

今回インストールが必要なライブラリ

ta-libのインストール

- テクニカル指標分析ライブラリ

- Google Colaboratoryの場合

```
!wget http://prdownloads.sourceforge.net/ta-lib/ta-lib-0.4.0-src.tar.gz
!tar -xzf ta-lib-0.4.0-src.tar.gz
%cd ta-lib
!./configure --prefix=/usr
!make
!make install
!pip install Ta-Lib
```

- aconda promptで以下を実行

```
conda install -c conda-forge ta-lib
```

[参考サイト](#)

mplfinanceのインストール

- ロウソク足のチャート作成ライブラリ

- Google Colaboratoryの場合

```
!pip install mplfinance
```

- aconda promptで以下を実行

```
conda install -c conda-forge mplfinance
```

[参考サイト](#)

prophetのインストール

- Facebookが開発した時系列予測ライブラリ

- Google Colaboratoryの場合

```
!pip install fbprophet
```

- aconda promptで以下を実行

- 1. conda install -c anaconda ephem
- 1. conda install -c conda-forge pystan (数分掛かる)
- 1. conda install -c conda-forge fbprophet

pandas_datareaderのインストール

- 日経平均、NASDAQ、個別銘柄の株価データ取得ライブラリ
- Google colaboryの場合
初期設定でインストールされているはずだが、されていなければ以下を実行
pip install pandas-datareader
- Anacondaの場合
pip install pandas-datareader

Google Colaboratoyrへ上記を纏めてインストール

In []:

```
# ta-lib
!wget http://prdownloads.sourceforge.net/ta-lib/ta-lib-0.4.0-src.tar.gz
!tar -xzf ta-lib-0.4.0-src.tar.gz
%cd ta-lib
!./configure --prefix=/usr
!make
!make install
!pip install Ta-Lib

# mplfinance
!pip install mplfinance

# prophet
!pip install fbprophet

# pandas_datareader
!pip install pandas-datareader

# japanize-matplotlib (グラフ日本語対応)
!pip install japanize-matplotlib
```

データ取得

日経平均の取得

- 使用ライブラリ: pandas_datareader
 - 第1引数: 銘柄: 日経225(コード: ^NKX)
 - 第2引数: 取得サイト: stooq(各種経済データを公開しているポーランドのサイト)
 - 第3引数: 開始日: 2019-06-01
 - 第4引数: 終了日: 2020-06-01
- pandas_datareader を用いて、2020/6/1-2021/6/1までの日経平均株価データを取得
- [stooqサイト](#)
- 以下では、df_N225 のデータフレームにデータを代入するものとする

In [4]:

```
# 使用するライブラリ
from pandas_datareader import data
import pandas as pd
import datetime as dt # 日付型データ生成ライブラリ

#-----
# 設定項目: DataReaderの引数
#-----
code = "^NKX" # 銘柄コード
```

```

site = "stooq" # データ取得サイト
start = dt.date(2019, 6, 1) # 開始(年, 月, 日)
end = dt.date(2020, 6, 1) # 終了(年, 月, 日)
#-----

# 取得データをpnadasのデータフレームに挿入
# 時系列を昇順に並び替えるために`sort_index()`を付ける
df_N225 = data.DataReader(code, site, start, end).sort_index()

display(df_N225.head(5)) # 先頭5行分データ表示
display(df_N225.tail(5)) # 末尾5行分データ表示

```

	Open	High	Low	Close	Volume
Date					
2019-06-03	20327.87	20438.03	20305.74	20410.88	695247100
2019-06-04	20435.86	20464.57	20290.16	20408.54	713001500
2019-06-05	20667.89	20800.64	20646.15	20776.10	744705100
2019-06-06	20745.84	20842.28	20745.84	20774.04	614232500
2019-06-07	20859.78	20907.06	20817.07	20884.71	522701300

	Open	High	Low	Close	Volume
Date					
2020-05-26	20927.96	21327.86	20918.11	21271.17	871911300
2020-05-27	21249.31	21473.68	21143.37	21419.23	1120168200
2020-05-28	21612.90	21917.43	21580.50	21916.31	1340312400
2020-05-29	21811.52	21955.44	21710.80	21877.89	1531900700
2020-06-01	21910.89	22161.39	21898.99	22062.39	721657900

- Open : 始値
- High : 高値
- Low : 安値
- Close : 終値
- Volume : 出来高 (取引が成立した株数)

時系列データを扱う場合はインデックス (行名) に年月日が入る (通常は0,1,...)

個別銘柄の株価の取得

- [個別銘柄のコード取得](#)
 - 例: トヨタ自動車 : 7203
 - codeに代入する場合、コードの後ろに .JP を付ける : 7203.JP
- 以下では、df_stock のデータフレームにデータを代入するものとする

In [6]:

```

#-----
# 設定項目 : DataReaderの引数
#-----
code = "7203.JP" # 銘柄コード

```

```

site = "stooq" # データ取得サイト
start = dt.date(2019, 6, 1) # 開始(年, 月, 日)
end = dt.date(2020, 6, 1) # 終了(年, 月, 日)
#-----

# 取得データをpnadasのデータフレームに挿入
df_stock = data.DataReader(code, site, start, end).sort_index()

display(df_stock.head(5)) # 先頭5行分データ表示
display(df_stock.tail(5)) # 末尾5行分データ表示

```

	Open	High	Low	Close	Volume
Date					
2019-06-03	6117.78	6155.61	6087.72	6150.75	5824532
2019-06-04	6158.51	6187.60	6083.84	6175.00	5232695
2019-06-05	6263.24	6344.69	6255.48	6332.09	5796998
2019-06-06	6336.94	6386.39	6319.48	6348.57	4284764
2019-06-07	6387.36	6410.63	6358.27	6388.32	3635589

	Open	High	Low	Close	Volume
Date					
2020-05-26	6440.0	6680.0	6426.0	6649.0	7740600
2020-05-27	6640.0	6775.0	6592.0	6740.0	7699300
2020-05-28	6839.0	6980.0	6826.0	6973.0	9734100
2020-05-29	6900.0	6945.0	6755.0	6755.0	21832300
2020-06-01	6785.0	6820.0	6685.0	6720.0	4506700

練習1

興味のある企業の2019-06-01から2020-06-01までの株価をDataReaderを使って取得してみましょう。

データを挿入するpandasのデータフレーム名(変数名)は `df_stock` とする。

個別銘柄のコードは以下で検索してください。

- [個別銘柄のコード取得](#)

In []:

時系列グラフ作成

日経平均のグラフを作成

In [7]:

```

import matplotlib.pyplot as plt # グラフ作成
import japanize_matplotlib # 日本語対応
%matplotlib inline

