Home

Jump to bottom

Myokyeong Kim edited this page on Mar 31, 2022 · 3 revisions

Clone this wiki locally

```
https://github.com/baeksumin/k-project.wiki.git
```

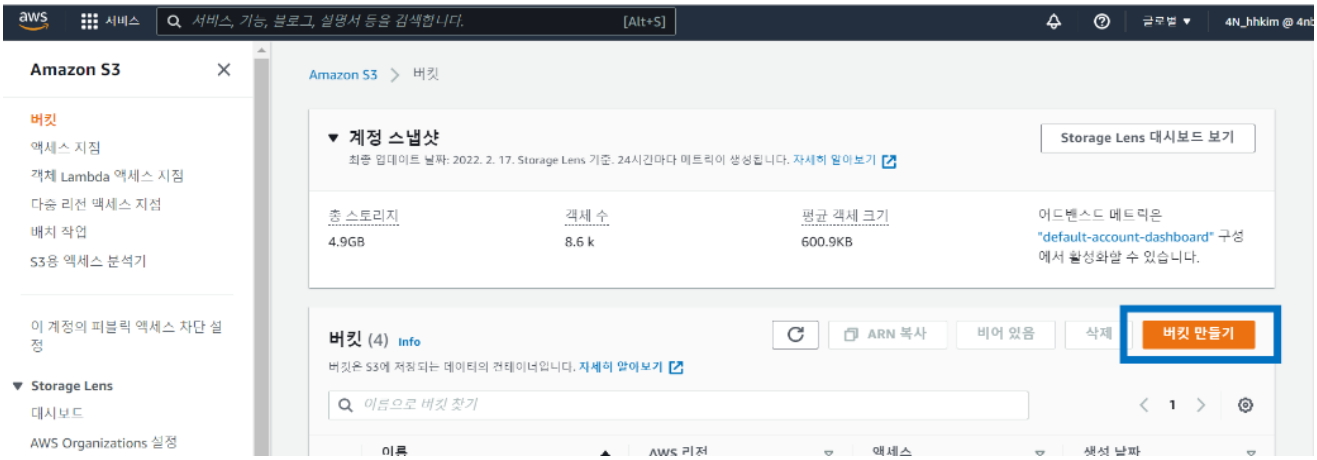
AWS S3 Bucket 생성

Jump to bottom

Myokyeong Kim edited this page on Feb 19, 2022 · 1 revision

1. S3 Bucket 생성

- AWS 검색창에 S3 검색 후 버킷 만들기 클릭

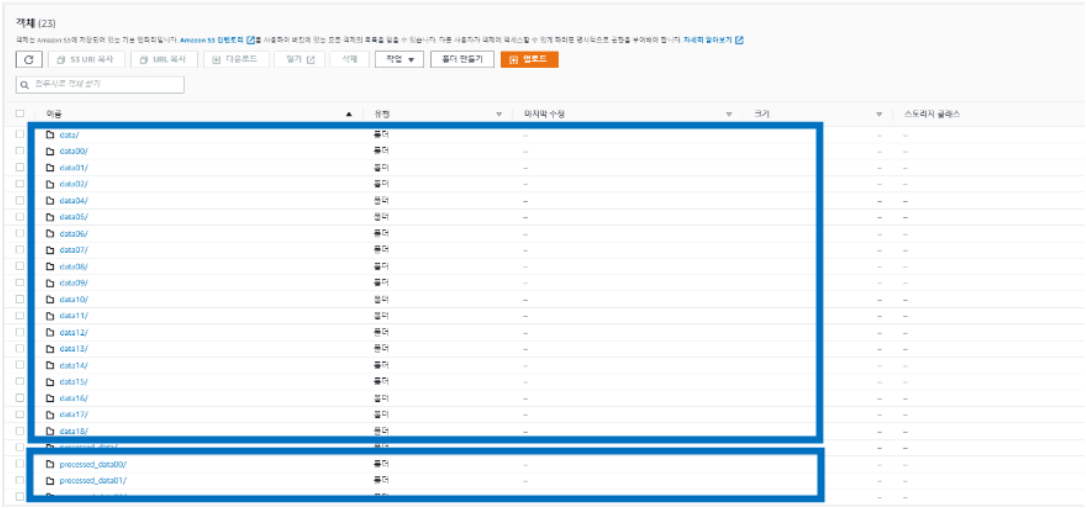


2. S3 Bucket 이름 설정

- sagemaker에서 잘 연동되도록 버킷이름에 'sagemaker' 단어가 들어가도록 해주며, AWS 리전은 아시아 태평양(서울)ap-northeast-2 로 설정해준다.

※ 버킷 이름 설정 시 주의사항
https://docs.aws.amazon.com/AmazonS3/latest/userguide/bucketnamingrules.html

3. S3 Bucket 데이터 저장



생성한 S3 Bucket에 다음과 같이 2개 유형의 파일을 만들어준다.

1. 전처리 할 데이터 폴더 (raw data)
    - 세션유지기간, 인스턴스 유형 등을 고려하여 한번에 작업 할 수 있는 양만큼 나누어 파일로 저장한다.

2. 전처리 작업 후 데이터가 저장되는 폴더
    - sagemaker에서 전처리 작업 후 데이터가 저장되는 폴더로 빈 폴더를 생성해준다.

