**퀀트 투자 쿡북**

R을 이용한 나만의 투자 포트폴리오 만들기

여는글

Part 1 재료 수집 및 다듬기

1. 퀀트 투자의 심장: 데이터와 프로그래밍
2. 데이터 구하기
3. 퀀트 투자와 프로그래밍
4. R 프로그램
5. 퀀트 투자에 유용한 R 패키지
6. API를 이용한 데이터 수집
7. API를 이용한 Quandl 데이터 다운로드
8. getSymbols() 함수를 이용한 API 다운로드
9. 크롤링 이해하기
10. 파이프 오퍼레이터 (%>%)
11. HTML 이해하기
12. GET과 POST 방식 이해하기
13. 크롤링 예제
14. 금융 데이터 수집하기
15. 주식티커 다운로드
16. 수정주가 다운로드
17. 재무제표 및 가치지표 다운로드
18. 데이터 정리하기
19. 주가 정리하기
20. 재무제표 정리하기
21. 가치지표 정리하기
22. 미국 데이터 다운로드 및 정리하기
23. 주식티커 다운로드
24. API 이용한 수정주가 다운로드
25. 재무제표 및 가치지표 다운로드
26. 주가, 재무제표 및 가치지표 정리하기

Part 2 레시피를 이용한 종목 선정 및 포트폴리오 구성

1. 퀀트 투자 전략을 이용한 종목선정
2. 주식의 베타 계산
3. 저변동성 전략
4. 모멘텀 전략
5. 밸류 전략
6. 퀄리티 전략
7. 마법공식
8. 소형주 내 밸류 전략
9. QMV 전략(퀄리티, 모멘텀, 밸류)
10. 포트폴리오 구성
11. 최적화 기본: 뉴튼 방법
12. 최소분산 포트폴리오
13. 최대분산효과 포트폴리오
14. 위험균형 포트폴리오

Part 3 결과물 평가하기

1. 포트폴리오 백테스트
2. Return.Portfolio() 함수
3. 전통적인 60대 40 포트폴리오 백테스트
4. 시점 선택 전략 백테스트
5. 동적 자산배분 백테스트
6. 포트폴리오 성과 및 위험 측정

책을 마치며

**Part 1 재료 수집 및 다듬기**

**화려하고 복잡한 걸작을 요리할 필요는 없다.**

**다만 신선한 재료로 좋은 음식을 요리하라.**

**- 줄리아 차일드**

**1장. 퀀트 투자의 심장: 데이터와 프로그래밍**

몇 년 전까지만 하더라도 ‘퀀트 투자’는 일반 투자자들에게 매우 낯선 영역이었던 반면, 최근에는 각종 커뮤니티와 매체를 통해 이제는 익숙한 단어가 되었습니다. ‘퀀트 투자’에서 말하는 ‘퀀트’란 모형을 기반으로 금융상품의 가격을 산정하거나, 이를 바탕으로 투자를 하는 사람을 말합니다. ‘퀀트Quant’라는 단어가 ‘계량적’을 의미하는 퀀티터티브Quantitative에서 앞 글자를 따온 점을 생각하면 쉽게 이해가 될 것입니다.

‘퀀트 투자’ 역시 이와 비슷한 의미입니다. 사람들이 일반적으로 투자법이라 알고 있는 산업과 기업을 분석하여 가치를 매기는 ‘정성적인’ 투자법과는 달리, 수학과 통계를 기반으로 투자 전략을 만들고 이를 바탕으로 투자하는 ‘정량적인’ 투자법을 의미합니다.

퀀트 투자의 단계는 요리 과정과 비교하면 이해가 쉽습니다. 요리의 기본이 되는 요리 재료는 퀀트 투자의 기본이 되는 데이터와 같습니다. 데이터가 없으면 그 어떤 전략도 만들 수 없기 때문입니다. 요리 기구는 컴퓨터 프로그래밍과 같습니다. 요리 재료를 다양한 도구를 이용하여 가공하듯이, 가공되지 않은 수많은 데이터를 프로그래밍을 이용하여 가공하여야 합니다. 요리의 순서인 레시피는, 퀀트 투자의 전략과 같으며, 요리 과정을 통해 나온 음식은 투자 결과와 같습니다. 물론 이 모든 요리를 수행하는 요리사는 투자를 수행하는 투자자 본인 이겠지요.

**[표] 요리 과정과 퀀트 투자와의 비교**

|  |  |
| --- | --- |
| **요리 과정** | **퀀트 투자** |
| 요리 재료 | 데이터 |
| 요리 도구 | 프로그래밍 |
| 레시피 | 투자 전략 (종목 선택) |
| 데코레이션 | 포트폴리오 구성 |
| 음식 | 투자 결과 |

첫번째 파트에서는 요리의 재료와 도구에 해당하는 데이터와 프로그래밍에 살펴보도록 하겠습니다. 요리의 기본은 신선하고 풍부한 재료 이듯이, 퀀트 투자에 있어서도 신빙성이 있고 풍부한 데이터가 생명입니다. 유통기한이 지난 재료로 만든 음식은 먹을 수 없듯이, 현재의 상황을 반영하지 못하는 과거 데이터는 연구 목적으로는 활용할 수 있을지 몰라도 현재 시점에서 투자를 하기에는 사용하기 어려운 면이 있습니다. 신뢰할 수 없는 데이터로 만들어진 전략은 누구도 믿지 못할 것이며, 데이터가 풍부할 수록 더욱 다양한 전략을 실험해 볼 수 있기 때문입니다.

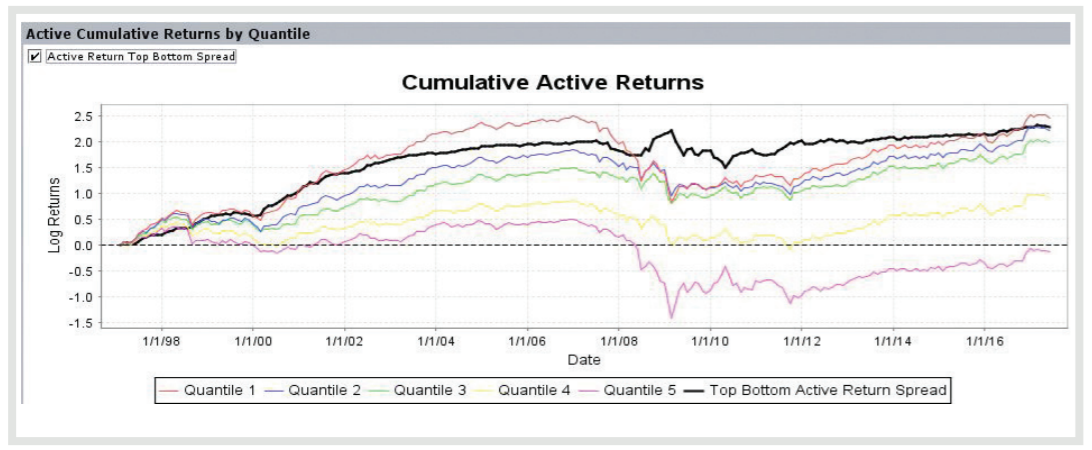
요리 재료가 준비된 후 요리를 하기 편하도록 썰거나 다지는 작업을 하듯이, 데이터 역시 분석을 위해 가공해줄 필요가 있습니다. 우리가 수집한 원시 데이터는 노이즈가 있는 경우가 대부분이므로, 프로그래밍을 통해 클랜징 작업을 해주어야 합니다. 예를 들어 인터넷에서 주가를 다운 받을 경우 “30,000” 처럼 글자 형태로 구해지는 경우가 많으며, 수익률 계산을 위해서는 30000 이라는 숫자 형태로 바꿔줄 필요가 있습니다. 본격적인 요리를 위해서는 냄비나 팬 등의 기구가 필요 하듯이, 코딩 혹은 프로그래밍을 통해 데이터를 다듬고 전략을 실험해 볼 수 있습니다.

**[1] 데이터 구하기**

요리 재료를 구하는 가장 간단한 방법은 마트에 가는 것이며, 배송을 통해 집에서 받아볼 수도 있습니다. 심지어 최근에는 가정식 대체식품HMR: Home Meal Replacement의 발전과 함께, 요리에 필요한 가공된 모든 재료를 한번에 구입할 수도 있습니다. 그러나 편리성과 가격은 비례하는 법입니다. 조금 더 저렴하게 요리 재료를 구입하기 위해서는 열심히 발품을 팔아야 하며, 가공되지 않은 원재료 그대로의 것을 산 뒤 내가 직접 가공하여야 합니다. 비용을 전혀 지불하지 않는 방법은 직접 재배하거나 사냥을 하러 다녀야겠죠.

퀀트 투자에 필요한 데이터 역시 이와 비슷합니다. 여러 데이터 제공업자들이 제공하는 서비스를 이용한다면 필요한 데이터를 매우 손쉽게 구할 수 있습니다. 글로벌 데이터 수집에는 블룸버그 혹은 Factset이, 국내 데이터 수집에는 DataGuide 혹은 Quantiwise가 흔히 사용됩니다. 물론 비용을 더 지불한다면 단순 데이터 수집뿐만 아니라 즉석에서 백테스트 및 성과 평가까지 가능합니다. Factset에서 판매하는 Alpha Testing 기능, 혹은 S&P Global에서 판매하는 ClariFI®를 사용한다면, 전세계 주식을 대상으로 원하는 전략의 백테스트 결과를 마우스 클릭과 몇 번의 동작만으로 수행할 수 있습니다.

**[그림] ClariFI®의 백테스트 기능**

****

이러한 데이터 제공업자들을 이용하는 방법의 최대 단점은 바로 비용입니다. 블룸버그 단말기의 경우 1년 사용료가 대리 한 명의 연봉과 비슷하여, 흔히들 ‘블대리’라 부르기도 합니다. 국내 데이터 업체의 경우 이보다 저렴하기는 하지만, 역시 1년 사용료가 수백 만원 정도로, 일반 개인 투자자들이 감당하기에는 부담이 가는 비용입니다.

해외데이터의 경우 Quandl이나 tiingo 등의 업체가 제공하는 서비스를 통해 상대적으로 저렴한 가격으로 데이터를 구할 수 있기도 합니다. 물론 대형 데이터 제공업체에 비해 데이터의 종류나 기간은 짧은 편이지만, 대부분의 퀀트 투자자들이 주식을 대상으로 한다는 점을 생각해보면 충분한 데이터를 얻을 수 있습니다. tiingo에서는 전세계 64,386개 주식의 30년 이상 가격 정보, 21,352개 주식의 12년 이상 재무정보를 월 $10에 받을 수 있으며[[1]](#endnote-1), 한정된 종목과 용량에 한해서는 무료로 데이터를 받을 수도 있습니다. 더군다나 API를 통해 프로그램 내에서 직접 데이터를 받을 수 있다는 편의성도 존재합니다.

**[그림] tiingo 제공 데이터 종류 및 가격**



그러나 아쉽게도 이러한 데이터에서 한국 시장의 정보는 소외가 되어있습니다. 따라서 돈을 들이지 않고 국내 데이터를 얻기 위해서는 직접 발품을 파는 수밖에 없습니다. 야후 파이낸스[[2]](#endnote-2)에서 제공하는 금융데이터를 API를 통해 받을 수도 있으며, 국내 금융 사이트들에서 제공하는 정보를 크롤링하여 데이터를 가공할 수도 있습니다. 이에 대해서는 각각 2장과 3장에서 다루도록 하겠습니다.

**[그림] NAVER 금융 제공 재무정보**



아래의 표는 국내의 금융 데이터를 구할 수 있는 사이트의 주소들 입니다. 해당 사이트의 정보를 잘만 활용한다면 장기간의 주가 및 재무정보를 무료로 수집할 수 있습니다. 물론 데이터 제공업체가 제공하는 깔끔한 형태의 데이터가 아니므로 클랜징 작업이 필요하다는 점, 그리고 상장폐지된 기업의 경우 데이터를 구하는 점이 힘들다는 단점이 있습니다. 그러나 비용이 들지 않는 다는 점, 그리고 현재 시점에서 투자 종목을 선택할 때는 상장폐지된 기업이 필요하지 않는다는 점을 고려하면, 이는 큰 문제가 되지 않는다 생각합니다.

**[표] 국내 금융관련 사이트 주소**

|  |  |
| --- | --- |
| **사이트명** | **주소** |
| 한국거래소 | http://krx.co.kr/ |
| Company Guide | http://comp.fnguide.com/ |
| Company Wise | http://comp.wisereport.co.kr/ |
| NAVER 금융 | http://finance.naver.com/ |
| Daum 금융 | http://finance.daum.net/ |
| 한국은행 경제통계시스템 | http://ecos.bok.or.kr/ |

**[2] 퀀트 투자와 프로그래밍**

좋은 재료를 구했다고 이를 바로 요리할 수는 없습니다. 감자볶음을 예로 들면, 단순히 감자를 구했다고 바로 요리가 완성되지 않습니다. 요리를 하기 위해서는 채칼을 이용해서 감자를 요리하기 편한 형태로 다듬어야 합니다. 물론 비용을 많이 지불한다면 채 썬 형태의 재료를 구하겠지만, 비용을 덜 지불한다면 껍질이 까진 정도의 감자를 구할 수 있을 것입니다. 비용을 아예 지불하지 않기 위해서 직접 논이나 밭으로 가 감자를 캐는 방법도 있습니다.

원하는 형태로 재료가 준비되었다면 이를 조리할 도구 역시 필요합니다. 감자를 볶을 팬과 볶음용 스푼이 있어야 제대로 된 요리를 할 수 있습니다. 열 전도율이 지나치게 떨어지는 팬을 사용할 경우 요리 과정도 오래 걸리며 맛 또한 떨어집니다. 볶음용 스푼이 제 역할을 하지 못한다면 역시나 골고루 볶아지지 않아 맛이 떨어집니다. 이처럼 요리도구의 성능과 요리의 맛은 비례하기 마련입니다.

퀀트 투자에서 이러한 요리 도구의 역할을 해주는 것이 바로 프로그래밍 입니다. 우리가 구한 데이터는 연구나 투자에 바로 사용할 수 있는 형태로 주어지는 경우가 거의 없기 때문에, 이를 목적에 맞게 처리하는 과정을 거쳐야 합니다. 이를 흔히 데이터 클랜징 작업이라 합니다. 또한, 데이터가 정제된 이후 이를 활용한 투자 전략의 백테스트나 종목 선정을 위해서도 프로그래밍은 필수입니다.

**[표] 요리 재료, 요리 도구와 퀀트 투자의 비교**

|  |  |
| --- | --- |
| **요리 재료, 요리 도구** | **퀀트 투자** |
| 감자 | 데이터 |
| 채 썰어진 감자 | 유료 데이터 (고급) |
| 껍질이 까진 감자 | 유료 데이터 (일반) |
| 직접 채취 | 무료 데이터 |
| 요리 재료 다듬기 | 클랜징 작업 |
| 채칼, 팬, 볶음용 스푼 | 프로그래밍 |

물론 모든 퀀트 투자에서 프로그래밍이 필수인 것은 아닙니다. 엑셀을 이용하여도 간단한 형태의 백테스트 및 종목 선정은 얼마든지 가능합니다. 그러나 응용성 및 효율성의 측면에서 엑셀을 이용하는 것은 매우 비효율적입니다.

데이터를 수집하고 클랜징 작업을 하는 경우, 몇 종목 되지 않는다면 엑셀을 이용하여도 충분히 가능합니다. 그러나 종목 수가 수 천 종목을 넘어갈 경우, 데이터를 손으로 일일이 처리하는 것은 사실상 불가능에 가깝습니다. 이러한 단순 반복 작업의 경우 프로그래밍을 이용한다면 훨씬 효율적으로 작업을 수행할 수 있습니다.

과거 12개월 누적 수익률이 높은 30종목에 투자하는 모멘텀 전략의 백테스트를 한다고 가정합시다. 처음에는 엑셀을 통해 백테스트를 하는 것이 편하다고 생각될 수 있습니다. 그러나 만일 12개월이 아닌 6개월 누적 수익률로 백테스트를 하고자 한다면 어떨까요? 다시 6개월 누적 수익률을 구하는 작업을 위해 명령어를 바꾸고 드래그를 해야 할 것입니다. 그러나 프로그래밍을 이용한다면 n=12 였던 부분을 n=6으로 변경한 후, 단지 클릭을 하면 됩니다.

전체 데이터가 100MB 정도라 가정할 때, 투자 전략이 계속해서 늘어날 경우는 어떨까요? 엑셀에서 A라는 전략을 백테스트 하기 위해서는 해당 데이터를 이용하여 작업을 한 후 저장합니다. 그 후 B라는 전략을 새롭게 백테스트 하려면 해당 데이터를 새로운 엑셀 파일에 복사하여 작업한 후 저장해야 합니다. 결과적으로 10개의 전략만 백테스트 하더라도 100MB 짜리 엑셀파일이 10개, 즉 1GB 정도의 엑셀 파일이 쌓이게 됩니다. 만일 데이터가 바뀔 경우, 다시 10개 엑셀 시트의 데이터를 일일이 바꿔야 하는 귀찮음도 감수해야 합니다. 물론 하나의 엑셀 파일 내에서 모든 전략을 수행할 수도 있지만, 이는 엄청난 속도 저하의 문제가 있습니다.

그러나 프로그래밍을 이용한다면 어떨까요? 백테스트을 수행하는 프로그래밍 스크립트는 불과 몇 KB도 되지 않습니다. 10개의 전략에 대한 스크립트 파일을 합해도 1MB가 되지 않습니다. 데이터가 바뀌더라도 원본 데이터 파일 하나만 수정해주면 모든 작업이 끝납니다.

물론 대부분의 사람들에게 프로그래밍은 매우 낯선 도구입니다. 그러나 퀀트 투자에 필요한 프로그래밍은 매우 반복적이기에 단어와 구문 몇 개만 익숙해지면 사용하는데 큰 어려움이 없습니다. 또한 개발자들의 프로그래밍에 비하면 엄청나게 쉬운 수준이므로, 비교적 빠른 시간 내에 원하는 전략을 테스트하고 수행하는 정도의 능력을 갖출 수도 있습니다.

최근에는 프로그래밍을 공부할 수 있는 좋은 온라인 사이트 및 학원도 늘어나고 있으므로, 프로그래밍을 처음 시작하기에도 큰 어려움이 없습니다. DataCamp[[3]](#endnote-3)는 사람들이 즐겨 이용하는 온라인 프로그래밍 학원으로써 R, Python, SQL과 같은 언어를 단계별로 배워 나갈 수 있으며, 월 사용료 $25로 모든 과정을 수강할 수 있습니다. 직접 코드를 입력해야 과정이 진행된다는 점에서 초보자들도 능동적으로 학습할 수 있는 장점이 있습니다.

**[3] R 프로그램**

인간이 사용하는 언어의 종류가 다양하듯이, 프로그램 언어의 종류 역시 매우 다양합니다. 대략700여개 이상의 프로그램 언어 중[[4]](#endnote-4) 사람들이 대중적으로 사용하는 언어는 그리 많지 않으므로, 대중성과 효율성을 위해 사용량이 많은 언어를 이용하는 것이 좋습니다.

아래 그림은 프로그래밍 언어의 사용 통계 순위입니다. 이 중 R과 Python은 금융에서 대표적으로 사용되는 언어로써, 10위권 내의 매우 대중적인 언어이기도 합니다. 해당 언어가 많이 사용되는 가장 큰 이유는 무료인 점, 그리고 일반인들도 사용하기에 매우 편한 형태로 언어가 구성되어 있다는 점입니다. 이 외에도 Matlab, VBA 등의 언어도 많이 사용됩니다.

**[그림] 2017년 기준 프로그래밍 언어 사용 통계 순위**

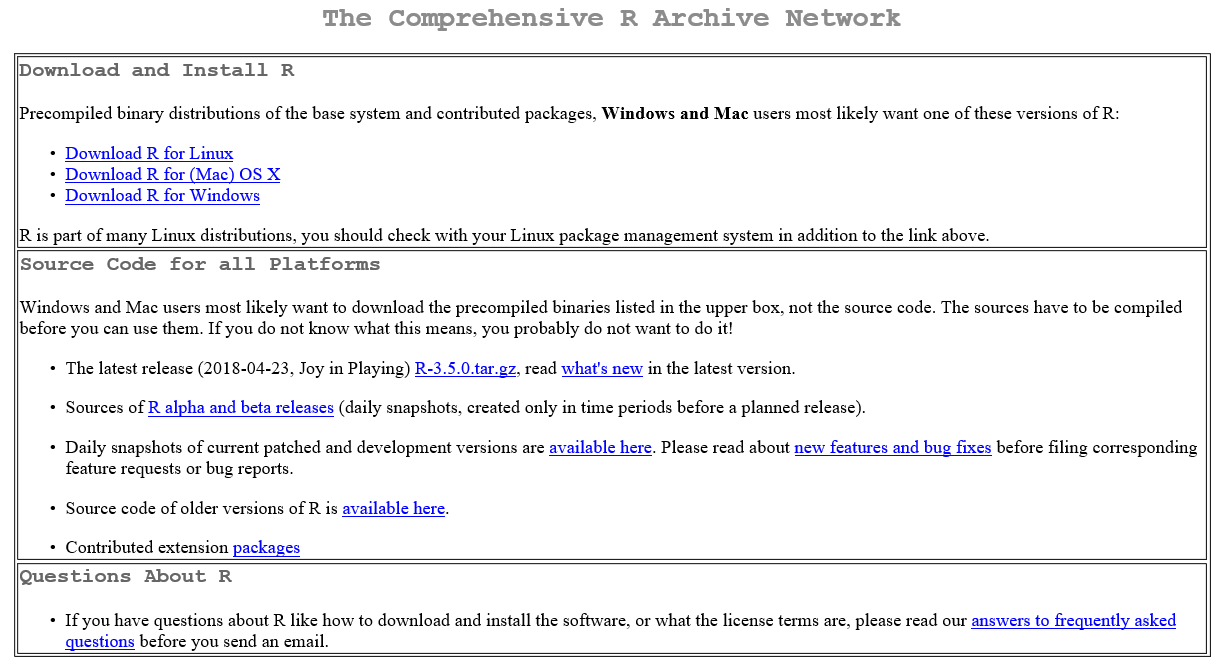


이러한 프로그래밍 언어 중 본 책에서는 R을 이용하였습니다. R의 장점은 무료라는 점 이외에도, 타 언어는 비교할 수 없는 다양한 패키지 입니다. 두터운 사용자 층을 기반으로 하여 R에는 상상할 수 없을 정도로 다양한 패키지가 존재하며, 특히 통계나 계량분석과 관련된 패키지는 독보적이라 할 수 있습니다.

**R과 RStudio 다운로드 받기**

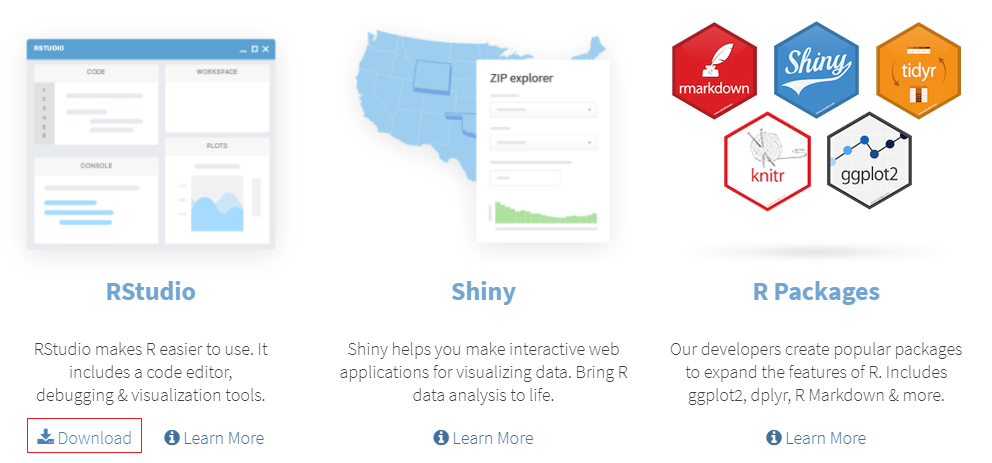
R을 다운로드 받는 방법은 매우 간단합니다. <https://cran.r-project.org/index.html>에 접속하여 Download and Install R 항목에서 본인이 사용하는 OS에 해당하는 항목을 선택한 후, base 항목을 선택하면 최신 버전의 R 설치 파일이 다운로드 됩니다. 설치 시 주의사항은 컴퓨터 이름이 한글로 되어있을 경우 프로그래밍 시 오류가 나므로, 컴퓨터 이름을 영어로 변경하기를 권장합니다.

**[그림] R 공식 홈페이지의 다운로드 항목**



R의 기본 콘솔인 RGui의 경우 인터페이스 및 사용환경이 지나치게 불편하므로, 대부분의 사용자들은 R의 IDE Integrated Development Environment 인 RStudio를 추가적으로 설치하여 사용합니다. RStudio의 공식 홈페이지인 <https://www.rstudio.com/>에 접속하여 Download를 클릭한 후, 페이지 하단의 Installers for Supported Platforms 항목에서 본인의 OS에 맞는 RStudio 설치 파일을 다운로드 받으면 됩니다. RStudio 역시나 무료로 이용이 가능합니다.

**[그림] RStudio공식 홈페이지의 다운로드 항목**



**[4] 퀀트 투자에 유용한 R 패키지**

여러 연구자 및 실무자들의 헌신적인 노력과 함께, R에는 금융 연구와 퀀트 투자를 위한 다양한 패키지들이 만들어져 있으며, 누구나 무료로 이용이 가능합니다. 그 중 해당 책의 내용에 사용되는 패키지 중 중요하다고 생각되는 것 위주로 소개하도록 하겠습니다. 각 패키지에 대한 자세한 설명은 구글에서 패키지 명으로 검색한 후, PDF 파일을 통해 확인할 수 있습니다.

* **quantmod**

이름에서 알 수 있듯이 퀀트 투자에 매우 유용한 패키지 입니다. API를 이용하여 데이터를 다운로드 받는 getSymbols() 함수는 너무나 많이 사용되는 함수이며, 이 외에도 볼린져밴드, 이동평균선, 상대 강도 지수RSI 등 여러 기술적 지표들을 주가 차트에 나타낼 수도 있습니다.

* **PerformanceAnalytics**

포트폴리오의 성과와 위험을 측정하는데 역시나 매우 유용한 패키지 입니다. 백테스트에 사용되는 Return.portfolio() 함수는 포트폴리오 단위의 백테스트에 필수적인 함수이며, 여러 종류의 차트를 아름답게 그릴 수 있습니다. 또한 성과를 측정하는 너무나 많은 함수를 이용하여, 포트폴리오의 성과뿐만 아니라 위험을 파악할 수 있습니다.

* **xts**

기본적으로 금융 데이터는 시계열 형태이며, 해당 패키지는 여러 데이터들을 시계열 형태eXtensible Time Series로 변형시켜 줍니다. 일별 수익률을 월별 수익률 혹은 연도별 수익률로 로 변환하는 apply.monthly()와 apply.yearly() 함수, 데이터들의 특정 시점을 찾아주는 endpoints() 함수 역시 백테스트에 필수적으로 사용되는 함수입니다. 해당 패키지는 PerformanceAnalytics 패키지 설치 시 자동으로 함께 설치됩니다.

* **zoo**

해당 패키지 역시 시계열 데이터를 다루는데 유용함 함수가 존재합니다. rollapply() 함수는 apply() 함수를 전체 데이터가 아닌 롤링-윈도우 기법으로 활용할 수 있게 해주며, na 데이터를 채워주는 na.locf() 함수는 시계열 데이터의 결측치를 보정할 때 매우 유용합니다.

* **rvest & httr**

데이터를 웹에서 직접 채집하기 위해서는 크롤링이 필수이며, rvest와 httr은 이에 사용되는 패키지 입니다. rvest의 경우 html 문서의 데이터를 가져오는데 사용되는 패키지이며, 웹 페이지에서 데이터를 크롤링 한 후 원하는 데이터만 뽑는데 필요한 여러 함수들이 포함되어 있습니다. httr의 경우 http의 표준 요청을 수행해주는 패키지로써 단순히 데이터를 받는 GET() 함수와 사용자가 필요한 값을 선택하여 요청하는 POST() 함수가 대표적으로 사용됩니다.

* **HenryQuant**

저자가 만든 패키지로써 금융 데이터 다운로드 및 정리, 포트폴리오 구성에 특화된 패키지입니다. 본 책에서 배울 대부분의 내용들이 패키지 내 함수로 만들어져 있으며, 퀀트 투자에 필요한 내용들을 지속적으로 업데이트 해 나가고 있습니다.

**패키지 설치하기**

R의 대부분 패키지는 CRAN에서 관리되며, 이러한 패키지의 경우 매우 간단하게 설치가 가능합니다. install.packages() 함수를 통해 쉽게 설치가 가능합니다. 다음은 quantmod 패키지의 설치 예시입니다. CRAN에서 접속하여 패키지를 다운받은 후 설치가 됨이 확인됩니다.

|  |
| --- |
| install.packages("quantmod")  Installing package into ‘C:/Users/Henry/Documents/R/win-library/3.5’  (as ‘lib’ is unspecified)  trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/quantmod\_0.4-13.zip'  Content type 'application/zip' length 942492 bytes (920 KB)  downloaded 920 KB  package ‘quantmod’ successfully unpacked and MD5 sums checked  The downloaded binary packages are in  C:\Users\Henry\AppData\Local\Temp\RtmpMdU8X9\downloaded\_packages |

간혹 서버 오류 등의 이유로 CRAN에 연결되지 않아 패키지가 설치되지 않는 경우가 발생하기도 합니다. 이럴 경우 직접 패키지 파일을 다운로드 받아 설치를 할 수도 있습니다. 구글에서 quantmod를 검색한 후 CRAN의 해당 패키지 페이지[[5]](#endnote-5)에 접속합니다. 그 후 Package source의 **quantmod\_0.4-13.tar.gz** 파일을 클릭하여 압축 파일을 다운로드 받습니다.

