柄コードを設定
    feats["code"] = code

    return feats

@classmethod
def create_model(cls, dfs, codes, label, model_path="..../model"):

```

2. 財務諸表で株価の先行きを予測しよう

```
"""
Args:
    dfs (dict) : dict of pd.DataFrame include stock_fin, stock_price
    codes (list[int]): A local code for a listed company
    label (str): prediction target label
Returns:
    RandomForestRegressor
"""

# 特徴量を取得
buff = []
for code in codes:
    buff.append(cls.get_features_for_predict(cls.dfs, code))
feature = pd.concat(buff)
# 特徴量と目的変数を一致させて、データを分割
train_X, train_y, _, _, _, _ = cls.get_features_and_label(
    dfs, codes, feature, label
)
# モデル作成
model = RandomForestRegressor(random_state=0)
model.fit(train_X, train_y)

return model

@classmethod
def save_model(cls, model, label, model_path="..../model"):
    """
Args:
    model (RandomForestRegressor): trained model
    label (str): prediction target label
    model_path (str): path to save model
Returns:
    -
"""

# tag::save_model_partial[]
# モデル保存先ディレクトリを作成
os.makedirs(model_path, exist_ok=True)
with open(os.path.join(model_path, f"my_model_{label}.pkl"), "wb") as f:
    # モデルをpickle形式で保存
    pickle.dump(model, f)
# end::save_model_partial[]

@classmethod
def get_model(cls, model_path="..../model", labels=None):
    """Get model method

Args:
    model_path (str): Path to the trained model directory.

Returns:
    bool: The return value. True for success, False otherwise.

"""

if cls.models is None:
    cls.models = {}
if labels is None:
    labels = cls.TARGET_LABELS
try:
    for label in labels:
        m = os.path.join(model_path, f"my_model_{label}.pkl")
```

```

    with open(m, "rb") as f:
        # pickle形式で保存されているモデルを読み込み
        cls.models[label] = pickle.load(f)
    return True
except Exception as e:
    print(e)
    return False

@classmethod
def train_and_save_model(cls, inputs, labels=None, codes=None):
    """Predict method

    Args:
        inputs (str) : paths to the dataset files
        labels (array) : labels which is used in prediction model
        codes (array) : target codes

    Returns:
        Dict[pd.DataFrame]: Inference for the given input.
    """
    if cls.dfs is None:
        cls.get_dataset(inputs)
        cls.get_codes(cls.dfs)
    if codes is None:
        codes = cls.codes
    if labels is None:
        labels = cls.TARGET_LABELS
    for label in labels:
        print(label)
        model = cls.create_model(cls.dfs, codes=codes, label=label)
        cls.save_model(model, label)

@classmethod
def predict(cls, inputs, labels=None, codes=None):
    """Predict method

    Args:
        inputs (dict[str]): paths to the dataset files
        labels (list[str]): target label names
        codes (list[int]): target codes

    Returns:
        str: Inference for the given input.
    """

# データ読み込み
if cls.dfs is None:
    cls.get_dataset(inputs)
    cls.get_codes(cls.dfs)

# 予測対象の銘柄コードと目的変数を設定
if codes is None:
    codes = cls.codes
if labels is None:
    labels = cls.TARGET_LABELS

# 特徴量を作成
buff = []
for code in codes:
    buff.append(cls.get_features_for_predict(cls.dfs, code))
feats = pd.concat(buff)

```

2. 財務諸表で株価の先行きを予測しよう

```
# 結果を以下のcsv形式で出力する
# 1列目:datetimeとcodeをつなげたもの(Ex 2016-05-09-1301)
# 2列目:label_high_20[終値→最高値への変化率
# 3列目:label_low_20[終値→最安値への変化率
# headerはなし、B列C列はfloat64

# 日付と銘柄コードに絞り込み
df = feats.loc[:, ["code"]].copy()
# codeを出力形式の1列目と一致させる
df.loc[:, "code"] = df.index.strftime("%Y-%m-%d") + df.loc[:, "code"].astype(str)

# 出力対象列を定義
output_columns = ["code"]

# 目的変数毎に予測
ret = {}
for label in labels:
    # 予測実施
    df[label] = cls.models[label].predict(feats)
    # 出力対象列に追加
    output_columns.append(label)

out = io.StringIO()
df.to_csv(out, header=False, index=False, columns=output_columns)

return out.getvalue()
```

モジュールの追加

https://docs.anaconda.com/anaconda/packages/py3.7_linux-64/ この表の [In Installer] にチェックが入っているものが、すでにインストールされています（ただしバージョンは異なります）。

Runtime環境にモジュールを追加するためには requirements.txt に追記します。requirements.txtに記載したモジュールは実行時にpipでインストールされます。

モジュールを追加する際はRuntime環境でインストールおよび使用可能かをご確認ください。本チュートリアルで評価のために使用したSHAPのように、一部のモジュールはインストール時にビルドが必要となるものもあります。そのためRuntime環境では使用することのできないモジュールもあります。以下のように実行環境の docker container内でインストールすることで確認可能です。

```
$ docker run --rm -it continuumio/anaconda3:2019.03 bash
# pip install tensorflow==2.4.0
```

requirements.txt には以下のようにモジュールのバージョンを指定して記載します。これは、モデルを提出してから全ての評価が完了するまで数ヶ月かかるためその間にモジュールの最新バージョンがリリースされても影響を受けないようにするためです。

```
tensorflow==2.4.0
```

requirements.txt の作成には pip freeze コマンドを使用すると便利です。

```
$ docker run --rm -it continuumio/anaconda3:2019.03 bash
# pip install [インストールするモジュール]
# pip freeze
```

デバッグ方法

通常

```
$ pip install -r requirements.txt # モジュールが必要な場合は pip でインストールします
$ cd src # ソースディレクトリに移動
$ python # python の実行
Python 3.7.3 (default, Mar 27 2019, 16:54:48)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> DATASET_DIR= "/path/to" # データ保存先ディレクトリ
>>> from predictor import ScoringService # モジュール読み込み
>>> inputs = ScoringService.get_inputs(DATASET_DIR) # 推論入力データ取得
>>> ScoringService.get_model() # モデルの取得
True
>>> ScoringService.predict(inputs) # 推論の実行
'[推論結果]'
```

pyenvを使用

```
$ pyenv local anaconda3-2019.03
```

以降 通常のインストールの場合と同様のデバッグ方法です。

Docker を使用

```
$ docker run -it -v $(pwd):/opt/ml continuumio/anaconda3:2019.03 /bin/bash
$ cd /opt/ml
```

以降は、通常のインストールの場合と同様のデバッグ方法になります。

5.zip で圧縮して提出する。

指定されたディレクトリ構成で学習済みモデルを保存した上で、predictor.pyを作成したら、以下のようにzipで圧縮して提出します。

2. 財務諸表で株価の先行きを予測しよう