Pages 6

Find a page...

Clone this wiki locally

```
https://github.com/baeksumin/k-project.wiki.git
```

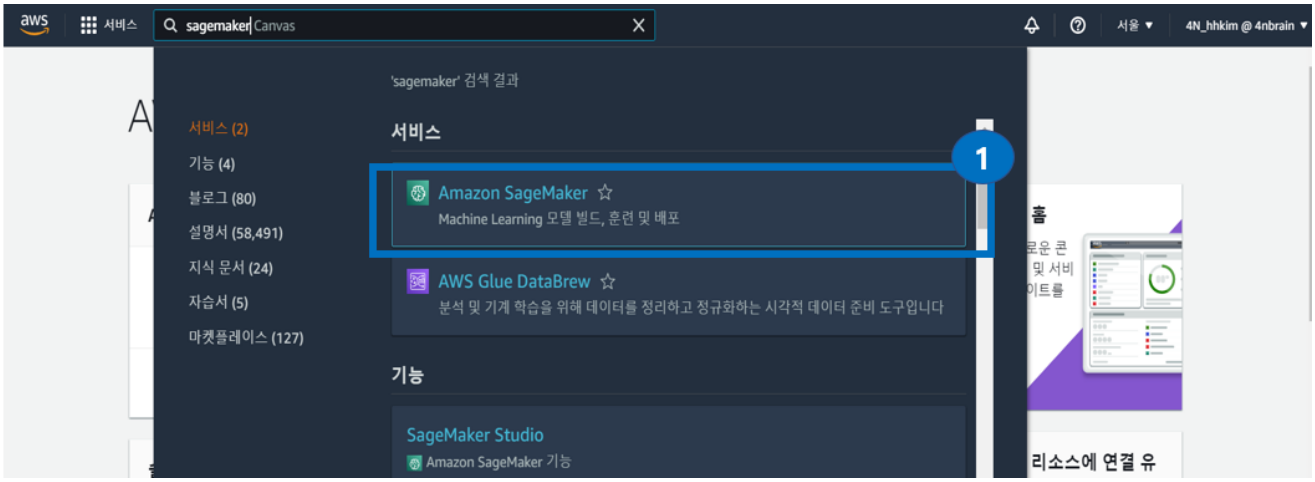
AWS Sagemaker 노트북 인스턴스 생성

Jump to bottom

Myokyeong Kim edited this page on Feb 19, 2022 · 1 revision

1. 노트북 인스턴스 생성

- AWS 검색창에 sagemaker 검색 후 `노트북 인스턴스 생성` 클릭



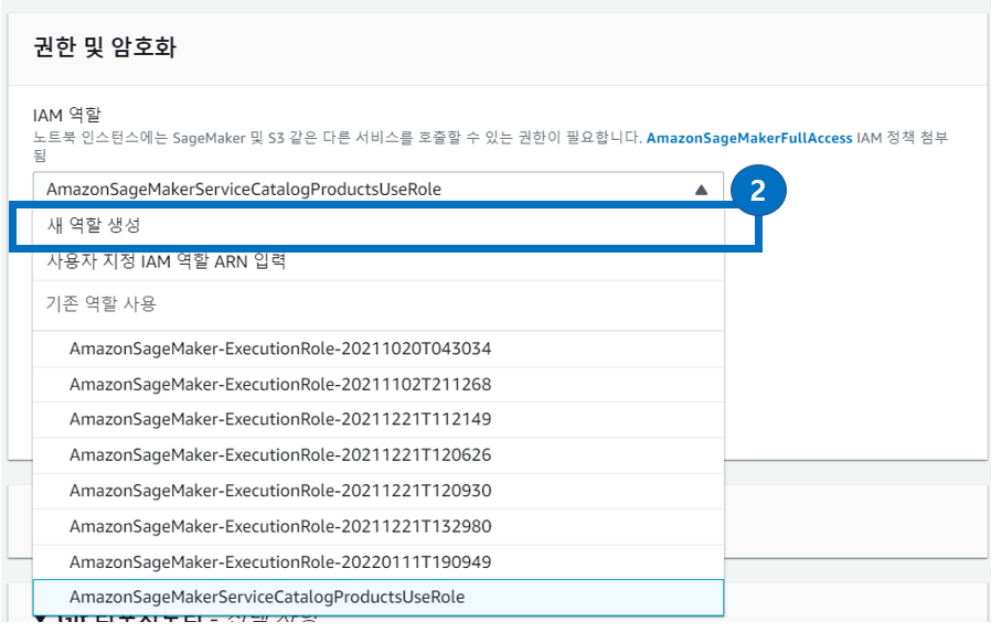
2. 노트북 인스턴스 설정

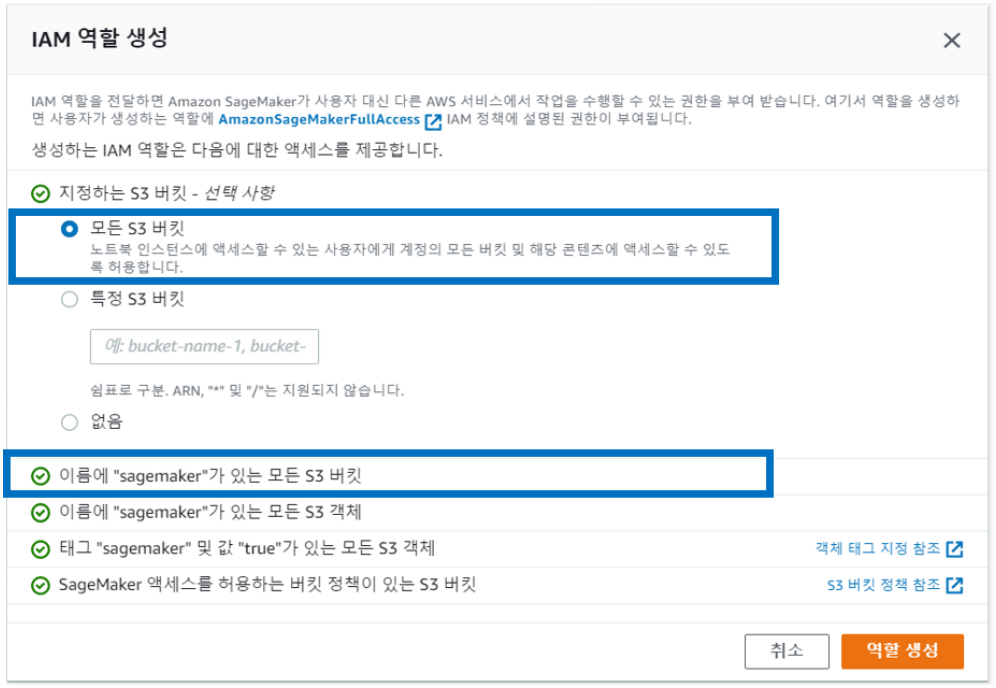
- **노트북 인스턴스 이름, 유형 설정**

  노트북 인스턴스 이름과 유형을 설정해준다.