그 후 RStudio의 메뉴에서 Tools → Install Packages 항목을 선택한 후 Install from 메뉴를 Repository(CRAN)에서 Package Archive File로 변경합니다. 즉, CRAN에 접속하여 패키지를 설치하는 것이 아닌, 직접 다운로드 받은 압축파일을 통해 설치하도록 합니다. 그 후 팝업창에서 다운로드 받은 패키지 압축파일을 선택한 후 Install를 누르면 역시나 패키지가 설치됩니다.

최근에는 Github를 통해 본인의 패키지나 코드를 제한적으로 공유하는 경우도 있으며, HenryQuant 패키지도 이러한 예입니다. 이러한 경우 먼저 devtools 패키지를 다운로드 받은 후, 해당 패키지의 install\_github()함수를 이용하여 직접 github에서 패키지를 다운로드 받아 설치가 가능합니다. 아래는 HenryQuant 패키지의 설치 예시입니다.

|  |
| --- |
| install.packages("devtools")  devtools::install\_github("hyunyulhenry/HenryQuant")  Downloading GitHub repo hyunyulhenry/HenryQuant@master  from URL https://api.github.com/repos/hyunyulhenry/HenryQuant/zipball/master  Installing HenryQuant  ………………  \*\* building package indices  \*\* testing if installed package can be loaded  \*\*\* arch - i386  \*\*\* arch - x64  \* DONE (HenryQuant)  In R CMD INSTALL |

패키지를 다운로드 받았다고 바로 사용이 가능한 것은 아닙니다. library() 함수를 통해 해당 패키지를 불러온 후 사용이 가능하며, 해당 작업은 Rstudio를 열 때마다 실행해 주어야 합니다. 패키지를 설치할 때와는 달리, 괄호 안에 큰 따옴표(“”)는 입력하지 않아도 됩니다. quantmod 패키지를 불러오는 예시는 다음과 같습니다.

|  |
| --- |
| library(quantmod)  필요한 패키지를 로딩중입니다: xts  필요한 패키지를 로딩중입니다: zoo  다음의 패키지를 부착합니다: ‘zoo’  The following object is masked from ‘package:timeSeries’:  time<-  The following objects are masked from ‘package:base’:  as.Date, as.Date.numeric  필요한 패키지를 로딩중입니다: TTR  다음의 패키지를 부착합니다: ‘TTR’  The following object is masked from ‘package:fBasics’:  volatility  Version 0.4-0 included new data defaults. See ?getSymbols.  Learn from a quantmod author: https://www.datacamp.com/courses/importing-and-managing-financial-data-in-r |

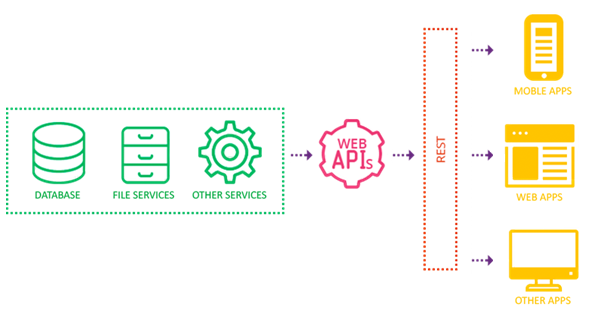
패키지들은 종종 업데이트 되어 새로운 명령어가 추가되거나, 오류가 고쳐지기도 합니다. RStudio의 메뉴에서 Tools → Check for Package Updates 항목을 선택하여, 업데이트 된 패키지를 새롭게 다운로드 받을 수 있습니다.

**2장. API를 이용한 데이터 수집**

본 장과 다음 장에서는 본격적으로 데이터를 수집하는 방법에 대해 배우도록 하겠습니다. 그 중 해당 장에서는 API를 이용하여 데이터를 수집하는 방법에 대해 살펴보도록 합니다.

API에 대한 개념을 간단히 설명하면 다음과 같습니다. API 제공자는 본인이 가진 데이터베이스를 다른 누군가가 쉽게 사용할 수 있는 형태로 가지고 있으며, 해당 데이터베이스에 접근할 수 있는 열쇠인 API 주소를 가진 사람은 이를 언제든지 사용할 수 있습니다.

**[그림] API 개념**

****

API 주소만 가지고 있다면 데이터를 언제, 어디서, 누구나 쉽게 이용할 수 있다는 장점이 있습니다. 또한 대부분의 경우 사용자가 필요한 데이터만을 가지고 있으므로 접속 속도가 빠르며, 데이터를 가공하는 번거로움도 줄어듭니다.

해외의 경우 금융 데이터도 API의 형태로 제공하는 업체가 많으므로, 이를 잘만 활용한다면 매우 손쉽게 퀀트 투자에 필요한 재료를 수집할 수 있습니다.

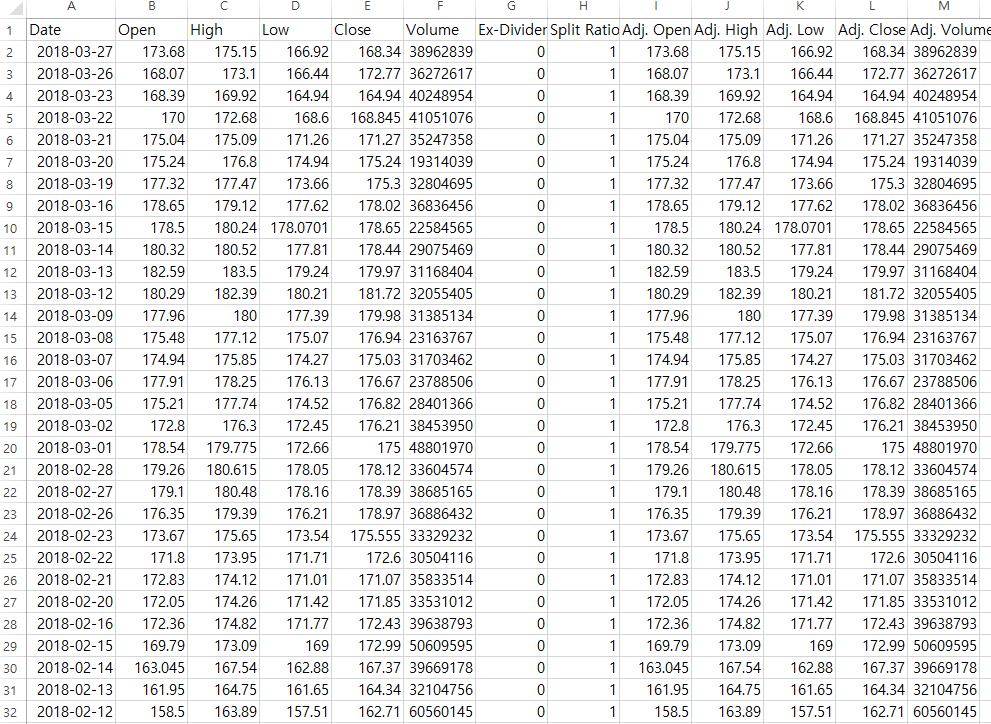
**[1] API를 이용한 Quandl 데이터 다운로드**

데이터 제공업체 Quandl은 일부 데이터를 무료로 제공하며, API를 통해서 이를 다운로드 받을 수 있습니다.[[6]](#endnote-6) 해당 책에서는 예제로써 애플(AAPL)의 주가를 다운로드 받도록 하겠습니다. csv 형식의 API 주소는 다음과 같습니다.

|  |
| --- |
| <https://www.quandl.com/api/v3/datasets/WIKI/AAPL/data.csv?api_key=xw3NU3xLUZ7vZgrz5QnG> |

위 주소를 웹 브라우저 주소 창에 직접 입력하면 csv 형식의 파일이 다운로드 되며, 이를 열어보면 애플의 주가에 해당하는 정보들이 다운로드 됨이 확인됩니다.

**[그림] API 주소를 이용한 데이터 다운로드**

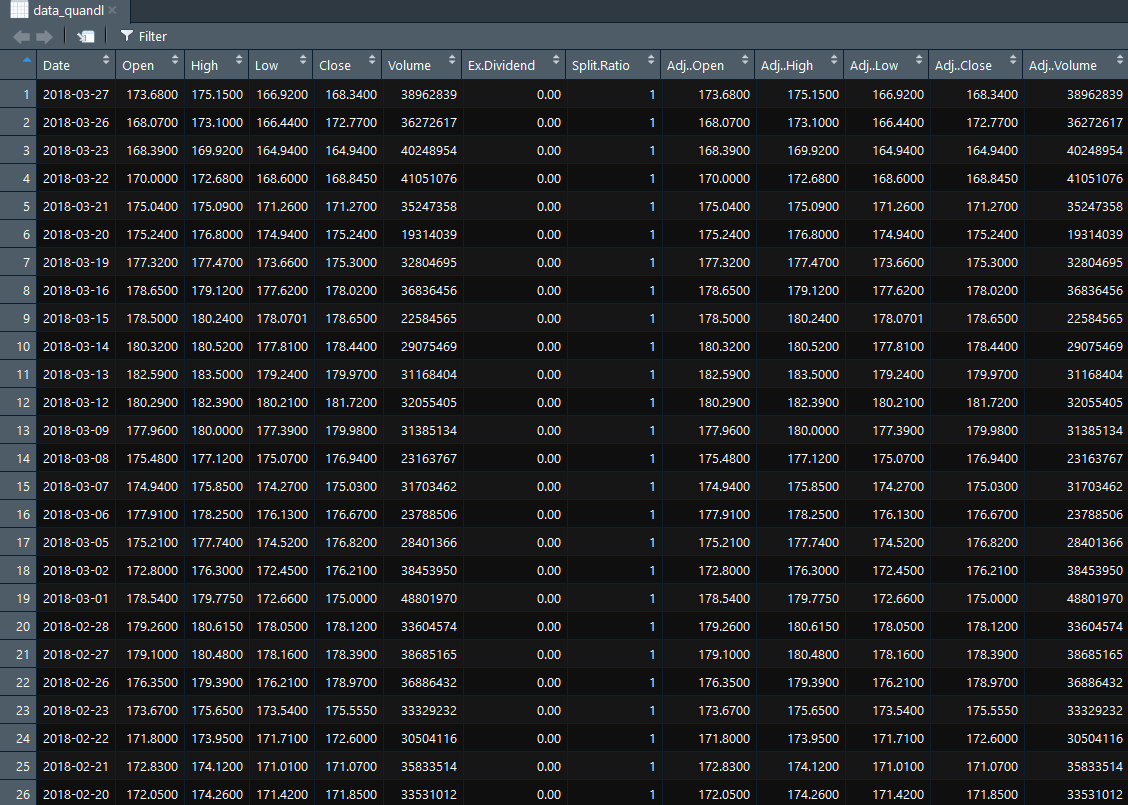
****

그러나 웹 브라우저에 해당 주소를 입력하여 csv 파일을 다운로드 받고, 이를 다시 R에서 불러오는 작업은 무척이나 비효율 적입니다. R 내에서 API 주소를 이용하여 직접 데이터를 다운로드 받아올 수 있습니다.

|  |
| --- |
| url1 = "https://www.quandl.com/api/v3/datasets/WIKI/AAPL/data.csv?api\_key=xw3NU3xLUZ7vZgrz5QnG"  data\_quandl = read.csv(url1) |

url1에 해당 주소를 입력한 후, read.csv() 함수를 이용하여 간단하게 csv 파일을 불러올 수 있습니다.

**[그림] R 내에서 API 주소를 이용한 데이터 다운로드**

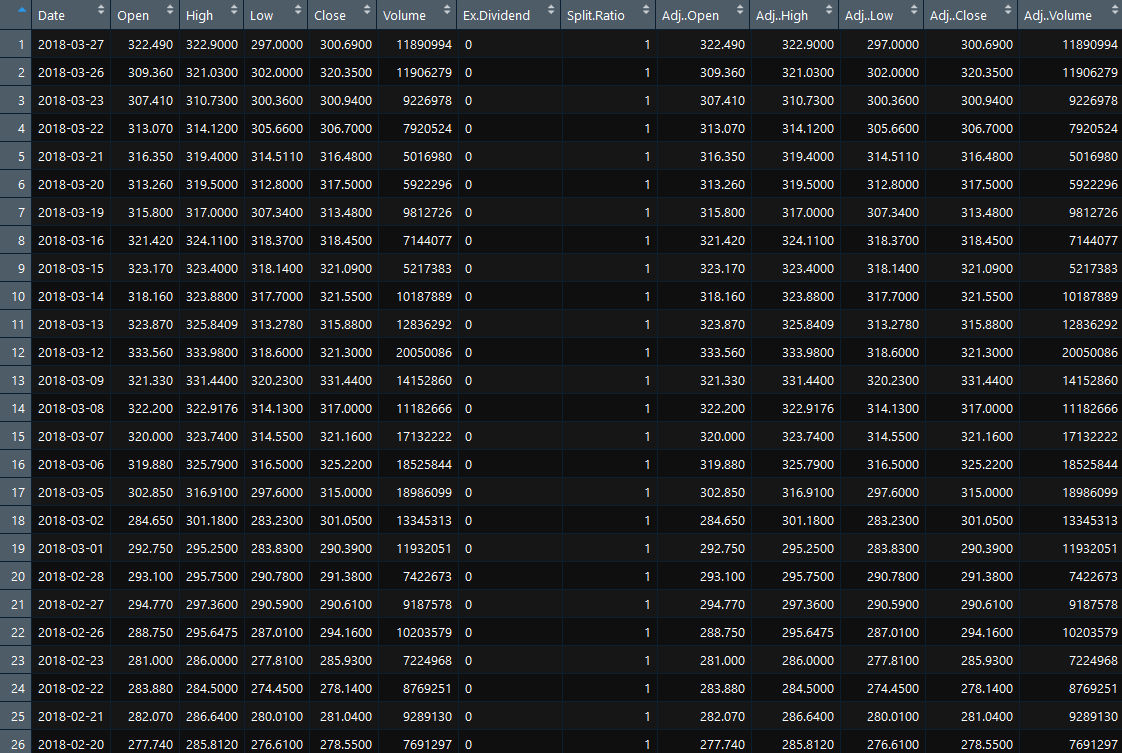


API 주소를 조금만 수정하면, 다른 종목의 데이터도 다운로드 받을 수 있습니다. 위의 주소에서 애플의 티커에 해당하는 **AAPL**를 넷플렉스의 티커에 해당하는 **NFLX**로 변경하여 데이터를 다운로드 받아보도록 하겠습니다.

|  |
| --- |
| url2 = "https://www.quandl.com/api/v3/datasets/WIKI/NFLX/data.csv?api\_key=xw3NU3xLUZ7vZgrz5QnG"  data\_quandl2 = read.csv(url2) |

역시나 넷플릭스의 주가를 손쉽게 다운로드 받을 수 있음이 확인됩니다.

**[그림] API 주소를 이용한 넷플릭스 데이터 다운로드**



**[2] getSymbols() 함수를 이용한 API 다운로드**

앞선 예에서 API 주소를 이용할 경우 매우 간단하게 데이터를 수집할 있음을 살펴 보았습니다. 그러나 해당 방법에는 여러 단점 또한 존재합니다. 먼저, 원하는 항목에 대한 API를 일일이 얻는 것이 힘든 일입니다. 또한 Quandl의 경우 무료로 얻을 수 있는 정보에 제한이 있으며, 다운로드 양에 대한 제한도 있습니다. 한 두 종목의 경우 해당 방법으로 데이터를 수집할 수 있지만, 전 종목의 데이터를 해당 방법으로 구하는 것은 사실상 불가능 합니다.

다행히 야후 파이낸스에서도 주가 데이터를 무료로 제공하며, quantmod 패키지의 getSymbols() 함수는 해당 API에 접속하여 데이터를 다운로드 받아옵니다.

**주가 다운로드**

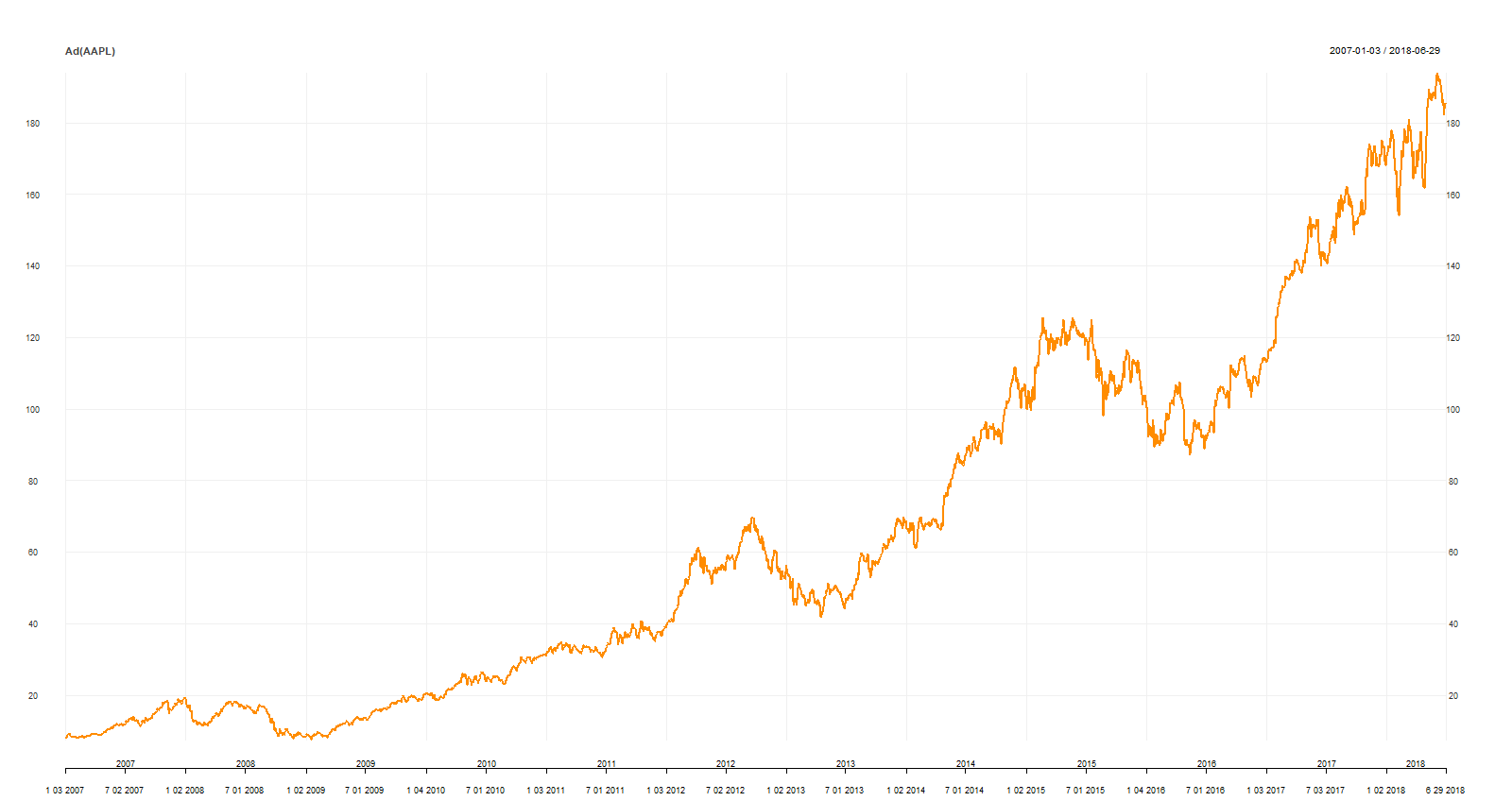
getSymbols() 함수의 기본적인 사용법은 매우 간단합니다. 괄호 안에 다운로드 받고자 하는 종목의 티커를 입력하면 됩니다. 아래의 코드는 애플의 티커인 “AAPL”을 입력한 경우이며, 티커와 동일한 변수인 AAPL이 생성되며, 주가 데이터가 다운로드 된 후 xts 형태로 입력됩니다.

|  |
| --- |
| library(quantmod)  getSymbols("AAPL")  head(AAPL)  AAPL.Open AAPL.High AAPL.Low AAPL.Close AAPL.Volume AAPL.Adjusted  2007-01-03 12.32714 12.36857 11.70000 11.97143 309579900 8.073009  2007-01-04 12.00714 12.27857 11.97429 12.23714 211815100 8.252192  2007-01-05 12.25286 12.31428 12.05714 12.15000 208685400 8.193427  2007-01-08 12.28000 12.36143 12.18286 12.21000 199276700 8.233888  2007-01-09 12.35000 13.28286 12.16429 13.22429 837324600 8.917878  2007-01-10 13.53571 13.97143 13.35000 13.85714 738220000 9.344647 |

다운로드 결과로써 총 6개의 열이 생성됩니다. Open은 시가, High는 고가, Low는 저가, Close는 종가를 의미합니다. 또한, Volume은 거래량을 의미하며, Adjusted는 배당이 반영된 수정주가[[7]](#endnote-7)를 의미합니다. 이 중 가장 많이 사용되는 데이터는 Adjusted, 즉 배당이 반영된 수정주가입니다. Ad() 함수는 getSymbols() 함수를 통해 다운로드 받은 데이터에서 수정주가만을 선택하여 줍니다. 시계열 그래프를 그려주는 chart\_Series()와 함께 수정주가를 그리면 다음과 같습니다.

|  |
| --- |
| chart\_Series(Ad(AAPL)) |

**[그림] 애플의 수정주가 그래프**



시계열 기간을 입력하지 않을 경우 2007년 1월부터 현재까지의 데이터가 다운로드 되며, 추가적인 입력 변수를 통해 원하는 기간의 데이터를 다운로드 받을 수도 있습니다.

|  |
| --- |
| getSymbols("AAPL", from = "2000-01-01", to = "2017-12-31")  head(AAPL)  AAPL.Open AAPL.High AAPL.Low AAPL.Close AAPL.Volume AAPL.Adjusted  2000-01-03 3.745536 4.017857 3.631696 3.997768 133949200 2.695920  2000-01-04 3.866071 3.950893 3.613839 3.660714 128094400 2.468626  2000-01-05 3.705357 3.948661 3.678571 3.714286 194580400 2.504751  2000-01-06 3.790179 3.821429 3.392857 3.392857 191993200 2.287994  2000-01-07 3.446429 3.607143 3.410714 3.553571 115183600 2.396373  2000-01-10 3.642857 3.651786 3.383929 3.491071 126266000 2.354226  tail(AAPL)  AAPL.Open AAPL.High AAPL.Low AAPL.Close AAPL.Volume AAPL.Adjusted  2017-12-21 174.17 176.02 174.10 175.01 20949900 173.6298  2017-12-22 174.68 175.42 174.50 175.01 16349400 173.6298  2017-12-26 170.80 171.47 169.68 170.57 33185500 169.2248  2017-12-27 170.10 170.78 169.71 170.60 21498200 169.2546  2017-12-28 171.00 171.85 170.48 171.08 16480200 169.7308  2017-12-29 170.52 170.59 169.22 169.23 25999900 167.8954 |

from에는 시작 기간을, to에는 종료 기간을 입력하여 주면, 해당 시점의 데이터가 다운로드 받아짐이 확인됩니다.

|  |
| --- |
| data = getSymbols("AAPL", from = "2000-01-01", to = "2017-12-31", auto.assign = FALSE)  head(data)  AAPL.Open AAPL.High AAPL.Low AAPL.Close AAPL.Volume AAPL.Adjusted  2000-01-03 3.745536 4.017857 3.631696 3.997768 133949200 2.695920  2000-01-04 3.866071 3.950893 3.613839 3.660714 128094400 2.468626  2000-01-05 3.705357 3.948661 3.678571 3.714286 194580400 2.504751  2000-01-06 3.790179 3.821429 3.392857 3.392857 191993200 2.287994  2000-01-07 3.446429 3.607143 3.410714 3.553571 115183600 2.396373  2000-01-10 3.642857 3.651786 3.383929 3.491071 126266000 2.354226  tail(data)  AAPL.Open AAPL.High AAPL.Low AAPL.Close AAPL.Volume AAPL.Adjusted  2017-12-21 174.17 176.02 174.10 175.01 20949900 173.6298  2017-12-22 174.68 175.42 174.50 175.01 16349400 173.6298  2017-12-26 170.80 171.47 169.68 170.57 33185500 169.2248  2017-12-27 170.10 170.78 169.71 170.60 21498200 169.2546  2017-12-28 171.00 171.85 170.48 171.08 16480200 169.7308  2017-12-29 170.52 170.59 169.22 169.23 25999900 167.8954 |

getSymbols() 함수를 통해 다운로드 받은 데이터는 자동으로 티커와 동일한 변수명에 저장됩니다. 만일 티커명이 아닌 원하는 변수명에 데이터를 저장하고 싶을 경우, 추가적인 입력 변수가 필요합니다. 아래 코드처럼 데이터를 저장하고자 하는 변수를 지정한 후, auto.assign = FALSE 변수를 추가적으로 입력하면 AAPL 데이터가 data 변수에 데이터가 저장됩니다.

|  |
| --- |
| ticker = c("FB", "NVDA")  getSymbols(ticker)  head(FB)  FB.Open FB.High FB.Low FB.Close FB.Volume FB.Adjusted  2012-05-18 42.05 45.00 38.00 38.23 573576400 38.23  2012-05-21 36.53 36.66 33.00 34.03 168192700 34.03  2012-05-22 32.61 33.59 30.94 31.00 101786600 31.00  2012-05-23 31.37 32.50 31.36 32.00 73600000 32.00  2012-05-24 32.95 33.21 31.77 33.03 50237200 33.03  2012-05-25 32.90 32.95 31.11 31.91 37149800 31.91  head(NVDA)  NVDA.Open NVDA.High NVDA.Low NVDA.Close NVDA.Volume NVDA.Adjusted  2007-01-03 24.71333 25.01333 23.19333 24.05333 28870500 22.27390  2007-01-04 23.96667 24.05333 23.35333 23.94000 19932400 22.16896  2007-01-05 23.37333 23.46667 22.28000 22.44000 31083600 20.77992  2007-01-08 22.52000 23.04000 22.13333 22.60667 16431700 20.93426  2007-01-09 22.64000 22.79333 22.14000 22.16667 19104100 20.52681  2007-01-10 21.93333 23.46667 21.60000 23.26000 27718600 21.53926 |

한번에 여러 종목의 주가를 다운로드 받을 수도 있습니다. 아래 예제와 같이 페이스북과 엔비디아의 티커인 “FB”와 “NVDA”을 ticker 변수에 입력하여 주고, getSymbols() 함수에 티커들을 입력한 변수를 넣어주면 두 종목의 주가가 동시에 다운로드 됩니다.

**국내 종목 주가 다운로드**

getSymbols() 함수를 이용하면 미국뿐 아니라 국내 종목의 주가를 다운로드 받을 수도 있습니다. 국내 종목의 티커는 총 6자리로 구성되어 있으며, 해당 함수에 입력되는 티커는 코스피 상장 종목의 경우 “티커.KS”, 코스닥 상장 종목의 경우 “티커.KQ”의 형태로 입력해 주어야 합니다.

|  |
| --- |
| getSymbols("005930.KS", from = "2000-01-01", to = "2017-12-31")  tail(Ad(`005930.KS`))  005930.KS.Adjusted  2017-12-19 18180.67  2017-12-21 17327.35  2017-12-22 17524.81  2017-12-26 16995.89  2017-12-27 31420.12  2017-12-28 32438.60 |

먼저 코스피 상장종목의 예시입니다. 삼성전자의 티커인 005930에 .KS를 붙여 함수에 입력할 경우, 티커명에 해당하는 005930.KS 변수명에 데이터가 저장됩니다. 변수명에 콤마(.)가 있는 관계로, Ad 함수를 통해 수정주가를 확인하고자 할 때는 변수명의 앞뒤에 억음부호(`)를 붙여주어야 합니다.

해당 데이터를 살펴보면 17년 12월 26일 수정주가는 16995.89인 반면, 다음날인 12월 27일의 수정주가는 31420.12로써 하루 동안 무려 2배 가까이 상승한 것처럼 보이며, 이는 잘못된 데이터가 서버에 저장되어 있기 때문입니다. 물론 시간이 지나면서 오류가 있는 데이터는 수정이 되지만, 신뢰성이 떨어지는 데이터를 사용하는 것은 권장하지 않습니다. 따라서 국내 종목의 경우 배당이 반영된 값 보다는 단순 종가를 사용하기를 권장합니다.

|  |
| --- |
| > tail(Cl(`005930.KS`))  005930.KS.Close  2017-12-19 51560  2017-12-21 49140  2017-12-22 49700  2017-12-26 48200  2017-12-27 49360  2017-12-28 50960 |

Cl() 함수는 Close, 즉 종가만을 선택하여 주며, 사용 방법은 기존 Ad() 함수와 동일합니다. 비록 배당을 고려할 수는 없지만, 전반적으로 오류가 없는 데이터를 사용할 수 있습니다.

|  |
| --- |
| getSymbols("068760.KQ", from = "2000-01-01", to = "2017-12-31")  > tail(Cl(`068760.KQ`))  068760.KQ.Close  2017-12-19 58921.6  2017-12-21 56372.6  2017-12-22 59215.7  2017-12-26 60686.3  2017-12-27 62000.0  2017-12-28 61200.0 |

이번엔 코스닥 상장종목의 예시입니다. 셀트리온제약의 티커인 068670에 .KQ를 붙여 함수에 입력합니다. 역시나 데이터가 다운로드 되어 티커명의 변수에 저장됩니다.

**FRED 데이터 다운로드**

미국 및 각국의 중요 경제지표 데이터를 살펴볼 때, 가장 많이 참조되는 곳 중 하나가 미 연방 준비 은행에서 관리하는 Fred Economic Date 입니다. getSymbols() 함수를 통해 FRED 데이터를 다운로드 받을 수 있습니다. 먼저 미 국채 10년물 금리를 다운로드 받는 예제를 살펴보도록 하겠습니다.

|  |
| --- |
| getSymbols("DGS10", src=”FRED”)  chart\_Series(DGS10) |

먼저 미 국채 10년물 금리에 해당하는 티커인 “DGS10”을 입력해 줍니다. 그 후, 데이터 소스에 해당하는 src에 “FRED”를 입력해 주면, 해당 데이터가 다운로드 됩니다. chart\_Series를 통해 해당 데이터를 그래프로 나타내면 다음과 같습니다.