```
$ ls
model requirements.txt src
$ zip -v submit.zip requirements.txt src/*.py model/*.pkl
updating: requirements.txt  (in=0) (out=0) (stored 0%)
updating: src/predictor.py  (in=11408) (out=2417) (deflated 79%)
updating: model/my_model_label_high_20.pkl .  (in=18919345) (out=5071005) (deflated 73%)
updating: model/my_model_label_low_20.pkl . (in=18704305) (out=5006613) (deflated 73%)
total bytes=37635058, compressed=10080035 -> 73% savings
```

3. ニュースでポートフォリオを構築しよう

3.1. 概要

本章は問題2の公開時に公開を予定しております。

スケジュール

日時	内容
2021年3月12日(金)	コンペティション開始
2021年5月9日(日)	モデル提出締切
2021年5月10日(月)～5月28日(金)	モデル評価期間
2021年7月頃	入賞者の決定

4. tips集

本コンペティションで関連する金融・データ解析一般のtipsを紹介

4.1. 金融用語集

本コンペティションで必要となる専門用語を解説してくれているサイトをご紹介します。もし、専門用語で困った場合は、下記のリンク先のコンテンツを確認していただければ幸いです。

タイトル	運営元	URL
証券用語解説集	野村證券	https://www.nomura.co.jp/terms/
金融・証券用語解説集	大和証券	https://www.daiwa.jp/glossary/
ファイナンス用語集	みずほ証券	https://glossary.mizuho-sc.com/
初めてでもわかりやすい用語集	SMBC日興証券	https://www.smbcnikko.co.jp/terms/index.html
用語解説	三菱UFJモルガン・スタンレー証券株式会社	https://www.sc.mufg.jp/learn/terms/index.html
金融用語解説(知るぽると)	金融広報中央委員会	https://www.shiruporuto.jp/public/document/container/yogo/
金融・証券用語集	日本証券業協会	https://www.jsda.or.jp/jikan/word/
用語集	EY新日本有限責任監査法人	https://www.shinnihon.or.jp/corporate-accounting/glossary/
会計監査用語解説集	日本公認会計士協会	https://jicpa.or.jp/cpainfo/introduction/keyword/
財務諸表等の用語、様式及び作成方法に関する規則	e-GOV	https://elaws.e-gov.go.jp/document?lawid=338M50000040059
用語集	野村アセットマネジメント	https://www.nomura-am.co.jp/basicknowledge/word/
用語集	アセットマネジメントOne	http://www.am-one.co.jp/shisankeisei/glossary/
用語集	大和アセットマネジメント	https://www.daiwa-am.co.jp/guide/term/
わかりやすい用語集	三井住友DSアセットマネジメント	https://www.smd-am.co.jp/learning/glossary/

4.2. 東証マネ部

「東証 マネ部！」は身近なお金の話から、プロが教える資産運用のノウハウまで、資産形成についてわかりやすく解説するサイトです。今回は、コンペに関係がありそうな記事をピックアップしてみました。コンペティションの息抜きにぜひご確認ください。

タイトル	リンク
投資に不可欠な財務三表の見方	https://money-bu-jpx.com/news/article022723/
財務ニュースを読む	https://money-bu-jpx.com/news/article028193/
長期投資に欠かせない運用コストを意識しよう	https://money-bu-jpx.com/news/article004555/
プロの投資家が注目する指標「ROE」とは？	https://money-bu-jpx.com/news/article005169/
ディープラーニングが拓く「AI投資」の可能性	https://money-bu-jpx.com/news/article008127/
AIが導く金融市场の未来	https://money-bu-jpx.com/news/article008332/
4大投資指標のワナ～解析力の鍛錬～	https://money-bu-jpx.com/news/article012308/
AIを使った市場の予測に挑む「AlpacaJapan」	https://money-bu-jpx.com/news/article015448/
「AIと資産運用」	https://money-bu-jpx.com/news/article016485/
投資に役立つ「会社四季報」活用のポイント	https://money-bu-jpx.com/news/article020860/
最低投資金額50万円以下の銘柄特集	https://money-bu-jpx.com/news/article022924/
クチコミで投資を楽しめるアプリ「ferci(フェルシー)」	https://money-bu-jpx.com/news/article023550/
コロナ後の世界	https://money-bu-jpx.com/news/article024736/
株式場況を読む～専門用語の理解～	https://money-bu-jpx.com/news/article026712/
決算ニュースを読む～会計用語の理解～	https://money-bu-jpx.com/news/article027285/
公的統計を補完する「オルタナティブデータ」とは？	https://money-bu-jpx.com/news/article028023/

4.3. 可視化テクニック

seaborn(複雑な設定をしなくても綺麗なグラフが沢山用意されている)やcufflinks(時系列データに特化した可視化ライブラリ)等の可視化の方法がある。

```
import seaborn as sns
import cufflinks as cf
import plotly
```

4.4. 簡単な正規化の例

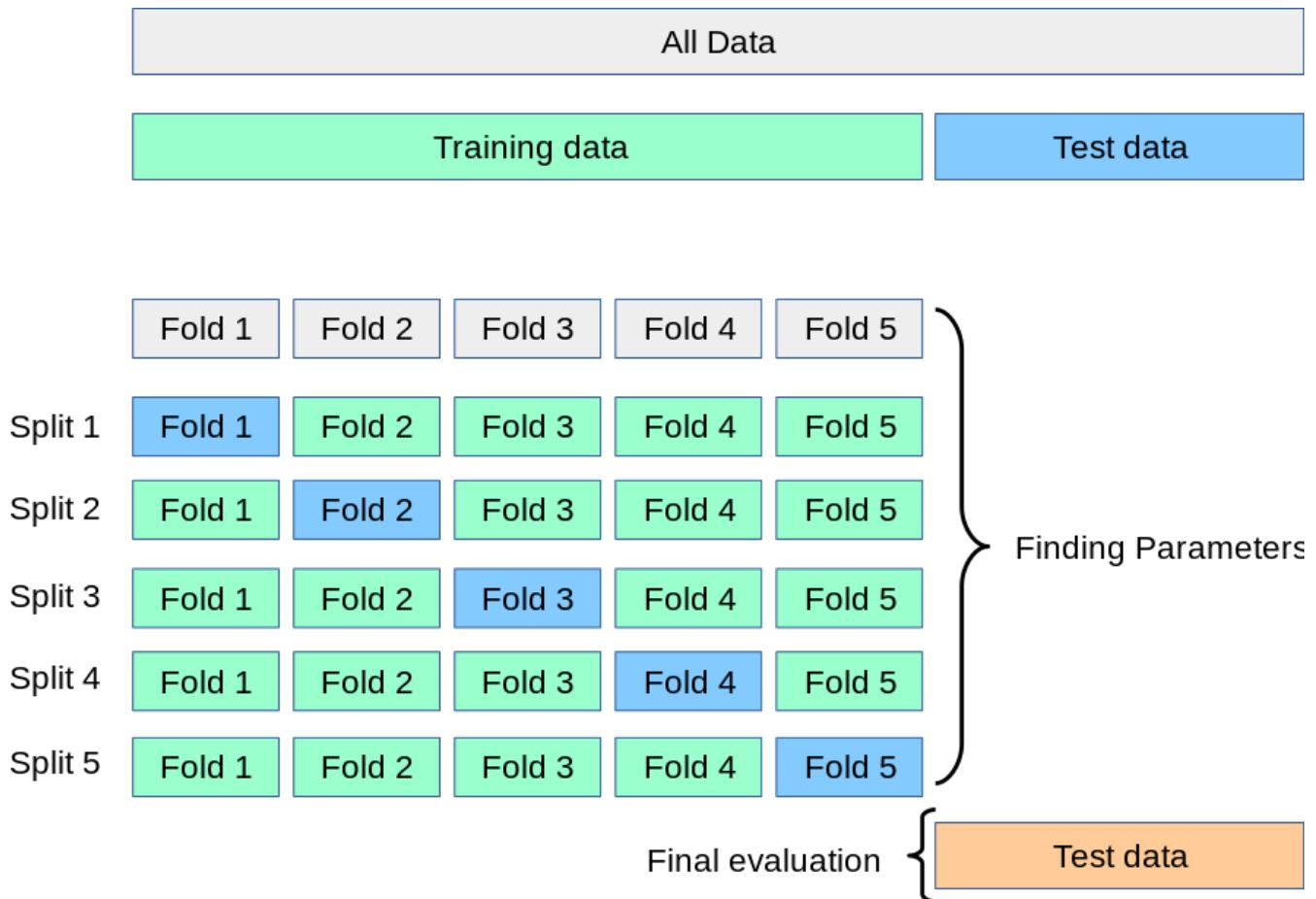
ここでは、`sklearn.preprocessing` を利用した正規化(Standard Scaler)を紹介する。

標準化された値 = (元の値 - 平均) / 標準偏差

という正規化を行う処理であり、変数からデータセット全体の平均を引くことで下の例では、平均値が[0.5, 0.5]、標準偏差が0.5なので、2→3になっている理由は、 $(2-0.5)/0.5=3$ となるからである。