```

```

#-----
# 設定項目
#-----
# x軸, y軸に対応するデータを取得
name = "日経平均" # 企業名など
date = df_N225.index # インデックスの日時を取得
price = df_N225['Close'] # 終値を取得
#-----

# グラフ作成前に設定するオプション
plt.figure(figsize=(14, 4)) # 図のサイズ (横×縦)

# グラフ作成
# x軸、y軸、凡例タイトル、線幅、色を指定
plt.plot(date, price, label=name, linewidth = 2, color = 'gray')

# 追加オプション
plt.title(name, fontsize=15) # グラフタイトル
plt.xlabel('日', fontsize=15) # x軸ラベル
plt.ylabel('価格', fontsize=15) # y軸ラベル
plt.xticks(fontsize=15) # x軸の目盛文字サイズ
plt.yticks(fontsize=15) # y軸の目盛文字サイズ
plt.legend(fontsize=15, loc="best") # 凡例
plt.grid() # 補助線追加

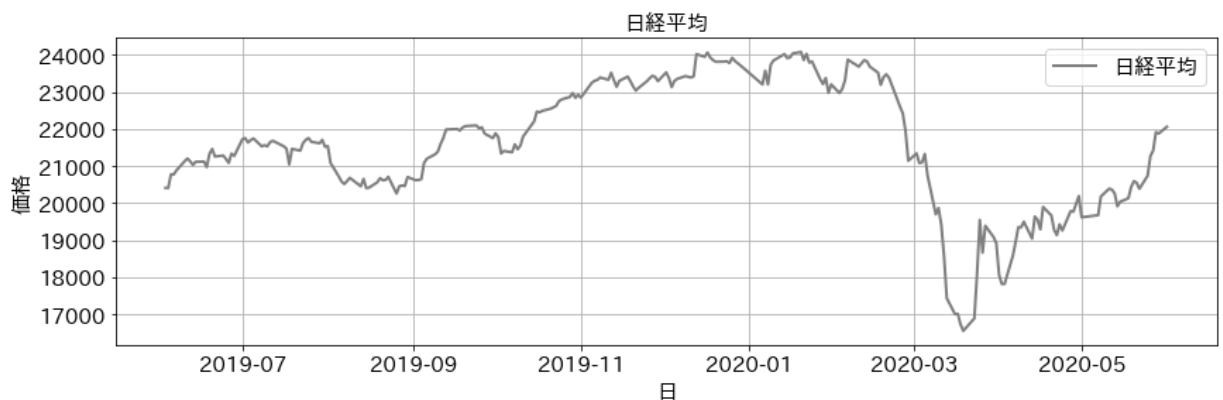
# 出力ファイル名を指定 (Googleドライブの場合)
outfile = "drive/My Drive/Colab Notebooks/05_グラフ_"+name+".png"

# 出力ファイル名を指定 (Anacondaの場合, Google Colaboratoryの場合は削除してください)
outfile = "05_グラフ_"+name+".png"

plt.savefig(outfile, bbox_inches='tight') # 画像の保存

plt.show()

```



練習2

練習1で取得したデータの時系列グラフを作成しよう

In []:

データ抽出

日時のインデックスによる抽出

- データフレーム名["日付"]
- データフレーム名["日付":"日付"]

2020年4月10日から4月15日までのデータを抽出

In [8]:

```
df_N225_term = df_N225["2020-04-10":"2020-04-15"]

display(df_N225_term)
```

	Open	High	Low	Close	Volume
Date					
2020-04-10	19500.07	19500.07	19236.26	19498.50	874941000
2020-04-13	19312.04	19354.50	19015.50	19043.40	656321900
2020-04-14	19150.30	19705.99	19098.52	19638.81	818776500
2020-04-15	19589.25	19659.47	19476.34	19550.09	940973900

曜日ごとの抽出

- データフレーム名.index.day_name() == "曜日名(英語)"
- 曜日名: Sunday (日)、Monday (月)、Tuesday (火)、Wednesday (水)、Thursday (木)、Friday (金)、Saturday (土)

水曜日のみ抽出

In [9]:

```
# 水曜日の行かどうかをTrue(真)かFalse(偽)で判定
TrueFalse = df_N225.index.day_name() == "Wednesday"

# 真の行のみ抽出
df_N225_day = df_N225[TrueFalse]

# 表示
display(df_N225_day)
```

	Open	High	Low	Close	Volume
Date					
2019-06-05	20667.89	20800.64	20646.15	20776.10	744705100
2019-06-12	21130.39	21259.49	21118.75	21129.72	582359200
2019-06-19	21223.17	21358.72	21215.37	21333.87	660883800
2019-06-26	21067.68	21129.04	21035.84	21086.59	519055100
2019-07-03	21684.07	21708.69	21566.82	21638.16	567018300
2019-07-10	21499.46	21601.86	21488.40	21533.48	607163200
2019-07-17	21474.63	21488.24	21380.55	21469.18	549148600
2019-07-24	21726.98	21744.88	21677.71	21709.57	526958200
2019-07-31	21526.38	21589.11	21476.07	21521.53	788832500
2019-08-07	20548.07	20570.19	20406.80	20516.56	758843600
2019-08-14	20669.99	20696.78	20581.17	20655.13	609364000

	Open	High	Low	Close	Volume
Date					
2019-08-21	20489.97	20626.05	20482.89	20618.57	529844100
2019-08-28	20474.31	20511.21	20433.94	20479.42	514888300
2019-09-04	20578.67	20694.35	20554.96	20649.14	515282800
2019-09-11	21466.66	21619.21	21437.82	21597.76	1101417600
2019-09-18	22024.11	22027.86	21942.03	21960.71	693981600
2019-09-25	21961.82	22035.83	21906.15	22020.15	616056300
2019-10-02	21744.62	21795.01	21725.23	21778.61	633258000
2019-10-09	21359.84	21467.77	21359.84	21456.38	554124900
2019-10-16	22479.57	22611.16	22434.35	22472.92	787212900
2019-10-23	22619.77	22648.49	22458.10	22625.38	680221100
2019-10-30	22953.17	22960.73	22829.52	22843.12	1015704900
2019-11-06	23343.51	23352.56	23247.41	23303.82	774062900
2019-11-13	23439.25	23452.63	23270.93	23319.87	634084500
2019-11-20	23176.49	23303.17	23086.31	23148.57	657193900
2019-11-27	23452.85	23507.31	23419.36	23437.77	521840600
2019-12-04	23186.74	23203.77	23044.78	23135.23	581240600
2019-12-11	23420.32	23438.43	23333.63	23391.86	613409000
2019-12-18	24023.27	24046.01	23920.03	23934.43	662677200
2019-12-25	23822.69	23823.61	23782.87	23782.87	315893400
2020-01-08	23217.49	23303.21	22951.46	23204.76	794100900
2020-01-15	23923.48	23997.39	23876.06	23916.58	571553500
2020-01-22	23840.64	24040.87	23837.05	24031.35	489705800
2020-01-29	23309.32	23392.61	23214.67	23379.40	549198400
2020-02-05	23351.47	23414.11	23241.64	23319.56	703366800
2020-02-12	23741.21	23869.36	23693.72	23861.21	746482500
2020-02-19	23329.33	23467.81	23238.98	23400.70	615892700
2020-02-26	22374.14	22456.55	22127.67	22426.19	993403400
2020-03-04	20897.20	21245.48	20862.05	21100.06	946262600
2020-03-11	19721.68	19974.83	19392.92	19416.06	1292294100
2020-03-18	17154.08	17396.84	16698.46	16726.55	1772187500
2020-03-25	18446.80	19564.38	18446.80	19546.63	1476925700
2020-04-01	18686.12	18783.22	17871.62	18065.41	1057750900
2020-04-08	19047.76	19448.78	18730.80	19353.24	1059482100
2020-04-15	19589.25	19659.47	19476.34	19550.09	940973900
2020-04-22	19109.18	19137.95	18858.25	19137.95	764031100

	Open	High	Low	Close	Volume
Date					
2020-05-13	20140.92	20329.27	20056.46	20267.05	795084000
2020-05-20	20454.49	20684.46	20454.49	20595.15	717576200
2020-05-27	21249.31	21473.68	21143.37	21419.23	1120168200

練習4

(1) 練習1で取得したデータで2020年5月1日から5月31日までのデータを抽出してみよう

In []:

(2) 練習1で取得したデータで金曜日のデータを抽出してみよう

In []:

時系列データの集計

- 集計方法

列名.resample("集計単位").agg(["処理1", "処理2", ...])