**[그림] 미 국채 10년물 금리 그래프**



각 항목 별 티커를 찾는 방법은 매우 간단합니다. 먼저 FRED의 홈페이지[[8]](#endnote-8)에 접속하여, 원하는 데이터를 검색합니다. 예시로써 원/달러 환율에 해당하는 **South Korea / U.S. Foreign Exchange Rate** 를 검색하여 원하는 페이지에 접속합니다. 이 중 페이지 주소에서 /series/ 다음에 위치하는 **DEXKOUS** 가 해당 항목의 티커입니다.

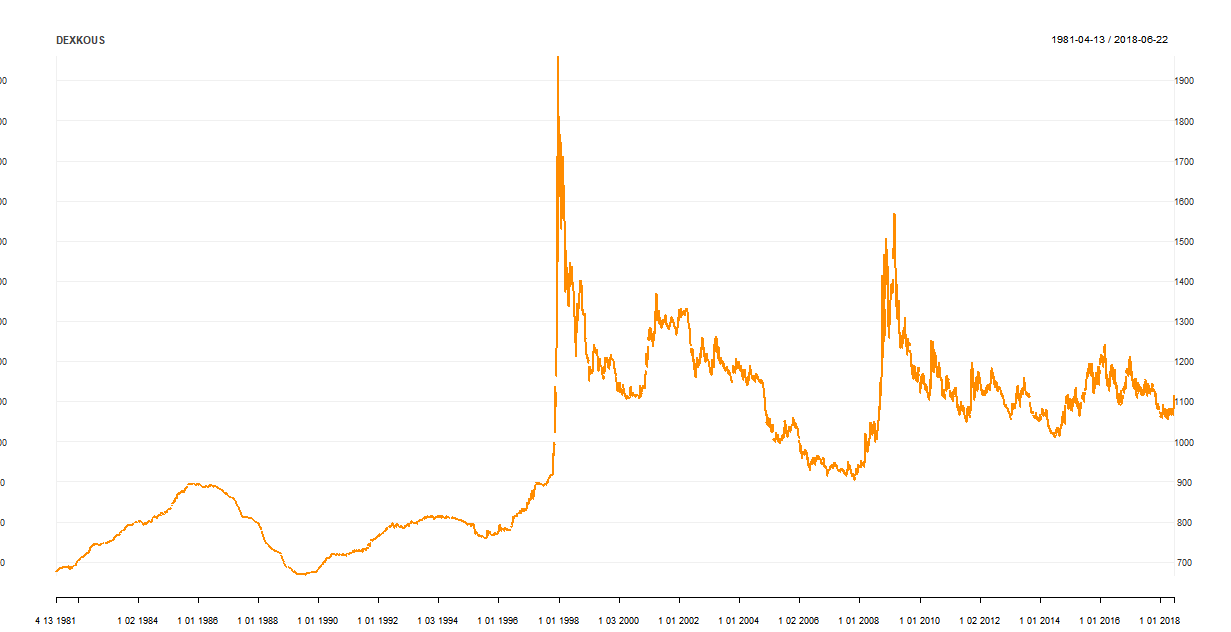
**[그림] FRED 사이트 내 원/달러 환율의 티커 확인**



해당 티커를 입력하면, 홈페이지와 동일한 데이터가 다운로드 됨이 확인됩니다. 이 외에도 509,000 여개의 방대한 FRED 데이터를 해당 함수를 통해 손쉽게 R에서 다운로드 받을 수 있습니다.

|  |
| --- |
| getSymbols("DEXKOUS", src="FRED")  tail(DEXKOUS)  DEXKOUS  2018-06-15 1101.89  2018-06-18 1105.07  2018-06-19 1110.64  2018-06-20 1107.42  2018-06-21 1110.01  2018-06-22 1114.49  chart\_Series(DEXKOUS) |

**[그림] 원/달러 환율 그래프**



**3장. 크롤링을 이해하기**

앞선 장에서 볼 수 있듯이 API를 이용할 경우 데이터를 매우 쉽게 수집할 수 있지만, 국내 주식 데이터를 다운로드 받기에는 한계가 있으며, 우리가 원하는 데이터가 API의 형태로 제공된다는 보장도 없습니다. 따라서 우리는 요리에 필요한 감자를 캐기 위해 논과 밭으로 가듯이, 데이터를 얻기 위해 직접 찾아나서야 합니다.

다행히도 금융 사이트들에는 주가, 재무정보 등 우리가 원하는 대부분의 주식 정보가 제공되고 있으며, API를 활용할 수 없는 경우에도 크롤링을 통해 이러한 데이터를 수집할 수 있습니다.

크롤링 혹은 스크래핑이란 웹사이트에서 원하는 정보를 수집하는 기술입니다. 대부분의 금융 사이트들이 간단한 형태로 작성되어 있어, 몇 가지 기술만 익히면 어렵지 않게 데이터를 크롤링 할 수 있습니다. 그러나 크롤링에 대한 대부분의 강의나 설명이 파이썬을 이용한 방법으로써, 초보자가 R을 이용한 크롤링을 배우는데는 어려움이 있습니다.

해당 장에서는 크롤링에 대한 간단한 설명과 주식티커, 수정주가, 재무제표 및 가치지표를 크롤링 하는법에 대해 알아보도록 합니다.

크롤링을 할 때는 주의해야 할 점이 있습니다. 특정 사이트의 페이지를 쉬지 않고 크롤링을 하는 행위를 ‘무한 크롤링’이라 합니다. 이러한 경우 해당 사이트의 자원을 독점하게 되어 타인의 사용을 막게 되며, 사이트에 부하를 주게 됩니다. 일부 사이트에서는 동일한 IP로 쉬지 않고 크롤링을 할 경우 접속을 막아버리는 경우도 있습니다. 따라서 하나의 페이지를 크롤링 한 후, 1~2초 가량 정지한 후 다시 다음 페이지를 크롤링 할 필요가 있습니다.

**[1] 파이프 오퍼레이터 (%>%)**

논과 밭에서 신선한 감자를 캐왔다 한들 이를 바로 사용할 수는 없습니다. 먼저 흙을 털어내고 깨끗이 씻어야 합니다. 그 후 껍질을 벗긴 후 채 썰기를 해야 요리에 필요한 재료로써 준비가 됩니다. 즉, 얻어진 재료를 바탕으로 단계별로 손질을 해 나갑니다.

그런데 재료를 다듬는 단계별로 서로 다른 주방에서 일을 처리한다면 어떻게 될까요? 흙을 털어내고 씻는 과정은 1번 주방에서 하고, 이를 다시 2번 주방으로 들고가 껍질을 벗긴 후, 다시 3번 주방으로 가서 채 썰기를 한다면 말입니다. 얼핏 보기에도 매우 비효율적인 과정입니다. 동일한 재료를 손질한다면, 동일한 주방에서 손질을 마치는 것이 가장 효율적입니다.

금융 데이터 크롤링 역시 감자의 재료 손질 단계와 비슷합니다. 먼저 우리가 원하는 데이터가 있는 페이지를 크롤링 한 후, 한 단계씩 구조를 내려가며 최종적으로 텍스트 혹은 테이블을 가져옵니다. R 내에서 동일한 데이터를 대상으로 연속적으로 작업하게 해주는 오퍼레이터(연산자)가 바로 파이프 오퍼레이터 입니다.

해당 오퍼레이터는 magrittr 패키지를 설치하면 사용할 수 있으며, 크롤링에 꼭 필요한 rvest 패키지를 설치할 경우 자동으로 magrittr 패키지가 설치되어 사용할 수 있습니다. magrittr 이라는 이름은 파이프 그림과 ‘이것은 파이프가 아니다’라는 문구로 모순적 어법을 창조한 르네 마그리트René Magritte의 이름에서 따왔습니다.

**[그림] 르네 마그리트의 ‘이미지의 반역’과 magrittr 패키지의 심블**



흔히 프로그래밍에서 x라는 데이터를 F()라는 함수에 넣어 결과값을 확인하고 싶을 경우, F(x)의 방법을 사용합니다. 예를 들어 3과 5라는 데이터 중 큰 값을 찾고 싶을 때는 max(3,5)를 통해 확인합니다. 이를 통해 나온 결과 값을 또 다시 G()라는 함수에 넣어 결과값을 확인하고자 할 경우, 비슷한 과정을 거칩니다. max(3,5)를 통해 나온 값의 제곱근을 구하고자 할 경우 result = max(3,5)를 통해 첫 번째 결과값을 저장하고, sqrt(result)를 통해 두 번째 결과값을 계산합니다. 물론 sqrt(max(3,5))와 같은 표현법으로 한번에 표현할 수 있습니다.

이러한 표현의 단점은, 계산하는 함수가 많아질수록 저장하는 변수가 늘어나고 혹은 괄호가 지나치게 길어집니다. 첫 번째 방법처럼 일일이 결과값을 저장하는 방법은 재료를 다듬는 단계별로 서로 다른 주방에서 작업을 하는 것과 같으며, 두 번째 방법처럼 괄호를 통해 함수를 계속 엮을 경우, 주방이 지나치게 지저분해지는 단점이 있습니다. 그러나 파이프 오퍼레이터인 **%>%**를 사용할 경우, 함수 간의 관계를 매우 직관적으로 이해할 수 있습니다. 이를 정리하면 아래 표와 같습니다.

**[표] 파이프 오퍼레이터의 표현과 내용 비교**

|  |  |
| --- | --- |
| **내용** | **표현** |
| F(x) | x %>% F |
| G(F(x)) | x %>% F %>% G |

다음으로 파이프 오퍼레이터의 간단한 예제[[9]](#endnote-9)를 통해 사용법을 살펴보도록 하겠습니다. 먼저 다음과 같은 10개의 숫자가 있다고 가정합니다.

|  |
| --- |
| x = c(0.3078, 0.2577, 0.5523, 0.0564, 0.4685,  0.4838, 0.8124, 0.3703, 0.5466, 0.1703) |

우리가 원하는 과정은 **1) 각 값들의 로그값을 구할 것, 2) 로그값들의 계차를 구할 것, 3) 구해진 계차의 지수값을 구할 것, 4) 소수 둘째 자리까지 반올림할 것** 입니다. 즉 log(), diff(), exp(), round()에 대한 값을 순차적으로 구하고자 합니다.

|  |
| --- |
| x1 = log(x)  x2 = diff(x1)  x3 = exp(x2)  round(x3, 2)  [1] 0.84 2.14 0.10 8.31 1.03 1.68 0.46 1.48 0.31 |

첫 번째 방법은, 단계 별 함수의 결과값을 변수에 저장하고, 저장된 변수를 다시 불러와 함수에 넣고 계산하는 방법입니다. 전반적인 계산 과정을 확인하기에는 좋지만, 매번 변수에 저장하고 불러오고 하는 과정은 매우 비효율 적이며, 코드 또한 불필요하게 길어집니다.

|  |
| --- |
| round(exp(diff(log(x))), 2)  [1] 0.84 2.14 0.10 8.31 1.03 1.68 0.46 1.48 0.31 |

두 번째는 괄호를 통해 감싸는 방법입니다. 앞선 방법에 비해 코드는 짧아졌지만, 계산 과정을 알아보기에는 매우 불편한 방법으로 코드가 짜여 있습니다.

|  |
| --- |
| library(magrittr)  x %>% log() %>% diff() %>% exp() %>% round(., 2)  [1] 0.84 2.14 0.10 8.31 1.03 1.68 0.46 1.48 0.31 |

마지막으로 파이프 오퍼레이터를 사용하는 방법입니다. 코드도 짧으며, 계산 과정을 한눈에 파악하기도 좋습니다. 맨 왼쪽에는 원하는 변수를 입력하며, %>% 뒤에는 차례대로 계산하고자 하는 함수를 입력합니다. 변수의 입력값을 ()로 비워둘 경우, 오퍼레이터의 좌측에 있는 값이 자동으로 계산됩니다. 반면 round와 같이 입력값이 2개 이상 필요할 경우, 콤마(.)는 자동으로 오퍼레이터의 좌측 값으로 인식됩니다.

파이프 오퍼레이터는 크롤링 뿐만이 아닌, 모든 코드에 사용할 수 있습니다. 이를 통해 훨씬 깔끔하면서도 데이터 처리과정을 직관적으로 이해할 수 있습니다.

**[2] HTML 이해하기**

논과 밭에서 감자를 직접 캔 후, 흙을 제거하고 껍질을 벗겨야 비로소 우리가 먹을 수 있는 식재료의 형태가 됩니다. 크롤링을 통해 데이터를 얻을 경우 웹페이지에 존재하는 데이터를 일괄적으로 받으므로 투자에 불필요한 데이터도 같이 받아질 때가 있습니다. 따라서 크롤링 작업 이후에는 필요한 데이터만을 찾아낸 후 분석하기 편한 형태로 처리하는 ‘데이터 클랜징’ 작업이 필수입니다.

우리가 사용하는 웹페이지는 대부분 하이퍼텍스트 마크업언어Hyper Text Markup Language, 즉 HTML을 이용하여 만들어집니다. 따라서 해당 언어의 구조를 간략하게 이해한다면 크롤링 이후 원하는 데이터만을 찾는데 매우 도움이 됩니다. 아래 표는 HTML 표현 및 웹페이지에 표시되는 내용[[10]](#endnote-10)의 비교입니다.

**[표] HTML과 웹페이지 표시 내용의 비교**

|  |  |
| --- | --- |
| **HTML** | <html>  <head>  <title>Title of Page</title>  </head>  <body>  <h2>This is page heading</h2>  <p>This is first <strong>paragraph text</strong></p>  </body>  </html> |
| **웹페이지**  **표시 내용** |  |

HTML 언어에서 <>로 표시된 부분은 태그입니다. 먼저 <head>는 제목 부분을, <title>은 제목 내용을 나타내는 태그이며, <title>뒤에 적힌 **Title of Page** 글자가 웹페이지의 상단에 그대로 위치함이 확인됩니다. 태그를 마칠 때는 태그명 앞에 슬래쉬(</태그명>)를 입력하여 줍니다.

<body>는 웹페이지에 나타나는 부분인 본문을 의미합니다. 먼저 <h2>에서 h는 제목의 역할을 하며, 2는 우선순위 및 크기를 나타냅니다. <p>는 새로운 본문이 시작되는 곳을 의미합니다. <strong>은 굵은 글씨를 의미하며, 해당 태그로 둘러쌓인 ‘paragraph text’ 라는 단어가 웹페이지에서는 굵게 표현됨이 확인됩니다.

태그의 특징 중 하나는 속성attribute을 추가할 수 있으며, 이를 통해 해당 내용을 어떻게 보여줄지 설정할 수 있습니다. 아래 표는 <p> 태그에 style 속성을 추가한 표현 및 웹페이지 표시내용의 비교입니다.

**[표] style 속성과 웹페이지 표시 내용의 비교**

|  |  |
| --- | --- |
| **HTML** | <p style="color: red; font-size: 11;"> 1. Red 11 </p>  <p style="color: blue; font-size: 13;"> 2. Blue 13 </p>  <p style="color:yellow; font-size: 15;">  <span style="background-color: black;"> 3. Yellow & Black 15 </span></p> |
| **웹페이지**  **표시 내용** |  |

style 속성은 글자의 크기, 모양 등을 표현하는데 사용됩니다. color는 글자의 색을, font-size는 글자의 크기를 나타내며, 각각 설정값에 따라 글자의 색 및 크기가 변경됨이 확인됩니다. 또한, background-color는 글자의 배경색을 나타내며, <span> 태그를 통해 적용되었음이 확인됩니다.

크롤링을 통한 데이터 수집에 있어 실무적으로 가장 알아야 하는 속성은 id와 class 입니다. 두 속성은 일종의 책갈피 역할을 합니다. 아래 표는 div 태그에 id 속성과 class 속성을 추가한 표현 및 웹페이지 표시내용의 비교입니다.

**[표] id와 class 속성의 웹페이지 표시 내용의 비교**

|  |  |
| --- | --- |
| **HTML** | <div id="div1">ID uses # for CSS</div>  <div class="div2">Class uses . for CSS</div> |
| **웹페이지**  **표시 내용** |  |

첫 번째 줄에는 **“div1”**의 id 속성을 지정해 주었으며, 두 번째 줄에는 **“div2”**의 class 속성을 지정해 주었습니다. 웹페이지에 표시되는 내용을 봤을 때는 아무런 역할도 하지 않는 것처럼 보입니다. 둘 간의 차이는 개발자도구 화면[[11]](#endnote-11)을 통해 확인할 수 있습니다.

**[표] id와 class 속성의 개발자도구 화면 비교**

|  |  |
| --- | --- |
| ID |  |
| Class |  |

먼저 우측의 개발자도구 화면에서 Elements 탭을 선택합니다. 첫번째 줄에 해당하는 id 속성의“div1” 부분으로 마우스를 움직이면, 좌측의 웹페이지 화면에 해당 태그의 정보인 div#div1이 표시됩니다. 다음으로 class 속성의 “div2” 부분으로 마우스를 움직이면 이번에는 태그의 정보가 div.div2로 표시됩니다. 즉 id 속성은 **‘태그#속성명’**, class 속성은 **‘태그.속성명‘**으로 인식됩니다.

이 외에도 id와 class 속성의 차이는 다음과 같습니다. id 속성은 문서안에서 한번만 사용 가능 하며, 보통 레이어에 사용됩니다. 반면에 class 속성은 문서안에서 중복으로 사용 가능 하며 일관성 있는 스타일을 적용시 사용됩니다.

­

각 클래스 별로 일관된 스타일을 적용하는 방법의 예시입니다.

**[표]** **class 속성별 스타일 적용**

|  |  |
| --- | --- |
| **HTML** | <style type="text/css">  .low-level { color: blue; }  .high-level { color: red; font-weight: bold; }  </style>  <p class="low-level"> Low: machine language, assembly </p>  <p class="high-level"> High: c, c++, java </p>  <p class="high-level"> We use R!! </p> |
| **웹페이지**  **표시 내용** |  |

먼저 style 태그를 통해 각 클래스 별 스타일을 정의해줍니다. 이는 프로그래밍에서 함수와 비슷한 역할을 합니다. .low-level, 즉 속성명이 low-level인 클래스는 푸른색을 적용하며, .high-level, 즉 속성명이 high-level인 클래스는 붉은색 및 굵은 표시를 적용합니다.

그 후 첫번째 단락인 **Low: machine language, assembly**은 low-level 클래스가, 두번째와 세번째 단락인 **High: c, c++, java** 및 **We use R!!**은 high-level 클래스로 설정되었습니다. 웹페이지의 표시 내용을 살펴보면, style 태그에서 적용한 것처럼 low-level 클래스에는 푸른 글씨가, high-level 클래스에는 붉은 글씨 및 굵은 표시가 적용되었음이 확인됩니다.

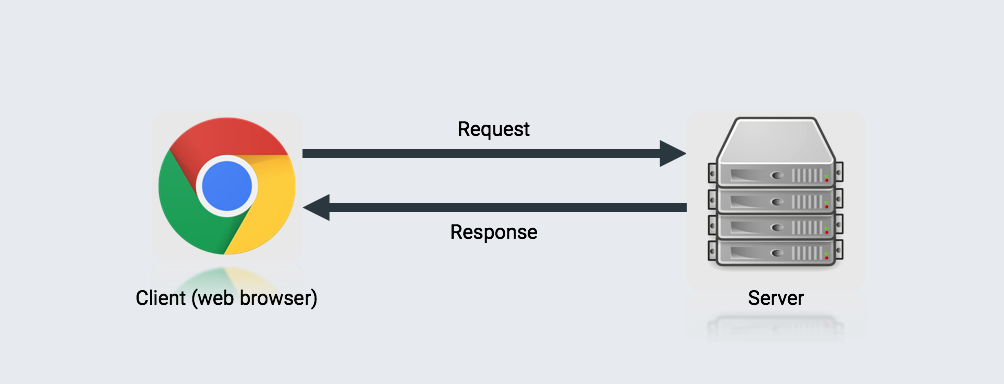
크롤링에 필요한 HTML 관련 지식은 해당 장에서 언급한 정도로도 충분하다고 생각됩니다. 추가적인 정보가 필요하거나 내용이 궁금하신 분들은 아래 사이트를 참고하기 바랍니다.

* 웨버 스터디: <http://webberstudy.com/>
* w3schools: <https://www.w3schools.in/html-tutorial/>

**[3] GET과 POST 방식 이해하기**

우리가 인터넷에 접속하여 서버에 파일을 요청하면, 서버는 이에 해당하는 파일을 우리에게 보내줍니다. 이러한 과정을 사람이 수행하기 편하고 시각적으로 보기 편하도록 만들어 진 것이 크롬과 같은 웹브라우저 이며, 서버의 주소를 기억하기 쉽게하기 위해 만든 것이 인터넷 주소 입니다. 우리가 서버에 데이터를 요청하는 형태는 다양하지만 크롤링에서는 주로 GET과 POST 방식을 사용합니다.

**[그림] 클라이언트와 서버 간의 요청/응답 과정**

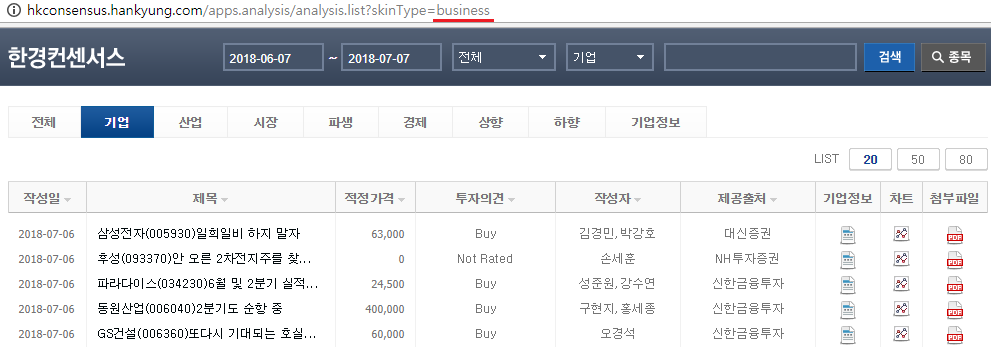


**GET 방식**

GET 방식은 인터넷 주소를 기준으로, 이에 해당하는 데이터나 파일을 요청하는 것입니다. 주로 클라이언트가 요청하는 쿼리를 앰퍼샌드(&) 혹은 물음표(?) 형식으로 결합하여 서버에 전달됩니다.

한경컨센서스(<http://hkconsensus.hankyung.com/>)에 접속한 후 전체 REPORT를 선택하면, 홈페이지의 주소 뒤에 **/apps.analysis/analysis.list**가 붙으며 이에 해당하는 페이지의 내용을 보여줍니다. 상단의 탭에서 기업을 선택하면, 주소의 끝부분에 **?skinType=business**가 추가되며 이에 해당하는 페이지의 내용을 보여줍니다. 즉, 해당 페이지는 GET 방식을 사용하고 있으며 입력종류는 skinType, 이에 해당하는 기업 탭의 입력값은 business 임을 알 수 있습니다.

**[그림] 한경 컨센서스 기업 REPORT 페이지**

****

이번에는 파생 탭을 선택하여 봅니다. 역시나 홈페이지 주소가 변경되며 해당 주소에 맞는 내용이 나타납니다. 주소의 끝부분이 **?skinType=derivative** 로 변경되며, 입력 값이 변경됨에 따라 페이지의 내용이 이에 맞게 변하는 모습이 확인됩니다. 여러 다른 탭들을 눌러보면 **?skinType=** 뒷부분의 입력값이 변함에 따라 이에 해당하는 페이지로 내용이 변경됨이 확인됩니다.

다시 기업 탭을 선택한 후, 다음 페이지를 확인하기 위해 하단의 2를 클릭합니다. 기존 주소인 **?skinType=business** 뒤에 추가로 sdate와 edate, 그리고 now\_page 쿼리가 추가됨이 확인됩니다. sdate에 검색 기간의 시작시점, edate에 검색 기간의 종료시점, now\_page에 원하는 페이지를 수기로 입력해도 이에 해당하는 페이지의 데이터를 보여줍니다. 이처럼 GET 방식으로 데이터를 요청할 경우, 웹 페이지 주소를 수정하여 원하는 종류의 데이터를 받아올 수 있습니다.

**[그림] 쿼리 추가로 인한 url의 변경**



**POST 방식**

POST 방식은 사용자가 필요한 값을 추가해서 요청하는 방법입니다. GET 방식과의 차이는 클라이언트가 요청하는 쿼리를 body에 넣어서 전송하므로, 요청 내역을 직접적으로 볼 수 없습니다.

한국거래소 상장공시시스템(<http://kind.krx.co.kr/>)에 접속하여 전체메뉴보기를 누른 후, 상장법인상세정보 중 상장종목현황을 선택합니다. 웹 페이지 주소가 바뀌며, 상장종목현황이 보여집니다.

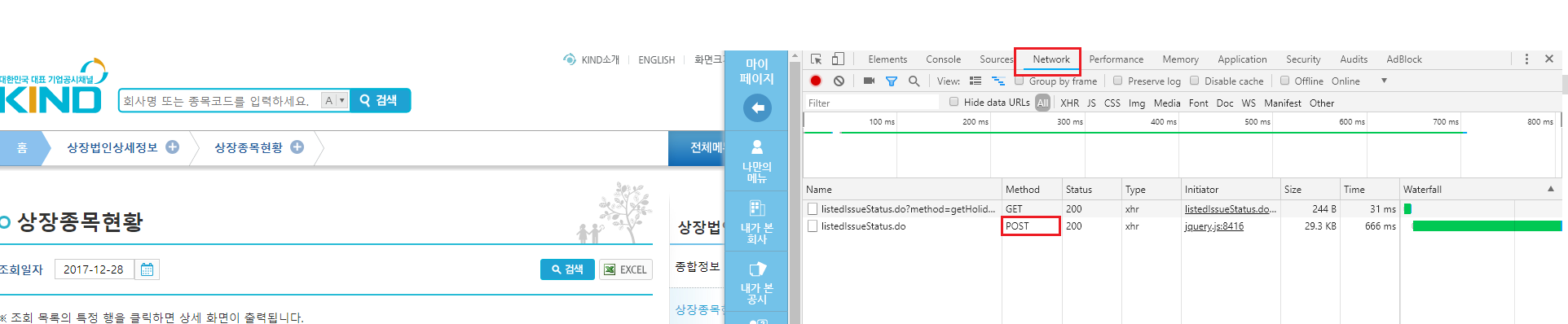
**[그림] 상장공시시스템의 상장종목현황 메뉴**



이번엔 조회일자를 2017-12-28로 선택한 후, 검색을 눌러보도록 합니다. 페이지의 내용은 선택일 기준으로 변경되었지만, 주소는 변경되지 않고 그대로 남아있습니다. GET 방식에서는 선택항목에 따라 웹 페이지 주소가 변경되었지만, POST 방식을 사용하여 서버에 데이터를 요청하는 해당 사이트는 그렇지 않음이 확인됩니다.

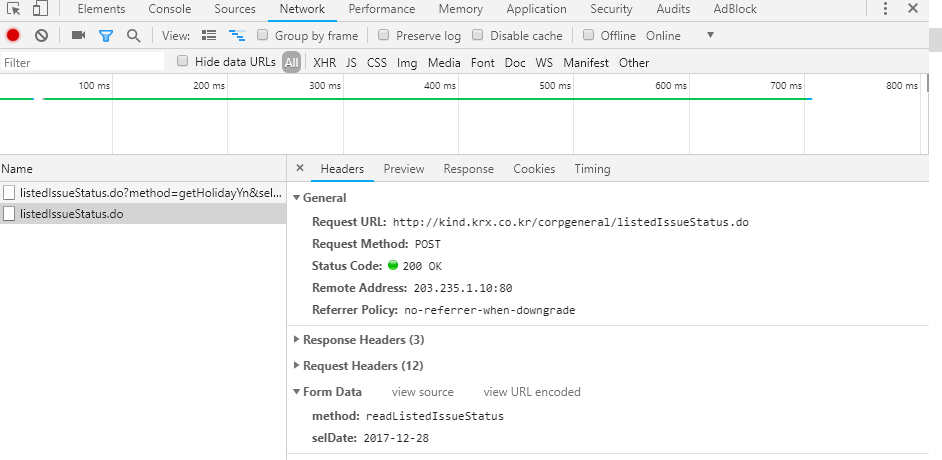
POST 방식의 데이터 요청과정을 살펴보기 위해서는 개발자도구를 이용하면 되며, 개발자도구를 연 상태에서 다시 한번 ‘검색’을 클릭해 봅니다. 개발자도구의 Network 탭을 클릭하면, ‘검색’을 클릭함과 함게 브라우저와 서버간의 통신 과정을 살펴볼 수 있습니다. 이 중 listedIssueStatus.do 라는 항목이 POST 형태임을 알 수 있습니다.

**[그림] 크롬 개발자도구의 Network 화면**



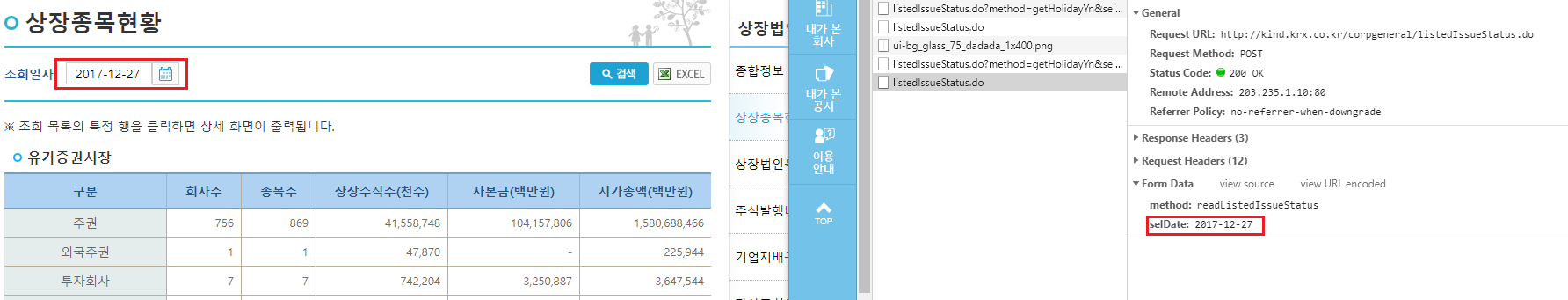
해당 메뉴를 클릭하면 통신 과정을 좀 더 자세히 알 수 있습니다. 가장 하단의 Form Data에 서버에 데이터를 요청하는 내역이 있습니다. method에는 readListIssueStatus, selDate에는 2017-12-28라는 값이 있습니다.