```
>>> from sklearn.preprocessing import StandardScaler
>>> data = [[0, 0], [0, 0], [1, 1], [1, 1]]
>>> scaler = StandardScaler()
>>> print(scaler.fit(data))
StandardScaler()
>>> print(scaler.mean_)
[0.5 0.5]
>>> print(scaler.transform(data))
[[-1. -1.]
 [-1. -1.]
 [ 1.  1.]
 [ 1.  1.]]
>>> print(scaler.transform([[2, 2]]))
[[3. 3.]]
```

4.5. 交差検証



上図のSplit 3の場合、検証期間の一つ前の訓練期間では、将来情報(目的変数:将来リターンetc)を含むデータを学習させているため、検証期間の前半の一部と訓練期間の後半の一部のデータが重なります。ですので、検証期間からデータが重なっている部分を取り除く必要があります。

検証期間の後に将来の情報が含まれていることに関しては、これを含まないような検証方法としてウォークフォワード法が挙げられるが、それは単一のパスに対してのみ検証が行われるため、過学習しやすいです。交差検証で将来の情報を学習させる場合は、検証期間と訓練期間で情報が被っていなければ情報リークにはなりません。

同様に、テスト期間の後半部分であるが、今度はテスト期間後の訓練期間で特徴量の生成方法によってはデータが漏れてしまう可能性があるので、テスト期間の後半部分も取り除く必要があります。

4.6. Colaboratory

Colaboratory (略称: Colab) は、ブラウザから Python を記述、実行できるサービスです。本コンペティションでもチュートリアルのコードはColabの提供をしております。

- Colaboratory へようこそ (<https://colab.research.google.com/notebooks/welcome.ipynb?hl=ja>)

4.7. 参考になるコンペ

本コンペに関連したコンペティションを紹介する

タイトル	概要	URL
Two Sigma: Using News to Predict Stock Movements	ニュースデータを用いた株価の予測	https://www.kaggle.com/c/two-sigma-financial-news
Fintech Data Championship	日本株式のポートフォリオで次の1カ月後に、最も上昇する組み合わせを検討	https://compass.labbase.jp/articles/296
Algorithmic Trading Challenge	投資戦略に関する収益率・取引高等のデータから、各戦略への資金の割り当ての重みを最適化	https://signate.jp/competitions/146
大手ヘッジファンドX: 金融モデリングチャレンジ	独自に導出した様々な特徴量から将来の動きを予想	https://signate.jp/competitions/53
財務・非財務情報を活用した株主価値予測	会計年度2014-2017年の各企業の財務・非財務情報から、会計年度2018年の期末時価総額を予測	https://www.nishika.com/competitions/4/summary
The Winton Stock Market Challenge	過去の株価とマスクされた特徴量から将来の株価を予想	https://www.kaggle.com/c/the-winton-stock-market-challenge
Numerai	マスクされた銘柄・特徴量から週次で予測結果を提出	https://numer.ai/
Quantconnect	条件を満たしたモデルでリターンを追求	https://www.quantconnect.com/competitions
Bloomberg	ESG要素を効果的に投資判断に組み込んでいるかなどの評価	https://www.bloomberg.co.jp/company/stories/investment_contest_2020/
Jane Street Market Prediction	株の取引戦略を採用するかどうかを予測	https://www.kaggle.com/c/jane-street-market-prediction/

4.8. 参考になる書籍

本コンペに参考になる書籍を紹介致します。

タイトル	著者	概要
ファイナンス機械学習—金融市場分析を変える機械学習アルゴリズムの理論と実践	マルコス・ロペス・デ・プラド	機械学習を用いて金融データを分析するまでの知識を網羅的に学ぶことができる。金融のドメイン知識をデータサイエンティストが学ぶには最適な本。
アセットマネージャーのためのファイナンス機械学習	マルコス・ロペス・デ・プラド	ファイナンス機械学習に続き、ノイズ除去、クラスタリング、ラベリング、特徴量の重要度分析などの本コンペでも関連の深いトピックを学ぶことができる。ただし、「ファイナンス機械学習」と比較すると網羅性はないので、2冊目として読むことを推奨。
Kaggleで勝つデータ分析の技術	門脇 大輔 他	モデルのチューニング・アンサンブルなど機械学習のコンペに関連するテクニックを効率的に学ぶことができる。
株を買うなら最低限知っておきたいファンダメンタル投資の教科書 改訂版	足立 武志	ファンダメンタル分析に必要な基礎知識を学ぶことができる。
経済・ファイナンスデータの計量時系列分析	沖本竜義	ARIMAモデルをベースとした古典的な時系列解析を学ぶことができる。ARIMAモデル自体を予測モデルとして使うことは、コンペではまれだが、金融データに対して時系列分析を実施するまでの実務上の課題なども学ぶことができる。

4.9. ファンダメンタル分析の活用方法

ファンダメンタル分析とは企業の成長性、収益性、割安性、安定性、効率性などを分析し、投資判断などに活用する手法です。ここでは成長性、収益性、割安性、安定性、効率性の代表的な指標を紹介します。

成長性: 成長性とは売上や利益の増加が継続しているかを指し、企業価値の増大に直結する指標です。本コンペのデータでは、純資産や営業利益の上昇率に着目することで、その銘柄の成長性を測ることが出来ます。営業利益の上昇率を計算する場合は、季節性を考慮し、前年同期比と比較することが多いです。また、年間を通して同一の指標で評価を行いたいときは直近4半期のデータで移動平均などを取る方法があります。成長性について利用される様々な指標として売上高増加率、経常利益増加率、営業利益増加率、総資本増加率、純資本増加率、従業員増加率、一株当たり当期純利益（EPS）（成長性分析で企業の成長度を測る 知っておくべき指標や分析方法 より引用 <https://keiei.freee.co.jp/articles/c0201686>)などがあります。なお、過去の成長性を解析することはできますが、その成長性が長期間に渡って継続するかを予測することは、様々な要因が関係するため難しい問題です。また、株価そのものではなく成長性自体をモデルの予測対象にして、その予測に基づき投資を行うスタイルも存在します。

収益性: 収益性は営業利益率や経常利益率を計算します。それぞれ営業利益、経常利益を売上で割ることで計算されます。営業利益率や経常利益率が高い企業は優れたビジネスモデルを保持していたり、販管費を低く保つオペレーションが徹底されてたりすることが多く優良企業を判断するまでの指標となっています。そして、この収益性の変化率を成長性と扱うこともできます。

一般的に投資家が期待する収益性は、業種において大きく異なることに注意してください。例えば、情報通信

と食品に期待される営業利益率は大きく異なることが想定されます。この場合、銘柄情報にはセクター情報が含まれているため、セクターの平均の営業利益率を計算し、その差分を計算することでセクター平均からの上振れ、下振れを特徴量にすることができます。

割安性: 株式が割安であるとは、その企業の株価が企業価値と比較して安いということです。代表的な指標としてはPBRが挙げられます。この指標は、企業の純資産と発行済み株式数を割って、1株当たりの純資産を計算します。そして、現在の株価をこの1株当たりの純資産で割ってPBRが求めます。この指標が1を割っているのは、その企業を解散すれば保持する資産で利益が計上されるような状態を指します。PBRが高い銘柄をグロース銘柄、低い銘柄をバリュー銘柄として扱いそれぞれ異なる特性を持った銘柄として分析することもあり、企業の状態を知る上で重要な指標です。

現状の日本マーケットにおいては、PBRが1未満の銘柄が数多くあります(Yahooファイナンス 低PBRランキングより引用 <https://info.finance.yahoo.co.jp/ranking/?kd=12>)。これらの極めて割安な銘柄はディープバリュー株と呼ばれています。PBRが低い理由として以下のような理由も考えられますので、PBRをもって一概に割安銘柄とするのではなく、各企業のその他の決算情報を参考するなど、複数の情報を考慮することも重要です。