指定方法	内容	区切り位置	主な処理	内容
AS	年	年初	min	最小
A	年	年末	max	最大
MS	月	月初	sum	合計
M	月	月末	mean	平均
W	週	日曜	first	最初の値
D	日	0時	interpolate	補間
H	時	0分		
T	分	0秒		
S	秒			

- 集計単位

月ごとの平均値、最小値、最大値の集計

- 集計単位: M
- 処理: mean, min, max

In [11]:

```
df_N225_M_agg = df_N225["Close"].resample("M").agg(["mean", "max", "min"])
display(df_N225_M_agg)
```

mean max min

	Date	mean	max	min
	Date			
	2019-06-30	21060.207000	21462.86	20408.54
	2019-07-31	21593.676364	21756.55	21046.24
	2019-08-31	20629.677143	21540.99	20261.04
	2019-09-30	21585.458947	22098.84	20620.19
	2019-10-31	22197.474762	22974.13	21341.74
	2019-11-30	23278.087500	23520.01	22850.77
	2019-12-31	23660.375714	24066.12	23135.23
	2020-01-31	23642.921579	24083.51	22977.75
	2020-02-29	23180.366111	23873.59	21142.96
	2020-03-31	18974.003810	21344.08	16552.83
	2020-04-30	19208.355238	20193.69	17818.72
	2020-05-31	20543.261111	21916.31	19619.35
	2020-06-30	22062.390000	22062.39	22062.39

100日ごとの平均値、最小値、最大値の集計

- 集計単位: 100D
- 処理: mean, min, max

In [12]:

```
df_N225_100D_agg = df_N225["Close"].resample("100D").agg(["mean", "max", "min"])
display(df_N225_100D_agg)
```

		mean	max	min
	Date			
	2019-06-03	21091.125143	21756.55	20261.04
	2019-09-11	22763.274030	24066.12	21341.74
	2019-12-20	22120.391270	24083.51	16552.83
	2020-03-29	19838.536429	22062.39	17818.72

練習5

練習1で取得したデータで週単位の平均値、最小値、最大値を集計してみよう

- 集計単位: W
- 処理: mean, min, max

In []:

移動平均の分析

移動平均の計算

- 移動平均は、ある一定区間ごとの平均値を区間をずらしながら求めたもの。移動平均を用いてグラフを作成すると、長期的な傾向を表す滑らかな曲線が得られる
- 今回は、終値(Close)の5日平均（短期）、25日平均（中期）、50日平均（長期）を求める
- pandasの機能である 列名.rolling(window=日数).mean() を利用

日経平均の移動平均を計算

In [10]:

```
span_short = 5 # 短期用の日数
span_midium = 25 # 中期用の日数
span_long = 50 # 長期用の日数

# 移動平均の結果を代入
df_N225['sma_short'] = df_N225['Close'].rolling(window=span_short).mean()
df_N225['sma_midium'] = df_N225['Close'].rolling(window=span_midium).mean()
df_N225['sma_long'] = df_N225['Close'].rolling(window=span_long).mean()

display(df_N225.head(5)) # 先頭5行を表示
display(df_N225.tail(5)) # 末尾5行を表示
```

	Open	High	Low	Close	Volume	sma_short	sma_midium	sma_long
Date								
2019-06-03	20327.87	20438.03	20305.74	20410.88	695247100	NaN	NaN	NaN
2019-06-04	20435.86	20464.57	20290.16	20408.54	713001500	NaN	NaN	NaN
2019-06-05	20667.89	20800.64	20646.15	20776.10	744705100	NaN	NaN	NaN
2019-06-06	20745.84	20842.28	20745.84	20774.04	614232500	NaN	NaN	NaN
2019-06-07	20859.78	20907.06	20817.07	20884.71	522701300	20650.854	NaN	NaN

	Open	High	Low	Close	Volume	sma_short	sma_midium	sma_long
Date								
2020-05-26	20927.96	21327.86	20918.11	21271.17	871911300	20709.688	20011.2048	19224.4638
2020-05-27	21249.31	21473.68	21143.37	21419.23	1120168200	20874.504	20096.3660	19264.5272
2020-05-28	21612.90	21917.43	21580.50	21916.31	1340312400	21147.304	20177.1280	19331.6608
2020-05-29	21811.52	21955.44	21710.80	21877.89	1531900700	21445.250	20265.4788	19420.5976
2020-06-01	21910.89	22161.39	21898.99	22062.39	721657900	21709.398	20376.7432	19521.8046

移動平均の日数分のデータ集まった時に平均値が計算できるので、それまではNaN(欠損)とな

る

例えば、5日移動平均の場合、5日目から値が得られる

移動平均のグラフ作成

In [12]:

```
#-----
# 設定項目
#-----
name      = "日経平均"          # 銘柄名(任意の名前)
date      = df_N225.index       # インデックスの日時を取得
price     = df_N225['Close']    # 終値を取得
sma_short = df_N225['sma_short'] # 5日移動平均を取得
sma_midium = df_N225['sma_midium'] # 25日移動平均を取得
sma_long  = df_N225['sma_long']  # 50日移動平均を取得
#-----

# グラフ作成前に設定するオプション
plt.figure(figsize=(16, 4)) # 図のサイズ (横×縦)

# グラフ作成
# x軸、y軸、凡例タイトル、線幅、色
plt.plot(date, price, label = name, linewidth = 2, color = 'gray')
plt.plot(date, sma_short, label = '移動平均 (短期)', linewidth = 1, color = 'red')
plt.plot(date, sma_midium, label = '移動平均 (中期)', linewidth = 1, color = 'blue')
plt.plot(date, sma_long, label = '移動平均 (長期)', linewidth = 1, color = 'green')

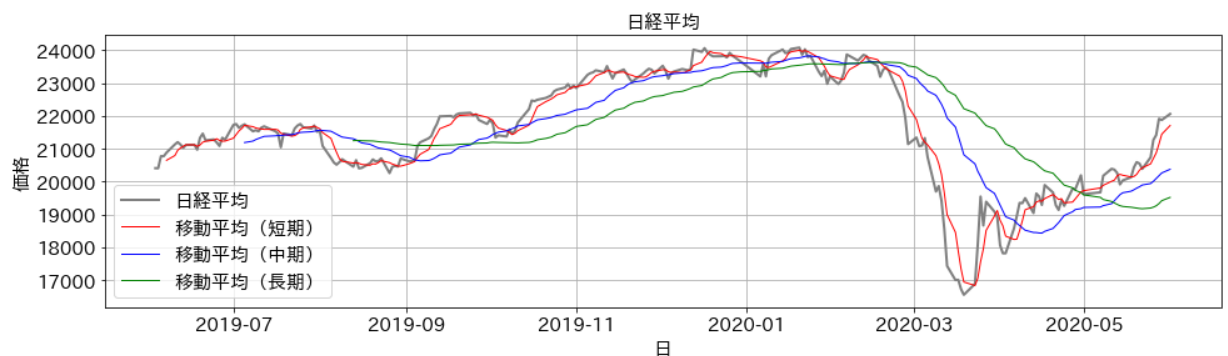
# 追加オプション
plt.title(name, fontsize=15)      # グラフタイトル
plt.xlabel('日', fontsize=15)     # x軸ラベル
plt.ylabel('価格', fontsize=15)  # y軸ラベル
plt.xticks(fontsize=15)          # x軸の目盛文字サイズ
plt.yticks(fontsize=15)          # y軸の目盛文字サイズ
plt.legend(fontsize=15, loc="best") # 凡例
plt.grid()                       # 補助線追加

# 出力ファイル名を指定 (Googleドライブの場合)
outfile = "drive/My Drive/Colab Notebooks/05_グラフ_"+name+"_移動平均.png"

# # 出力ファイル名を指定 (Anacondaの場合, Google Colaboratoryの場合は削除してください)
outfile = "05_グラフ_"+name+"_移動平均.png"

plt.savefig(outfile, bbox_inches='tight') # 画像の保存

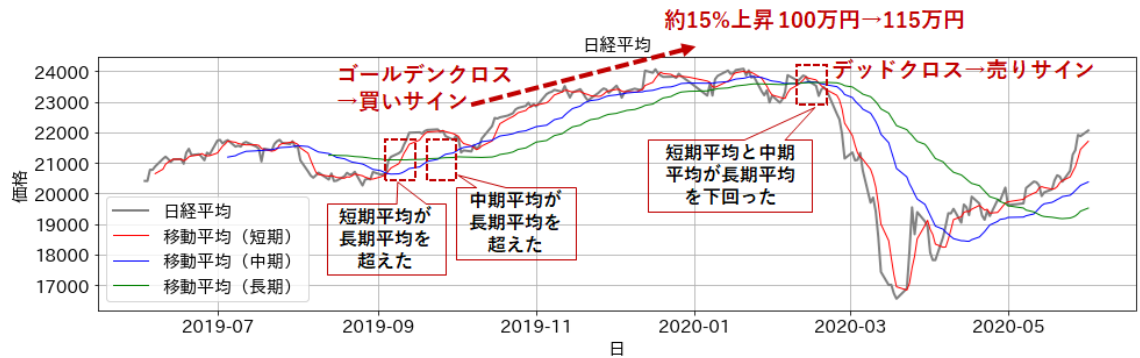
plt.show()
```