**[그림] POST 방식의 서버 요청 내역**



이번에는 조회일자를 하루 전인 2017-12-27로 선택하여 검색을 누릅니다. 새로운 데이터 요청이 있음에 따라 개발자도구 화면에 이에 해당하는 내용들이 추가되며, 가장 하단에 이번에도listedIssueStatus.do 메뉴가 생생됩니다. 해당 메뉴를 선택해보면, method는 기존과 동일하지만 selDate가 2017-12-27로 변경되었습니다. 즉 POST 방식은 요청하는 데이터에 대한 쿼리가 body를 통해 전송되며, 이를 웹브라우저를 통해 확인할 수는 없습니다.

**[그림] 요청내역 변경에 따른 Form Data의 변경**



**[4] 크롤링 예제**

크롤링의 일반적인 과정은 httr 패키지의 GET() 혹은 POST() 함수를 이용하여 데이터를 다운로드 받은 후, rvest 패키지의 함수들을 이용하여 원하는 데이터를 찾아내는 과정으로 이루어집니다. 해당 장에서는 GET 방식의 예제로 인기도서 순위 중 책 제목만을 추출하는 과정을, POST 방식의 예제로 상장공시시스템의 상장종목현황 테이블을 가져오는 과정을 살펴보도록 하겠습니다.

**GET 방식을 이용하여 인기도서 제목 추출하기**

먼저 온라인서점인 예스24(<http://www.yes24.com/>)에 접속하여 **‘R 프로그래밍’**을 검색합니다. 검색 후 웹주소가 아래와 같이 변경되며, 페이지가 변경됩니다.

|  |
| --- |
| <http://www.yes24.com/searchcorner/Search?keywordAd=&keyword=&domain=ALL&qdomain=%C0%FC%C3%BC&Wcode=001_005&query=R+%C7%C1%B7%CE%B1%D7%B7%A1%B9%D6> |

주소에 ?와 &가 결합되어 있는 점으로 보아 GET 방식으로 데이터를 요청하는 것으로 짐작 되며, 마지막 query= 항목 뒷부분이 우리가 검색한 검색어임을 유추할 수 있습니다.

GET 방식의 페이지 정보를 크롤링하는 방법은 다음과 같습니다.

|  |
| --- |
| library(rvest)  library(httr)  url = "http://www.yes24.com/searchcorner/Search?keywordAd=&keyword=&domain=ALL&qdomain=%C0%FC%EC%B2%3F&query=R+%C7%C1%B7%CE%B1%D7%B7%A1%B9%D6"  data = GET(url) |

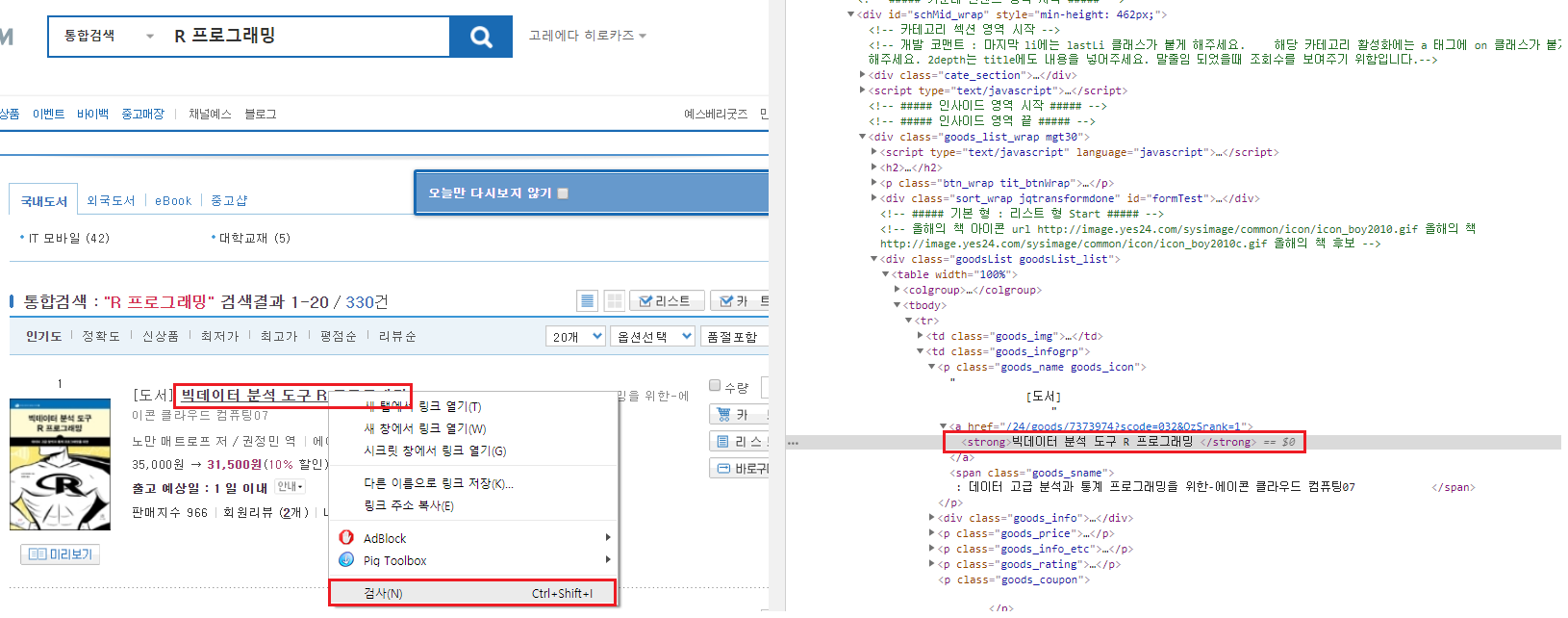
먼저 크롤링의 필수 패키지인 rvest와 httr 패키지를 열어줍니다. 웹페이지 주소를 url 변수에 지정해준 후, httr의 GET() 함수를 이용하면 GET 방식으로 해당 페이지의 데이터를 다운로드 받게 됩니다. 데이터가 저장된 data 변수를 확인해보면 다음과 같습니다.

|  |
| --- |
| > data  Response [http://www.yes24.com/searchcorner/Search?keywordAd=&keyword=&domain=ALL&qdomain=%C0%FC%EC%B2%3F&query=R+%C7%C1%B7%CE%B1%D7%B7%A1%B9%D6]  Date: 2018-07-20 08:04  Status: 200  Content-Type: text/html; charset=ks\_c\_5601-1987  Size: 343 kB  <!DOCTYPE html>  <html lang="ko">  <head>  <meta http-equiv="X-UA-Compatible" content="IE=Edge" />  <meta http-equiv="Content-Type" content="text/html;charset=euc-kr" />  <meta name="viewport" content="width=1170" />  ... |
|  |

Response는 데이터 요청에 내역이며, <!DOCTYPE html> 하단의 내용은 해당 페이지의 HTML 정보입니다. Status가 200 값을 나타낼 경우 데이터 요청에 대한 응답이 성공적으로 진행되었음을 의미합니다.

받아진 데이터 중 책 제목만을 추출하기 위해서는 해당 항목의 node가 무엇인지 알 필요가 있으며, 개발자도구를 이용하면 쉽게 찾을 수 있습니다. 먼저 웹페이지에서 최종 목적 데이터인 책 제목에 마우스를 올린 후 우클릭을 하여 검사 항목을 누르면, 개발자도구 화면에서 해당 부분에 대한 HTML 부분이 보여집니다.

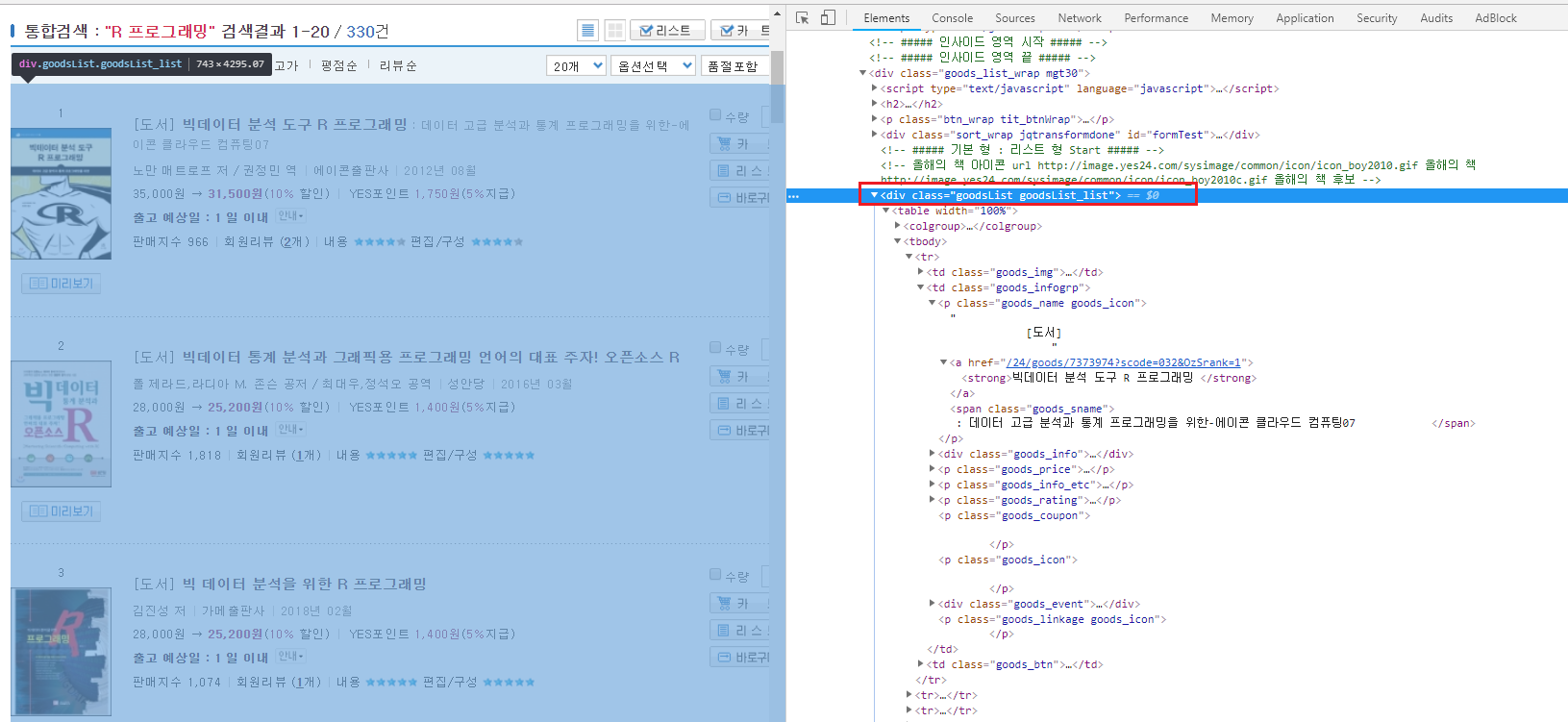
**[그림] 인기도서 순위 중 책제목의 HTML 항목 찾기**



검사 항목을 누를 경우 개발자도구 화면이 열리며, 책제목이 있는 부분에 회색 음영처리가 되며, 책제목은 <strong> 태그 가운데 있음이 확인됩니다. HTML 부분을 마우스로 움직여보면, 해당 부분이 해당하는 내용이 웹페이지에 음영처리가 됩니다.

HTML 부분에서 마우스를 위로 올려보면, 우리가 찾는 인기순위 부분은 <div class="goodsList goodsList\_list"> 태그에 위치해 있음이 확인됩니다. 개발자도구 화면에서 ‘goodsList goodsList\_list’ 를 검색하면 통합검색, 중고샵, 리뷰, 기사 및 인터뷰 총 4개 테이블의 클래스가 해당 이름으로 설정되어 있음을 확인할 수 있으며, 그 중 우리가 원하는 인기도서는 첫번째에 위치하고 있습니다.

**[그림] 원하는 데이터가 포함된 class 찾기**



원하는 데이터가 포함된 class를 찾았다면, 마우스를 아래로 내려가며 데이터의 구조를 파악해 나가면 됩니다. 각 태그들이 웹페이지에서 해당하는 내용은 아래 표와 같습니다.

**[표] HTML 태그와 웹페이지 내용의 비교**

|  |  |
| --- | --- |
| **HTML 태그** | **웹페이지 내용** |
| <div class="goodsList goodsList\_list"> | 전체 표 |
| <td class="goods\_infogrp"> | 책 관련 정보 |
| <p class="goods\_name goods\_icon"> | 책 제목 및 부제목 |
| <a href="/24/goods/7373974?scode=032&amp;OzSrank=1"> | 책 제목 및 링크 |
| <strong> | 책 제목 |

위에서 찾아낸 HTML 구조를 이용하여, 우리가 원하는 책제목을 찾아나가는 코드를 아래와 같습니다.

|  |
| --- |
| x = read\_html(data)  html\_node(x, ".goodsList.goodsList\_list") %>%  html\_nodes(".goods\_infogrp") %>%  html\_nodes(".goods\_name.goods\_icon") %>%  html\_nodes("a") %>%  html\_text() |

다운로드 받아진 페이지 정보를 바탕으로, 인기도서 순위 중 책 제목만을 추출하는 과정을 진행하도록 합니다. 먼저 rvest 패키지의 read\_html() 함수를 이용하여 HTML 정보만을 추출합니다. 그 후 html\_node() 혹은 html\_nodes() 함수를 이용하여 원하는 정보를 찾아 내려갑니다.

html\_nodes() 함수는 **html\_nodes(x, css, xpath)** 형태로 이루어지며, x에 HTML 값을 입력한 후 찾고자 하는 node인 css 혹은 xpath를 입력하면 이에 해당하는 값들을 찾아줍니다. html\_nodes() 함수의 경우 node에 해당하는 모든 데이터를 보여주며, html\_node() 함수의 경우 이 중 첫번째 데이터만을 보여줍니다.

원하는 css가 태그인 경우 “태그명”을, id를 찾는 경우 “#id”를, class를 찾는 경우 “.class”를 입력하면 됩니다. 또한 class 이름에 공백이 있을 시, 공백 부분을 콤마(.)로 대체하여 주면 됩니다.

개발자도구 화면에서 파악한 HTML 구조와 html\_nodes() 함수를 이용하여 원하는 데이터를 찾아나면 됩니다. 먼저 해당 페이지에는 총 4개의 goodsList goodsList\_list 클래스가 존재하므로, html\_node() 함수를 이용하여 첫번째 클래스만을 찾아냅니다. 클래스를 찾는 경우이므로 이름 앞에 콤마(.)를 붙여주며, 공백 부분도 콤마로 대체하여 줍니다. 그 후, 테이블 내에 존재하는 모든 클래스를 찾기 위해 html\_nodes() 함수를 이용하며, 이들의 연결은 파이프 오퍼레이터(%>%)를 이용합니다. html\_text() 함수는 HTML 언어에서 태그 및 속성을 제외하고 텍스트 부분만을 표시해주며, 우리가 원하는 책제목만을 추출하여 줍니다.

위의 코드를 실행하면 다음과 같이 ‘R 프로그래밍’으로 검색한 페이지의 인기순위 상위 20개 책제목이 추출됩니다.

|  |
| --- |
| [1] "빅데이터 분석 도구 R 프로그래밍 "  [2] "빅데이터 통계 분석과 그래픽용 프로그래밍 언어의 대표 주자! 오픈소스 R"  [3] "빅 데이터 분석을 위한 R 프로그래밍"  [4] "R 병렬 프로그래밍"  [5] "손에 잡히는 R 프로그래밍"  [6] "효율적인 R 프로그래밍"  [7] "손에 잡히는 R 프로그래밍"  [8] "R 프로그래밍 기초 & 활용"  [9] "R에서 객체지향 프로그래밍 사용하기"  [10] "R 프로그래밍"  [11] "R 프로그래밍 레퍼런스 북"  [12] "R Shiny 프로그래밍 가이드 "  [13] "컴퓨터 비전공자를 위한 R언어를 활용한 기초컴퓨터 프로그래밍"  [14] "R입문 및 기초 프로그래밍"  [15] "R 프로그래밍"  [16] "R 통계 프로그래밍 입문 "  [17] "R을 활용한 통계 프로그래밍 입문"  [18] "빅데이터 분석을 위한 R 프로그래밍"  [19] "R 병렬 프로그래밍"  [20] "알찬 R프로그래밍" |

**POST 방식을 이용하여 상장종목현황 추출하기**

이전의 예제였던 상장공시시스템의 상장종목현황 페이지를 접속합니다. POST 방식의 경우 요청하는 데이터에 대한 쿼리가 body의 형태를 통해 전송되며, 개발자도구를 통해 해당 쿼리에 대한 내용을 직접 확인할 수 있습니다.

개발자도구를 열고 조회일자를 2017-12-28로 선택하여 검색을 누른 후, listedIssueStatus.do 항목을 살펴보면 Form Data를 통해 서버에 데이터를 요청하는 내역을 확인할 수 있음을 이미 배웠습니다. method에는 readListIssueStatus, selDate에는 2017-12-28라는 값이 있으며, 이를 이용하여 해당 데이터를 다운로드 받는 방법은 다음과 같습니다.

|  |
| --- |
| url = "http://kind.krx.co.kr/corpgeneral/listedIssueStatus.do?method=loadInitPage"  data = POST(url,  query=list(  method = 'readListedIssueStatus',  selDate = '2017-12-28')  ) |

웹페이지 주소를 url 변수에 지정해준 후, httr의 POST() 함수를 이용하면 POST 방식으로 해당 내역의 데이터를 요청하게 됩니다. 이 중 query 부분에는 Form Data에 있는 쿼리 내용들을 list의 형태로 입력해주면 됩니다. 데이터가 저장된 data 변수를 확인해보면 다음과 같습니다.

|  |
| --- |
| > data  Response [http://kind.krx.co.kr/corpgeneral/listedIssueStatus.do?method=loadInitPage]  Date: 2018-07-20 16:16  Status: 200  Content-Type: text/html; charset=UTF-8  Size: 63.7 kB  <!DOCTYPE html>  <html lang="ko">  <head>  <meta http-equiv="X-UA-Compatible" content="IE=edge" />  <meta http-equiv="content-type" content="text/html; charset=utf-8" />  <meta http-equiv="content-language" content="kr" />  ... |

받아진 데이터 중 유가증권시장, 코스닥시장, 코넥스시장에 대한 표를 추출하고자 합니다. 앞서 살펴본 html\_nodes 함수를 이용할 수도 있지만, 표 형식의 데이터를 추출할 때는 훨씬 효율적인 방법이 있습니다.

|  |
| --- |
| data\_table = read\_html(data) %>%  html\_table() |

먼저 read\_html() 함수를 이용하여 HTML 정보를 추출한 후, rvest 패키지의 html\_table() 함수를 이용합니다. 해당 함수는 HTML에 존재하는 표 형태의 데이터를 찾아내어 데이터프레임 형식으로 읽어옵니다. 위 코드를 실행하면 data\_table에 3개의 데이터프레임이 리스트 형태로 저장됩니다. 해당 변수를 확인하면 다음과 같습니다.

|  |
| --- |
| > data\_table  [[1]]  구분 회사수 종목수 상장주식수(천주) 자본금(백만원) 시가총액(백만원)  1 주권 756 869 41,580,793 104,168,829 1,601,670,789  2 외국주권 1 1 47,870 - 226,423  3 투자회사 7 7 742,204 3,250,887 3,669,787  4 부동산투자회사 4 4 64,487 149,054 120,874  5 선박투자회사 6 6 62,472 312,358 133,040  6 소계 774 887 42,497,825 107,881,138 1,605,820,912  7 신주인수권증권 12 13 279,208 - 259,253  8 신주인수권증서 1 1 4,557 - 962  9 ELW 5 1,930 26,762,414 - 6,211,279  10 ETN - 184 449,100 - 5,199,361  11 ETF 325 325 1,949,942 - 35,574,913  12 수익증권 12 27 1,440,409 1,551,952 1,463,850  [[2]]  구분 회사수 종목수 상장주식수(천주) 자본금(백만원) 시가총액(백만원)  1 주권 1,198 1,201 32,129,584 14,766,746 276,741,261  2 기업인수목적회사 50 50 274,845 27,484 548,523  3 외국주권 15 15 1,221,589 4,914 1,898,334  4 소계 1,263 1,266 33,626,018 14,799,145 279,188,118  5 주식예탁증권(DR) 4 4 132,038 - 3,551,933  6 신주인수권증권 12 12 89,063 - 38,494  [[3]]  구분 회사수 종목수 상장주식수(천주) 자본금(백만원) 시가총액(백만원)  1 주권 154 154 670,598 314,976 4,908,055  2 소계 154 154 670,598 314,976 4,908,055 |

리스트의 첫번째 항목은 유가증권시장 테이블이, 두번째 항목은 코스닥시장 테이블이, 세번째 항목은 코넥스시장 항목이 저장되어 있음이 확인됩니다. 이처럼 표 형태의 데이터를 추출할 때는 html\_nodes() 함수를 이용한 후 클랜징 과정을 거치는 것 보다, html\_table() 함수를 이용하는 것이 훨씬 효율적입니다.

**4장. 금융 데이터 수집하기**

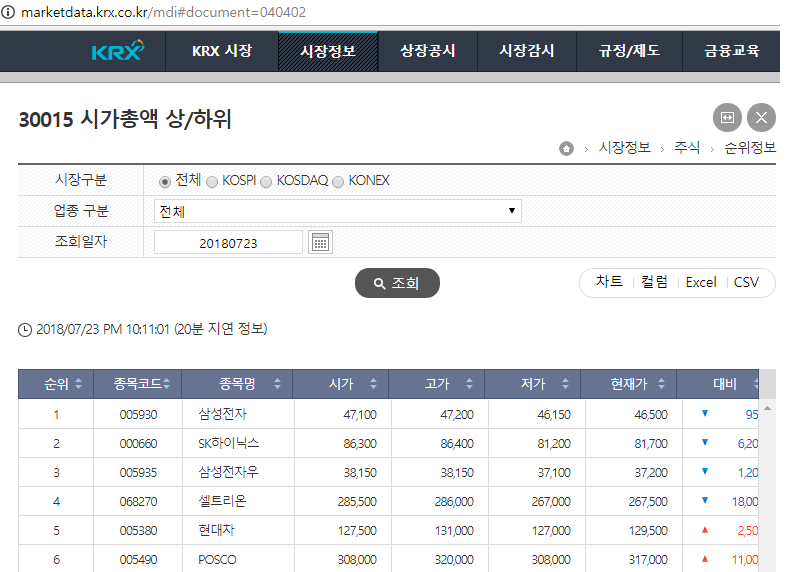
API와 크롤링을 이용한다면 비용을 지불하지 않고 얼마든지 금융 데이터를 수집할 수 있습니다. 해당 책에서는 기본적인 퀀트 투자에 필요한 주가, 재무재표, 가치지표의 수집에 대해 배우겠지만 이 외에도 애널리스트 추정치, 공시, 경제지표 등 다양한 정보를 수집할 수 있습니다. 또한 국내 데이터뿐만 아니라 해외 모든 데이터도 얼마든지 수집할 수 있습니다.

**[1] 주식티커 크롤링**

각 종목들의 주가 및 재무정보를 수집하기 위해서는 먼저 주식의 티커를 다운로드 받아야 합니다. 국내 상장 전종목의 티커를 구하는 방법은 여러가지가 있지만, 한국거래소에서 제공하는 데이터를 이용하는 것이 가장 정확합니다.

먼저 한국거래소(<http://krx.co.kr/>) 홈페이지 접속한 후 **‘시장정보 → 주식 → 순위정보 → 시가총액 상/하위’**를 선택하면 국내 상장된 전종목의 티커 및 각종 정보가 표로 나타납니다. 화면 단축번호는 30015이며, 해당 화면의 주소는 <http://marketdata.krx.co.kr/mdi#document=040402> 입니다.

**[그림] 한국거래소 전종목 데이터**



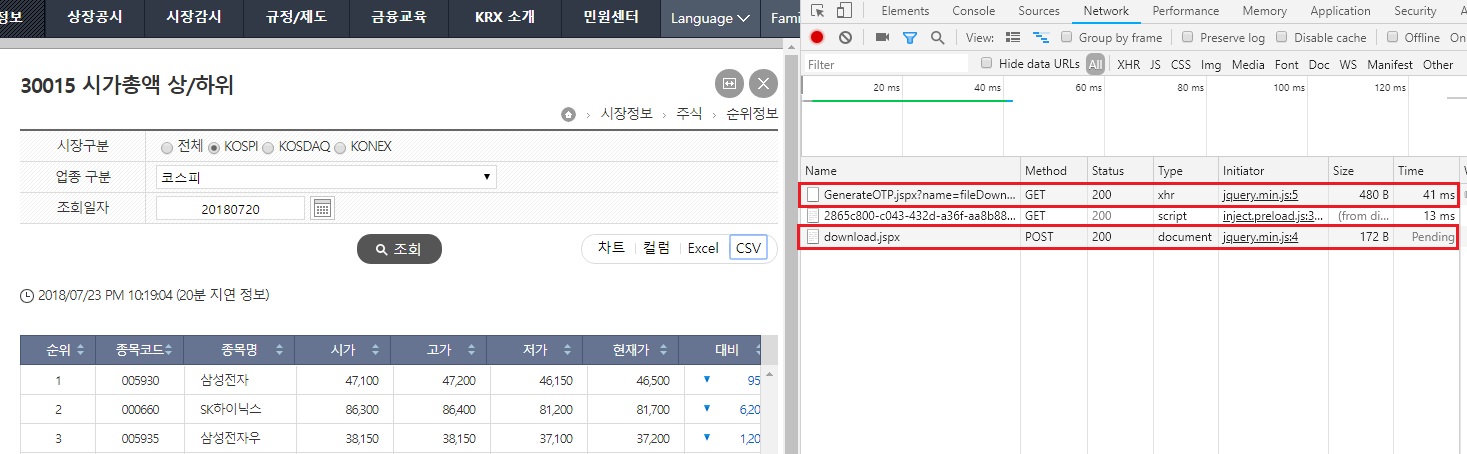
Excel 혹은 CSV 버튼을 눌러 파일을 다운로드 받은 후, 이를 다시 R로 읽는 과정을 통해 해당 데이터를 불러올 수도 있습니다. 그러나 매번 이러한 작업을 하는 것은 매우 비효율적이므로, 크롤링을 통해 데이터를 읽어오는 방법을 살펴보도록 하겠습니다.

해당 페이지에 있는 시장구분, 업종 구분, 조회일자를 바꾸어가며 조회를 눌러도 웹페이지 주소가 변경되지 않는 것으로 보아, 해당 페이지는 POST 방식으로 데이터를 요청함을 알 수 있습니다.

개발자도구 화면을 연 상태에서 KOSPI 항목을, 조회일자를 2018-07-20으로 선택한 후 CSV 버튼을 누르면, 데이터 요청 및 csv 파일 다운로드 과정을 확인할 수 있습니다. 한국거래소에서 제공하는 데이터는 다음과 같은 절차를 거쳐 파일이 다운로드 됩니다.

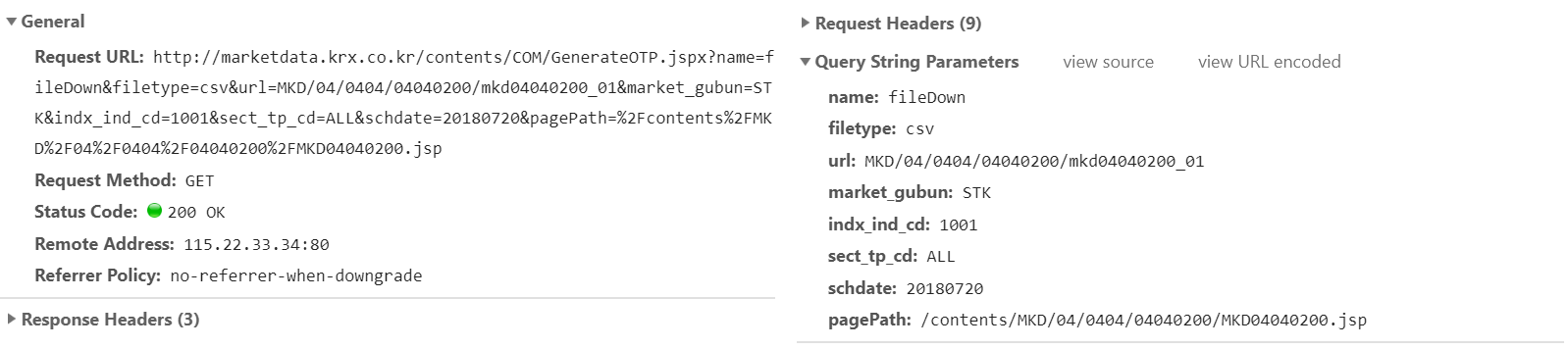
|  |
| --- |
| 1. <http://marketdata.krx.co.kr/contents/COM/GenerateOTP.jspx> 주소에 query를 전송하여 OTP 코드 생성 2. 위에서 생성된 OTP를 <http://file.krx.co.kr/download.jspx> 주소에 query로 사용하여 데이터 다운로드 |

**[그림] csv 파일 데이터 요청 과정**

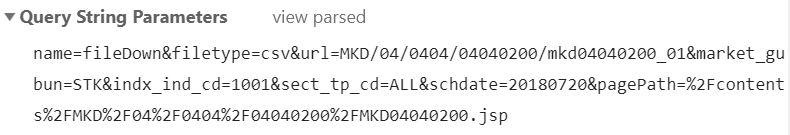


먼저, ①번 과정인 GenerateOTP.jspx?name=fileDown… 항목을 클릭하면 다음과 같은 내역을 볼 수 있습니다. Query String Parameters에 각 쿼리들의 항목이 입력되어 있으며 market\_gubun의 STK는 KOSPI임을, schdate는 선택일자를 의미합니다. 해당 쿼리들이 앰퍼샌드(&) 혹은 물음표(?)의 형태로 결합하여, Request URL인 <http://marketdata.krx.co.kr/contents/COM/GenerateOTP.jspx>에 결합됨이 확인됩니다. Query String Parameters 우측의 view source를 클릭하면, 결합된 내역을 확인할 수도 있습니다.