- ・ 株式市場全体が調整局面にあるなどの理由により、企業実態より株価が売り込まれている。
- ・ 含み損の実現や業績の悪化による純資産の減少を株価が先取りして織り込んで下落している。
- ・ 不人気のため安値に放置されている。

安定性: 安定性は大型株や安定性が重視される金融・銀行のような特定のセクターで重視される指標です。安定性を計算する代表的な指標は「自己資本比率」です。「自己資本比率」は総資本における自己資本の割合を計算します。自己資本比率が高ければ、自己資本が多い、つまり返済義務のないお金を潤沢に持っているということになるので「中長期的に見て倒産しにくい会社」ということができ(自己資本比率 | 会社経営の「安全性」をあらわす指標 より引用 <https://advisors-freee.jp/article/category/cat-big-03/cat-small-08/9011/>)株式の中長期保有を行う上で倒産リスクを減らすための重要なチェックポイントとなっています。

ROEの計算では、当期純利益と自己資本の割合を見ますが、自己資本が低い株はROEが高くなるので注意が必要です。つまり、借り入れを増やしリスクを取っている銘柄ではROEが高くなりやすく、安定性が低くなる可能性があります。また、小型株においては安定性よりも成長性を重視することが多いことに注意してください。特定のセクターや特定の領域において重要視される指標をモデルに投入する場合、セクターの情報を特徴量として投入するなど、一緒に銘柄を分類できる情報をモデルに投入することでそれらの特性を学ぶことができます。

効率性: 効率性は、近年注目されているROEやROAから計算されるファンダメンタルです。ROEは純利益 ÷ 総資産 × 100として計算され、自己資本（純資産）に対してどれだけの利益が生み出されたのかを示します。ROAは借り入れなどを含む総資産を使ってどれだけ利益を生み出したかを表す指標です。ROE/ROAは効率性を示す指標として注目されており、政府が2017年に公表した成長戦略「未来投資戦略2017」において「《KPI》大企業（TOPIX500）のROAについて、2025年までに欧米企業に遜色のない水準を目指す。」(未来投資戦略2017 より引用 https://www.kantei.go.jp/jp/singi/keizaisaisei/pdf/miraitousi2017_sisaku.pdf)というKPIが設定され注目を浴びています。ROEが高い銘柄ほど効率性が高いと考えることができ、企業価値を高めるための施策としてROEの向上、または維持を目標とする企業が多く、ROE重視の流れの背景は「投資指標としてのROE」(https://www.tr.mufg.jp/houjin/jutaku/pdf/u201503_1.pdf)で詳しく解説されています。株式投資は同一のお金を投資するなら、どの会社に投資するのが最もうまく活用して企業価値を増できるか、ということを探す側面を持っています。効率性はお金を上手く活用する企業を探す指標として利用することができます。

上記は本コンペティションで提供されるデータで計算可能ですので、様々なファンダメンタルを勉強することは特徴量設計の次の一步に繋がるでしょう。

4.10. テクニカル分析の活用方法

テクニカル分析は将来の株価の変化を過去に発生した価格や出来高等の時系列パターンから予想・分析しようとする手法であり、メジャーな分析手法の一つです。テクニカル分析にはトレンド分析、オシレーター分析、フォーメーション分析、ローソク足分析(テクニカル指標一覧より引用 <https://info.monex.co.jp/technical-analysis/indicators/>)があり、2章においてもオシレーターの代表的な分析手法の一つである「移動平均乖離率」を特徴量の一つとして採用しています。

ここでは、テクニカル分析を更に活用するための注意事項を説明します。テクニカル分析を特徴量として採用する場合、そのテクニカル分析が新規の情報をすでに投入済みの特徴量と比較して、どの程度保持しているか、という観点から考えることが重要です。

例えば移動平均と移動平均乖離率はそれぞれ似たような情報を保持していることが容易に想像ができます。特徴量として考えた場合「20日移動平均乖離率」の方が0平均となるため定常性を仮定することができ、扱いやすいことから移動平均乖離率を2章では採用しています。ストキャスティクス(詳細は <https://info.monex.co.jp/technical-analysis/indicators/006.html> を参照)やRSI(詳細は <https://info.monex.co.jp/technical-analysis/indicators/005.html> を参照)のようなオシレーター系の分析も似通った情報を保持していることが推測できます。

機械学習のモデル構築において、若干のパラメータを変更したテクニカル分析を複数個特徴量として投入することは殆どの場合、良い結果に結びつきません。パラメータ違いのテクニカル分析や同一種類のテクニカル分析は、ほぼ同一の情報を保持していることが多く、複数個特徴量を投入しても、パフォーマンスが向上する新規の情報を発見することが難しいためです。また、時系列解析はサンプル数が限られていることが多く、特徴量の種類を多くすると学習に必要な十分なデータを確保できないため、無闇矢鱈に特徴量を増やすべきではなく、特徴量自体にパフォーマンス向上に結びつく新しい情報が含まれることを重視しましょう。

例えば、2章にあるような移動平均乖離率と標準偏差の組み合わせは、お互いに直近時系列に対してトレンドとボラティリティという異なる情報となるためパフォーマンス向上に期待が持てます。テクニカル分析は順張り系、逆張り系のような系統でくくることができる(第1回 数多くあるテクニカル指標を体系的に解説より引用 <https://kabu.com/investment/guide/technical/01.html>)のでそれらの系統の中から選択したり、異なる系統として時間時系列に対して依存しない手法、例えば、過去の高値圏にどれだけ近づいているかを計算するような方法なども考えられます。

4.11. ファクター分析の活用方法

ファクター分析とは、投資が産み出すリターンを説明するファクターを定義しそのファクターの挙動から分析を行う手法です。例えば、ファーマ-フレンチの3ファクターモデルでは株式投資が産み出すリターンをリスクプレミアム、時価総額リスクファクター、簿価時価比率リスクファクターの3要素に分解してます。(Wikipedia ファーマ-フレンチの3ファクターモデルより引用 <https://ja.wikipedia.org/wiki/%E3%83%95%E3%82%A1%E3%83%BC%E3%83%9E-%E3%83%95%E3%83%AC%E3%83%B3%E3%83%81%E3%81%AE%E3%83%95%E3%82%A1%E3%82%AF%E3%82%BF%E3%83%BC%E3%83%A2%E3%83%87%E3%83%AB>)

株式投資におけるファクター分析を知っておくことは、どのような特徴量を設計するかという考察を行うために役立ちます。

例えば、主要なファクターである時価総額について考えてみましょう。本コンペで利用できるデータを利用し、株価と発行済株式数から各銘柄の時価総額を計算することができます。一般的にマーケットは小型株が優位なときもあれば、大型株が優位なときもありますからどちらに投資するべきかをかんたんに決めるることはできません。しかし、この時価総額が株価のリターンの説明要因の一つであるということを知っていれば、少なくとも特徴量に追加することで何らかの学習を行える可能性があると考える事ができ、時価総額が入力データに含まれていない場合、モデルは時価総額ファクターに関連した相場変動を理解することが難しくなりますので、特徴量として投入するべきだらうことが推測されます。

また、近年新たな投資手法として現れたスマートベータ投資もこのファクター分析を利用したアプローチの一つであり、ファクター分析は新しい投資手法を考える上で、基礎知識の一つになっています。スマートベータ投資とは特定のファクターをベンチマークとして制御する手法であり、高配当、バリュー、低リスク、最小分散、クオリティ、モメンタム、これらの組み合わせなどが制御するファクターの候補となっています(スマートベータとリターン特性についてより引用 <https://www.mizuho-ir.co.jp/publication/report/2020/fe36.html>)。

ファクター分析で扱われているファクターはリターンの要因として定義され、株価にも何らかの影響力があると考えられたものです。様々なファクターについて勉強することが特徴量設計の次の一步に繋がるでしょう。

4.12. 複数個のモデルの出力をアンサンブルするアプローチ