結果の見方 (売り買いの判断)

- ゴールデンクロス：短期移動平均が中期または長期平均を下から上に超えたとき「買い」

- デッドクロス：短期移動平均が中期または長期平均を上から下に超えたとき「売り」
- 必ずゴールデンクロスやデッドクロスの法則は、売買のタイミングの一例



練習6

練習1で取得したデータの移動平均のグラフを書いてみよう。

作成したグラフから、ゴールデンクロスやデッドクロスがあるかを確認してみよう。

In []:

MACD（マックディー）の分析

- 時系列データの長期変動（トレンド）を見る指標で、売買シグナルとして用いられる
 - 0以上 → 上昇トレンド
 - 0未満 → 下降トレンド
 - 0ラインを超え始めたら → 買い
 - 0ラインを下回り始めたら → 売り
 - [参考:岡三オンライン](#)
- 計算式
 - $MACD = \text{短期EMA} - \text{長期EMA}$ (EMA:指数平滑移動平均)
 - $MACD \text{シグナル} = MACD \text{の移動平均線}$

MACDの計算

- 使用ライブラリ
 - `talib.MACD`(データ, 短期移動平均の日数, 長期移動平均の日数, MACDシグナルの移動平均の日数) を使用
 - 基本の設定値
 - 短期移動平均の日数:12
 - 長期移動平均の日数:26
 - MACDシグナルの移動平均の日数: 9

日経平均のMACD

In [13]:

```
import talib
```

```
#-----
```

```

# 設定項目
#-----
price = df_N225['Close'] # 終値を取得
#-----

# 第1出力: MACD
# 第2出力: MACDシグナル
# 第3出力: MACDのヒストグラム作成用データ
macd, macdSig, macdHist = talib.MACD(price,          # 終値
                                     fastperiod=12,   # 短期の移動平均の日数
                                     slowperiod=26,   # 長期の移動平均の日数
                                     signalperiod=9    # MACDシグナルの移動平均の日数
                                    )

# pandasデータフレームに追加
df_N225['macd'] = macd
df_N225['macdSig'] = macdSig
df_N225['macdHist'] = macdHist

display(df_N225.head(5))

```

	Open	High	Low	Close	Volume	sma_short	sma_midium	sma_long	macd
Date									
2019-06-03	20327.87	20438.03	20305.74	20410.88	695247100	NaN	NaN	NaN	NaN
2019-06-04	20435.86	20464.57	20290.16	20408.54	713001500	NaN	NaN	NaN	NaN
2019-06-05	20667.89	20800.64	20646.15	20776.10	744705100	NaN	NaN	NaN	NaN
2019-06-06	20745.84	20842.28	20745.84	20774.04	614232500	NaN	NaN	NaN	NaN
2019-06-07	20859.78	20907.06	20817.07	20884.71	522701300	20650.854	NaN	NaN	NaN

MACDの計算には移動平均を使うため前半データはNaN(欠損)となる

MACDのグラフ作成

```

In [76]: import matplotlib.pyplot as plt
import warnings # 警告を表示しないようにするライブラリ
warnings.simplefilter('ignore') # 警告を表示しない設定

#-----
# 設定項目
#-----
name = "日経平均" # 銘柄名(任意の名前)
date = df_N225.index # インデックスの日時を取得
price = df_N225['Close'] # 終値を取得
macd = df_N225['macdHist'] # macdの作図用データ
#-----

#-----
# グラフ1: 日経平均
#-----
# グラフサイズの変更
plt.figure(figsize=(16, 4))

```

```

# グラフ作成
plt.plot(date, price, label=name, linewidth=2, color="gray")

# 追加オプション
plt.title(name, fontsize=15)
plt.xlabel('日', fontsize=15)
plt.ylabel('価格', fontsize=15)
plt.legend(fontsize=15, loc="upper left")
plt.grid()
plt.show()

#-----
# グラフ2: MACD
#-----
# グラフサイズの変更
plt.figure(figsize=(16, 4))

# グラフ作成
plt.fill_between(date, macd, label='MACD', color='grey', alpha=0.5)

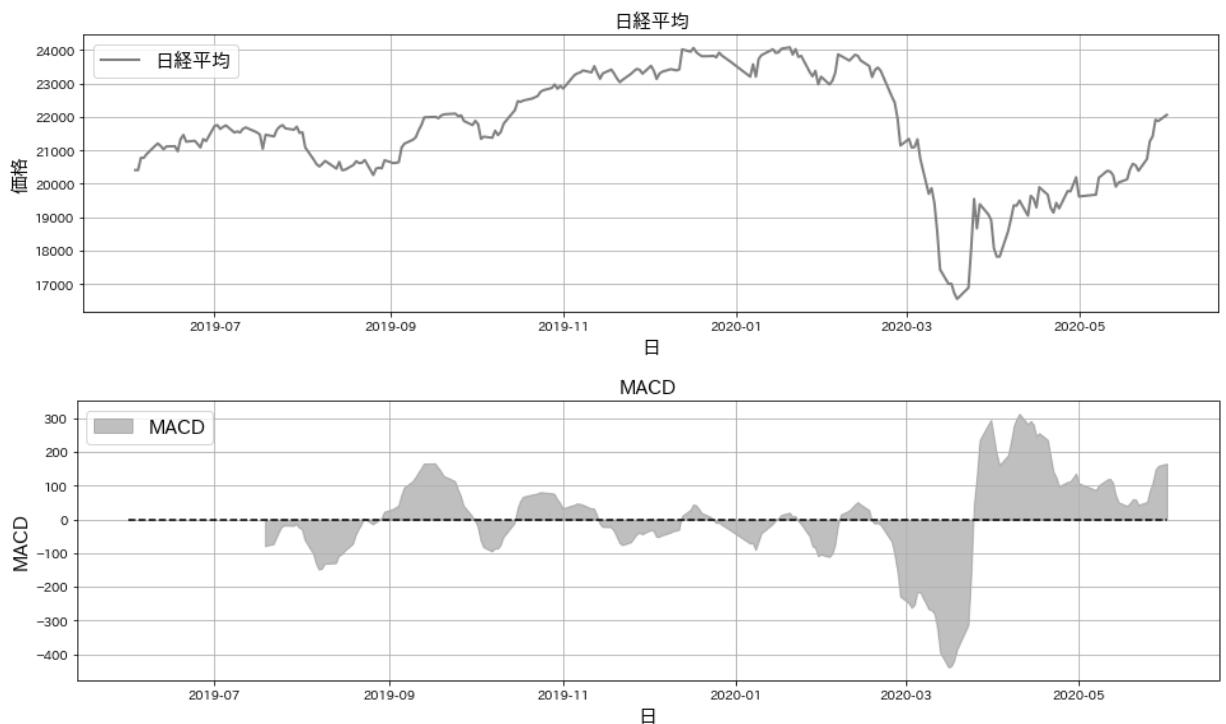
# 追加オプション
plt.title("MACD", fontsize=15)
plt.xlabel("日", fontsize=15)
plt.ylabel("MACD", fontsize=15)
plt.legend(fontsize=15, loc="upper left")
plt.grid()
plt.hlines(0, start, end, 'black', linestyle='dashed') # 水平線追加

# 出力ファイル名を指定 (Googleドライブの場合)
outfile = "drive/My Drive/Colab Notebooks/05_グラフ_"+name+"_MACD.png"

# 出力ファイル名を指定 (Anacondaの場合, Google Colaboratoryの場合は削除してください)
outfile = "05_グラフ_"+name+"_MACD.png"

plt.savefig(outfile, bbox_inches='tight') # 画像の保存
plt.show()

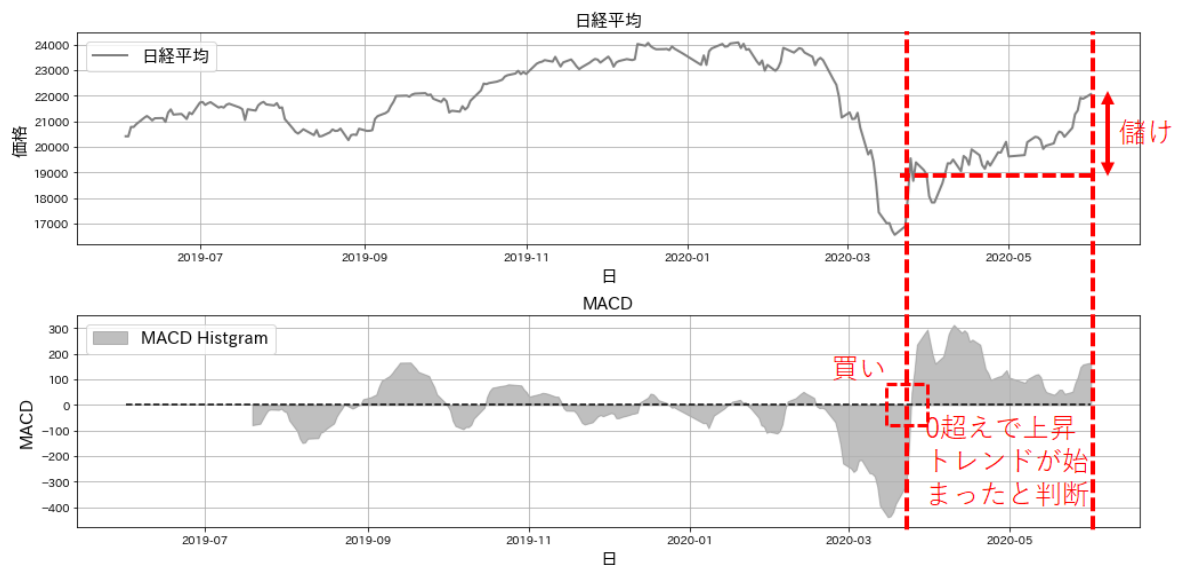
```