**[그림] OTP 코드 생성**

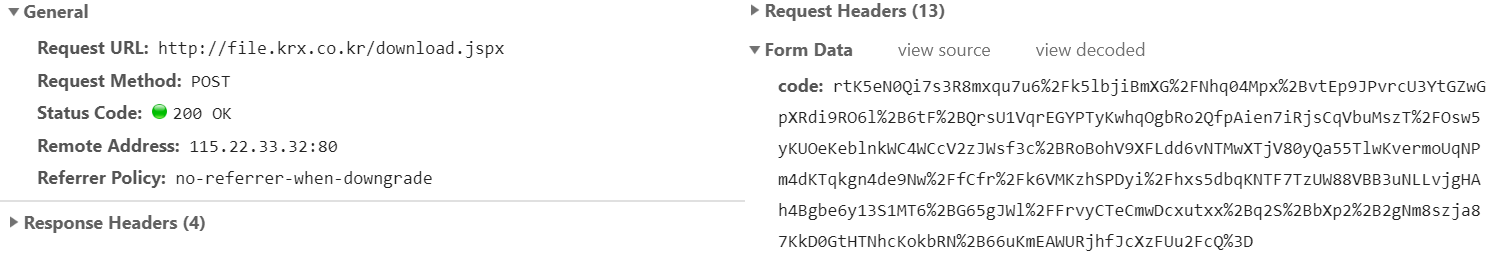


**[그림] url 형태의 쿼리 결합**



다음으로 ②번 과정인 download.jspx 항목을 클릭하면 다음과 같은 내역을 볼 수 있습니다. Form Data 항목의 code는 위 과정에서 발급된 OTP 코드이며, Request URL에 해당 내역의 데이터를 POST 방식으로 요청합니다.

**[그림] OTP 코드를 이용한 데이터 요청 과정**



OTP 코드를 생성하는 ①번 과정을 R 코드로 나타내면 다음과 같습니다.

|  |
| --- |
| library(rvest)  library(httr)  library(readr)  gen\_otp\_url = "http://marketdata.krx.co.kr/contents/COM/GenerateOTP.jspx"  gen\_otp\_query = list(name = "fileDown",  filetype = "csv",  url = "MKD/04/0404/04040200/mkd04040200\_01",  market\_gubun = "STK",  indx\_ind\_cd = "1001",  sect\_tp\_cd = "ALL",  schdate = "20180720",  pagePath = "/contents/MKD/04/0404/04040200/MKD04040200.jsp")  otp = POST(gen\_otp\_url, query = gen\_otp\_query)  otp\_code = read\_html(otp) %>% html\_text |

먼저 rvest, httr 및 csv 데이터를 읽는데 필요한 readr 패키지를 열어줍니다. gen\_otp\_url 변수에는 OTP 코드를 요청할 주소를 입력하며, gen\_otp\_query 변수에는 Query String Parameters의 내역들을 리스트 형태로 입력해 줍니다.

POST() 함수를 이용하여 url 및 쿼리에 관한 데이터를 요청한 후, read\_html() 함수와 html\_text() 함수를 이용하여 텍스트 형태의 OTP 코드만을 추출합니다.

|  |
| --- |
| > otp\_code  [1] "rtK5eN0Qi7s3R8mxqu7u664PgLZC4vpnSkGgDQ6NoOzgSmIeuPbgJdI+e4CjmJYYshcaScCFdVZygi59rWd/jhbIvGSwxFdKnL5LcpJ2CZNAbZ0S5di72R8r77IiNzb3eP6E+2gnFF8r8DkZ5F8CkM62/c3YqeG1/ZpGVqbmaxyqBRVAQEIMKjDIpy2YdPg22a5wG4bYeMp4p0SAco7QGayLvw6iyKc1AeukzusH96Ij236M9sIxs79XXB2KgA3ADMMBQQgZbWRrykwuHFl4l7agP9ZnE+mIAdLJJKvPj/rUfGec9GWeeyhfsZuX7FFOKO3XlbQeITgg8OFBHzCNnkW0YncPe2we/OoECZYuB1E=" |

위에서 생성된 OTP 코드를 바탕으로 csv 데이터를 요청하는 ②번 과정을 코드로 나타내면 아래와 같습니다.

|  |
| --- |
| down\_url = "http://file.krx.co.kr/download.jspx"  down\_query = list(code = otp\_code)  down\_data = POST(down\_url, query = down\_query)  down\_data = read\_html(down\_data) %>% html\_text  down\_kospi = read\_csv(down\_data) |

down\_url 변수에는 데이터를 요청할 웹주소를, down\_query 변수에는 위에서 추출한 otp\_code를 입력하며, POST() 함수를 이용하여 데이터를 요청합니다. 그 후 read\_html()과 html\_text() 함수를 이용하여 텍스트 데이터만을 추출합니다. down\_data 변수를 확인해보면 다음과 같은 csv 형태의 데이터가 저장되어 있습니다.

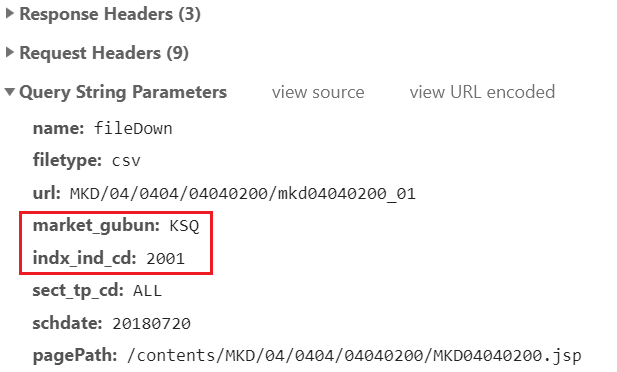
|  |
| --- |
| > down\_data  [1] "순위,종목코드,종목명,현재가,대비,등락률,거래량,거래대금,시가,고가,저가,시가총액,시가총액비중(%),상장주식수,외국인 보유주식수,외국인 지분율(%),총카운트＼n1,005930,삼성전자,＼"47,450＼",＼"550＼",＼"1.2＼",＼"10,474,547＼",＼"493,877,704,023＼",＼"47,000＼",＼"47,600＼",＼"46,700＼",＼"304,596,957,015,000＼",＼"19.89＼",＼"6,419,324,700＼",＼"3,371,001,313＼",＼"52.51＼",897＼n2,000660,SK하이닉스,＼"87,900＼",＼"-2,400＼",＼"-2.7＼",＼"3,726,334＼",＼"328,519,209,200＼",＼"89,700＼",＼"90,000＼",＼"87,200＼",＼"63,991,407,883,500＼",＼"4.18＼",＼"728,002,365＼",＼"368,873,286＼",＼"50.67＼",＼n3,068270,셀트리온,＼"285,500＼",＼"-3,000＼",＼"-1.0＼",＼"388,902＼",＼"111,032,990,000＼",＼"288,000＼",＼"290,000＼",＼"282,500＼",＼"35,800,530,306,500＼",＼"2.34＼",＼"125,395,903＼",＼"28,071,450＼",＼"22.39＼",＼n4,005935,삼성전자우,＼"38,400＼",＼"350＼",＼"0.9＼",＼"784,231＼",＼"29,781,765,050＼",＼"38,000＼",＼"38,400＼",＼"37,500＼",＼"34,699,353,600,000＼" …. |

csv 형태의 데이터를 테이블 형태로 읽어오기 위해서는 readr 패키지의 read\_csv() 함수를 사용하면 됩니다. down\_kospi 변수에는 데이터 프레임 형태로 데이터가 저장[[12]](#endnote-12)됨이 확인됩니다.

|  |
| --- |
| > head(down\_kospi)  # A tibble: 6 x 17  순위 종목코드 종목명 현재가 대비 등락률 거래량 거래대금 시가 고가 저가  <int> <chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  1 1 005930 삼성전자~ 4.74e4 550. 1.20 1.05e7 4.94e11 4.70e4 4.76e4 4.67e4  2 2 000660 SK하이닉~ 8.79e4 -2400. -2.70 3.73e6 3.29e11 8.97e4 9.00e4 8.72e4  3 3 068270 셀트리온~ 2.86e5 -3000. -1.00 3.89e5 1.11e11 2.88e5 2.90e5 2.82e5  4 4 005935 삼성전자우~ 3.84e4 350. 0.900 7.84e5 2.98e10 3.80e4 3.84e4 3.75e4  5 5 207940 삼성바이오~ 4.24e5 1000. 0.200 1.09e5 4.65e10 4.26e5 4.30e5 4.20e5  6 6 005380 현대차 1.27e5 3000. 2.40 4.64e5 5.86e10 1.25e5 1.28e5 1.23e5  # ... with 6 more variables: 시가총액 <dbl>, `시가총액비중(%)` <dbl>,  # 상장주식수 <dbl>, `외국인 보유주식수` <dbl>, `외국인 지분율(%)` <dbl>,  # 총카운트 <int> |

다음으로 코스닥 종목에 대한 데이터를 읽어오는 방법입니다. 개발자도구 화면을 연 상태에서 KOSDAQ 항목을, 조회일자를 2018-07-20으로 선택한 후 CSV 버튼을 누릅니다. 기존 코스피 데이터를 받는 경우의 query와 거의 비슷하며 market\_gubun과 indx\_ind\_cd값이 각각 KSQ와 2001로 변경되었습니다.

**[그림] 코스닥 종목의 OTP 코드 생성**



위의 변경된 사항을 이용하여 코스닥의 데이터를 읽어오는 코드는 아래와 같습니다.

|  |
| --- |
| gen\_otp\_url = "http://marketdata.krx.co.kr/contents/COM/GenerateOTP.jspx"  gen\_otp\_query = list(name = "fileDown",  filetype = "csv",  url = "MKD/04/0404/04040200/mkd04040200\_01",  market\_gubun = "KSQ",  indx\_ind\_cd = "2001",  sect\_tp\_cd = "ALL",  schdate = "20180720",  pagePath = "/contents/MKD/04/0404/04040200/MKD04040200.jsp")  otp = POST(gen\_otp\_url, query = gen\_otp\_query)  otp\_code = read\_html(otp) %>% html\_text  down\_url = "http://file.krx.co.kr/download.jspx"  down\_query = list(code = otp\_code)  down\_data = POST(down\_url, query = down\_query)  down\_data = read\_html(down\_data) %>% html\_text  down\_kosdaq = read\_csv(down\_data) |

기존 코드와 대부분의 내용이 동일하며, OTP 코드 생성시 필요한 쿼리 부분의 market\_gubun 및 indx\_ind\_cd 값만 코스닥 내역에 맞게 변경되었습니다. down\_kosdaq 변수를 확인해 보면, 코스닥 종목의 정보가 저장됨이 확인됩니다.

|  |
| --- |
| > head(down\_kosdaq)  # A tibble: 6 x 17  순위 종목코드 종목명 현재가 대비 등락률 거래량 거래대금 시가 고가  <int> <chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  1 1 091990 셀트리온헬스케~ 98200 -1200 -1.2 3.54e5 3.50e10 99400 100300  2 2 035760 CJ ENM 223100 -6900 -3 3.31e5 7.38e10 230800 230900  3 3 086900 메디톡스 794300 34300 4.5 3.61e4 2.84e10 773000 794300  4 4 215600 신라젠 61800 -400 -0.6 2.04e6 1.28e11 61900 65200  5 5 151910 나노스 8420 290 3.6 3.04e5 2.54e 9 8000 8600  6 6 084990 바이로메드~ 224400 -7300 -3.2 2.46e5 5.47e10 230400 230700  # ... with 7 more variables: 저가 <dbl>, 시가총액 <dbl>,  # `시가총액비중(%)` <dbl>, 상장주식수 <dbl>, `외국인 보유주식수` <dbl>,  # `외국인 지분율(%)` <dbl>, 총카운트 <int> |

다운받은 코스피 및 코스닥 데이터를 합친는 과정은 다음과 같습니다.

|  |
| --- |
| down\_kospi$market = "KS"  down\_kosdaq$market = "KQ"  ticker = rbind(down\_kospi, down\_kosdaq)  ticker$순위 = NULL |

각 종목의 거래 시장 구분을 위하여 코스피 종목의 market 열에는 “KS”, 코스닥 종목의 market 열에는 “KQ”를 입력합니다. rbind() 함수를 이용하여 코스피 데이터와 코스닥 데이터를 하나로 합친 후, ticker 변수에 저장해주도록 합니다. 순위에 해당하는 항목은 필요가 없는 데이터이므로. ticker$순위 = NULL 코드를 통해 해당 열을 삭제해 줍니다.

|  |
| --- |
| ticker = data.frame(ticker)  ticker = ticker[!grepl("스팩", ticker$종목명), ]  ticker = ticker[substr(ticker$종목명, nchar(ticker$종목명), nchar(ticker$종목명)) != "우", ]  ticker = ticker[substr(ticker$종목명, nchar(ticker$종목명)-1, nchar(ticker$종목명)) != "우B", ]  ticker = ticker[substr(ticker$종목명, nchar(ticker$종목명)-1, nchar(ticker$종목명)) != "우C", ]  ticker = ticker[substr(ticker$종목명, nchar(ticker$종목명), nchar(ticker$종목명)) != "호", ]  write.csv(ticker, "KOR\_ticker.csv") |

위에서 구한 데이터에는 스팩[[13]](#endnote-13)이나 우선주, 선박펀드와 같이 일반적인 주식이 아닌 종목들도 포함되어 있으므로, 이를 제거해주도록 합니다. 먼저 ticker 변수를 데이터 프레임 형태로 변경해줍니다. grepl() 함수는 grepl(pattern, x)의 형식으로 사용되며, x 내에서 pattern의 존재 여부를 TRUE와 FALSE로 반환합니다. 즉, grepl("스팩", ticker$종목명)은 ticker 변수의 ‘종목명’ 열 중에서 ‘스팩’이라는 단어가 존재하는 행은 TRUE, 그렇지 않은 행은 FALSE를 반환합니다. 또한 R에서 느낌표(!)는 Not의 의미이므로, !grepl("스팩", ticker$종목명)는 결과적으로 종목명에 ‘스팩’이라는 단어가 존재하지 않는 행을 의미합니다. 이를 통해 ticker 변수 중 스팩이 아닌 종목만을 선택하여 다시 ticker에 저장해주도록 합니다.

또한 우선주는 ‘삼성전자우’, ‘현대차우’ 와 같이 종목명의 끝이 ‘우’로 끝나는 경우가 있으며, ‘현대차2우B’, ‘현대차3우B’, ‘루트로닉3우C’와 같이 ‘우B’ 혹은 ‘우C’로 끝나는 경우도 존재합니다. substr() 함수는 substr(x, start, stop) 형식으로 이루어지며, x 문자열에서 start 부터 stop 까지의 문자를 추출합니다. nchar() 함수는 문자열의 길이를 계산하는 함수이며, !=는 같지 않음을 나타내는 표현식입니다. 즉 종목명의 마지막 한글자가 “우”가 아닌 종목, 또는 마지막 두글자가 “우B” 혹은 “우C”가 아닌 종목을 선택하여 ticker 저장해주도록 합니다.[[14]](#endnote-14)

마지막으로 바다로19호, 하나니켈1호와 같이 선박펀드나 광물펀드가 상장된 경우도 있습니다. 종목명의 마지막 한글자가 “호”가 아닌 종목을 선택함으로써 해당 종목들을 제거할 수 있습니다. 이러한 과정이 모두 완료되었으면, csv 파일로 저장함으로써 작업을 완료합니다.

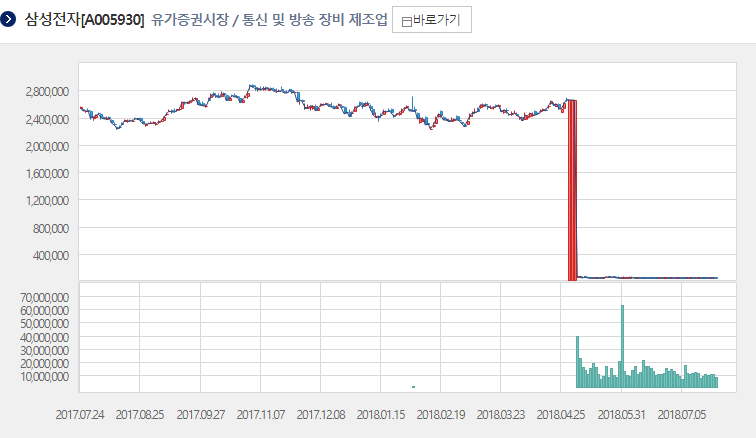
위에서 살펴본 코드 중 schdate 값에 “yyyymmdd” 형식으로 특정일을 입력하면, 해당일의 코스피 및 코스닥 데이터를 다운로드 받을 수 있습니다. 단, “20180101”처럼 영업일이 아닌 날짜를 입력할 시에는 오류가 발생합니다.

**[2] 수정주가 다운로드**

주가 데이터를 수집할 수 있는 방법은 여러가지가 있습니다. 이전 챕터에서 설명한 것처럼 한국거래소에서 제공하는 데이터를 이용하는 것이 가장 정확하지만 여기에는 한가지 문제가 있습니다.

한국거래소 홈페이지에서 **‘시장정보 → 주식 → 종목정보 → 일자별시세’**를 선택하면 종목별 일자별 주가를 확인할 수 있습니다. 삼성전자의 주가 추이를 살펴보면 2,500,000만원대의 주가가 2018년 5월 초 갑자기 하락한 것으로 나타납니다. 이는 수정주가가 적용되지 않으므로 인해 발생한 왜곡입니다. 거래소에서 제공하는 데이터를 그대로 사용할 경우 5월 4일의 수익률은 -98%라는 결과가 나오며, 한국거래소에서 제공하는 주가를 사용하기에는 무리가 있습니다.

**[그림] 한국거래소 제공 삼성전자 주가 추이**



삼성전자는 5월 4일부로 1주당 가액을 5,000원에서 100원으로 낮추는 액면분할을 실시하였습니다. 따라서 기존 5월 3일의 주가는 2,650,000원을 50으로 나눈 53,000원으로 수정되어야 합니다. 수정주가를 적용할 경우, 5월 4일의 수익률은 -2.08%로 제대로 된 결과가 나오게 됩니다.

**[표] 수정 전 및 수정 후 주가 계산**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **일자** | **수정 전** | | **수정 후** | |
| **주가** | **수익률** | **주가** | **수익률** |
| 5월 3일 | 2,650,000 |  | 53,000\* |  |
| 5월 4일 | 51,900 | -98.04% | 51,900 | -2.08% |
| \*: 2,650,000을 50으로 나눈 값 | | | | |

**야후 API를 이용한 국내 주가 다운로드**

2장에서 quantmod 패키지의 getSymbols() 함수를 통해 야후 파이낸스 API를 이용한 주가 다운로드 방법을 학습하였습니다. 이를 이용하여 전종목의 주가를 다운로드 받는 코드는 다음과 같습니다.

|  |
| --- |
| ticker = read.csv("KOR\_ticker.csv", row.names = 1)  library(quantmod)  for (i in 1 : nrow(ticker)) {  name = paste0(ticker[i, '종목코드'], ".", ticker[i, 'market'])  price = xts(NA, order.by = Sys.Date())    price = Cl(getSymbols(name, auto.assign = FALSE))  write.csv(data.frame(price), paste0(name, "\_price.csv"))  print(c(name, i / nrow(ticker)))  Sys.sleep(3)  } |

먼저 전 챕터에서 만든 국내 상장 전종목의 티커를 불러옵니다. row.names은 n번째 열을 행이름으로 선택할 지에 대한 변수이며, 첫번째 열을 지정해 줍니다. 그 후, getSymbols() 함수[[15]](#endnote-15)를 사용하기 위해 quantmod 패키지를 불러옵니다.

for loop 구문 내의 내용은 다음과 같습니다. 순서별로 티커의 종목코드 및 market을 불러온 후, 이를 결합한 후 getSymbols() 함수에 대입하여 각 종목의 주가를 다운로드 받은 후, 이를 csv 파일로 저장합니다. 상세한 내용은 다음과 같습니다.

이해를 위해 i=1, 즉 첫번째 기업인 삼성전자의 예시를 살펴봅니다. paste0() 함수는 괄호 내 입력된 값들을 띄어쓰기 없이 문자열의 형태로 결합합니다. ticker[i, '종목코드']인 005930, 콤마(.), ticker[i, 'market']인 KS를 결합하면 005930.KS의 값이 반환됩니다. 이를 name 변수에 저장한 후, price 변수에 NA값으로만 이루어진 xts 데이터를 생성합니다. xts() 함수는 시계열 데이터를 생성하는 함수이며 xts(x, order.by = index(x))의 형태로 이루어 집니다. x는 시계열로 만들고자 하는 데이터, order.by는 인덱스로 사용하고자 하는 시간 형태의 데이터입니다. Sys.Date() 함수는 현재 날짜를 가져오는 함수이며, 이를 통해 만들어진 데이터는 다음과 같습니다.

|  |
| --- |
| > xts(NA, order.by = Sys.Date())  [,1]  2018-07-27 NA |

이후 name 변수를 getSymbols() 함수의 입력변수로 사용하며, price 변수에 저장하기 위해 auto.assign 값은 FALSE로 지정합니다. Cl() 함수를 이용하여 다운로드 받은 값 중 종가만을 선택하여 price에 저장합니다.

다운로드 받아진 주가는 xts의 형태로 저장되며, 이를 csv의 형태로 저장할 경우 인덱스 값인 일자가 사라지게 됩니다. 따라서 data.frame() 변수를 이용하여 데이터 프레임 형태로 변환하며, csv 파일이름은 ‘005930.KS\_price.csv’로 지정[[16]](#endnote-16)합니다. 그 후 다운로드의 진행 상황을 알기 위해 티커 이름, 그리고 총 진행률을 계산하여 print() 함수를 이용하여 출력합니다.

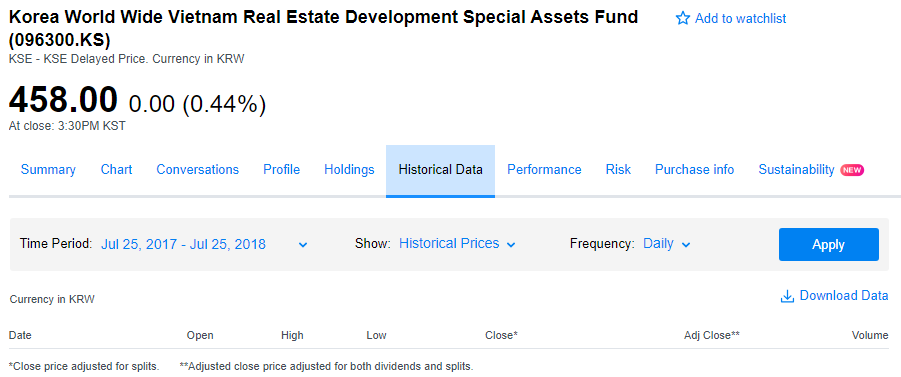
그 후 Sys.sleep() 함수를 이용하여 3초간 프로세스 수행을 지연합니다. 만일 첫번째 종목의 데이터를 다운로드 한 후 쉬지 않고 다음 종목의 데이터를 계속해서 다운로드 할 경우 서버에 지나친 트래픽 부담을 일으키고, 해당 서버는 사용자의 IP를 차단시킵니다. 이를 방지하기 위해 데이터 다운로드가 끝나면 3초 이상 쉰 후 다음 데이터를 다운로드 받는 것이 좋습니다.

위의 코드를 실행할 경우 티커 순서대로 주가 다운로드 및 csv 저장을 수행합니다. 그러나 특정 종목에서 에러가 나타난 후 코드가 멈춰버리는 일이 발생합니다.

|  |
| --- |
| Warning: 096300.KS download failed; trying again.  Error: 096300.KS download failed after two attempts. Error message:  HTTP error 404. |

오류 내용을 살펴보면 **096300.KS**, 즉 **베트남개발1** 종목의 데이터를 다운로드 받는데 실패하여 코드가 멈췄습니다. 실제로 야후 파이낸스에 접속해보면 해당 데이터에 대한 주가 정보는 제공하지 않습니다. 이 외에도 일부 종목에 대한 주가 정보가 제공되지 않으며, 이로 인해 코드가 멈춰버리는 일을 방지하기 위한 추가적인 작업이 필요합니다.

**[그림] 야후 파이낸스 내 베트남개발1 주가 정보**



**tryCatch() 함수를 이용한 오류 무시하기**

위의 경우처럼 일부 종목은 야후 파이낸스에서 주가 데이터를 제공하지 않으며, 이로 인해 발생하는 오류를 무시하고 for loop 구문의 다음 루프로 넘어간다면 코드가 멈추지 않고 끝까지 실행됩니다. 여러 프로그래밍 언어에서 try, catch, finally 등을 사용하여 예외를 처리할 수 있으며, R에서도 tryCatch() 함수를 통해 오류 발생시 예외 처리를 할 수 있습니다. 해당 함수의 사용법은 다음과 같습니다.

|  |
| --- |
| result = tryCatch({  expression  }, warning = function(w) {  warning-handler-code  }, error = function(e) {  error-handler-code  }, finally = {  cleanup-code  } |

먼저 expression는 기본적으로 실행하고자 하는 코드입니다. warning-handler-code는 경고 발생 시 실행되는 코드이며, error-handler-code는 오류 발생시 실행되는 코드입니다. cleanup-code는 오류 여부에 관계없이 반드시 수행되는 코드입니다.

위에서 작성한 코드에 tryCatch() 함수를 적용하면 다음과 같습니다.

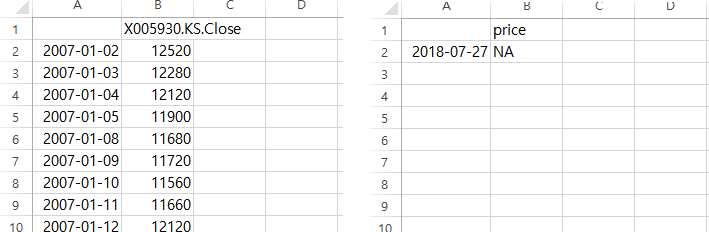
|  |
| --- |
| for (i in 1 : nrow(ticker)) {  name = paste0(ticker[i, '종목코드'], ".", ticker[i, 'market'])  price = xts(NA, order.by = Sys.Date())    tryCatch({  price = Cl(getSymbols(name, auto.assign = FALSE))  }, error = function(e) {  print(paste0("Error in Ticker: ", name))  })  price = price[!duplicated(index(price))]    write.csv(data.frame(price), paste0(name, "\_price.csv"))  print(c(name, i / nrow(ticker)))  Sys.sleep(3)  } |

대부분의 코드는 동일하며, tryCatch() 함수를 통한 예외처리가 getSymbols() 함수의 앞뒤에 추가되었습니다. getSymbols() 함수를 통해 데이터를 받지 못하고 오류가 발생할 경우 function(e) {} 안의 내용인 print(paste0("Error in Ticker: ", name)) 명령어를 통해, 오류가 발생한 티커를 출력한 후 다음 loop로 넘어갑니다. 이를 통해 어떠한 종목이 오류가 발생했는지 알 수 있으며, 사후적으로 오류 발생의 원인을 조사할 수도 있습니다.

또한 일부 종목의 경우 (시간외 거래량 반영 등으로 인해) 동일 일자의 주가가 두번 나오는 현상이 발생합니다. 이 중 중복된 일자 중 하나의 데이터만 선택하기 위해 !duplicated() 코드를 이용하여 줍니다. duplicated() 함수는 중복되는 데이터를 찾아주며, R에서 느낌표(!)는 Not의 의미입니다. 따라서 !duplicated()는 중복되지 않은 데이터를 의미합니다. price[!duplicated(index(price))] 코드를 통해 price의 인덱스, 즉 일자가 중복되지 않은 데이터만을 선택합니다.

위 코드를 실행하면 오류 발생으로 인한 멈춤없이 모든 종목의 주가가 다운로드 됩니다. 작업 공간의 폴더를 확인해보면, ‘티커.ks(kq)\_price.csv’의 형태로 파일이 저장되어 있음도 확인됩니다. 아래 그림의 좌측은 삼성전자의 주가 데이터인 005930.KS\_price.csv 파일이며, 우측은 베트남개발1의 데이터인 096300.KS\_price.csv 파일입니다. 오류가 발생하지 않을 경우에는 주가 데이터를 제대로 받아오며, 오류가 발생할 시에는 NA 데이터를 저장합니다.

**[그림] 작업 공간 폴더 내 주가 데이터 파일 비교**



해당 과정을 완료하고 나면 다음과 같은 메시지가 뜹니다. 이는 for loop 과정에서 경고들이 발생하였음을 알려주는 것이며, warnings() 함수를 통해 경고 내용들을 확인할 수 있습니다.