アンサンブルの活用について説明します。機械学習における「アンサンブル」とは、複数のモデルを組み合わせることでパフォーマンスの高いモデルを作成する手法を意味します。アンサンブルに使うモデルは多様性があればある程基本的には好ましく、多様性のあるモデルを作り、最終的にアンサンブルすることで、単一モデルでは達成できないパフォーマンスを最終的に達成することができる可能性があります。

アンサンブルにはシンプルに複数のモデルの出力の平均を取るモデル平均法、scikit-learnライブラリの StandardScaler(<https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>)による標準化を実施して分布をある程度揃えてから足し合わせる方法などがあります。また、より高度な手法であるスタッキングもscikit-learnライブラリには StackingClassifier/StackingRegressorがあるため容易に実施(スタッキングで分類・回帰(scikit-learn) より引用 <https://qiita.com/maskot1977/items/de7383898123fa378d86>)することができます。kaggleで活用されている様々なアンサンブルの方法がKAGGLE ENSEMBLING GUIDE(<https://mlwave.com/kaggle-ensembling-guide/>)では紹介されており、英文ですが読む価値があります。

ここでは、時価総額に着目して学習の対象を分けて複数個のモデルを作るアプローチを考えてみます。一般的に時価総額が低い銘柄は大きな値動きが発生しやすいことが知られています。これは市場の流動性に違いがあり、同一額の投資が発生してもより小型株のほうがインパクトが大きいのです。今回のコンペティションでは全銘柄を予測対象としますが、実際にモデルを作ると全銘柄の時価総額の下位の銘柄を学習対象から除外するとよりモデルのパフォーマンスが高くなる現象が発生することがあります。これは時価総額が低い銘柄はよりランダム性が高く、またファンダメンタルに従わないことが多いのです。

では以下の2つのモデルをつくった場合を考えています。

A: 全銘柄を対象に学習を実施したモデル

B: 時価総額の下位の銘柄を学習から除外した上でパフォーマンスをチューニングしたモデル

この2つが異なることを学習したとした場合、Bのモデルの出力に対してStandardScalerによる標準化を実施してAのモデルの出力に足し合わせると、Bのモデルが学んだことをBの学習対象の銘柄に足し合わせることができます。Bの出力に対してはStandardScalerによる標準化を実施しているため分布が0平均となっており、学習対象外の銘柄に対する影響も限定することができます。

異なる特性をもったデータを学習させることで、モデルが異なることを学び、結果的にモデル同士の出力の相関が低くなります。モデル間の出力の相関が低いもの同士をアンサンブルさせると、相関の高いもの同士のときよりアンサンブルの効果は高く出ることが多いため、上記のように学習対象を変えることで複数のモデルを作り、アンサンブルを実施するようなアプローチは有用と考えることができます。

4.13. プライベート期間の性能の向上のために考慮すべきこと

金融時系列はサンプル数が少ないため、特徴量を大量に生成し特徴量選択をすべてモデルに任せるアプローチを採用する場合、ライブ性能の劣化に注意する必要があります。特徴量をどんどん増やすと訓練期間のパフォーマンスは伸びても、評価期間のパフォーマンスの向上が止まることがあります。100種類のデータを使うモデルと10種類のデータを使うモデルが、訓練期間で同一の精度を達成した場合、評価期間・ライブ期間では少ないデータを使ったモデルの方が高い性能を期待できる可能性が高いことが経験的にわかっています。これは、説明変数の取りうるパターンが多いと未来で同じ現象を発生する確率が下がっていくためと考えられます。モデルの複雑さとデータへの適合度とのバランスを取るためにオッカムの剃刀(Wikipedia オッカムの剃刀より引用 <https://ja.wikipedia.org/wiki/%E3%82%AA%E3%83%83%E3%82%AB%E3%83%A0%E3%81%AE%E5%89%83%E5%88%80>)的な発想が必要となります。

4.14.

本コンペでは利用できないが、モデルを将来的に発展させるために検討する価値のある外部データ

日本株のデータ以外に特徴量設計に利用可能と思われるデータを紹介します。本コンペではあらかじめ定められたデータしか利用することはできませんが、今後手元で新たなモデルを作成するときの知識としてご活用下さい。なお、日本でも証券会社がAPIで個人にデータ提供を開始(kabu STATION API より引用 <https://kabu.com/company/lp/lp90.html>)する事例がでてきました。利用時は各社の利用規約をよく読んでから利用するようにしてください。

為替データ: ドル円の値動きが株式市場に影響を与えることはよく知られています。特に日本市場の株式は輸出関連銘柄が多いため、ドル円の値動きは直接収益に関連します。よって、為替データを説明変数に採用するアプローチも想定されます。為替データはいくつかのFX会社がAPIを提供しています。

金利データ: 金利データとは例えば米国債10年金利のような各国の債券の金利を指します。一般的に機関投資家は債券と株式の両方を投資対象とするため、債券市場の動きは株式市場に反映されます。こちらはAPIによるデータ習得は容易ではありませんが、そこまで数は多くないため、説明変数として利用することは難しくありません。

米国株データ: Alpaca US(<https://alpaca.markets/data>)やIEX Cloud(<https://iexcloud.io/>)など様々な米国株の株価/ETFのデータを無料、もしくは格安で提供するサービスが存在します。日本株式のマーケットは米国株式のマーケットクローズ後にオープンするため米国市場が日本市場に与える影響を解析することで様々な情報を分析することができます。

他にも政府が発表するGDPなどの各国の経済動向、金などのコモディティ市場も密接に株式市場の長期トレンドの形成に関わっていますので、様々な外部データの利用を検討することで新たなモデルを作ることができる可能性があります。

4.15. モデルの再学習の運用について

長期間に渡りモデルを運用しようとする場合、再学習の運用を考慮しておく必要があります。再学習の運用とは、2019年までのデータを用いてモデルを学習し2020年で運用した場合、2020年末に2020年のデータを用いて再学習を行うのようなことを指します。再学習を行うにあたっていくつか考慮しておくべきポイントを紹介します。

4.15.1. 再学習を行う必要があるほど新規のデータが存在するか

再学習は新規のデータが溜まった時に実施します。モデルにも依存しますが、モデルが挙動を変えるには既存の学習に用いたデータ量と比較してある程度の量の新規データが存在しないと再学習を実施してもモデルの挙動は変わりませんこの新規データ量の割合はモデルによって変わりますが筆者の感覚では5%以上の新規データ量がないとモデルはほぼ同じ挙動をすると感じています。毎日モデルをトレーニングするような運用も可能ですが、実質サンプル数が極少量増えただけではほぼ同じ挙動となるでしょう。

4.15.2. 再学習を行うことでモデルの出力分布が変わらないか

こちらはシステムで運用する場合に発生しやすい問題です。例えば投資において購入タイミングをモデルの出力が0.8以上のようなしきい値を用いた実装をしていると、再学習を行うことでモデルの出力が変わると、想像以上に購入したり、逆に購入を実施しなくなったりします。再学習を行った場合、直近数週間程度のデータは利用せずに古いモデルと新しいモデルで出力を比較し、出力の分布が変わっていないことを確認するのが良いでしょう。

- LSTMMorCNNによる株価予測
- 欠損値補完の方法のその他の方法（平均値埋めやリストワיז法、多重代入法）
- ホールドアウト検証以外の検証（k-fold）について
- XBRLについて触れる（優先度低、簡単な説明と参考記事の紹介）

(2月以降ニュース問題の際に書いてもらうもの)

- 形態素解析、言語コーパス
- 適時開示
- 発注時の売買手数料

- ・投資系アプリの紹介 ferociなど

5. J-QuantsAPI

5.1. 概要

ここでは、各種データをダウンロードできるJ-QuantsAPIについてご紹介します。APIの詳細な仕様はこちら("https://jpx-jquants.com/apidoc.html")をご確認ください。

5.2. APIの利用

APIを利用するには、SIGNATEでのコンペティションへのご登録("https://signate.jp/competitions/423")とJ-QuantsAPIの利用登録("https://jpx-jquants.com/")が必要となります。

5.3. 必要なパッケージのインポート