結果の見方（売り買いの判断）

- 0以上 → 上昇トレンド
- 0未満 → 下降トレンド
- 0ラインを超え始めたら → 買い
- 0ラインを下回り始めたら → 売り

実際の株トレーダーはもう少し複雑な分析を行って判断しているので興味のある方は調べてください



練習7

練習1で取得したデータのMACDのグラフを書いてみよう。グラフから、買いのタイミングがあるかを確かめてみよう。

In []:

RSIの分析

- 買われ過ぎ、売られ過ぎの指標
 - 20-30%を下回る → 売られすぎ → 逆張りの 買い
 - 70-80%を上回る → 買われすぎ → 逆張りの 売り
 - 参考:[岡三オンライン](#)

RSIの計算

- 使用ライブラリ
 - `RSI = talib.RSI(データ, 何日間のRSIを計算するか)` を使用

日経平均のRSI

In [16]:

```
#-----
# 設定項目
#-----
price = df_N225['Close']      # 終値を取得
#-----
```



```

#-----
# グラフサイズの変更
plt.figure(figsize=(16, 4))

# グラフ作成
plt.plot(date, rsi, label='RSI', linewidth=2, color='gray')

# 追加オプション
plt.title('RSI', fontsize=15)           # グラフタイトル
plt.xlabel('日', fontsize=15)           # x軸ラベル
plt.ylabel('RSI', fontsize=15)          # y軸ラベル
plt.legend(fontsize=15, loc="best")     # 凡例
plt.grid()                             # 補助線追加
plt.hlines([30, 50, 70], start, end, 'blue', linestyle='dashed') # 30, 50, 70%ラインに水色

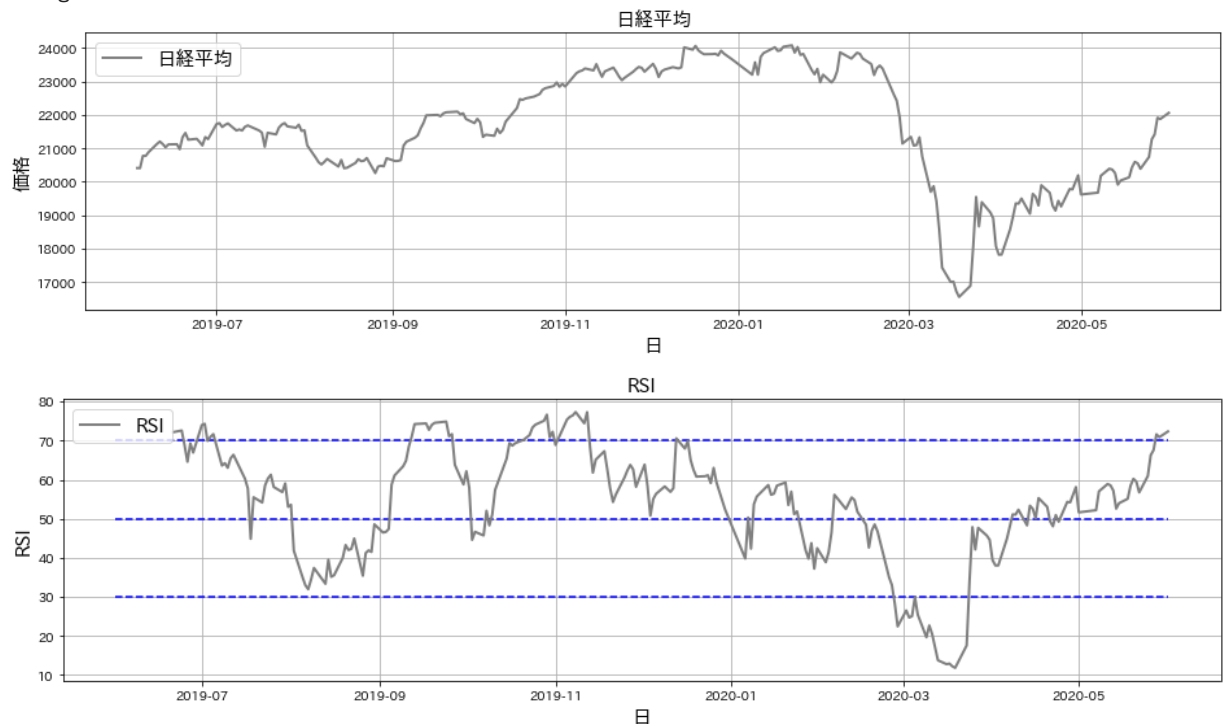
# 出力ファイル名を指定 (Googleドライブの場合)
outfile = "drive/My Drive/Colab Notebooks/05_グラフ_"+name+"_RSI.png"

# # 出力ファイル名を指定 (Anacondaの場合, Google Colaboratoryの場合は削除してください)
outfile = "05_グラフ_"+name+"_RSI.png"

plt.savefig(outfile, bbox_inches='tight') # 画像の保存
plt.show()