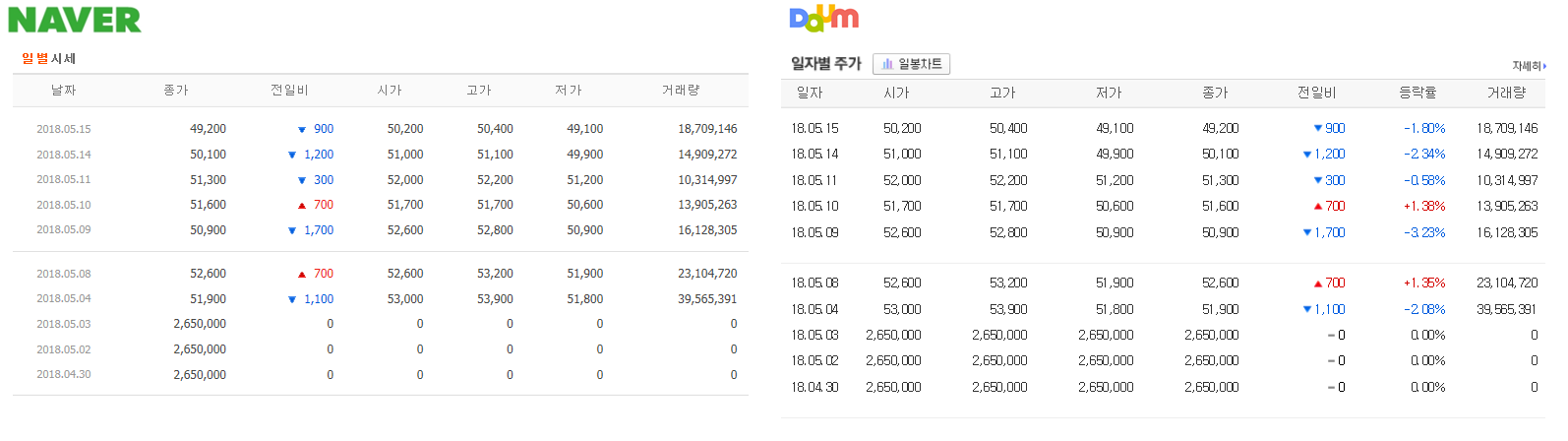
|  |
| --- |
| There were 50 or more warnings (use warnings() to see the first 50)  > warnings()  경고 메시지:  1: 068270.KS contains missing values. Some functions will not work if objects contain missing values in the middle of the series. Consider using na.omit(), na.approx(), na.fill(), etc to remove or replace them.  ... |

경고 메시지를 확인해보면, 해당 종목들의 데이터에서 결측치가 발생하였다는 내용입니다. 이처럼 야후 파이낸스 API를 이용하여 주가를 받을 경우 다수의 종목에서 특정일 혹은 몇 달간의 주가가 비어있거나 NA로 표시되는 경우가 있습니다. 이러한 문제는 na.omit(), na.approx(). na.fill() 등의 함수로 처리를 해주어야 하며, 이에 대해서는 다음 장에서 다루도록 하겠습니다.

**크롤링을 이용한 국내 주가 다운로드**

getSymbols() 함수를 통해 야후 파이낸스 API를 이용할 경우 국내 전종목 주가 데이터를 매우 짧은 코드로 다운로드 받을 수 있으며, 이미 정제된 데이터이므로 데이터 클랜징을 할 필요도 없습니다. 그러나 야후 파이낸스 API가 언제까지나 제공된다는 보장이 없습니다. 실제 구글 파이낸스 API를 통한 금융 데이터 다운로드도 가능했지만, 2018년 5월 이후 해당 서비스가 종료되어 더는 활용할 수 없습니다. 야후 파이낸스 API 또한 과거에 서비스가 중지되었던 경우가 있으며, 서비스를 종료하지 않는다는 보장이 없습니다.

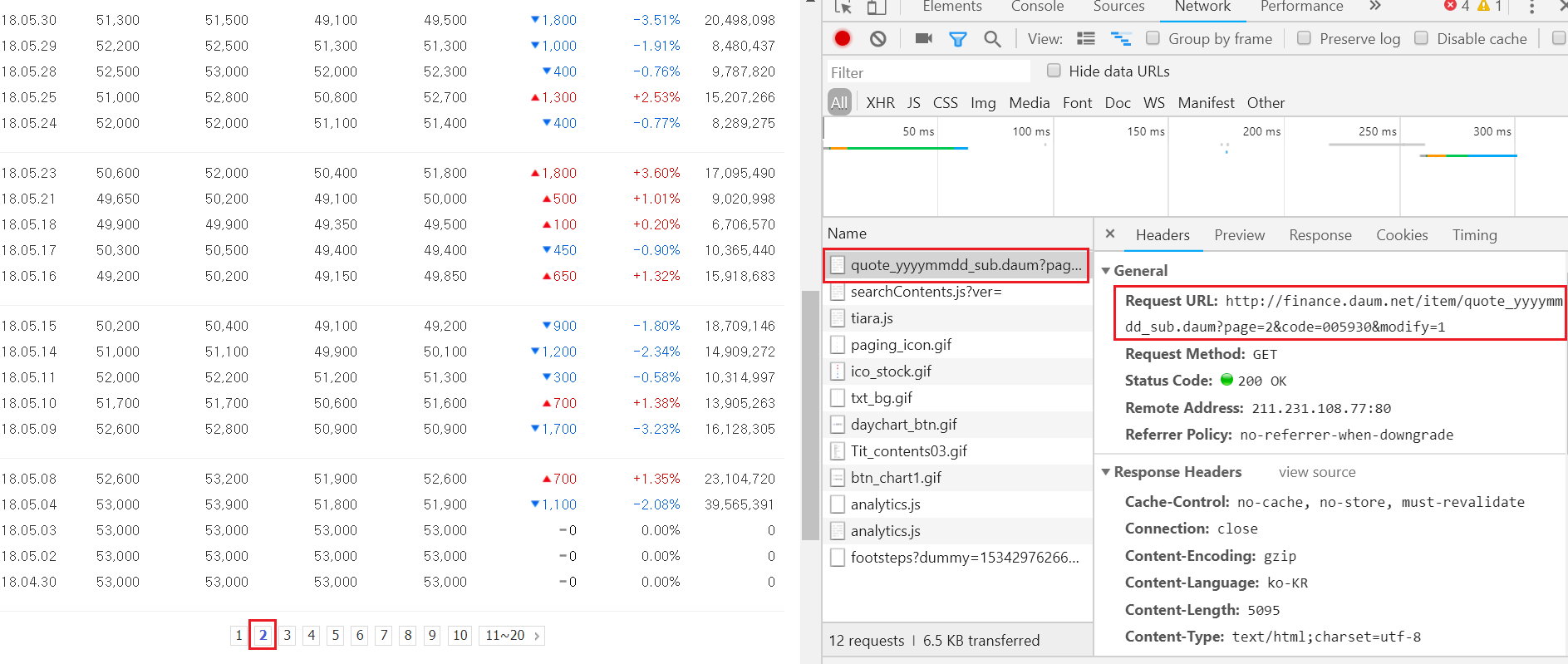
이러한 문제의 대안으로 크롤링을 이용하여 주가 데이터를 구하는 방법을 알아둘 필요가 있습니다. 국내 포털 사이트 중 네이버 금융[[17]](#endnote-17)과 다음 금융[[18]](#endnote-18)에서는 주가 데이터를 무료로 제공하고 있습니다. 그러나 네이버 금융은 수정주가를 제공하지 않는 반면, 다음 금융은 수정주가를 제공해 줍니다. 아래 그림은 삼성전자의 액면분할 시기의 주가 비교이며, 이러한 이유로 다음 금융의 주가 데이터를 이용하는 것이 훨씬 바람직합니다.

**[그림] 네이버 금융과 다음 금융 수정주가 제공 비교**

먼저 다음 금융 웹사이트에 접속한 후, ‘삼성전자 검색 → 현재가 → 일자별’을 클릭합니다. ‘수정주가 적용’ 항목이 기본값으로 선택되어 있으며, 해당 항목을 해제할 경우 수정주가가 아닌 일반주가를 보여줍니다.

개발자도구 화면을 연 후, 하단의 버튼을 누르면 해당 데이터를 어디에서 불러오는지 알 수 있습니다.

**[그림] 다음 금융의 수정주가 데이터 주소**



quote\_yyyymmdd\_sub.daum?... 항목의 Request URL을 살펴보면 주소가 다음과 같습니다.

|  |
| --- |
| http://finance.daum.net/item/quote\_yyyymmdd\_sub.daum?page=2&code=005930&modify=1 |

앰퍼샌드(&)와 물음표(?) 형식으로 주소가 구성되어 있는것으로 보아 GET 방식임을 알 수 있습니다. page= 다음의 숫자는 페이지 숫자를, code= 다음의 숫자는 종목 티커를, modify= 다음의 숫자는 수정주가 여부를 의미합니다. 따라서 페이지 숫자와 종목 티커를 변수로 지정하여, 전종목의 주가 데이터를 다운로드 받을 수 있습니다. 이를 코드로 나타내면 다음과 같습니다.

|  |
| --- |
| ticker = read.csv("KOR\_ticker.csv", row.names = 1)  library(rvest)  library(httr)  library(lubridate)  library(xts)  for (i in 1:nrow(ticker)) {  name = ticker[i, '종목코드'] %>% as.character()  price = list(xts(NA, order.by = Sys.Date()))  tryCatch({  for (j in 1:10) {  url = paste0("http://finance.daum.net/item/quote\_yyyymmdd\_sub.daum?page=",  j,"&code=",name,"&modify=1")  Sys.setlocale("LC\_ALL", "English")  data = GET(url)  data\_table = read\_html(data) %>%  html\_table()  Sys.setlocale("LC\_ALL", "Korean")  data\_table = data\_table[[1]][c('일자', '종가')]  data\_table[data\_table == ""] = NA  data\_table = na.omit(data\_table)  data\_table$일자 = ymd(data\_table$일자)  data\_table$종가 = gsub(",", "", data\_table$종가) %>% as.numeric()  rownames(data\_table) = data\_table$일자  data\_table$일자 = NULL  data\_table = as.xts(data\_table)  price[[j]] = data\_table  Sys.sleep(0.5)  }  }, error = function(e) {  print(paste0("Error in Ticker: ", name))}  )  price = do.call(rbind, price)  price = price[!duplicated(index(price))]  write.csv(data.frame(price\_list), paste0(name, "\_price.csv"))  print(c(name, i / nrow(ticker)))  Sys.sleep(3)  } |

전 챕터에서 만든 국내 상장 전종목의 티커를 불러와 ticker 변수에 저장한 후, 기존 rvest와 httr 패키지 외에 데이터 클랜징 작업에 필요한 lubridate, xts 패키지를 불러옵니다. for loop 구문 중 i=1인 삼성전자의 예시를 살펴보면 다음과 같습니다. 먼저 ticker[i, '종목코드']는 티커인 005930을 의미하며, as.character() 함수를 이용하여 문자열의 형태로 name 변수에 저장합니다. 그 후 NA로 구성된 xts 형태의 데이터를 리스트 형태에 넣은 후, 크롤링한 데이터들이 저장될 변수인 price에 저장합니다.

앞서와 동일하게 오류가 발생할 경우 예외처리를 하기 위한 tryCatch() 함수를 사용합니다. 다음으로 for (j in 1:10) 코드를 이용하여 j 변수에 1부터 10까지의 루프를 실행합니다. 이는 n번째 페이지 숫자를 의미하며 첫번째부터 열번째 페이지까지 데이터를 크롤링 함을 의미합니다. 한 페이지 당 영엉일 기준 30일의 주가가 존재하며, 10개 페이지를 크롤링 할 경우 10 × 30 = 300, 1년 2개월 정도의 데이터를 크롤링하게 됩니다. 물론 종료 숫자를 10보다 크게 할 경우 더욱 긴 시계열 데이터가 구할 수 있지만 그만큼 다운로드 시간 또한 증가하며, 현실적으로 퀀트 투자에 필요한 주가 데이터는 1년으로도 충분합니다.

j=1, 즉 첫번째 페이지 예시를 살펴보면 url 변수에 name 부분에는 005930, j 부분에는 1이 들어갑니다. 이를 paste0() 함수로 결합하면 다음과 같습니다.

|  |
| --- |
| > url  [1] "http://finance.daum.net/item/quote\_yyyymmdd\_sub.daum?page=1&code=005930&modify=1" |

앞선 예제와 같이 GET() 함수를 통해 url의 정보를 받아오며, read\_html() 함수와 html\_table() 함수를 통해 HTML 정보 추출 및 표 형태의 데이터 프레임을 찾습니다.

또한 해당 코드의 앞뒤로 Sys.setlocale("LC\_ALL", "English"), Sys.setlocale("LC\_ALL", "Korean") 코드가 있습니다. R을 이용한 크롤링 중 고질적인 문제인 하나는, 한글이 들어간 웹 페이지 정보를 크롤링 할 경우 인코딩 문제가 발생한다는 점입니다. 이를 해결하는 가장 간단한 방법은 크롤링 이전에는 로케일 언어를 영어로, 크롤링 이후 분석을 시작할 때는 로케일 언어를 한글로 변경하면 됩니다.

data\_table 변수의 첫번째 리스트는 주가 데이터의 표 항목이며, 두번째 리스트는 웹 페이지 하단에 위치한 페이지 번호입니다. 따라서 우리는 첫번째 리스트만이 필요합니다. 첫번째 리스트를 확인하면 다음과 같습니다.

|  |
| --- |
| > head(data\_table[[1]])  일자 시가 고가 저가 종가 전일비 등락률 거래량  1  2 18.07.27 46,450 47,000 46,450 46,900 -0 0.00% 4,752,164  3 18.07.26 46,100 47,000 46,000 46,900 ▲750 +1.63% 7,374,946  4 18.07.25 46,250 46,550 45,900 46,150 -0 0.00% 7,222,471  5 18.07.24 46,350 46,600 45,950 46,150 ▼350 -0.75% 8,261,363  6 18.07.23 47,100 47,200 46,150 46,500 ▼950 -2.00% 10,823,782 |

이 중 우리는 일자와 종가 데이터만이 필요합니다. 이를 data\_table[[1]][c('일자', '종가')] 코드를 통해 첫번째 리스트의 일자와 종가 열을 추출하면 다음과 같습니다.

|  |
| --- |
| > head(data\_table)  일자 종가  1  2 18.07.27 46,900  3 18.07.26 46,900  4 18.07.25 46,150  5 18.07.24 46,150  6 18.07.23 46,500 |

원하는 데이터가 추출되었다면, 이제는 데이터 클랜징 작업이 필요합니다. 현재 일자와 종가 열의 데이터 항목은 모두 문자열 형태이므로, 일자의 데이터는 날짜 형태, 종가 데이터는 숫자 형태로 바꾸어 주어야 합니다.

먼저 1행과 같이 데이터가 비어 있는(“”) 행들이 존재합니다. 이러한 행들을 삭제해 주기 위해 data\_table[data\_table == ""] 명령어를 통해 “” 값들을 NA 값들로 변경해준 후, na.omit() 함수를 통해 NA 값이 포함된 행을 모두 삭제해 줍니다.

다음으로 일자 열의 데이터를 날짜 형태로 바꾸어 줍니다. lubridate 패키지 내 ymd() 함수는 문자 형태의 값을 yyyy-mm-dd 형식의 날짜 형태로 변경해 줍니다. 종가 열의 데이터를 숫자 형태로 바꾸어 주기 위해서는 먼저 쉼표(,) 문자를 없애주어야 합니다. gsub() 함수는 찾아바꾸기의 역할을 수행하며, gsub(pattern, replacement, x)의 형태로 이루어 집니다. pattern 항목은 찾을 내용, replacement는 바꿀 내용, x는 바꾸고자 하는 데이터를 의미합니다.

rownames(data\_table) = data\_table$일자 코드를 통해 data\_table 변수의 일자 열을 행이름으로 지정해준 후, NULL로 지정해주어 삭제해줍니다. 그 후, as.xts() 함수를 통해 xts 형태의 데이터로 변경해 줍니다.

각 데이터 클랜징 단계 별 data\_table 변수의 내용 변경 과정은 아래 표와 같습니다.

**[표] 데이터 클랜징 단계 별 내용 변경 과정**

|  |  |  |  |
| --- | --- | --- | --- |
| **클랜징 이전** | **공백행 삭제** | **형태 변경** | **행이름 변경** |
| > head(data\_table)  일자 종가  1  2 18.07.27 46,900  3 18.07.26 46,900  4 18.07.25 46,150  5 18.07.24 46,150  6 18.07.23 46,500 | > head(data\_table)  일자 종가  2 18.07.27 46,900  3 18.07.26 46,900  4 18.07.25 46,150  5 18.07.24 46,150  6 18.07.23 46,500  9 18.07.20 47,450 | > head(data\_table)  일자 종가  2 2018-07-27 46900  3 2018-07-26 46900  4 2018-07-25 46150  5 2018-07-24 46150  6 2018-07-23 46500  9 2018-07-20 47450 | > head(data\_table)  종가  2018-06-18 46600  2018-06-19 47000  2018-06-20 47000  2018-06-21 47050  2018-06-22 47250  2018-06-25 46650 |

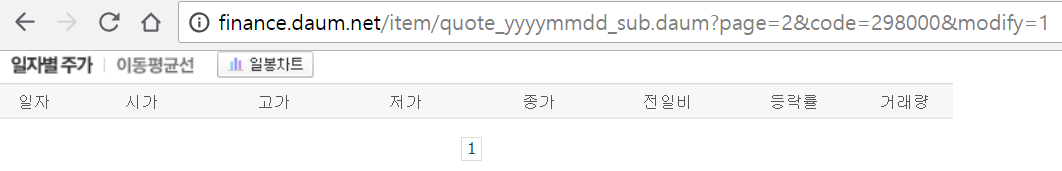
price\_list의 j번째 리스트에 해당 데이터를 저장하며, Sys.sleep() 함수를 통해 0.5초 정도의 프로세스지연을 줍니다.

위의 코드에서 발생할 수 있는 오류는 크게 2가지입니다. 먼저, 상장한지 얼마되지 않은 종목의 경우 시계열 데이터가 짧아 주가 페이지 수가 굉장히 작습니다. 예를 들어 효성의 인적분할로 생겨난 효성화학(298000)의 경우 상장일이 2018년 7월 13일이며, 2018년 7월 말 기준으로 주가 페이지가 하나 밖에 존재하지 않습니다. 반면 페이지의 for loop 구문을 담당하는 j는 1부터 10까지 페이지를 크롤링하며, j=2 단계에서의 url은 다음과 같습니다.

|  |
| --- |
| "http://finance.daum.net/item/quote\_yyyymmdd\_sub.daum?page=2&code=298000&modify=1" |

해당 url을 입력해보면 아래 그림과 같이 아무런 데이터가 나타나지 않으며, 해당 페이지를 크롤링 할 경우에도 오류가 발생합니다. 즉, 상장 후 일수가 짧은 관계로 주가 페이지가 10 이내로 존재하는 종목의 경우, j의 for loop 구문에서 오류가 발생하게 됩니다.

**[그림] 효성화학 주가 데이터의 2번째 페이지 내역**



시간 초과 오류도 종종 발생하는 오류입니다. 인터넷 접속 장애 혹은 속도 저하 등의 이유로 url에 대한 접속 시도가 특정 시간을 초과할 경우 오류가 발생합니다. 이는 인터넷 접속 환경으로 인한 오류이므로 코드의 문제가 아닌 예외적으로 발생하는 오류입니다. 오류가 발생했을 시 print(paste0("Error in Ticker: ", name)) 명령어를 통해 해당 티커에 해당하는 name을 출력합니다.

j 변수에 대한 for loop가 끝난후 list 형태로 저장되어 있는 pricㄷe 변수를 하나로 묶어주기 위해 do.call() 함수를 이용하며, 행의 형태로 묶어주기 위해 rbind() 함수를 이용합니다. 일부 종목의 경우 동일 일자의 주가가 두 페이지에 걸쳐 존재하는 경우가 있습니다. 이러한 경우 !duplicated 코드를 통해 price의 인덱스, 즉 일자가 중복되지 않은 데이터만을 선택합니다. xts 형태의 price 데이터를 데이터 프레임 형태로 변환 후, ‘티커\_price.csv’ 파일로 저장해 준 후, 다운로드 진행 상황을 출력합니다. 마지막으로 종목 별 3초간 프로세스 지연을 추가해 줍니다. 위 코드 역시 오류 발생으로 인한 멈춤없이 전 종목의 주가가 다운로드 되며, 작업 공간 폴더 내에 csv 파일이 저장되는 것도 확인할 수 있습니다.

전 종목 주가를 다운받는 방법으로 야후 파이낸스의 API를 이용하는 방법과 크롤링을 이용하는 방법을 살펴보았습니다. API를 이용할 경우 더욱 손쉽고 빠르게 정제된 데이터를 받을 수 있는 장점이 있는 반면, 데이터에 오류가 있거나 일부 종목의 데이터를 받을 수 없는 경우가 존재합니다. 반면 크롤링을 이용할 경우 상대적으로 시간을 더욱 소요되지만 비교적 신뢰할 만한 데이터를 빠짐없이 받을 수 있는 장점이 있습니다.

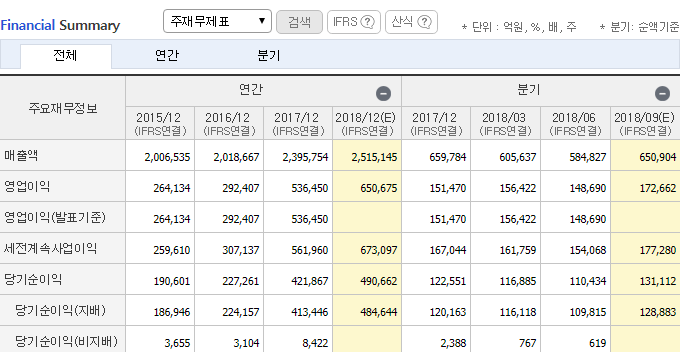
**[3] 재무제표 및 가치지표 크롤링**

밸류 전략, 퀄리티 전략 등 전통적 가치투자와 연관된 퀀트 투자 전략의 경우 재무제표 및 가치지표를 이용하여 포트폴리오를 구성합니다. NAVER 금융에서 제공하는 데이터를 이용할 경우, 잘 정리된 데이터를 크롤링을 통해 손쉽게 수집할 수 있습니다.

**재무제표 수집하기 (요약 재무제표)**

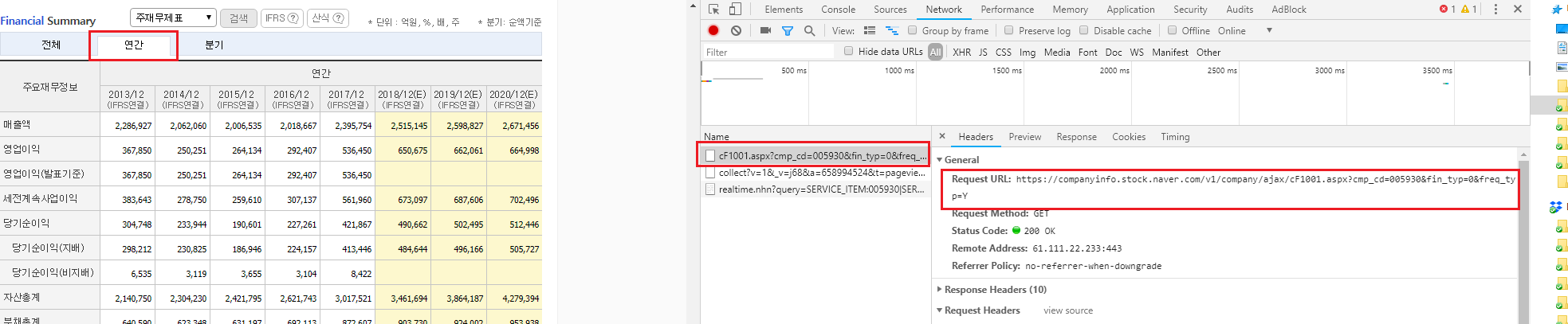
NAVER 금융([https://finance.naver.com/](http://comp.wisereport.co.kr/))에서 종목명 혹은 티커를 입력하면, 해당 종목의 다양한 정보들이 나타납니다. 이중 종목분석 메뉴의 기업현황 하단에 위치한 Financial Summary 부분에 핵심 데이터들이 요약되어 있습니다.

**[그림] 개별기업의 Financial Summary**

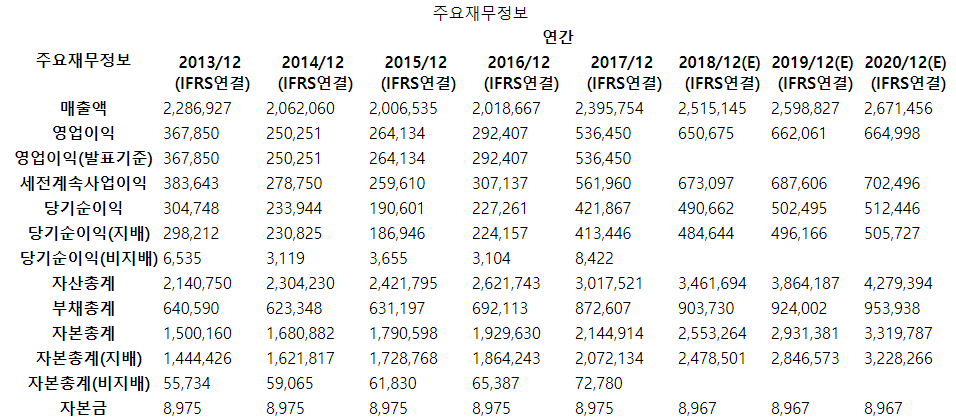


해당 화면은 JavaScript를 통해 다른 링크에서 HTML 문서를 가져온 다음 이를 보여주도록 구성되어 있으므로, 개발자도구 화면을 통해 원본 데이터가 있는 HTML 주소를 찾도록 합니다. 개발자도구 화면을 연 상태에서 ‘연간’ 부분을 클릭하면 cF1001.aspx?... 항목이 생성됩니다. 해당 항목의 Request URL에 접속할 경우, 웹페이지에서 보이는 데이터가 표로 나타납니다.

**[그림] 원본 HTML 문서 주소의 확인**



**[그림] 원본 HTML 데이터**



웹주소를 살펴보면 cmd\_cd, fin\_type, freq\_typ 총 3가지의 항목이 존재합니다. 개발자도구 화면을 연 채 Financial Summary의 항목들을 클릭해 보면, 각기 다른 항목을 클릭함에 따라 해당 부분이 변경되어 데이터를 읽어오는 HTML 문서의 주소 또한 바뀌는 것을 확인할 수 있습니다.

**[표] 웹주소의 각 항목별 구분**

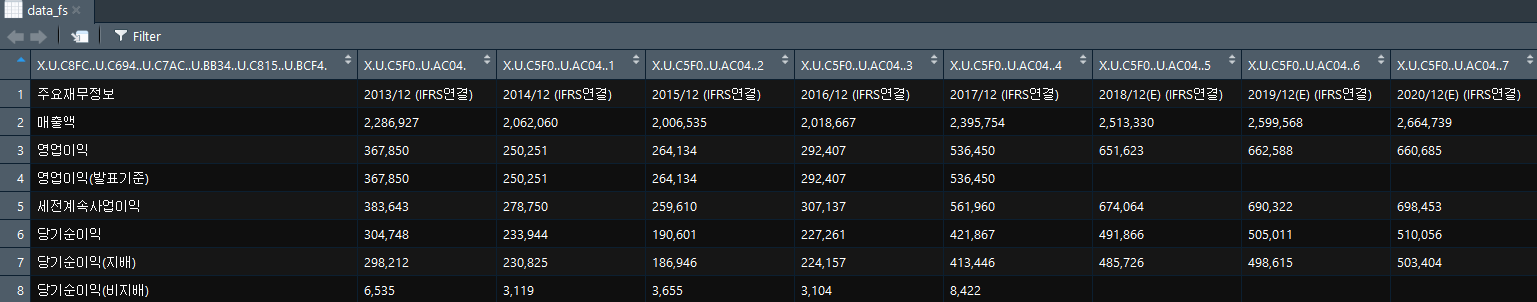
|  |  |  |
| --- | --- | --- |
| **주소** | **항목** | **구분** |
| cmd\_cd | 티커 | 6자리 종목 티커 |
| fin\_type | 재무제표 타입 | 0: IFRS 연결 1: GAAP 개별 2: GAAP 연결 3: IFRS 별도 |
| freq\_typ | 기간 | A: 전체 Y: 연간 Q: 분기 |

위의 테이블을 크롤링하는 코드는 다음과 같습니다.

|  |
| --- |
| ticker = read.csv("KOR\_ticker.csv", row.names = 1)  library(rvest)  library(httr)  for (i in 1:nrow(ticker)) {    name = ticker[i, '종목코드'] %>% as.character()  data\_fs = c()    tryCatch({    url = paste0("https://companyinfo.stock.naver.com/company/ajax/cF1001.aspx?cmp\_cd=",  name,"&fin\_typ=0&freq\_typ=Y")  Sys.setlocale("LC\_ALL", "English")  data = GET(url)  data\_fs = read\_html(data) %>%  html\_table(fill = TRUE) %>%  data.frame()  Sys.setlocale("LC\_ALL", "Korean")    data\_fs = data\_fs[, 1:6]    colnames(data\_fs) = data\_fs[1, ]  data\_fs = data\_fs[-1, ]  rownames(data\_fs) = data\_fs[, 1]  data\_fs = data\_fs[, -1]    for (j in 1:ncol(data\_fs)) {  data\_fs[, j] = gsub(",", "", data\_fs[, j]) %>% as.numeric  }  colnames(data\_fs) = substr(colnames(data\_fs),1,7)    }, error = function(e) {  data\_fs <<- NA  print(paste0("Error in Ticker: ", name))}  )    write.csv(data\_fs, paste0(name, "\_fs\_simple.csv"))  print(c(name, i / nrow(ticker)))  Sys.sleep(3)    } |

역시나 i=1인 삼성전자의 예시를 살펴보도록 하겠습니다. 국내 상장 전종목의 티커를 불러온 후, 크롤링에 필요한 rvest와 httr 패키지를 불러옵니다. 재무제표 데이터가 저장될 data\_fs 변수를 만들어 준 후, paste0() 함수를 이용하여 각종목에 맞는 url을 생성하여 줍니다. 그 후 GET(), read\_html() 및 html\_table() 함수를 차례로 실행하여 해당 url의 표 데이터를 가져오도록 합니다. 셀병합 등으로 모든 행의 열 개수가 일치하지 않을 경우 html\_table() 함수는 오류가 발생하며, fill = TRUE 인자를 추가해주어 병합된 셀들을 해제하여 줍니다. 그 후 data.frame() 함수를 이용하여 데이터 프레임 형태로 변경해주도록 합니다. data\_fs 변수에 저장된 데이터를 확인해보면, 웹페이지의 내용과 동일함을 알 수 있습니다.

**[그림] data\_fs 변수 데이터 확인**



앞선 예제와 동일하게 1열은 각 항목별 이름, 2~6열은 발표된 데이터, 7~9열은 추정 데이터이며, 데이터의 강건성을 위해 발표된 데이터인 6열까지만을 선택하여 줍니다.