- **권한 및 암호화 설정**

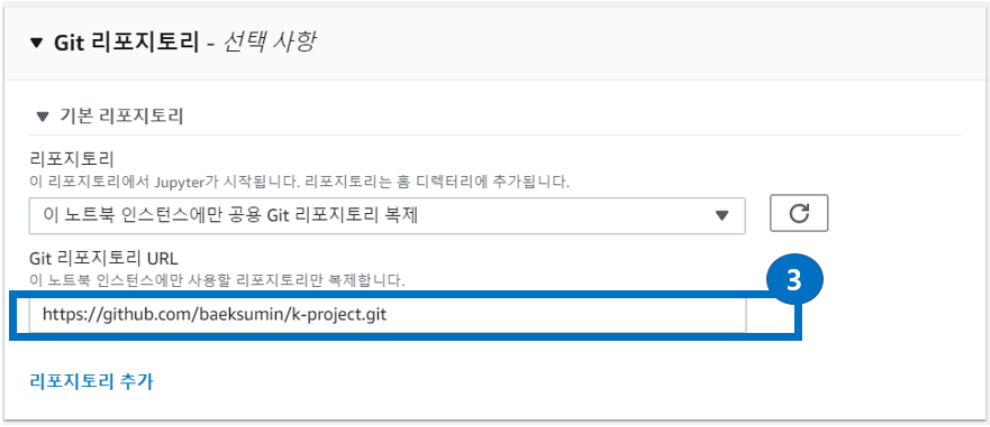
  IAM 역할 생성에서 지정하는 S3 버킷 에 모든 S3버킷 으로 설정해주며, 이름에 “sagemaker”가 있는 모든 S3버킷 에 체크가 되어있는지 확인해준다.





- **Git 리포지토리 설정**

  노트북 인스턴스와 https://github.com/baeksumin/k-project.git 과 연동시켜주도록 설정한다.



Pages 6

Find a page...

- Home
- AWS S3 Bucket 생성
- AWS Sagemaker 노트북 인스턴스 생성
  1. 노트북 인스턴스 생성
  2. 노트북 인스턴스 설정
- AWS Sagemaker에서 S3에 저장된 데이터 불러오기
- GA를 이용한 보조지표 파라미터 최적화
- Scikit Learn 컨테이너를 사용하여 데이터 사전 병렬 처리