```

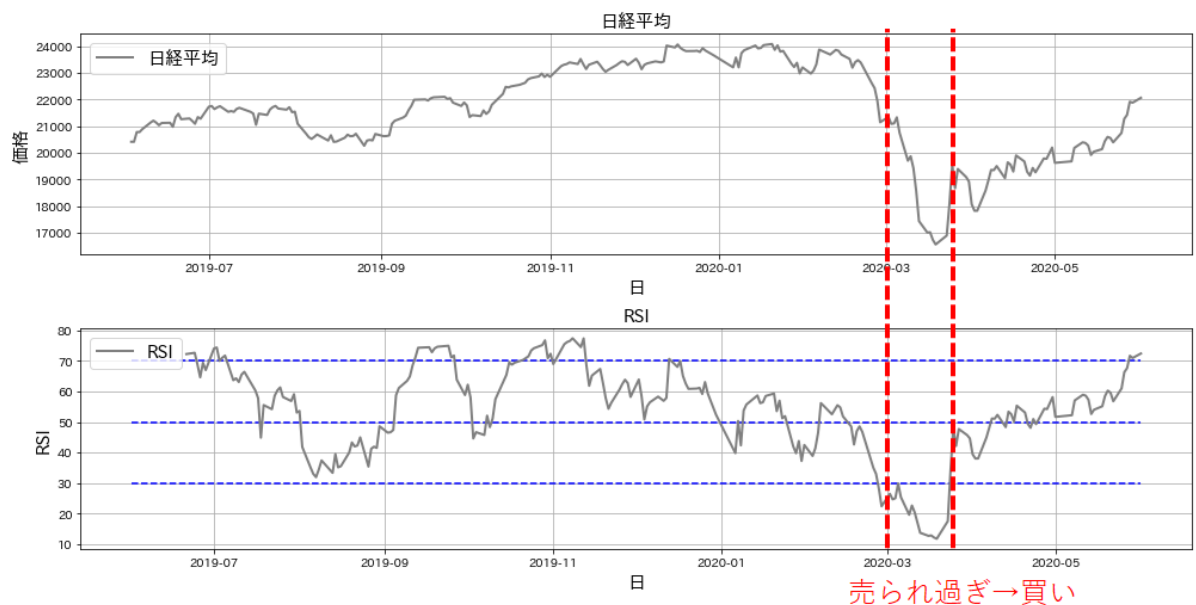
<Figure size 1008x504 with 0 Axes>



結果の見方

- 20-30%を下回る → 売られすぎ → 逆張りの 買い
- 70-80%を上回る → 買われすぎ → 逆張りの 売り

実際の株トレーダーはもう少し複雑な分析を行って判断しているので興味のある方は調べてください



練習8

練習1で取得したデータのRSIのグラフを書いてみよう。グラフから、買われ過ぎまたは売られ過ぎのサインがあるかを確認してみよう。

In []:

ボリンジャーバンドの分析

- 移動平均などの一定期間の平均に対して、一定期間の標準偏差を足したものと引いたもの
- 買われ過ぎ、売られ過ぎの判断に用いられる
- 一般に上限下限は $\pm 2\sigma$ を使う (σ :標準偏差) → 統計的95%はその範囲に価格が入るということ
 - 終値が $+2\sigma$ を超える → 買われすぎ
 - 終値が -2σ を超える → 売られすぎ
 - 参考:[岡三オンライン](#)

ボリンジャーバンドの計算

- 使用ライブラリ
 - 上限値、移動平均、下限値 = talib.BBANDS (データ, 平均と標準偏差を計算するための日数, 上限を何 σ にするか, 下限を何 σ にするか, 移動平均のタイプ)
 - 移動平均のタイプ: matype
 - 0: 単純移動平均
 - 1: 指数移動平均
 - 2: 加重移動平均

日経平均のボリンジャーバンド

In [20]:

```
#-----
# 設定項目
```

```
#-----
price = df_N225['Close']      # 終値を取得
#-----

upper, middle, lower = talib.BBANDS(price,      # 終値,
                                     timeperiod = 25, # 移動平均日数(平均と標準偏差を計算
                                     nbdevup = 2,    # 上限をσ(標準偏差)の何倍にするか
                                     nbdevdn = 2,    # 下限をσ(標準偏差)の何倍にするか
                                     matype = 0      # 単純移動平均
                                    )

# pandasデータフレームに追加
df_N225['upper'] = upper
df_N225['middle'] = middle
df_N225['lower'] = lower

display(df_N225)
```

	Open	High	Low	Close	Volume	sma_short	sma_midium	sma_long	
Date									
2019-06-03	20327.87	20438.03	20305.74	20410.88	695247100	NaN	NaN	NaN	
2019-06-04	20435.86	20464.57	20290.16	20408.54	713001500	NaN	NaN	NaN	
2019-06-05	20667.89	20800.64	20646.15	20776.10	744705100	NaN	NaN	NaN	
2019-06-06	20745.84	20842.28	20745.84	20774.04	614232500	NaN	NaN	NaN	
2019-06-07	20859.78	20907.06	20817.07	20884.71	522701300	20650.854	NaN	NaN	
...	
2020-05-26	20927.96	21327.86	20918.11	21271.17	871911300	20709.688	20011.2048	19224.4638	346
2020-05-27	21249.31	21473.68	21143.37	21419.23	1120168200	20874.504	20096.3660	19264.5272	398
2020-05-28	21612.90	21917.43	21580.50	21916.31	1340312400	21147.304	20177.1280	19331.6608	473
2020-05-29	21811.52	21955.44	21710.80	21877.89	1531900700	21445.250	20265.4788	19420.5976	524
2020-06-01	21910.89	22161.39	21898.99	22062.39	721657900	21709.398	20376.7432	19521.8046	572

242 rows × 15 columns



ボリンジャーバンドのグラフ作成

In [21]:

```
#-----
# 設定項目
#-----
name = "日経平均"      # 銘柄名(任意の名前)
date = df_N225.index   # インデックスの日時を取得
price = df_N225['Close'] # 終値を取得
```

```

upper = df_N225['upper'] # 上限値
middle = df_N225['middle'] # 移動平均
lower = df_N225['lower'] # 下限値
#-----

# グラフ描画枠作成（2つのグラフを並べるキャンパスを設定）
fig = plt.figure(figsize=(14, 7), tight_layout=True)

# グラフ作成前に設定するオプション
plt.figure(figsize=(16, 4)) # 図のサイズ（横×縦）

# グラフ作成
# x軸、y軸、凡例タイトル、線幅
plt.plot(date, price, label=name, linewidth=2, color='gray') # 日経平均
plt.plot(date, middle, label='移動平均', linewidth=1, color='red') # 移動平均
plt.fill_between(date, upper, lower, color='green', alpha=0.2) # ボリンジャー

# 追加オプション
plt.title(name, fontsize=15) # グラフタイトル
plt.xlabel('日', fontsize=15) # x軸ラベル
plt.ylabel('価格', fontsize=15) # y軸ラベル
plt.xticks(fontsize=10) # x軸の目盛文字サイズ
plt.yticks(fontsize=10) # y軸の目盛文字サイズ
plt.legend(fontsize=10, loc="upper left") # 凡例
plt.grid() # 補助線追加

# 出力ファイル名を指定（Googleドライブの場合）
outfile = "drive/My Drive/Colab Notebooks/05_グラフ_"+name+"_ボリンジャーバンド.png"

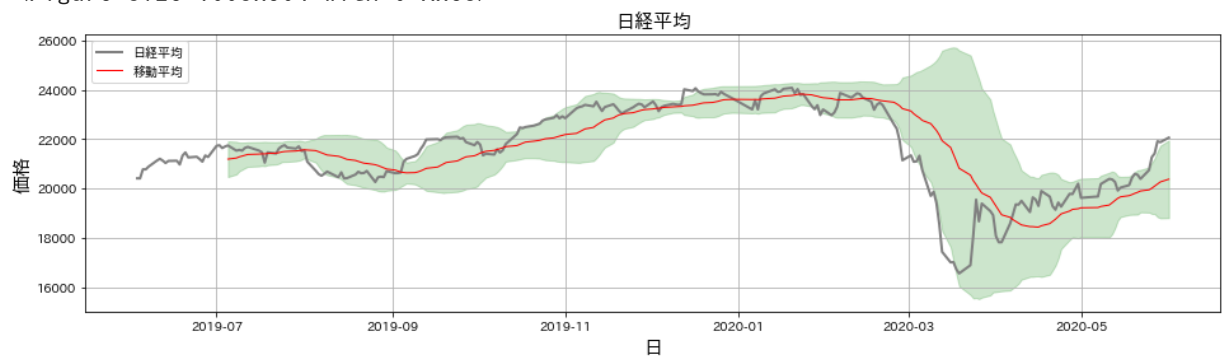
# 出力ファイル名を指定（Anacondaの場合、Google Colaboratoryの場合は削除してください）
outfile = "05_グラフ_"+name+"_ボリンジャーバンド.png"

plt.savefig(outfile, bbox_inches='tight') # 画像の保存

plt.show()