```
import os
import json
import requests
import base64
```

パッケージ名	目的
os	ディレクトリ、ファイル操作のため
json	レスポンスの加工のため
requests	APIのGETやPOSTを利用するため
base64	TDnetのファイルダウンロードAPIでBase64形式で返ってくるデータをデコードするため

5.4. Refresh API

初めにidTokenをリフレッシュするRefresh API("/refresh")をご紹介します。J-Quantsのログイン後の画面でご確認いただくことができるrefreshTokenを使用します。Refresh APIを使うためのサンプルコードは以下のようになります。

```

def call_refresh_api(refreshtoken: str):
    """
    idTokenをリフレッシュするメソッド。

    Parameters
    -----
    refreshtoken : str
        refreshtoken。ログイン後の画面からご確認いただけます。

    Returns
    -----
    resjson : dict
        新しいidtokenが格納されたAPIレスポンス(json形式)
    """
    headers = {"accept": "application/json"}
    data = {"refresh-token": refreshtoken}

    response = requests.post(
        "https://api.jpx-jquants.com/refresh", headers=headers, data=json.dumps(data)
    )

    resjson = json.loads(response.text)
    return resjson

```

このAPIを使うことで新しいidtokenを払い出すことができます。使用例は以下の通りです。

```

refreshtoken = <Your refreshtoken>
call_refresh_api(refreshtoken)

```

以下のようなレスポンスが返ります。なお、以下のレスポンスからもわかるとおり、idtokenの有効期限は1時間（3600sec）となっております。

```
{"idToken": "<Your New idtoken>",
"expiresIn": 3600}
```

5.5. 共通で使用するメソッド

TDnetのファイルダウンロードAPI以外は共通のメソッドで使用することができます。サンプルコードは以下の通りです。引数"apitype"に各APIを指定することでコールできます。

```

def call_jquants_api(params: dict, idtoken: str, apitype: str, code: str = None):
    """
    J-QuantsのAPIを試すメソッド。