Clone this wiki locally

```
https://github.com/baeksumin/k-project.wiki.git
```

AWS Sagemaker에서 S3에 저장된 데이터 불러오기

Jump to bottom

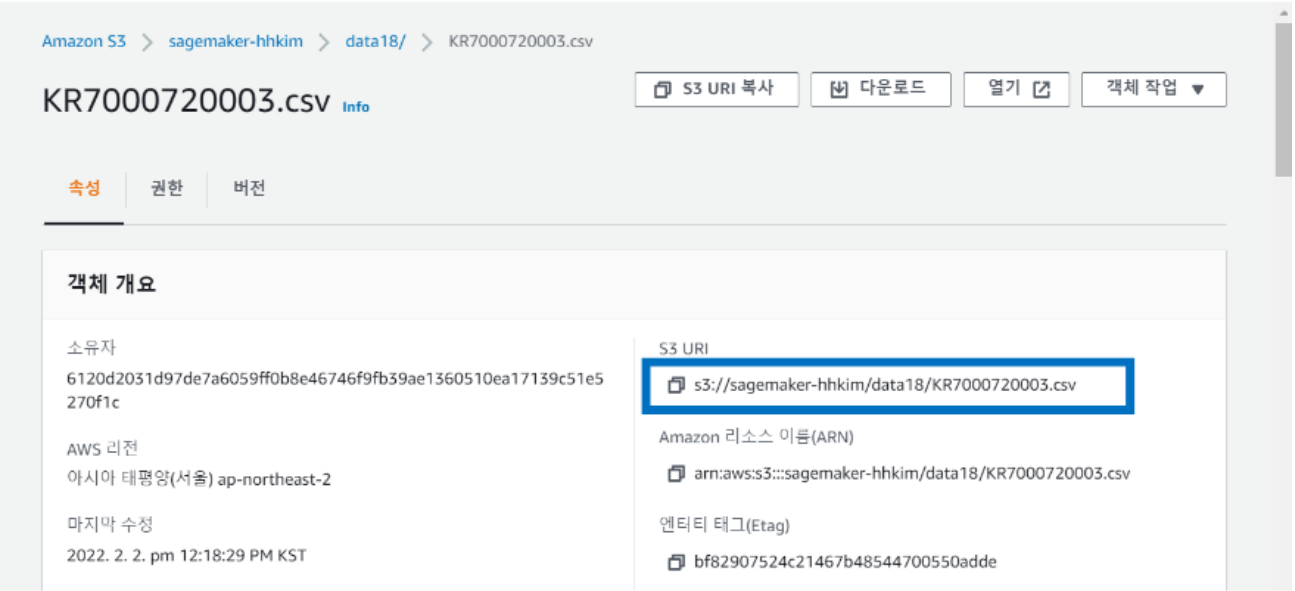
Myokyeong Kim edited this page on Feb 20, 2022 · 2 revisions

1. 필요한 패키지 설치

```
import boto3
import io
import pandas as pd
from sagemaker import get_execution_role
```

2. S3 데이터 불러오기

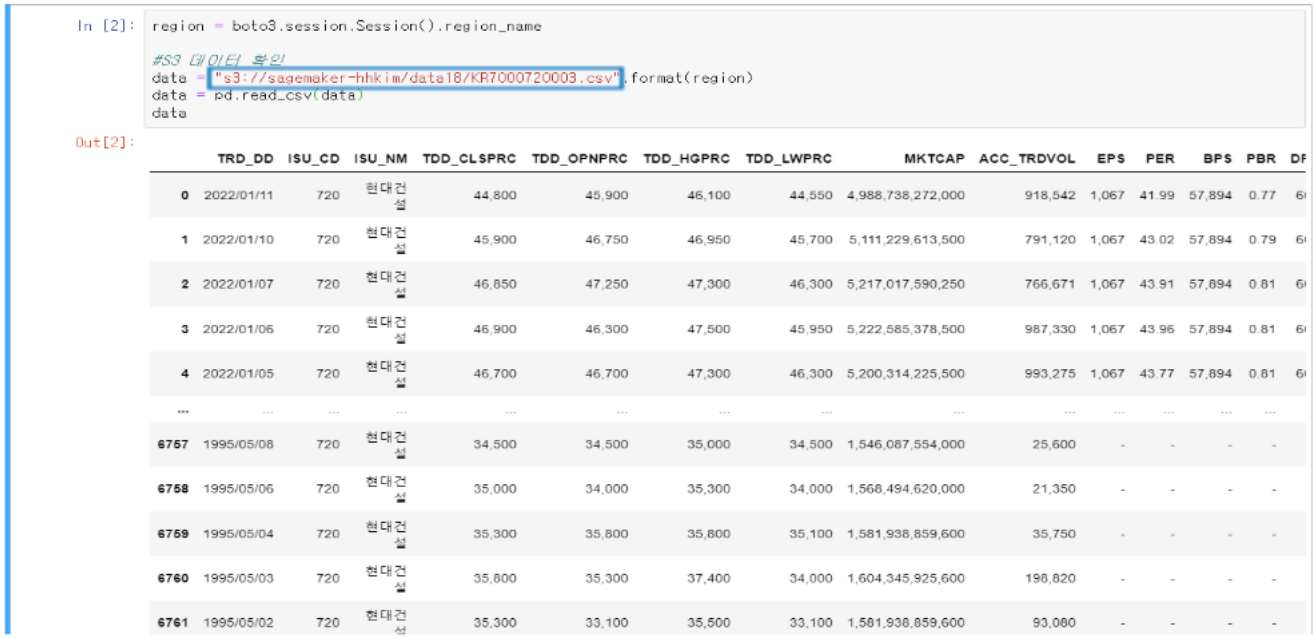
- S3에서 `S3 URI` 확인하기



- S3 데이터 불러오기

```
region = boto3.session.Session().region_name

#S3 데이터 확인
data = "s3://sagemaker-hhkim/data18/KR7000720003.csv".format(region)
data = pd.read_csv(data)
data
```



In [2]:
```
region = boto3.session.Session().region_name

#S3 데이터 확인
data = "s3://sagemaker-hhkim/data18/KR7000720003.csv".format(region)
data = pd.read_csv(data)
data
```

Out[2]:

	TRD_DD	ISU_CD	ISU_NM	TDD_CLSPRC	TDD_OPNPRC	TDD_HGPRC	TDD_LWPRC	MKTCAP	ACC_TRDVOL	EPS	PER	BPS	PBR	DI
0	2022/01/11	720	현대건설	44,800	45,900	46,100	44,550	4,988,738,272,000	918,542	1,067	41.99	57,894	0.77	6
1	2022/01/10	720	현대건설	45,900	46,750	46,950	45,700	5,111,229,613,500	791,120	1,067	43.02	57,894	0.79	6
2	2022/01/07	720	현대건설	46,650	47,250	47,300	46,300	5,217,017,590,250	766,671	1,067	43.91	57,894	0.81	6
3	2022/01/06	720	현대건설	46,900	46,300	47,500	45,950	5,222,585,378,500	987,330	1,067	43.96	57,894	0.81	6
4	2022/01/05	720	현대건설	46,700	46,700	47,300	46,300	5,200,314,225,500	993,275	1,067	43.77	57,894	0.81	6
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6757	1995/05/08	720	현대건설	34,500	34,500	35,000	34,500	1,546,087,554,000	25,600	-	-	-	-	
6758	1995/05/06	720	현대건설	35,000	34,000	35,300	34,000	1,568,494,620,000	21,350	-	-	-	-	
6759	1995/05/04	720	현대건설	35,300	35,800	35,800	35,100	1,581,938,859,600	35,750	-	-	-	-	
6760	1995/05/03	720	현대건설	35,800	35,300	37,400	34,000	1,604,345,925,600	196,820	-	-	-	-	
6761	1995/05/02	720	현대건설	35,300	33,100	35,500	33,100	1,581,938,859,600	93,080	-	-	-	-	

Clone this wiki locally

```
https://github.com/baeksumin/k-project.wiki.git
```

GA를 이용한 보조지표 파라미터 최적화

Jump to bottom

Myokyeong Kim edited this page on Mar 31, 2022 · 1 revision

1. 필요한 패키지 설치

- backtrader 설치

```
!pip install backtrader
```

- backtesting 설치

```
!pip install backtesting
```

- deap 설치

```
!pip install deap
```

- talib 설치하기

```
#talib 설치
!wget -q http://prdownloads.sourceforge.net/ta-lib/ta-lib-0.4.0-src.tar.gz 2>&1 > /dev/null
!tar xvzf ta-lib-0.4.0-src.tar.gz 2>&1 > /dev/null

import os

os.chdir('ta-lib')

!./configure --prefix=/usr 2>&1 > /dev/null
!make 2>&1 > /dev/null
!make install 2>&1 > /dev/null

os.chdir('../')
```

```
!pip install TA-Lib 2>&1 > /dev/null
```

2. 필요한 라이브러리 불러오기

```
import re
import sys
import json
import time
import talib
import pickle
import random
import logging
import os.path
import numpy as np
import pandas as pd
import seaborn as sns
from tqdm import trange
import backtrader as bt
from google.colab import drive
import matplotlib.pyplot as plt
from backtesting import Strategy
from backtesting import Backtest
import backtrader.feeds as btfeeds
from IPython.display import display, Image
from datetime import datetime, date, timedelta
from deap import base, creator, tools, algorithms
```

3. Backtrader Strategy 설정

- Golden Dead Cross

```
class SmaCross1(bt.Strategy):
  params = dict(
    pfast=50, # period for the fast moving average
    pslow=200 # period for the slow moving average
    )

  def __init__(self):
    sma1 = bt.ind.SMA(period = self.p.pfast) # fast moving average
    sma2 = bt.ind.SMA(period = self.p.pslow) # slow moving average
    self.crossover = bt.ind.CrossOver(sma1, sma2) # crossover signal

  def next(self):
    if not self.position: # not in the market
```

```
        if self.crossover > 0: # if fast crosses slow to the upside
          close = self.data.close[0] # 종가 값

          size = int(self.broker.getcash() / close) # 최대 구매 가능 개수
          self.buy(size=size) # 매수 size = 구매 개수 설정
      elif self.crossover < 0: # in the market & cross to the downside
          self.close() # 매도
```

- RSI

```
class RSI(bt.Strategy):
  params = dict(period=26)

  def __init__(self):
    self.rsi = bt.indicators.RSI(self.data.close, period=self.p.period)

  def next(self):
    if not self.position:  #아직 주식을 사지 않았다면

      if self.rsi <30 :
        self.order = self.buy()

    elif self. rsi >70 :
      self.order = self.sell()
```

- ROC

```
class ROC(bt.Strategy):
  params = dict(period=14)

  def __init__(self):
    self.roc = bt.indicators.ROC(self.data.close, period=self.p.period)

  def next(self):
    if not self.position:  #아직 주식을 사지 않았다면

      if self.roc > 0:
        self.order = self.buy()

    elif self.roc < 0:
      self.order = self.sell()
```

- Moving Average Envelop

```
class MAP(bt.Strategy):
  params = dict(period = 12, upperLimit = .07, lowerLimit = .07)

  def __init__(self):
    # self.sma = bt.ind.SMA(self.data.close, period = self.p.period) # fast moving average
```

```
        self.sma = bt.ind.SMA(period = self.p.period)
        self.ul = self.sma + (self.p.upperLimit * self.sma)
        self.ll = self.sma + (self.p.lowerLimit * self.sma)

    def next(self):
        if not self.position: # 아직 주식을 사지 않았다면
            if self.sma <= self.ll:
                close = self.data.close[0] # 종가 값
                size = int(self.broker.getcash() / close) # 최대 구매 가능 개수
                self.buy(size=size) # 매수 size = 구매 개수 설정

        elif self.sma > self.ul:
            self.sell() # 매도lf.sell() # 매도
```