데이터 클랜징 작업은 다음과 같습니다. 먼저 colnames(data\_fs) = data\_fs[1, ] 코드를 통해 첫번째 행을 열이름으로 변경해준 후, 열이름을 첫번째 행으로 변경해준 후, data\_fs = data\_fs[-1, ] 코드를 통해 해당 행을 삭제하여 줍니다. 그 후 rownames(data\_fs) = data\_fs[, 1] 코드를 통해 첫번째 열을 행으름으로 변경해 준 후, data\_fs = data\_fs[, -1] 코드를 통해 해당 열은 삭제해 줍니다. for loop 구문과 gsub(), as.numeric() 함수를 이용하여 모든 열에서 쉼표(,) 문자를 없애준 후 숫자 형태로 바꾸어 줍니다.

마지막으로 열이름을 변경해 주도록 합니다. 현재 열이름에는 \r\n\t 및 (IFRS연결)와 같이 불필요한 문자가 추가되어 있으므로, 열이름을 보기 좋게 수정해주도록 합니다.

|  |
| --- |
| > colnames(data\_fs)  [1] "2013/12\r\n\t\t\t\t\t\t\t\t(IFRS연결)" "2014/12\r\n\t\t\t\t\t\t\t\t(IFRS연결)"  [3] "2015/12\r\n\t\t\t\t\t\t\t\t(IFRS연결)" "2016/12\r\n\t\t\t\t\t\t\t\t(IFRS연결)"  [5] "2017/12\r\n\t\t\t\t\t\t\t\t(IFRS연결)" |

substr() 함수는 substr(x, start, stop) 형태로 이루어지며 x는 문자열, start는 시작점, stop은 끝을 의미합니다. 즉 x에서 시작점부터 끝지점에 해당하는 문자만을 추출하여 줍니다. 위의 예제에서는 substr(colnames(data\_fs),1,7) 코드를 통해 각 열이름의 첫번째부터 일곱번째까지 문자열을 추출하면 yyyy/mm 형태만 남게 되며, 이를 다시 열이름에 저장해주도록 합니다.

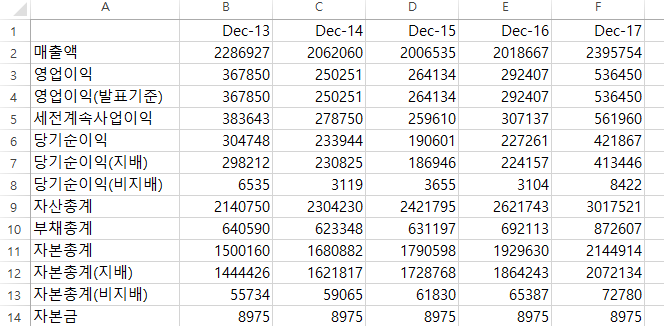
모든 작업이 끝난 후 데이터는 다음과 같습니다.

|  |
| --- |
| > head(data\_fs)  2013/12 2014/12 2015/12 2016/12 2017/12  매출액 2286927 2062060 2006535 2018667 2395754  영업이익 367850 250251 264134 292407 536450  영업이익(발표기준) 367850 250251 264134 292407 536450  세전계속사업이익 383643 278750 259610 307137 561960  당기순이익 304748 233944 190601 227261 421867  당기순이익(지배) 298212 230825 186946 224157 413446 |

error = function(e) {} 코드를 통해 오류가 발생했을 시 조건문을 추가해 줍니다. 오류가 발생하는 경우는 티커에 해당하는 페이지가 없는 경우이므로, data\_fs 변수를 NA로 처리해 줍니다. 이 중 <<- 오퍼레이터는 데이터를 전역 환경Global environment으로 지정하는 할당연산자입니다. 그 후 오류가 발생한 티커인 name을 출력한 후 다음 루프로 넘어가도록 합니다.

‘티커\_fs\_simple’ 이름의 csv 파일로 저장해 준 후, 다운로드 진행상황을 출력합니다. 마지막으로 종목 별 3초간 프로세스 지연을 추가해 줍니다. 위 코드 역시 오류 발생으로 인한 멈춤없이 전 종목의 요약 재무제표가 다운로드 되며, 작업 공간 폴더 내에 csv 파일이 저장되는 것도 확인할 수 있습니다.

**[그림] 요약 재무제표 데이터 파일**

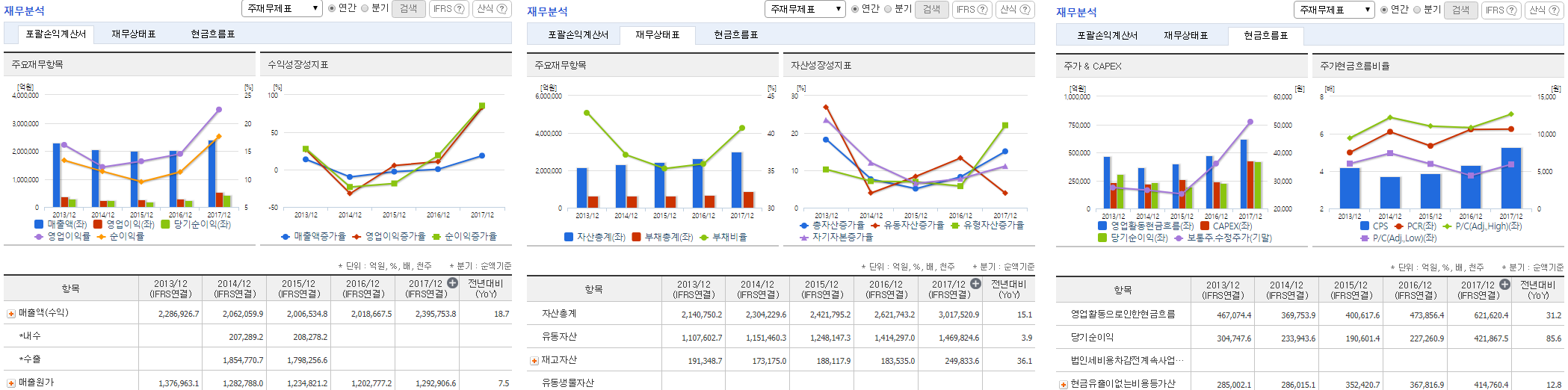


**재무제표 수집하기 (전 항목)**

앞서 수집한 요약 재무제표를 이용하여도 충분히 훌륭한 퀀트 투자 포트폴리오를 구성할 수 있습니다. 그러나 더욱 정교한 모형을 만들기 위해서는 재무제표의 모든 항목이 필요할 경우도 있으며, 이를 크롤링 하는 과정 또한 추가로 살펴보도록 하겠습니다.

특정 종목을 검색한 후 종목분석 항목에서 재무분석을 선택하면 포괄손익계산서, 재무상태표, 현금흐름표의 모든 항목을 살펴볼 수 있습니다.

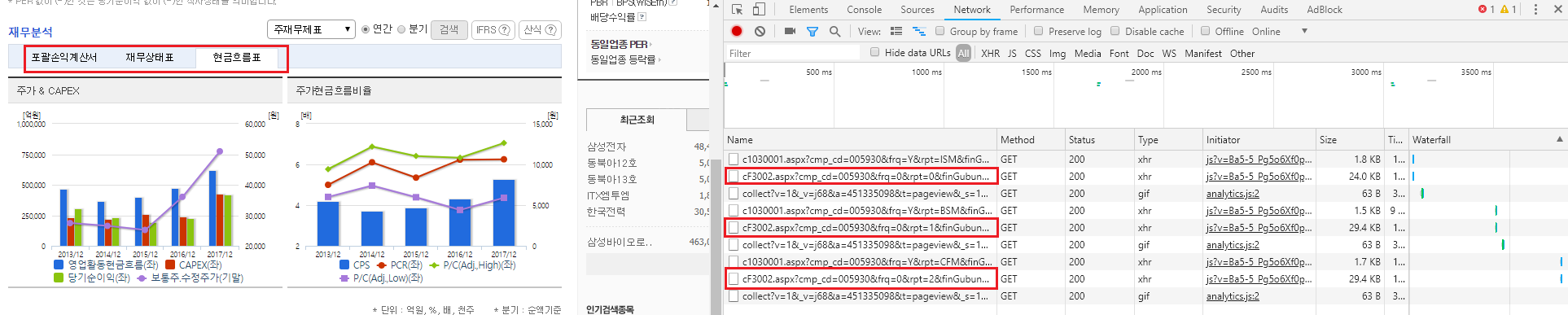
**[그림] 재무제표 세부 내역**



해당 페이지 역시 타 링크의 HTML 문서를 JavaScript가 보여주는 형식이므로, 원본이 되는 HTML 주소를 확인해야 합니다. 개발자도구 화면을 연 상태에서 포괄손익계산서, 재무상태표, 현금흐름표 항목을 각각 클릭하면 총 6개의 항목이 생성됩니다. c1030001.aspx?... 항목의 경우 그래프를 그리는 항목이며, cF3002.aspx?... 항목이 우리가 원하는 HTML 주소에서 데이터를 가지고 오는 항목입니다.

각 항목의 Request URL에 접속할 경우 기존에 우리가 보던 형태와는 다른, 매우 생소한 화면이 나타납니다. 중간중간 보이는 값들은 웹페이지의 내용과 동일한 반면, 구조 자체는 테이블의 형태를 보이고 있지 않습니다. 이는 해당 웹페이지가 JSON 형식으로 표현되었기 때문입니다. 기존에 우리가 살펴보았던 대부분의 웹페이지는 XML 형식으로 표현되었으며, 이는 문법이 복잡하고 엄격한 표현규칙으로 인해 데이터의 용량이 커진다는 단점이 있습니다. 반면 JSON 형식은 문법이 단순하여 데이터의 용량이 작아, 빠른 속도로 데이터를 교환할 수 있습니다.

**[그림] 재무재표 각 항목의 URL 확인**



**[그림] JSON 형태의 포괄손익계산서 항목**



포괄손익계산서, 재무상태표, 현금흐름표 각 항목의 URL을 살펴보면, 모든 항목이 동일하며 rpt= 뒷부분의 숫자에 따라 항목이 변경됨을 알 수 있습니다. 따라서 for loop 구문을 통해 cmd\_cd= 뒷부분의 티커와, rpt= 뒷부분의 숫자만 변경해주면 전 종목의 재무제표 세부사항을 수집할 수 있습니다.

**[표] 재무제표 항목 별 URL**

|  |  |
| --- | --- |
| **재무제표 항목** | **URL** |
| 포괄손익계산서 | http://comp.wisereport.co.kr/company/cF3002.aspx?cmp\_cd=005930&frq=0&rpt=0&finGubun=MAIN&frqTyp=0&cn= |
| 재무상태표 | http://comp.wisereport.co.kr/company/cF3002.aspx?cmp\_cd=005930&frq=0&rpt=1&finGubun=MAIN&frqTyp=0&cn= |
| 현금흐름표 | http://comp.wisereport.co.kr/company/cF3002.aspx?cmp\_cd=005930&frq=0&rpt=2&finGubun=MAIN&frqTyp=0&cn= |

|  |
| --- |
| ticker = read.csv("KOR\_ticker.csv", row.names = 1)  library(rvest)  library(httr)  library(tibble)  library(jsonlite)  for (i in 1:nrow(ticker)) {    name = ticker[i, '종목코드'] %>% as.character()  data\_fs = list()    tryCatch({    for (j in 0 : 2) {  url = paste0("https://companyinfo.stock.naver.com/v1/company/cF3002.aspx?cmp\_cd=",  name,"&frq=0&rpt=",j,"&finGubun=MAIN&frqTyp=0&cn=")  Sys.setlocale("LC\_ALL", "English")  data = fromJSON(url)  Sys.setlocale("LC\_ALL", "Korean")    yr\_name = data[[1]][1:5] %>% data.frame()  yr\_name = apply(yr\_name, 1, function(x) {substr(x, 1, 4)})    data = data[[2]]  data\_table = cbind(data$DATA1, data$DATA2, data$DATA3, data$DATA4, data$DATA5)  rownames(data\_table) = data$ACC\_NM  colnames(data\_table) = yr\_name    data\_fs[[j+1]] = data\_table  Sys.sleep(0.5)  }    data\_fs = do.call(rbind, data\_fs)  data\_fs = data\_fs[!duplicated(rownames(data\_fs)), ]  }, error = function(e) {  data\_fs <<- NA  print(paste0("Error in Ticker: ", name))}  )    write.csv(data\_fs, paste0(name, "\_fs\_all.csv"))  print(c(name, i / nrow(ticker)))  Sys.sleep(3)    } |

먼저 rvest와 httr 패키지 외에 JSON 형태의 웹페이지 크롤링에 필요한 jsonlite ㅍ ㅐ키지를 불러옵니다. 그 후 각 항목들이 저장될 data\_fs 변수를 빈 리스트 형태로 만들어 줍니다. 그 후, for (j in 0 : 2) 코드를 이용하여 j 변수에 0부터 2까지의 루프를 실행합니다. 이는 앞서 살펴본 각 재무제표 항목들의 url에서 rpt= 항목에 해당하는 숫자를 입력하기 위함입니다.

먼저 j=0인 포괄손익계산서 항목의 예시입니다. paste0() 함수를 이용하여 티커 및 재무제표 항목에 해당하는 url을 만들어 준 후, fromJSON() 함수를 이용하여 JSON 형태의 페이지를 읽어옵니다. data 변수에는 list 형태로 데이터가 저장되며, 첫번째 리스트에는 연도를 나타내는 문자열이, 두번째 리스트에는 각 항목별 데이터가 저장됩니다.

먼저 연도에 해당하는 첫번째 리스트의 내용은 다음과 같으며, 이 중 필요한 항목만을 선택한 후 정리해줄 필요가 있습니다.

|  |
| --- |
| > data[[1]]  [1] "2013/12<br />(IFRS연결)" "2014/12<br />(IFRS연결)" "2015/12<br />(IFRS연결)"  [4] "2016/12<br />(IFRS연결)" "2017/12<br />(IFRS연결)" "2018/12(E)<br />(IFRS연결)"  [7] "전년대비<br />(YoY)" "전년대비<br />(YoY)" |

총 8개의 문자열 중 발표된 데이터에 해당하는 부분은 첫번째(2013/12) 부터 다섯번째(2017/12) 까지이며, data[[1]][1:5] 코드를 통해 해당 부분만을 선택한 후, data.frame() 함수를 통해 데이터 프레임 형식으로 변경합니다. 그 후 apply 함수 내에 substr() 함수를 응용하여 첫번째부터 네번째까지의 문자열, 즉 yyyy 부분만 선택해 주도록 합니다. 결과는 다음과 같습니다.

|  |
| --- |
| > yr\_name = data[[1]][1:5] %>% data.frame()  > yr\_name = apply(yr\_name, 1, function(x) {substr(x, 1, 4)})  > yr\_name  [1] "2013" "2014" "2015" "2016" "2017" |

다음으로 각 항목별 데이터가 저장된 두번째 리스트를 data 변수에 저장합니다. 그 후 발표된 데이터에 해당하는 DATA1, DATA2, DATA3, DATA4, DATA5 열들을 cbind() 함수를 통해 열의 형태로 묶어줍니다. 그 후 행이름은 각 변수명을 나타내는 ACC\_NM 열을, 열이름은 위에서 생성한 yr\_name 변수를 입력합니다. 정리된 테이블은 다음과 같습니다.

|  |
| --- |
| > head(data\_table)  2013 2014 2015 2016 2017  매출액(수익) 2286927 2062059.9 2006534.8 2018667 2395754  \*내수 NA 207289.2 208278.2 NA NA  \*수출 NA 1854770.7 1798256.6 NA NA  ....제품매출액 NA NA NA NA NA  ....상품매출액 NA NA NA NA NA  ....제품상품매출액 NA NA NA NA NA |

해당 테이블을 data\_fs 리스트에 저장합니다. 현재 j값은 0인데 반해 리스트는 첫번째부터 저장이 가능하므로, j+1번째 리스트에 해당 데이터를 저장합니다. 그 후 0.5초의 프로세스지연을 준 후, 다음번째 j(재무제표 항목)를 다운로드 받습니다.

for loop 구문을 통해 data\_fs 리스트에는 세가지 재무제표 항목이 저장되어 있습니다. 첫번째 리스트에는 포괄손익계산서 항목이, 두번째 리스트에는 재무상태표 항목이, 세번째 리스트에는 현금흐름표 항목이 저장되어 있습니다.

|  |
| --- |
| > head(data\_fs[[1]])  2013 2014 2015 2016 2017  매출액(수익) 2286927 2062059.9 2006534.8 2018667 2395754  \*내수 NA 207289.2 208278.2 NA NA  \*수출 NA 1854770.7 1798256.6 NA NA  ....제품매출액 NA NA NA NA NA  ....상품매출액 NA NA NA NA NA  ....제품상품매출액 NA NA NA NA NA  > head(data\_fs[[2]])  2013 2014 2015 2016 2017  자산총계 2140750.2 2304229.58 2421795.2 2621743.24 3017520.90  유동자산 1107602.7 1151460.26 1248147.2 1414297.04 1469824.64  ....재고자산 191348.7 173175.04 188117.9 183535.03 249833.55  ........상품 NA NA NA NA NA  ........제품 74293.5 57734.48 57694.6 59053.39 73044.37  ........완성건물 NA NA NA NA NA  > head(data\_fs[[3]])  2013 2014 2015 2016 2017  영업활동으로인한현금흐름 467074.40 369753.89 400617.61 473856.4 621620.41  ....당기순이익 304747.64 233943.58 190601.44 227260.9 421867.47  ....법인세비용차감전계속사업이익 NA NA NA NA NA  ....현금유출이없는비용등가산 285002.09 286015.11 352420.74 367816.9 414760.37  ........금융비용 15686.63 19335.65 24660.42 25000.2 25906.83  ........이자비용 NA NA NA NA NA |

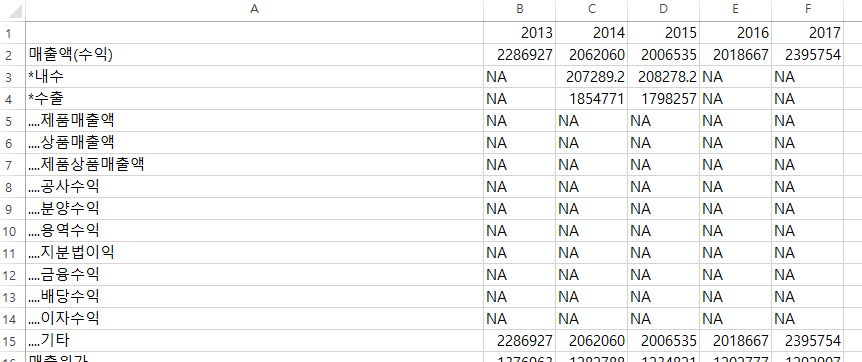
각 리스트에 저장되어 있는 재무제표 항목들을 do.call(rbind, data\_fs) 코드를 이용하여 행의 형태로 묶어줍니다. 그 후 nrow(data\_fs) 코드를 통해 행 개수를 확인하면, 총 742개 항목의 데이터가 저장되어 있습니다. 이 중 동일한 항목이 중복되어 있는 경우도 있습니다.

!duplicated(rownames(data\_fs)) 코드를 통해 data\_fs 테이블 중 행이름이 중복되지 않는 곳들의 인덱스를 찾아, 해당 행의 데이터만을 선택합니다. 해당 과정을 거치면 행 개수가 기존 742개에서 659개[[19]](#endnote-19)로 줄어드는 것이 확인됩니다.

|  |
| --- |
| > data\_fs = do.call(rbind, data\_fs)  > nrow(data\_fs)  [1] 742  > data\_fs = data\_fs[!duplicated(rownames(data\_fs)), ]  > nrow(data\_fs)  [1] 659 |

이후의 작업은 기존 예제와 동일하며, ‘티커\_fs\_all.csv’ 파일로 저장이 됩니다. 기존 요약 재무제표는 20개 내외의 항목만이 있었던 반면, 해당 방법을 이용할 경우 700개 내외의 항목을 수집할 수 있습니다.

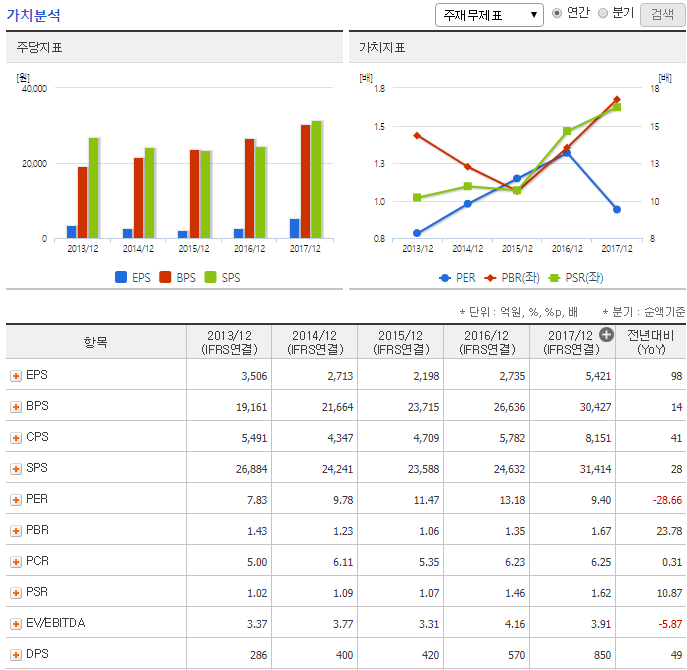
**[그림] 재무제표 전 항목 데이터 파일**



**가치지표 크롤링**

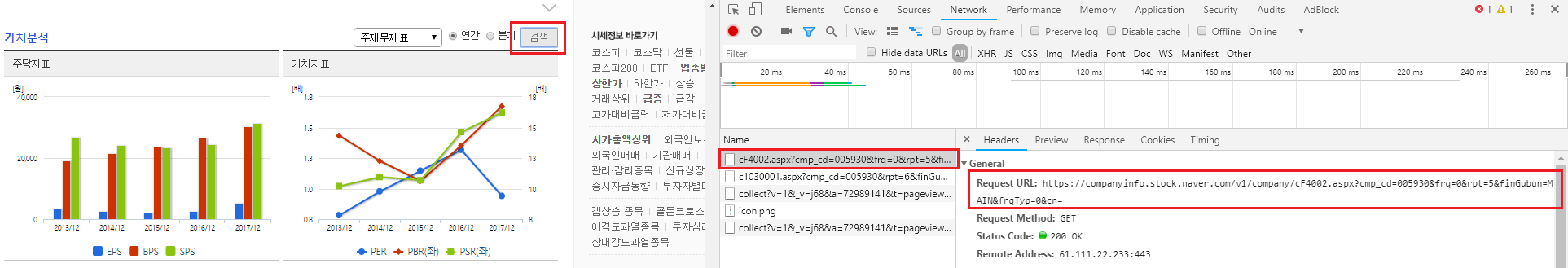
NAVER 금융의 투자지표 항목 하단의 가치분석 표에는 가치지표가 존재합니다. 이 중 전통적 가치투자 전략과 연관된 밸류 전략에는 PER, PBR, PCR, PSR 지표가 필요합니다.

**[그림] Company Wise의 가치지표**



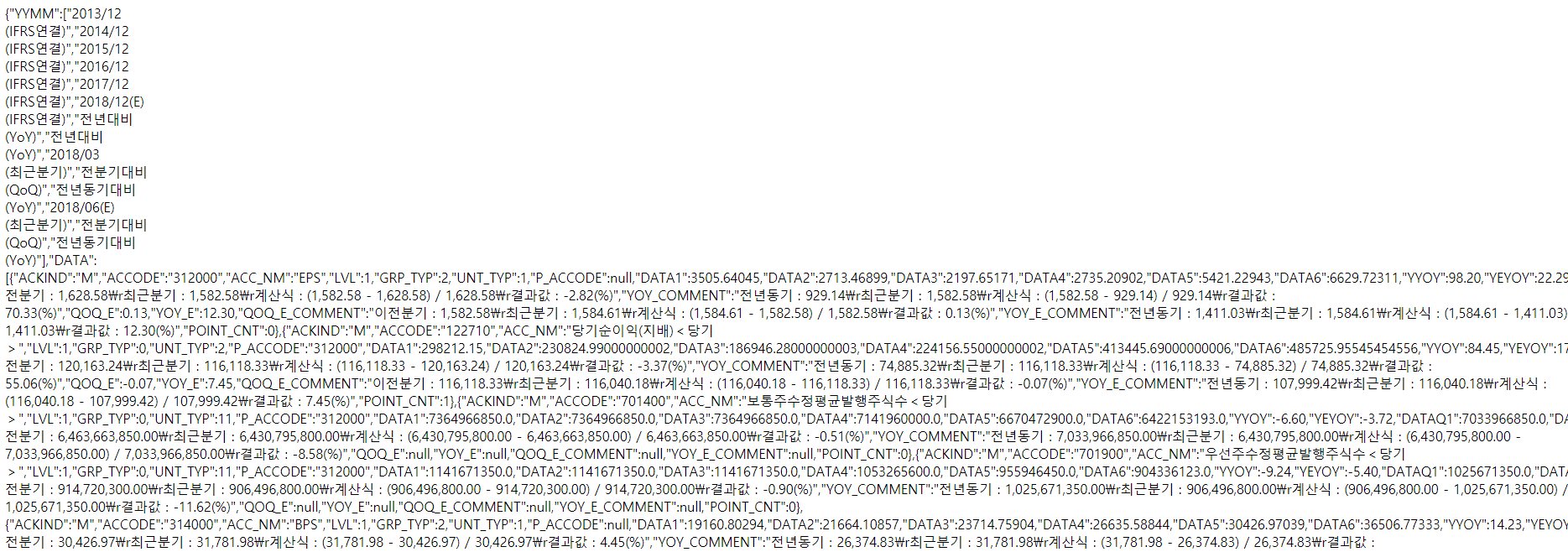
해당 데이터의 원본 HTML 데이터를 찾기 위해 크롬 개발자도구 화면을 연 상태에서 우측의 검색 버튼을 누릅니다. 상단의 cF4002.aspx?... 항목이 테이블로 된 데이터를 보여주는 항목이며, 하단의 c1030001.aspx?... 항목이 그래프를 보여주는 항목입니다.

**[그림] 원본 HTML 문서 주소의 확인**



cF4002.aspx?... 항목의 Request URL에 접속할 경우 해당 페이지 역시 JSON 형태로 작성되어 있음이 확인됩니다.

**[그림] JSON 형태의 웹페이지**



|  |
| --- |
| ticker = read.csv("KOR\_ticker.csv", row.names = 1)  library(rvest)  library(httr)  library(jsonlite)  for (i in 1:nrow(ticker)) {    name = ticker[i, '종목코드'] %>% as.character()  data\_value = c()    tryCatch({    url = paste0("http://comp.wisereport.co.kr/company/cF4002.aspx?cmp\_cd=",  name,"frq=0&rpt=5&finGubun=MAIN&frqTyp=0&cn=")  Sys.setlocale("LC\_ALL", "English")  data = fromJSON(url)  data = data[[2]] %>% data.frame()  Sys.setlocale("LC\_ALL", "Korean")    data\_table = cbind(data$DATA5)  rownames(data\_table) = data$ACC\_NM    type = c("PER", "PBR", "PCR", "PSR")  data\_value = data\_table[sapply(type, function(x) {which(rownames(data\_table) == x)}), ]    }, error = function(e) {  data\_value <<- NA  print(paste0("Error in Ticker: ", name))}  )  write.csv(data\_value, paste0(name, "\_value.csv"))  print(c(name, i / nrow(ticker)))  Sys.sleep(2)    } |

먼저 크롤링에 필요한 기존의 rvest, httr, jsonlite 패키지를 불러옵니다. 그 후 가치지표가 저장될 data\_value 변수를 만들어 준 후, paste0() 함수를 이용하여 각 종목에 맞는 url을 생성하여 줍니다.

JSON 형태의 페이지를 읽어오기 위해서는 jsonlite 패키지의 fromJSON() 함수를 사용하면 됩니다. data 변수에는 list 형태로 데이터가 저장되며, 첫번째 리스트에는 연도를 나타내는 문자열이, 두번째 리스트에는 각 항목별 데이터가 저장되어 있습니다.

|  |
| --- |
| > head(data[[1]])  [1] "2013/12<br />(IFRS연결)" "2014/12<br />(IFRS연결)" "2015/12<br />(IFRS연결)"  [4] "2016/12<br />(IFRS연결)" "2017/12<br />(IFRS연결)" "2018/12(E)<br />(IFRS연결)"  > head(names(data[[2]]))  [1] "ACKIND" "ACCODE" "ACC\_NM" "LVL" "GRP\_TYP" "UNT\_TYP" |

data[[2]] %>% data.frame() 코드를 통해 우리에게 필요한 두번째 리스트를 선택한 후, 이를 데이터 프레임 형태로 저장합니다. 이 중 우리에게 필요한 데이터는 최근 년도 발표 기준인 DATA5 열의 데이터이며, cbind(data$DATA5) 코드를 통해 해당 열의 데이터만을 data\_table 변수에 저장합니다. 또한 그 후, 각 변수의 이름을 나타내는 data$ACC\_NM 열을 행이름으로 지정해줍니다. data\_table에 저장된 데이터를 확인하면 다음과 같습니다.

|  |
| --- |
| > head(data\_table)  [,1]  EPS 5.421229e+03  당기순이익(지배)＜당기＞ 4.134457e+05  보통주수정평균발행주식수＜당기＞ 6.670473e+09  우선주수정평균발행주식수＜당기＞ 9.559465e+08  BPS 3.042697e+04  자본총계(지배)＜당기＞ 2.072134e+06 |

다음으로 가치지표인 PER, PBR, PCR, PSR 만을 선택해 보도록 하겠습니다. 먼저 해당 지표를 문자열 형태로 type 변수에 저장합니다. 그 후 sapply() 함수를 응용하여 행이름이 해당 문자열과 같은 행의 인덱스를 찾은 후, data\_table에서 해당 인덱스의 행들을 선택합니다.

sapply() 함수는 apply() 함수와 비슷하며, for loop 구문을 대체할 수 있는 가장 간단한 함수입니다. 해당 함수는 sapply(X, FUN, ..) 구조로 이루어지며 X는 변수를, FUN은 실행하고자 하는 함수를 의미합니다. 다음 코드는 sapply() 함수의 간단한 예제입니다.

|  |
| --- |
| > num = c(1,2,3)  > sapply(num, function(x) {x^2})  [1] 1 4 9 |

먼저 num 변수에 1,2,3 이라는 숫자를 입력합니다. 그 후 sapply() 함수에서 X에 해당하는 부분에 num을, FUN에 해당하는 부분에는 function(x) {x^2}을 입력합니다. 이는 num 변수 내 숫자들의 각 제곱을 나타내게 됩니다.