    Parameters
    -----
    params : dict
        リクエストパラメータ。
    idtoken : str
        idTokenはログイン後の画面からご確認いただけます。
    apitype: str
        APIの種類。“news”, “prices”, “lists”などがあります。
    code: str
        銘柄を指定するAPIの場合に設定します。

    Returns
    -----
    resjson : dict
        APIレスポンス(json形式)
    """

    datefrom = params.get("datefrom", None)
    dateto = params.get("dateto", None)
    date = params.get("date", None)
    includedetails = params.get("includedetails", "false")
    keyword = params.get("keyword", None)
    headline = params.get("headline", None)
    paramcode = params.get("code", None)
    nexttoken = params.get("nexttoken", None)
    headers = {"accept": "application/json", "Authorization": idtoken}
    data = {
        "from": datefrom,
        "to": dateto,
        "includeDetails": includedetails,
        "nexttoken": nexttoken,
        "date": date,
        "keyword": keyword,
        "headline": headline,
        "code": paramcode,
    }

    if code:
        code = "/" + code
        r = requests.get(
            "https://api.jpx-jquants.com/" + apitype + code,
            params=data,
            headers=headers,
        )
    else:
        r = requests.get(
            "https://api.jpx-jquants.com/" + apitype, params=data, headers=headers
        )
    resjson = json.loads(r.text)
    return resjson

```

5.6. Stock List API

銘柄一覧を取得するAPIについて紹介いたします。このAPIでは、企業名や業種区分などの基本情報を取得することができます。全銘柄の一覧を取得する"/lists"と銘柄コードを指定した"/lists/{code}"が利用できます。

```
idtk=<your idtoken>
# Codeを指定しない場合
paramdict = {}
paramdict["includedetails"] = "True"
call_jquants_api(paramdict, idtk, "lists")

# Codeを指定する場合
paramdict = {}
paramdict["includedetails"] = "True"
call_jquants_api(paramdict, idtk, "lists", "8697")
```

レスポンスは以下のようになります。

```
{"list": [{"33 Sector(name)": "Other Financing Business",
  "Effective Date": "20201230",
  "prediction_target": "True",
  "Section/Products": "First Section (Domestic)",
  "33 Sector(Code)": 7200.0,
  "Name (English)": "Japan Exchange Group, Inc.",
  "IssuedShareEquityQuote IssuedShare": 536351448.0,
  "Local Code": "8697"}]}
```

5.7. Price API

株価情報を取得するPrice APIをご紹介します。検索期間や銘柄コードを指定することで、四本値、売買高、前日比変化率などを取得することができます。銘柄コードを指定する場合は"/prices/{code}"でAPIをご利用ください。"includeDetails"をTrueにした場合は、全てのデータ系列を取得します。詳細は、<https://jpql.jquants.com/apidoc.html>をご確認ください。

```
idtk=<your idtoken>
# Codeを指定しない場合
paramdict = {}
paramdict["date"] = "2020-12-30"
paramdict["includedetails"] = "True"
call_jquants_api(paramdict, idtk, "prices")

# Codeを指定する場合
paramdict = {}
paramdict["datefrom"] = "2020-01-17"
paramdict["dateto"] = "2020-01-31"
paramdict["includedetails"] = "True"
call_jquants_api(paramdict, idtk, "prices", "8697")
```

レスポンスのサンプルは以下の通りです。

```
{"prices": [{"EndOfDayQuote Open": 2005.0,  
"EndOfDayQuote PreviousClose": 1972.0,  
"EndOfDayQuote CumulativeAdjustmentFactor": 1.0,  
"EndOfDayQuote VWAP": 1994.792,  
"EndOfDayQuote Low": 1989.0,  
"EndOfDayQuote PreviousExchangeOfficialClose": 1972.0,  
"EndOfDayQuote High": 2008.0,  
"EndOfDayQuote Date": "2020/01/20",  
"EndOfDayQuote Close": 1990.0,  
"EndOfDayQuote PreviousExchangeOfficialCloseDate": "2020/01/17",  
"EndOfDayQuote ExchangeOfficialClose": 1990.0,  
"EndOfDayQuote ChangeFromPreviousClose": 18.0,  
"EndOfDayQuote PercentChangeFromPreviousClose": 0.913,  
"EndOfDayQuote PreviousCloseDate": "2020/01/17",  
"Local Code": "8697",  
"EndOfDayQuote Volume": 528600.0},  
{ "EndOfDayQuote Open": 1989.0,  
"EndOfDayQuote PreviousClose": 1990.0,  
"EndOfDayQuote CumulativeAdjustmentFactor": 1.0,  
"EndOfDayQuote VWAP": 1976.539,  
"EndOfDayQuote Low": 1965.0,  
"EndOfDayQuote PreviousExchangeOfficialClose": 1990.0,  
"EndOfDayQuote High": 1995.0,  
"EndOfDayQuote Date": "2020/01/21",  
"EndOfDayQuote Close": 1977.0,  
"EndOfDayQuote PreviousExchangeOfficialCloseDate": "2020/01/20",  
"EndOfDayQuote ExchangeOfficialClose": 1977.0,  
"EndOfDayQuote ChangeFromPreviousClose": -13.0,  
"EndOfDayQuote PercentChangeFromPreviousClose": -0.653,  
"EndOfDayQuote PreviousCloseDate": "2020/01/20",  
"Local Code": "8697",  
"EndOfDayQuote Volume": 571000.0},  
... ]}
```

5.8. Stock Fin API

最後に各銘柄の財務諸表データを取得するAPIをご紹介します。特定の日の全銘柄の情報を取得する"/stockfins"と1銘柄の情報を取得する"/stockfins/{code}"がございます。

```

idtk=<your idtoken>
# Codeを指定しない場合
paramdict = {}
paramdict["date"] = "2020-12-30"
paramdict["includedetails"] = "True"
call_jquants_api(paramdict, idtk, "stockfins")

# Codeを指定する場合
paramdict = {}
paramdict["datefrom"] = "2020-01-01"
paramdict["dateto"] = "2020-12-30"
paramdict["includedetails"] = "True"
call_jquants_api(paramdict, idtk, "stockfins", "8697")

```

レスポンスは以下のようになります。

```
{
  "stockfin": [
    {
      "Result_FinancialStatement": {
        "TotalAssets": 56671198.0,
        "base_date": "2020/01/30",
        "Result_FinancialStatement": {
          "FiscalPeriodEnd": "2019/12",
          "ReportType": "Q3",
          "OrdinaryIncome": 48586.0,
          "CashFlowsFromOperatingActivities": "",
          "Local Code": "8697",
          "NetSales": 87433.0,
          "CashFlowsFromFinancingActivities": "",
          "CashFlowsFromInvestingActivities": "",
          "AccountingStandard": "ConsolidatedIFRS",
          "NetIncome": 33317.0,
          "OperatingIncome": 48176.0
        },
        "Result_FinancialStatement": {
          "TotalAssets": 56671198.0,
          "base_date": "2020/03/23",
          "Result_FinancialStatement": {
            "FiscalPeriodEnd": "2019/12",
            "ReportType": "Q3",
            "OrdinaryIncome": 48586.0,
            "CashFlowsFromOperatingActivities": "",
            "Local Code": "8697",
            "NetSales": 87433.0,
            "CashFlowsFromFinancingActivities": "",
            "CashFlowsFromInvestingActivities": "",
            "AccountingStandard": "ConsolidatedIFRS",
            "NetIncome": 33317.0,
            "OperatingIncome": 48176.0
          }
        }
      }
    }
  ]
}
```

5.9. Stock Labels

基準日から一定期間の株価の最大上昇率、最大下落率を取得するAPIをご紹介します。Stock Labels APIは期間や銘柄コードを指定することで該当する株価騰落率のデータを取得できます。