- Stochastic

```
class StochasticSR(bt.Strategy):
    '''Trading strategy that utilizes the Stochastic Oscillator indicator for oversold/over
    and previous support/resistance via Donchian Channels as well as a max loss in pips for
    # parameters for Stochastic Oscillator and max loss in pips
    # Donchian Channels to determine previous support/resistance levels will use the given
    # http://www.ta-guru.com/Book/TechnicalAnalysis/TechnicalIndicators/Stochastic.php5 for
    params = (('period', 14), ('pfast', 3), ('pslow', 3), ('upperLimit', 80), ('lowerLimit'

    def __init__(self):
        '''Initializes logger and variables required for the strategy implementation.'''
        # initialize logger for log function (set to critical to prevent any unwanted autol
        for handler in logging.root.handlers[:]:
            logging.root.removeHandler(handler)

        logging.basicConfig(format='%(message)s', level=logging.CRITICAL, handlers=[
            logging.FileHandler("LOG.log"),
            logging.StreamHandler()
            ])

        self.order = None
        self.donchian_stop_price = None
        self.price = None
        self.stop_price = None
        self.stop_donchian = None

        self.stochastic = bt.indicators.Stochastic(self.data, period=self.params.period, pe
        upperband=self.params.upperLimit, lowerband=self.params.lowerLimit)


    def next(self):
        '''Checks to see if Stochastic Oscillator, position, and order conditions meet the
        if self.order:
            # if there is a pending order, don't do anything
            return
        if self.position.size == 0:
```

```
            # When stochastic crosses back below 80, enter short position.
            if self.stochastic.lines.percD[-1] >= 80 and self.stochastic.lines.percD[0] <=
                # stop price at last support level in self.params.period periods
                self.donchian_stop_price = max(self.data.high.get(size=self.params.period))
                self.order = self.sell()
                # stop loss order for max loss of self.params.stop_pips pips
                self.stop_price = self.buy(exectype=bt.Order.Stop, price=self.data.close[0]
                # stop loss order for donchian SR price level
                self.stop_donchian = self.buy(exectype=bt.Order.Stop, price=self.donchian_s
            # when stochastic crosses back above 20, enter long position.
            elif self.stochastic.lines.percD[-1] <= 20 and self.stochastic.lines.percD[0] >
                # stop price at last resistance level in self.params.period periods
                self.donchian_stop_price = min(self.data.low.get(size=self.params.period))
                self.order = self.buy()
                # stop loss order for max loss of self.params.stop_pips pips
                self.stop_price = self.sell(exectype=bt.Order.Stop, price=self.data.close[0
                # stop loss order for donchian SR price level
                self.stop_donchian = self.sell(exectype=bt.Order.Stop, price=self.donchian_

        if self.position.size > 0:
            # When stochastic is above 70, close out of long position
            if (self.stochastic.lines.percD[0] >= 70):
                self.close(oco=self.stop_price)
        if self.position.size < 0:
            # When stochastic is below 30, close out of short position
            if (self.stochastic.lines.percD[0] <= 30):
                self.close(oco=self.stop_price)
```

4. Backtest 데이터 가공

수집한 데이터를 <Table 2> 같은 데이터프레임의 형태로 변환 필요

<Table 2> Sample Data

Date	Open	High	Low	Close	Volume
1995-05-02	121000.0	121000.0	118500.0	119500.0	139560.0
1995-05-03	119500.0	126500.0	119500.0	123500.0	382980.0
1995-05-04	124000.0	124500.0	122000.0	122500.0	175590.0
1995-05-06	122000.0	123500.0	122000.0	122000.0	47440.0
1995-05-08	122000.0	122000.0	120500.0	121000.0	91810.0

```
#시간순 재정렬
df = df.sort_values(by=['TRD_DD'])
df.reset_index(drop=True,inplace=True)
df['TRD_DD'] = pd.to_datetime(df['TRD_DD']) #datetime변환

#인풋 데이터 모양 맞춰주기(backtest에 들어갈 데이터 모양)
df_bt = df[['TRD_DD','TDD_OPNPRC','TDD_HGPRC','TDD_LWPRC','TDD_CLSPRC', 'ACC_TRDVOL']]
df_bt['TRD_DD'] = pd.to_datetime(df_bt['TRD_DD'])
df_bt.rename(columns={'TRD_DD':'Date', 'TDD_OPNPRC':'Open', 'TDD_HGPRC':'High','TDD_LWPRC
df_bt.set_index('Date',drop=True,inplace=True)


def backtest_data(data_bt):
  check_dtype = data_bt.dtype == 'object'
  if (check_dtype):
    return data_bt.str.replace(',','').astype('float')
  else :
    return data_bt.astype('float')


#데이터프레임 콤마(,) 제거 그리고 타입 소수로 변환
df_bt['Open'] = backtest_data(df_bt['Open'])
df_bt['High'] = backtest_data(df_bt['High'])
df_bt['Low'] = backtest_data(df_bt['Low'])
df_bt['Close'] = backtest_data(df_bt['Close'])
df_bt['Volume'] = backtest_data(df_bt['Volume'])
```