```

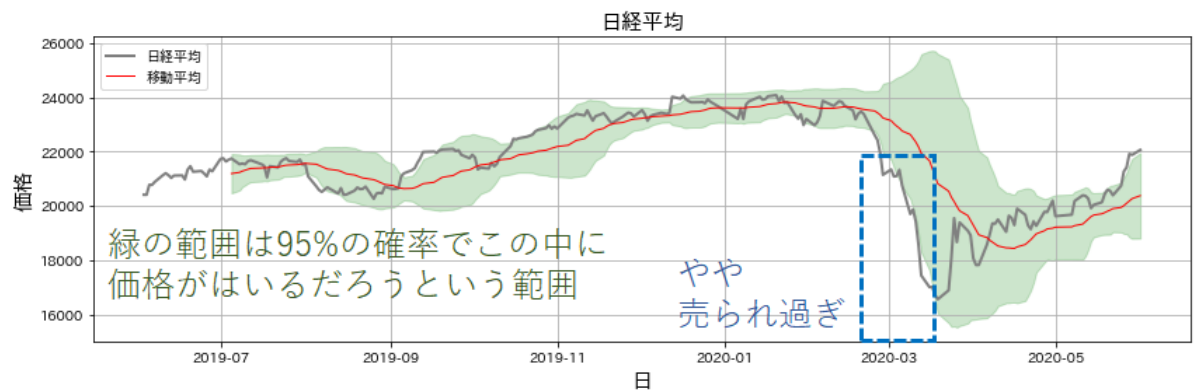
<Figure size 1008x504 with 0 Axes>



結果の見方

- 終値が $+2\sigma$ を超える → 買われすぎ
- 終値が -2σ を超える → 売られすぎ

実際の株トレーダーはもう少し複雑な分析を行って判断しているので興味のある方は調べてください



練習9

練習1で取得したデータのボリンジャーバンドのグラフを書いてみよう。グラフから、買われ過ぎまたは売られ過ぎのサインがあるかを確認してみよう。

In []:

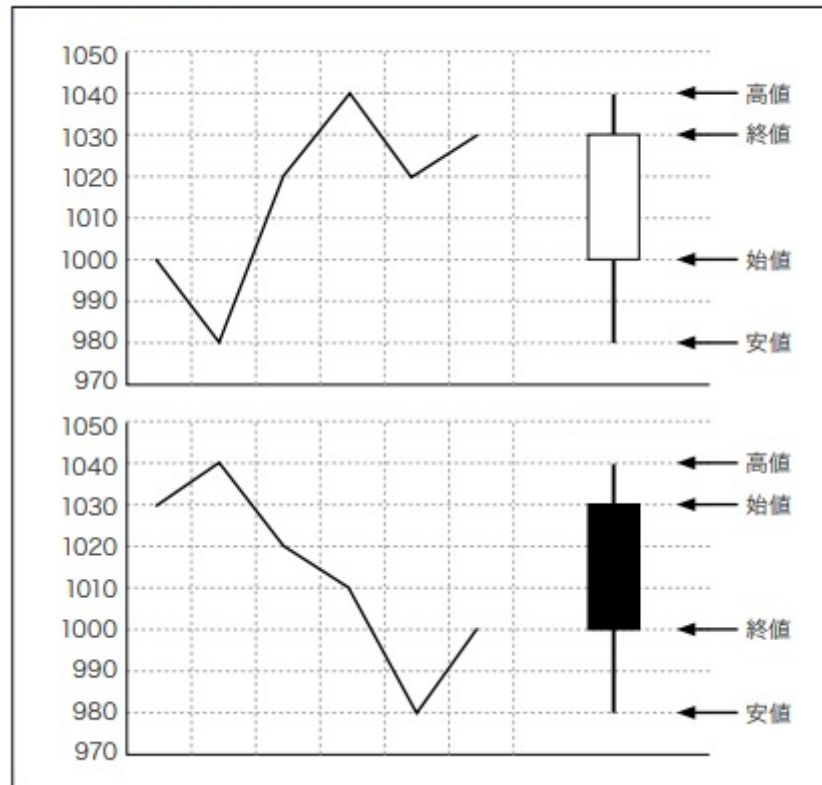
以降、付録

[参考] ロウソク足のチャート作成

- ローソク足のチャートは、過去の足取りと比較して現在の値位置が高いのか、安いのかを明瞭に示してくれるもの
 - 高値、低値、始値、終値で構成される
 - 参考：[岡三オンライン](#)
- 使用ライブラリ

- mplfinance を使用

図1-1 ● 1日の動き



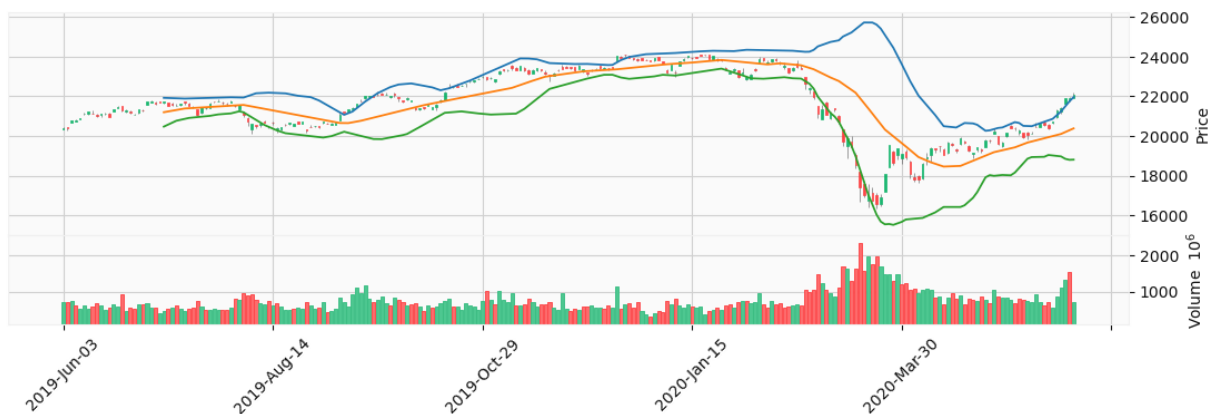
In [22]:

```
import mplfinance

#-----
# 設定項目
#-----
# 移動平均及びボリンジャーバンドのデータを抽出したデータフレーム
df_BBANDS = df_N225[['upper', 'middle', 'lower']]
#-----

# 追加プロットデータ作成
apd = mplfinance.make_addplot(df_BBANDS)

mplfinance.plot(df_N225,          # データ
                 figsize = (14, 4), # 画像サイズ
                 addplot = apd,      # 追加データ
                 type = 'candle',    # ロウソク足の指定
                 volume = True,      # 出来高 (Volume) を描画する場合 : True
                 style = 'yahoo',    # 表示スタイル
                 )
```



[参考] 将来予測

- 使用ライブラリ： prophet
- データフレームではデータの変数名を以下のように設定
 - Prophetで扱う時間データ: ds
 - 日経平均株価の調整済み終値(Adj Close): y