```
# Codeを指定しない場合
paramdict = {}
paramdict["date"] = "2018-05-31"
paramdict["includedetails"] = "true"
call_jquants_api(paramdict, idtk, "stocklabels")

# Codeを指定する場合
paramdict = {}
paramdict["from"] = "2020-02-01"
paramdict["to"] = "2020-02-28"
paramdict["includedetails"] = "true"
call_jquants_api(paramdict, idtk, "stocklabels", "1301")
```

レスポンスのサンプルは以下の通りです。

```
{"labels": [{"label_low_10": -0.01748,
  "label_low_20": -0.10699,
  "label_low_5": 0.0021,
  "label_high_20": 0.02203,
  "base_date": "2020-02-04",
  "label_high_10": 0.02203,
  "label_date_5": "2020-02-12",
  "label_date_10": "2020-02-19",
  "label_high_5": 0.02203,
  "label_date_20": "2020-03-05",
  "Local Code": "1301"}, {"label_low_10": -0.02507,
  "label_low_20": -0.11072,
  "label_low_5": -0.00557,
  "label_high_20": 0.01776,
  "base_date": "2020-02-05",
  "label_high_10": 0.01776,
  "label_date_5": "2020-02-13",
  "label_date_10": "2020-02-20",
  "label_high_5": 0.01776,
  "label_date_20": "2020-03-06",
  "Local Code": "1301}], "scrollId": "eyJMb2NhbCBDb2RlIjogIjEzMDEiLCAiYmFzZV9kYXR1IjogIjIwMjAtMDItMDUiQ=="}]
```

5.10. News API

News APIは現在公開しておりません。ヘッドライン、キーワード、期間などで該当するニュース記事データを検索できる予定です。

レスポンスのサンプルは以下の通りになる予定です。

```
{"news": [{"article_id": "TDSKDBDGXLASF21HM9_21022020000000",  
"publish_datetime": "2020-02-21T16:34:00Z",  
"media_code": "TNY",  
"media_name": "日本経済新聞電子版",  
"men_name": "",  
"headline": "日本取引所C E O、東商取のエネルギー市場「早期に統合したい」",  
"keywords": "最高経営責任者\n東京商品取引所\n日本取引所グループ\n清田瞭\nエネルギー市場  
\n大阪取引所\n統合\n定例\n早期",  
"classifications": "T 8 6 9 7 \nP D 5 2 1 \nN 0 0 4 0 4 3 1 \nN 0 0 7 5 1 0 7  
\nN 0 0 4 0 7 7 9",  
"stock_code": "8697"}, {"scrollId": "FGluY2x1ZGVfY29u"}]
```

5.11. TDnet API

適時開示を取得できるTDnetAPIは現在公開しておりません。レスポンスのサンプルは以下のようになる予定です。このAPIを利用することでbase64形式のデータをファイルに書き出すことも可能です。

```
{"tdnet": [{"pdfSumaryFlag": "1",  
"modifiedHistory": "1",  
"name": "J P X",  
"disclosureItems": ["11384"],  
"code": "86970",  
"disclosedDate": "2020-01-30",  
"datetime": "2020-01-30:12:00:00",  
"handlingType": None,  
"disclosedTime": "12:00:00",  
"pdfGeneralFlag": "1",  
"disclosureNumber": "20200129453073",  
"xbrlFlag": "1",  
"title": "2020年3月期 第3四半期決算短信〔I F R S〕(連結) "}, {"pdfSumaryFlag": "0",  
"modifiedHistory": "1",  
"name": "J P X",  
"disclosureItems": ["11804"],  
"code": "86970",  
"disclosedDate": "2020-01-30",  
"datetime": "2020-01-30:12:00:00",  
"handlingType": None,  
"disclosedTime": "12:00:00",  
"pdfGeneralFlag": "1",  
"disclosureNumber": "20200129453074",  
"xbrlFlag": "0",  
"title": "Consolidated financial results for the nine months ended December 31, 2019"}, ...]}
```

6. 参考文献

本文書の手順の実装であるビルドスクリプトやテーマでは次のプロダクトと技術資料が使われています。



プロダクト名の隣にライセンスを併記しました。商用利用等で制限のあるプロダクトはありませんが、それぞれライセンスを確認してください。

Template

- AsciidoctorとGradleでつくる文書執筆環境 - MIT License - <https://h1romas4.github.io/asciidoctor-gradle-template/index.html>

Font

- 源真ゴシック - SIL Open Font License 1.1 - <http://jikasei.me/font/genshin/>
- 源様明朝 - SIL Open Font License 1.1 - <https://github.com/ButTaiwan/genyo-font>
- Ricty Diminished - SIL Open Font License 1.1 - <https://github.com/edihbrandon/RictyDiminished>

Asciidoc

- Asciidoctor - MIT License - <https://asciidoctor.org/>
- Asciidoctorj - Apache License 2.0 - <https://github.com/asciidoctor/asciidoctorj>
- Asciidoctor.js - MIT License - <https://asciidoctor.org/docs/asciidoctor.js/>
- Asciidoctor PDF - MIT License - <https://asciidoctor.org/docs/asciidoctor-pdf/>
- Asciidoctor Gradle Plugin Suite - Apache License 2.0 - <https://github.com/asciidoctor/asciidoctor-gradle-plugin>
- asciidoctor-pdf-linewrap-ja - MIT License - <https://github.com/fuka/asciidoctor-pdf-linewrap-ja>

Build Tool

- SDKMAN - Apache License 2.0 - <https://sdkman.io/>
- Gradle - Apache License 2.0 - <https://gradle.org/>

Text Editor

- Visual Studio Code - Microsoft - <https://code.visualstudio.com/>
- asciidoctor-vscode - MIT License - <https://github.com/asciidoctor/asciidoctor-vscode>

Guide

- asciidoctor-pdfでかっこいいPDFを作る - <https://qiita.com/kuboaki/items/67774c5ebd41467b83e2>

7. ライセンス

本チュートリアルおよびハンズオンのソースコードは [CC BY-NC-ND 4.0](#) に従うライセンスで公開しています。

教育など非商用の目的での本チュートリアルの使用や再配布は自由に行うことが可能です。商用目的で本チュートリアルの全体またはその一部を無断で転載する行為は、これを固く禁じます。