5. GA를 이용한 보조지표 파라미터 최적화

```
try:

  random.seed(3)

  PARAM_NAMES = ["pfast", "pslow"]  #PARAM_NAMES 부분은 각 전략마다 최적화할 부분 적용.

  NGEN = 5  # 알고리즘 5번 반복.
  NPOP = 100 #인구 초기
  CXPB = 0.5  #교차 전략
  MUTPB = 0.3  #돌연변이 전략.


  #최소fintness 설정 (fitness값이 작을수록 좋도록 설정)
  creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
  creator.create('Individual', list, fitness=creator.FitnessMin)
```

- 가장 수익률이 높을 때의 파라미터 채택하도록 설정

```
def evaluate(individual, plot=False, log=False):

  strategy_params = {k: v for k, v in zip(PARAM_NAMES, individual)}

  cerebro = bt.Cerebro(stdstats=False)

  data = bt.feeds.PandasData(dataname = df_bt)

  cerebro.adddata(data)

  initial_capital = 1000000
  cerebro.broker.setcash(initial_capital)

  cerebro.addstrategy(SmaCross1, **strategy_params)

  cerebro.addanalyzer(bt.analyzers.DrawDown)

  cerebro.broker.setcommission(commission=0.0025, margin=False)  #수수료 설정

  strats = cerebro.run()

  profit = cerebro.broker.getvalue() - initial_capital

  if profit == 0:
    return [np.inf]

  # max_dd = strats[0].analyzers.drawdown.get_analysis()["max"]["moneydown"] # max.mone
  # fitness = profit / (max_dd if max_dd > 0 else 1)
  fitness = round(1 / profit, 15)

  if log:
    print(f"Starting Portfolio Value: {initial_capital:,.2f}")
    print(f"Final Portfolio Value:  {cerebro.broker.getvalue():,.2f}")
    print(f"Total Profit:       {profit:,.2f}")
    print(f"Profit / Max DD:     {fitness}")

  # if plot:
    # cerebro.plot()

  return [fitness]
```

- 최적화할 파라미터 수치 범위 지정해주기

```
toolbox = base.Toolbox()
toolbox.register("indices", random.sample, range(NPOP), NPOP)

# crossover strategy
toolbox.register("mate", tools.cxUniform, indpb=CXPB)
```

```
# mutation strategy
toolbox.register("mutate", tools.mutUniformInt, low=1, up=151, indpb=0.2)
# selection strategy
toolbox.register("select", tools.selTournament, tournsize=3)
# fitness function
toolbox.register("evaluate", evaluate)


# definition of an individual & a population
toolbox.register("attr_sma1", random.randint, 1, 100)   #최적화할 파라미터 수치 범위 지정
toolbox.register("attr_sma2", random.randint, 151, 251)  #최적화할 파라미터 수치 범위 지
toolbox.register(
  "individual",
  tools.initCycle,
  creator.Individual,
  (
    toolbox.attr_sma1,
    toolbox.attr_sma2,

  ),
)

toolbox.register("population", tools.initRepeat, list, toolbox.individual)

mean = np.ndarray(NGEN)
best = np.ndarray(NGEN)
hall_of_fame = tools.HallOfFame(maxsize=3)

t = time.perf_counter()
pop = toolbox.population(n=NPOP)
for g in trange(NGEN):
  # Select the next generation individuals
  offspring = toolbox.select(pop, len(pop))
  # Clone the selected individuals
  offspring = list(map(toolbox.clone, offspring))

  # Apply crossover on the offspring
  for child1, child2 in zip(offspring[::2], offspring[1::2]):
    if random.random() < CXPB:
      toolbox.mate(child1, child2)
      del child1.fitness.values
      del child2.fitness.values

  # Apply mutation on the offspring
  for mutant in offspring:
    if random.random() < MUTPB:
      toolbox.mutate(mutant)
      del mutant.fitness.values

  # Evaluate the individuals with an invalid fitness
  invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
  fitnesses = toolbox.map(toolbox.evaluate, invalid_ind)
  for ind, fit in zip(invalid_ind, fitnesses):
```

```
            ind.fitness.values = fit

        # The population is entirely replaced by the offspring
        pop[:] = offspring
        hall_of_fame.update(pop)
        print(
          "\n HALL OF FAME:\n"
          + "\n".join(
            [
              f"  {_}: {ind}, Fitness: {ind.fitness.values[0]}"
              for _, ind in enumerate(hall_of_fame)
            ]
          )
        )

        fitnesses = [
          ind.fitness.values[0] for ind in pop if not np.isinf(ind.fitness.values[0])
        ]
        mean[g] = np.mean(fitnesses)
        best[g] = np.max(fitnesses)

      end_t = time.perf_counter()
      print(f"Time Elapsed: {end_t - t:,.2f}")

      # 최적의 파라미터 값 출력
      OPTIMISED_STRATEGY_PARAMS = {
        k: v for k, v in zip(PARAM_NAMES, hall_of_fame[0])}
      GDC_params = list(OPTIMISED_STRATEGY_PARAMS.values())
      print('**GDC 파라미터 값: ', GDC_params)
      print('\n')

    except:
      GDC_params = [50, 200]
```