予測用データ作成

- 今回は、2016-08-01から2021-07-31の5年間の日経平均のデータから、その先1年後までを予測する

In [23]:

```
from pandas_datareader import data
import pandas as pd
import datetime as dt # 日付型データ生成

#-----
# 設定項目：DataReaderの引数
#-----
code = "^NKX"          # 銘柄コード
site = "stoq"          # データ取得サイト
start = dt.date(2016, 8, 1) # 開始(年, 月, 日)
end = dt.date(2021, 7, 31) # 終了(年, 月, 日)
#-----

# 新規データフレーム(df_prophet)にデータを挿入
df_prophet = data.DataReader(code, site, start, end)

# Prophetによる予測データを作成
df_prophet = df_prophet.reset_index() # indexの日時をデータ列に設定
df_prophet.rename(columns={'Date': 'ds'}, inplace=True) # 列名のDateをdsに変更
df_prophet.rename(columns={'Close': 'y'}, inplace=True) # 列名のCloseをyに変更

display(df_prophet.head(5)) # 5行分データ表示
```

	ds	Open	High	Low	y	Volume
0	2021-07-30	27677.89	27699.35	27272.49	27283.59	706422900
1	2021-07-29	27722.61	27798.05	27663.41	27782.42	677832300
2	2021-07-28	27674.99	27809.86	27466.99	27581.66	570860800
3	2021-07-27	27911.93	28036.20	27862.69	27970.22	516997200
4	2021-07-26	27990.47	28036.47	27789.35	27833.29	545421000

予測モデル作成

In [24]:

```
from fbprophet import Prophet # ライブラリ読み込み

# 予測モデルの準備
model = Prophet()

# データを予測モデルにフィット
model.fit(df_prophet)
```

```
INFO:numexpr.utils:NumExpr defaulting to 8 threads.  
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to  
override this.
```

```
Out[24]: <fbprophet.forecaster.Prophet at 0x1e8da14c550>
```

予測する期間を指定して予測

- 1行目: 予測期間設定
 - モデル名.make_future_dataframe(periods=予測期間, freq = 予測単位)
 - 予測期間: 365日先の場合 365
 - 予測単位: 日単位の場合 'd'
- 2行目: 土日の予測を除外(株価は土日は動かないため)
 - dt.weekdayにより、月0, 火1, 水2, 木3, 金4, 土5, 日6
- 3行目: データを挿入して予測
 - モデル名.predict(データ)
- 4行目: 予測結果の可視化

```
In [25]: future_data = model.make_future_dataframe(periods=365, freq = 'd')  
future_data = future_data[future_data['ds'].dt.weekday < 5]  
  
forecast_data = model.predict(future_data)  
  
fig = model.plot(forecast_data)
```



青の実線が予測値、水色の範囲は誤差範囲（95%信頼区間）

予測精度を上げるためには株価そのものではなく、データを対数変換したり、Prophetのパラメータを調整するなどが必要

時系列データの成分分解

- 原系列（元データ）を以下の3つに分解

- 傾向変動（トレンド）：季節要因や不規則要因の除いた大きな流れ
- 季節変動（周期性）：カレンダーに基づく要因
- 不規則変動（残差）：なんらかの不規則なイベントによる影響

- 使用ライブラリ

- `statmodels.api.tsa.seasonal_decompose`(時系列データ, `freq`=周期間隔)
-

- 今回は、2016-08-01から2021-07-31の5年間の日経平均の成分分解を行う

In [27]:

```
from statsmodels import api as sm
import japanize_matplotlib

#-----
# 設定項目：DataReaderの引数
#-----
code = "^NKX"          # 銘柄コード
site = "stooq"         # データ取得サイト
start = dt.date(2016, 8, 1) # 開始(年, 月, 日)
end = dt.date(2021, 7, 31) # 終了(年, 月, 日)
#-----

# 新規データフレーム(df_prophet)にデータを挿入
df_seasonal = data.DataReader(code, site, start, end)

# 月単位の平均値
df_seasonal_M = df_seasonal['Close'].resample('M').mean()
display(df_seasonal_M.head(5))

# 成分分解
res = sm.tsa.seasonal_decompose(df_seasonal_M, freq=12)
observed = res.observed # 原系列（元データ）
trend = res.trend       # 傾向変動（トレンド）成分
seasonal = res.seasonal # 季節変動（周期性）成分
residual = res.resid    # 不規則変動（残差）成分

fig = plt.figure(figsize=(14, 7), tight_layout=True) # グラフ描画枠作成、サイズ指定

# 元データ
ax1 = fig.add_subplot(4, 1, 1)
ax1.set_title('Original')
ax1.grid()
ax1.plot(observed)

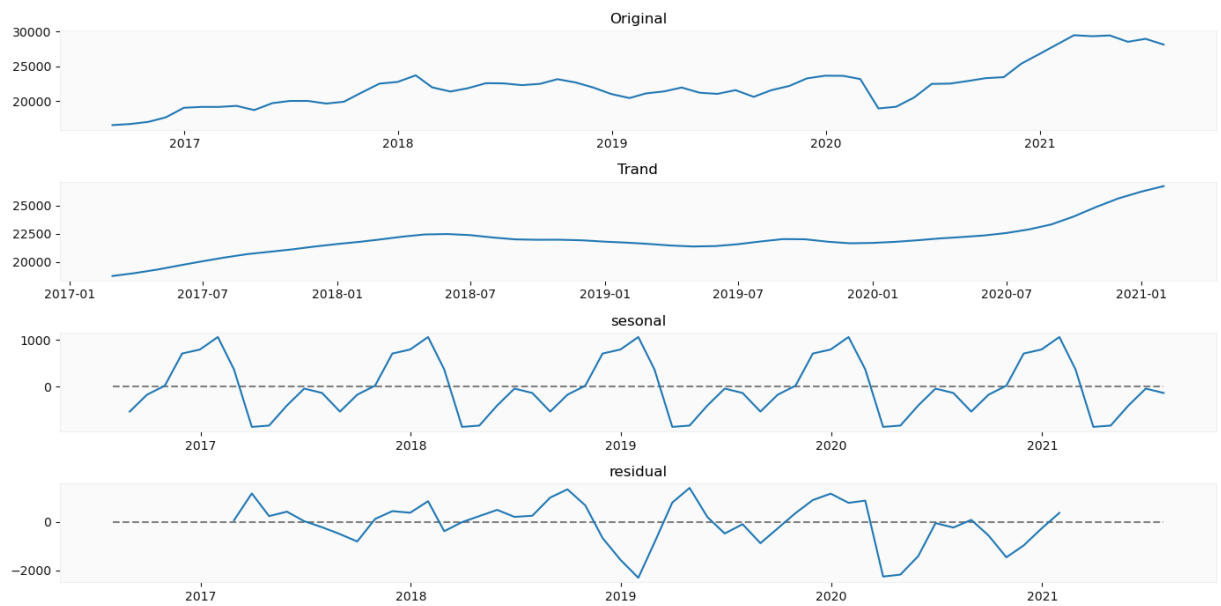
ax2 = fig.add_subplot(4, 1, 2)
ax2.set_title('Trand')
ax2.grid()
ax2.plot(trend)

ax3 = fig.add_subplot(4, 1, 3)
ax3.set_title('sesonal')
ax3.hlines([0], start, end, 'grey', linestyle='dashed') # 水平線追加
ax3.grid()
ax3.plot(seasonal)

ax4 = fig.add_subplot(4, 1, 4)
ax4.set_title('residual')
ax4.hlines([0], start, end, 'grey', linestyle='dashed') # 水平線追加
ax4.grid()
ax4.plot(residual)
```

```
Date
2016-08-31    16586.070455
2016-09-30    16737.041500
2016-10-31    17044.510500
2016-11-30    17689.540000
2016-12-31    19066.027619
Freq: M, Name: Close, dtype: float64
```

```
Out[27]: [ <matplotlib.lines.Line2D at 0x1e8eef69730>]
```



- トレンドを見るとコロナショックがあってもそれほど変化なし
 - 季節性をみると、冬場に上昇、夏場に減少する傾向
 - 不規則要因をみると、2020年3月の落ち込みはコロナショックと考えられる
-