6. -1,0,1로 이루어진 sell-buy signal로 변환

```
# 매도, 매수 전략 설정 후 GDC_sig 열 추가
first_cross = 0
for i in range(0, len(read_df)):
  if read_df['pfast'][i] < read_df['pslow'][i] and first_cross == 0:
    # print('Death cross on day', df['TRD_DD'][i], ':expect the price to continue to fall
    read_df['GDC_sig'][i] = 1
    first_cross=1
  elif read_df['pfast'][i] > read_df['pslow'][i] and first_cross ==1:
    # print('Golden cross on day', df['TRD_DD'][i], ':expect the price to continue to ris
    first_cross=0
```

```
            read_df['GDC_sig'][i] = -1
        else:
            read_df['GDC_sig'][i] = 0
```


Pages 6

Find a page...

- Home
- AWS S3 Bucket 생성
- AWS Sagemaker 노트북 인스턴스 생성
- AWS Sagemaker에서 S3에 저장된 데이터 불러오기
- GA를 이용한 보조지표 파라미터 최적화
    1. 필요한 패키지 설치
    2. 필요한 라이브러리 불러오기
    3. Backtrader Strategy 설정
    4. Backtest 데이터 가공
    5. GA를 이용한 보조지표 파라미터 최적화
    6. -1,0,1로 이루어진 sell-buy signal로 변환
- Scikit Learn 컨테이너를 사용하여 데이터 사전 병렬 처리

Clone this wiki locally

```
https://github.com/baeksumin/k-project.wiki.git
```

Scikit Learn 컨테이너를 사용하여 데이터 사전 병렬 처리

Jump to bottom

Myokyeong Kim edited this page on Feb 28, 2022 · 3 revisions

1. 필요한 라이브러리 불러오기

```
import boto3
import io
import pandas as pd
from sagemaker import get_execution_role
from sagemaker.sklearn.processing import SKLearnProcessor
from sagemaker.processing import ProcessingInput, ProcessingOutput
```

2. SKLearnProcessor 객체 만들기

```
role = get_execution_role()

sklearn_processor = SKLearnProcessor(
    framework_version="0.20.0",
    role=role,
    instance_type="ml.m5.xlarge",  #인스턴스 타입 설정
    instance_count=100             #인스턴스 개수 입력
    )
```

- 인스턴스 타입에 따라 전처리 작업 속도가 다르므로 인스턴스 타입과 개수를 고려하여 `SKLearnProcessor` 객체를 설정한다.
  (AWS sagemaker 인스턴스 타입 종류 확인하기)

  ※ 현재 (2022-02-04 기준) 는 100개가 최대 인스턴스로 되어 있다.

> 각 인스턴스별로 한 종목 (1995 ~ 2022.01.13 주가 데이터) 전처리 하는 데 소요되는 시간

인스턴스 타입	소요 시간
ml.t3.medium	약 2시간
ml.m5.large	약 30~40분
ml.m5.xlarge	약 20~30분

3. 사전 병렬 처리 Python 스크립트 작성하기

```
%%writefile preprocessing.py

import argparse
import os
import warnings
import pandas as pd
import numpy as np
from glob import glob

warnings.filterwarnings(action='ignore', category=DataConversionWarning)

if __name__ == "__main__":

    parser = argparse.ArgumentParser()
    args, _ = parser.parse_known_args()

    print("Received arguments {}".format(args))
    files = glob('/opt/ml/processing/input/*.csv')

    print("각 인스턴스 처리하는 파일 개수 : ", len(files))

    for i, f in enumerate(files):
        code = f[25:]                # ex) 'KR7099520009.csv' 부분만 불러오기
        df = pd.read_csv(f)            # 주가데이터 불러오기


        #불러온 주가데이터 전처리 작업 할 내용 여기에 작성


        output_path = os.path.join('/opt/ml/processing/processed_data' , code)    #저장할 경

        pd.DataFrame(df).to_csv(output_path, index=False)       #csv 파일로 저장
        print('Saving train data {}'.format(output_path))
```

- 작성된 preprocessing.py 확인하기

> [사전 처리 스크립트 참고 사이트]
>
> - https://github.com/aws/amazon-sagemaker-examples/blob/main/sagemaker_processing/scikit_learn_data_processing_and_model_evaluation/scikit_learn_data_processing_and_model_evaluation.ipynb
> - https://aws.amazon.com/ko/blogs/korea/amazon-sagemaker-processing-fully-managed-data-processing-and-model-evaluation/

4. sklearn_processor.run 작성하기

```
sklearn_processor.run(
    code="preprocessing.py",     # 3에서 작성한 사전 병렬 처리 Python 스크립트
    inputs=[ProcessingInput(source='s3://sagemaker-kproject/data18/',       # 전처리 작업할
                            s3_data_distribution_type='ShardedByS3Key',    # 병렬 처리 설정
                           destination="/opt/ml/processing/input")],


    outputs=[
        ProcessingOutput(source='/opt/ml/processing/processed_data',
                         destination = 's3://sagemaker-kproject/processed_data/')  #전처리
    ]
)
```

4. sklearn_processor.run 작성하기

Clone this wiki locally

```
https://github.com/baeksumin/k-project.wiki.git
```