이와 마찬가지로 sapply(type, function(x) {which(rownames(data\_table) == x)}) 코드는 data\_table의 행이름과 type의 문자가 같은지를 비교한 후, which() 명령어를 통해 TRUE에 해당하는 부분의 인덱스를 반환합니다. 아래 코드를 살펴보면 가치지표는 각각 17, 20, 23, 26번째 행에 위치하고 있습니다.

|  |
| --- |
| > sapply(type, function(x) {which(rownames(data\_table) == x)})  PER PBR PCR PSR  17 20 23 26 |

data\_table 변수 중 위의 행에 해당하는 값, 즉 가치지표에 해당하는 값들만이 data\_value 변수에 저장되게 됩니다.

|  |
| --- |
| > data\_value  PER PBR PCR PSR  9.40010 1.67483 6.25210 1.62220 |

이후의 작업은 기존 예제와 동일하며, ‘티커\_value’ 이름의 csv 파일로 저장이 됩니다.

**현재 주가를 고려한 가치지표 계산하기**

위에서 구해진 가치지표의 가장 큰 문제점은 현재의 주가를 반영하지 못한다는 점입니다. 예를 들어 2017년 PER 값의 경우, 의 형태로 계산됩니다. 만약 2018년 중후반의 시점에서 해당 페이지를 크롤링 하여도 2017년 말일 주가로 계산된 PER 값을 수집하며, 17년 말일과 데이터 수집 시점간의 주가 움직임을 고려하지 못합니다. 따라서 가치지표를 계산할 때에는 분자에 해당하는 주가의 경우 현재 시점의 주가를 사용하는 것이 바람직[[20]](#endnote-20)합니다.

가치지표에 계산되는 재무지표인 EPS, BPS, CPS, SPS는 앞선 예제에서 이미 구하였으므로, 현재 주가만 추가적으로 크롤링하여 계산해주면 됩니다.

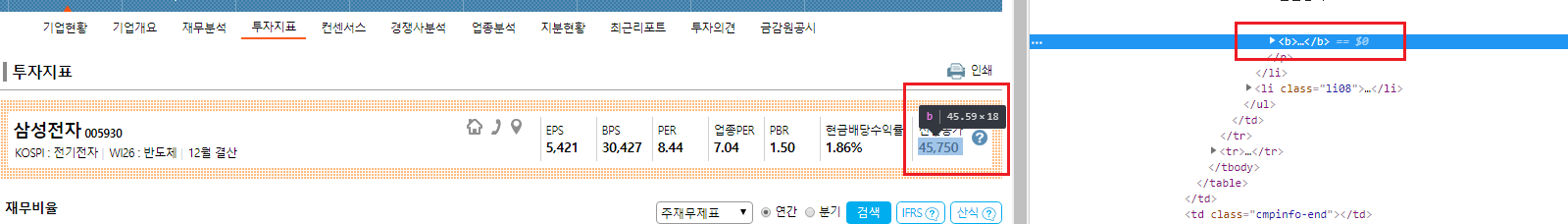
|  |
| --- |
| ticker = read.csv("KOR\_ticker.csv", row.names = 1)  for (i in 1:nrow(ticker)) {    name = ticker[i, '종목코드'] %>% as.character()  data\_value = c()    tryCatch({    url = paste0("http://comp.wisereport.co.kr/company/cF4002.aspx?cmp\_cd=",  name,"frq=0&rpt=5&finGubun=MAIN&frqTyp=0&cn=")  Sys.setlocale("LC\_ALL", "English")  data = fromJSON(url)  data = data[[2]] %>% data.frame()  Sys.setlocale("LC\_ALL", "Korean")    data\_table = cbind(data$DATA5)  rownames(data\_table) = data$ACC\_NM    type = c("EPS", "BPS", "CPS", "SPS")  data\_value = data\_table[sapply(type, function(x) {which(rownames(data\_table) == x)}), ]    url = paste0("http://comp.wisereport.co.kr/company/c1010001.aspx?cmp\_cd=",  name,"&cn=")  price = GET(url) %>%  read\_html() %>%  html\_nodes(xpath = '//\*[@id="comInfo"]/tbody/tr[1]/td[2]/ul/li[8]/p/b') %>%  html\_text()    price = gsub(",", "", price) %>% as.numeric()  data\_value = price / data\_value  names(data\_value) = c("PER", "PBR", "PCR", "PSR")    data\_value[is.infinite(data\_value)] = NA    }, error = function(e) {  data\_value <<- NA  print(paste0("Error in Ticker: ", name))}  )    write.csv(data\_value, paste0(name, "\_value.csv"))  print(c(name, i / nrow(ticker)))  Sys.sleep(2)    } |

앞선 예제에서 가치지표에 해당하는 테이블을 가져오는 코드는 동일하며, type 변수에 가치지표들의 분모 부분에 해당하는 EPS, BPS, CPS, SPS를 입력합니다. 그 후 sapply를 응용하여 해당 값들을 선택하면 다음과 같습니다.

|  |
| --- |
| > data\_value  EPS BPS CPS SPS  5421.229 30426.970 8150.882 31413.874 |

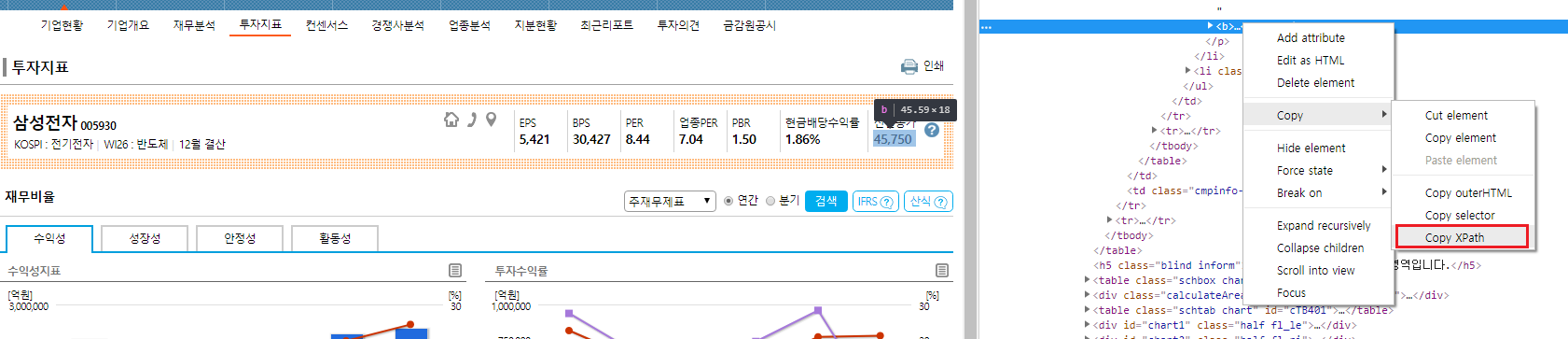
이제 주가를 해당 값들로 나누어지면 가치지표가 계산됩니다. 전일종가의 경우 웹페이지 상단 우측에 위치하고 있으므로, 해당 값을 크롤링하도록 합니다. 해당 전일종가 역시 XPath를 이용하여 손쉽게 매우 쉽게 크롤링 할 수 있습니다.

**[그림] 전일종가의 HTML 문서 부분 확인**



개발자도구 화면에서 전일 종가에 해당하는 부분에 마우스를 둔 채 우클릭 후 **Copy → Copy XPath**를 선택하면, 클립보드에 자동으로 XPath가 복사됩니다. 해당 XPath는 다음과 같습니다.

**[그림] 전일종가의 XPath의 복사 방법**



|  |
| --- |
| //\*[@id="comInfo"]/tbody/tr[1]/td[2]/ul/li[8]/p/ |

위의 XPath를 이용하여 전일종가를 크롤링하도록 합니다. url 변수에 해당 페이지의 웹주소를 새롭게 만들어 준 후, GET() 함수와 read\_html() 함수를 통해 해당 페이지의 HTML 정보를 저장합니다. 그 후 html\_nodes() 함수를 사용하며, xpath 항목에 위에서 구한 값을 입력해 줍니다. 해당 부분까지 진행할 경우, 전일 종가에 해당하는 45,750이 선택됩니다.

|  |
| --- |
| > GET(url) %>%  + read\_html() %>%  + html\_nodes(xpath = '//\*[@id="comInfo"]/tbody/tr[1]/td[2]/ul/li[8]/p/b')  {xml\_nodeset (1)}  [1] <b>45,750</b> |

이 중 텍스트 부분만을 추출하기 위해 html\_text() 함수를 추가적으로 사용하여 주면, price 변수에는 “45,750” 값이 저장됩니다. gsub() 함수를 통해 쉼표(,)를 없애주고, as.numeric() 함수를 사용하여 데이터를 45750 이라는 숫자 형태로 바꾸어 주도록 합니다. 그 후 price 변수를 data\_value 변수로 나누어 주면 현재 주가가 고려된 가치지표가 계산됩니다. 마지막으로 names() 함수를 통해 데이터들의 이름을 가치지표에 맞게 변경하여 줍니다.

|  |
| --- |
| > data\_value  PER PBR PCR PSR  8.439045 1.503600 5.612890 1.456363 |

이전 예제에서는 PER 값이 9.40이었던 반면, 새롭게 계산된 방법에서는 PER 값이 8.43으로 하락하였습니다. 이는 2017년 말일 (수정)주가가 50,960 이었던 반면, 2017년 8월 3일 기준 주가가 45,750으로 하락하였기 때문입니다.

일부 종목의 경우 재무데이터(EPS, BPS, CPS, SPS)가 0인 경우가 있으며, 주가를 0으로 나눌 경우 무한 값을 나타내는 Inf이 계산됩니다. is.infinite() 함수는 괄호 안의 값이 Inf 일 경우 True를 반환하는 함수이며, data\_value[is.infinite(data\_value)] = NA 코드를 통해 data\_value 변수에서 Inf에 해당하는 값을 NA로 변경하여 줍니다. 이후의 작업은 기존 예제와 동일하며, ‘티커\_value’ 이름의 csv 파일로 저장이 됩니다.

**4장. 데이터 정리하기**

앞장에서는 API와 크롤링을 통하여 퀀트 투자에 필요한 주가, 재무제표, 가치지표를 수집하는 방법에 대해 배웠습니다. 그러나 아래 표와 같이 각 종목 및 항목별로 나누어져 csv 파일로 저장되어 있습니다.

**[표] 각 항목별 파일 이름 및 세부 내역**

|  |  |  |
| --- | --- | --- |
| **파일 이름** | **데이터 항목** | **데이터 출처** |
| ticker.KS(KQ)\_price.csv | 수정주가 | 야후 파이낸스 API |
| ticker\_price.csv | 수정주가 | 다음 금융 크롤링 |
| ticker\_fs\_simple.csv | 재무제표 (요약) | NAVER 금융 크롤링 |
| ticker\_fs\_all.csv | 재무제표 (모든 항목) | NAVER 금융 크롤링 |
| ticker\_value.csv | 가치지표 | NAVER 금융 크롤링 |

이는 요리에 필요한 재료가 주방 여기저기에 흩어져 있는 것과 같습니다. 효율적으로 요리를 하기 위해서는 각 재료별로 준비를 해두어야 하듯이, 투자에 필요한 데이터들 역시 같은 항목끼리는 묶어서 저장해 둘 필요가 있습니다.

이번 장에서는 앞장에서 수집한 데이터들을 바탕으로 주가, 재무제표, 가치지표 별로 정리하는 과정을 살펴보도록 하겠습니다.

**[1] 주가 정리하기**

앞 장에서는 전 종목의 수정주가를 수집하는 두가지 방법에 대해 알아보았습니다. 각 종목 별로 ticker.KS(KQ)\_price.csv 혹은 ticker\_price.csv로 저장되어 있으며, for loop 구문을 이용하여 csv 파일을 불러온 후 데이터를 묶는 작업을 통해 하나의 파일로 합치는 방법에 대해 알아보도록 하겠습니다.

**야후 파이낸스를 통해 주가를 수집한 경우**

|  |
| --- |
| ticker = read.csv("KOR\_ticker.csv", row.names = 1)  library(quantmod)  library(magrittr)  price\_list = list()  for (i in 1 : nrow(ticker)) {    name = paste0(ticker[i, '종목코드'], ".", ticker[i, 'market'])  price\_list[[i]] = read.csv(paste0(name, "\_price.csv"), row.names = 1) %>%  as.xts()  } |

먼저 티커 리스트를 불러온 후, quantmod 패키지와 magrittr 패키지를 불러옵니다. 가격 데이터가 저장될 price\_list 변수를 리스트 형태로 만들어 준 후, for loop 구문을 통해 csv 파일을 순서대로 불러오도록 합니다.

먼저 i=1의 예를 들면, 티커, 콤마(.), 시장구분을 paste0() 함수를 이용하여 csv 파일 이름과 동일한 형태를 만들어 준 후 name 변수에 저장합니다. 그 후 read.csv() 함수를 통해 i번째 종목에 해당하는 csv 파일을 읽어온 후, as.xts() 함수를 통해 xts 형태로 변경해줍니다. 그 후 해당 데이터를 price\_list변수의 i번째 리스트에 저장해줍니다.

for loop 과정이 끝나면 price\_list의 i번째 리스트에는 i번째 종목의 종가가 입력되어 있습니다. 이를 do.call() 함수 내에서 cbind() 함수를 이용하여 열의 형태로 결합합니다. xts 형식의 데이터를 열의 방향으로 결합할 경우 SQL의 full outer join과 같은 방식으로 데이터를 결합합니다. 즉, 데이터 내 존재하는 모든 인덱스를 나열한 후 각 인덱스에 해당하는 값들을 찾아 넣으며, 해당하지 않는 부분의 값은 NA로 표시합니다. 이에 대한 간단한 예제는 다음과 같습니다.

|  |
| --- |
| > x = xts(c(1,2,4), order.by = as.Date(c("2018-01-01", "2018-01-02", "2018-01-04")))  > y = xts(c(2,3,4), order.by = as.Date(c("2018-01-02", "2018-01-03", "2018-01-04")))  > z = xts(c(3,4), order.by = as.Date(c("2018-01-03", "2018-01-04")))  > cbind(x,y,z)  ..1 ..2 ..3  2018-01-01 1 NA NA  2018-01-02 2 2 NA  2018-01-03 NA 3 3  2018-01-04 4 4 4 |

x에는 1,2,4일, y에는 1,3,4일, z에는 3,4일에 각각 데이터가 존재합니다. cbind() 함수를 통해 해당 데이터들을 열의 형태로 결합할 경우, 모든 인덱스인 1,2,3,4일을 나열한 후 해당 인덱스에 맞게 데이터를 찾거나 NA를 표시합니다.

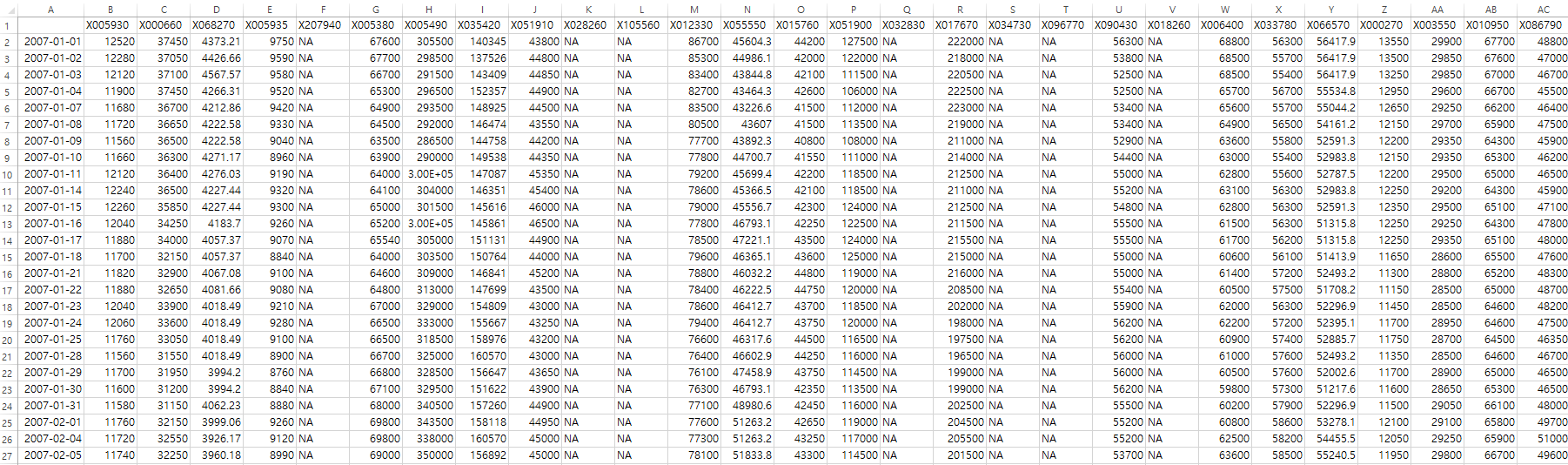
이전에 데이터를 다운로드 받는 과정에서 일부 종목에서 결측치가 발생하였다는 경고 메시지가 있었습니다. 예를 들어 야후 파이낸스 API를 이용할 경우 일부 종목은 몇 일 혹은 몇 달간의 데이터가 비는 경우가 발생하며, cbind() 함수를 통해 데이터를 결합할 경우 해당 기간의 주가는 NA로 나타납니다. 이러한 데이터 손실을 보정하고자 na.locf() 함수를 사용하도록 합니다. 해당 함수는 NA값을 직전의 값으로 대체하는 함수입니다. 간단한 예제는 다음과 같습니다.

|  |
| --- |
| x = c(1,3,NA,4)  na.locf(x)  > na.locf(x)  [1] 1 3 3 4 |

x 변수의 세번째 데이터에는 NA라는 결측치가 존재하며, na.locf() 함수를 사용할 경우 NA 값은 그 이전값인 3으로 대체됩니다. 이를 위의 예제에 적용하면, NA가 존재하는 일자의 주가는 직전일 혹은 NA가 존재하기 이전의 주가로 대체됩니다.

이후 열이름을 ticker의 종목코드 열에 해당하는 값으로 변경하여 줍니다. 마지막으로 데이터 프레임 형식으로 변경하여 준 후, ‘KOR\_price.csv’ 파일로 저장해주면 주가 정리작업이 완료됩니다. 해당 파일을 확인해보면, 각 일자별로 전 종목의 주가가 정리되어 있음이 확인됩니다.

**[그림] 주가 정리 파일**



**다음 금융을 통해 주가를 수집한 경우**

다음 금융의 테이블을 크롤링하여 주가를 수집한 경우도, 위와 거의 유사한 코드로 주가를 정리할 수 있습니다.

|  |
| --- |
| ticker = read.csv("KOR\_ticker.csv", row.names = 1)  price\_list = list()  for (i in 1 : nrow(ticker)) {    name = ticker[i, '종목코드'] %>% as.character()  price\_list[[i]] = read.csv(paste0(name, "\_price.csv"), row.names = 1) %>%  as.xts()    }  price\_list = do.call(cbind, price\_list)  price\_list = na.locf(price\_list)  colnames(price\_list) = ticker$종목코드  write.csv(data.frame(price\_list), "KOR\_price.csv") |

기존 예제와 대부분 동일하며 name 부분의 코드만 변경이 됩니다. name에 해당하는 부분이 기존과 달리 ticker의 종목코드에 해당 하는 부분만을 불러옵니다. 이는 크롤링으로 주가를 구한 경우 ‘ticker\_price.csv’의 형태로 파일을 저장하였기 때문입니다.

이를 제외하고는 이전 예제와 코드가 일치하며, 역시나 전 종목의 주가가 정리된 csv 파일을 저장합니다.

**[2] 재무제표 정리하기**

NAVER 금융을 이용하여 요약 재무제표 및 재무분석 항목 중 포괄손익계산서, 재무상태표, 현금흐름표의 모든 항목을 수집하는 방법에 대해 알아보았습니다. 그 중 모든 항목을 수집한 데이터를 바탕으로, 각 항목별로 정리하는 방법에 대해 알아보도록 하겠습니다. 요약 재무제표의 csv 파일 이름에 해당하는 부분만 변경하면, 항목 별 정리가 가능합니다.

**Company Wise를 통헤 재무제표 수집한 경우**

|  |
| --- |
| ticker = read.csv("KOR\_ticker.csv", row.names = 1)  library(magrittr)  library(data.table)  data\_csv = list()  for (i in 1 : nrow(ticker)) {    name = ticker[i, '종목코드'] %>% as.character()  data\_csv[[i]] = read.csv(paste0(name, "\_fs\_all.csv"), row.names = 1)  } |

먼저 티커 리스트를 불러온 후, magrittr 패키지와 data.table 패키지를 불러옵니다. 불러온 csv 파일의 데이터들이 저장될 data\_csv 변수를 리스트 형태로 만들어 준 후, for loop 구문을 통해 csv 파일을 순서대로 불러오도록 합니다.

|  |
| --- |
| > head(data\_csv[[1]])  X2013 X2014 X2015 X2016 X2017  매출액(수익) 2286927 2062059.9 2006534.8 2018667 2395754  \*내수 NA 207289.2 208278.2 NA NA  \*수출 NA 1854770.7 1798256.6 NA NA  ....제품매출액 NA NA NA NA NA  ....상품매출액 NA NA NA NA NA  ....제품상품매출액 NA NA NA NA NA  > head(data\_csv[[4]])  x  1 NA |

재무제표가 존재하는 종목의 경우 해당 항목이 데이터 프레임 형태로 저장이 됩니다. 반면에 삼성전자 우선주와 같이 Company Wise 홈페이지 내에서 재무제표가 존재하지 않는 종목의 경우리스트 내에 빈 데이터가 저장이 됩니다.

|  |
| --- |
| item = data\_csv[[1]] %>% rownames()  fs\_list = list()  for (i in 1 : length(item)) {  fs\_list[[i]] = lapply(data\_csv, function(x) {  if ( item[i] %in% rownames(x) ) {  x[which(rownames(x) == item[i]),]  } else {  matrix(NA, 1, 5) %>% data.frame()  }  })  } |

그 후 기준점이 될 재무제표 항목을 설정하며, 가장 대표적인 삼성전자의 항목을 기준으로 삼습니다. 이는 요약 재무제표의 경우 계정 항목이 모든 종목에서 동일하지만, 전 항목을 크롤링한 경우 주식의 섹터별로 재무제표 항목이 상이하므로 특정 종목에 맞추어 데이터를 정리할 필요가 있기 때문입니다.

data\_csv 변수의 첫번째 리스트는 삼성전자 종목의 데이터이며, 이 중 행이름 즉 재무제표 항목을 item 변수에 저장합니다. 그 후 각 재무제표 항목이 순서대로 정리될 fs\_list 변수를 리스트 형태로 만들어 줍니다.

다음의 for loop 과정은 전 종목을 대상으로 각 계정 항목과 일치하는 데이터들을 찾은 후, 이를 계정 항목별 리스트에 저장하는 단계입니다. 먼저 i=1의 예시입니다. csv 데이터가 저장된 data\_csv 변수를 대상으로 lapply() 함수를 실행합니다.

lapply() 함수는 기존에 살펴본 sapply() 함수와 거의 동일하며, 결과값이 벡터 형태가 아닌 리스트 형태로 반환됩니다. 구조는 sapply() 함수와 동일하게 lapply (X, FUN, ..) 구조로 이루어지며 간단한 예제는 다음과 같습니다.

|  |
| --- |
| > sapply(num, function(x) {x^2})  [1] 1 4 9  > lapply(num, function(x) {x^2})  [[1]]  [1] 1  [[2]]  [1] 4  [[3]]  [1] 9 |

num 변수에 해당하는 값들의 제곱이 sapply() 함수를 이용할 경우 벡터의 형태로 반환되었지만, lapply() 함수를 이용할 경우 리스트의 형태로 반환됨이 확인됩니다.

다시 코드로 돌아가면, function 부분은 다음과 같습니다. a %in% b 오퍼레이터는 a값이 b에 존재할 경우 TRUE, 아닐 경우 FALSE를 반환합니다. 따라서 item[1]인 ‘매출액(수익)’이 행이름에 존재한다면 item[i] %in% rownames(x) 코드는 TRUE를 반환하며, 행이름이 해당 항목인 부분의 데이터를 찾은 후 반환하도록 합니다.

그러나 티커에 해당하는 페이지가 없어 csv 파일에 데이터가 없는 종목, 혹은 계정 항목 중 ‘매출액(수익)’에 해당하는 값이 없는 종목의 경우는 NA로 이루어진 데이터 프레임을 반환합니다. 빈 데이터 프레임을 만드는 과정은 다음과 같습니다.

|  |
| --- |
| > matrix(NA, 1, 5) %>% data.frame()  X1 X2 X3 X4 X5  1 NA NA NA NA NA |

먼저 NA로 이루어진 1행, 5열의 매트릭스를 만들어 준 후, 이를 데이터 프레임 형식으로 바꾸어줍니다. 이를 통해 재무제표 데이터가 없는 종목의 경우 해당 항목이 NA로 처리되도록 해줍니다.

해당 과정을 거치면 fs\_list 변수의 첫번째 리스트에는 첫번째 계정 항목명에 해당하는 전종목의 데이터가 리스트의 형태로, 두번째 리스트에는 두번째 계정 항목명에 해당하는 전종목의 데이터가 리스트의 형태로 차례대로 쌓이게 됩니다. 즉, 계정 항목명(Company Wise의 경우 33개)의 개수만큼 리스트가 생성되고, 각 계정 항목명에는 이에 해당하는 전종목들의 데이터가 리스트의 형태로 저장되어 있습니다.

|  |
| --- |
| > head(fs\_list[[1]], 2)  [[1]]  X2013 X2014 X2015 X2016 X2017  매출액(수익) 2286927 2062060 2006535 2018667 2395754  [[2]]  X2013 X2014 X2015 X2016 X2017  매출액(수익) 141651 171255.7 187980 171979.8 301094.3  > head(fs\_list[[25]], 2)  [[1]]  X2013 X2014 X2015 X2016 X2017  매출총이익 909963.6 779271.9 771713.6 815890.3 1102847  [[2]]  X2013 X2014 X2015 X2016 X2017  매출총이익 53005.15 76638.41 82826.45 64108.36 174075.9 |

fs\_list의 첫번째 리스트에는 매출액에 해당하는 정보들이 각 종목별 순서대로 리스트의 형태로 저장되어 있습니다. 스물 다섯번째 리스트에는 매출총이익에 해당하는 정보들이 저장되어 있습니다.

|  |
| --- |
| fs\_list = lapply(fs\_list, function(x) {rbindlist(x) %>% data.frame()}) |

계정 항목별 정리가 완료되었으므로, 각 항목의 하부 리스트를 데이터 프레임 형식으로 묶어주도록 합니다. fs\_list 변수를 대상으로 lapply() 함수를 적용하며, function 부분에는 data.table 패키지의 rbindlist() 함수를 이용[[21]](#endnote-21)해 줍니다. 해당 함수는 데이터 테이블, 데이터 프레임 혹은 리스트 형식의 데이터를 매우 빠르게 행의 형태인 데이터 테이블로 묶어주는 함수입니다. 그 후 데이터 테이블 형태를 데이터 프레임 형태로 바꾸기 위해, data.frame() 함수를 사용해 줍니다.

|  |
| --- |
| > head(fs\_list[[1]])  X2013 X2014 X2015 X2016 X2017  1 2286926.6700 2062059.870 2006534.8200 2018667.45 2395753.760  2 141651.0200 171255.660 187979.9800 171979.75 301094.340  3 2262.0787 4710.455 6034.1277 6705.81 9490.800  4 NA NA NA NA NA  5 437.4701 1053.990 912.7803 2946.22 4646.293  6 618646.4986 650984.453 581923.4456 530835.13 606550.996 |

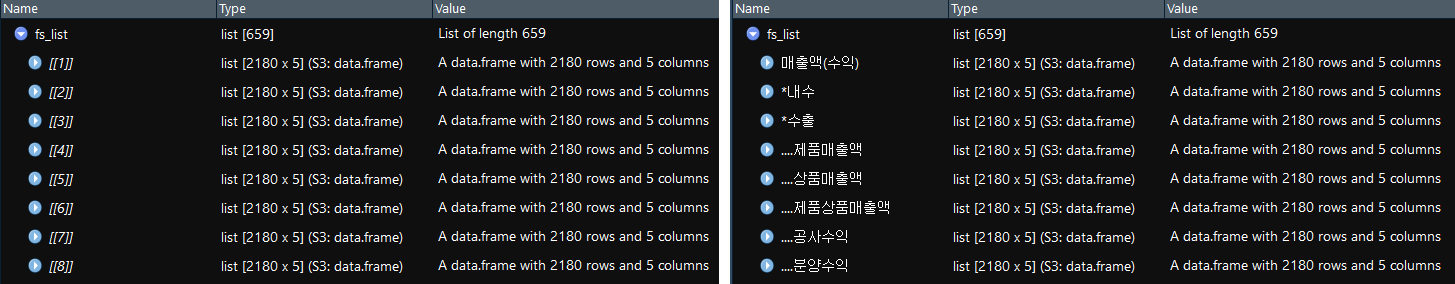
매출액에 해당하는 첫번째 리스트를 살펴보면, 리스트의 형태로 저장되어 있던 종목 별 데이터들이 데이터 프레임의 형태로 묶여져 있으며, 데이터가 존재하지 않는 종목은 NA로 처리되어 있습니다.

|  |
| --- |
| fs\_list = lapply(fs\_list, function(x) {  rownames(x) = ticker[,'종목코드'] %>% as.character()  return(x)  })  names(fs\_list) = item |

각 데이터 프레임의 행이름과, fs\_list의 이름을 변경해주도록 합니다. 위의 예를 살펴보면 데이터 프레임의 행이름이 1,2,3과 같이 단순 숫자의 형태로 되어 있습니다. lapply() 함수를 통해 행이름을 6자리 종목코드로 변경되도록 하며, return(x)를 통해 rownames(x)가 아닌 x 자체가 반환되도록 합니다.

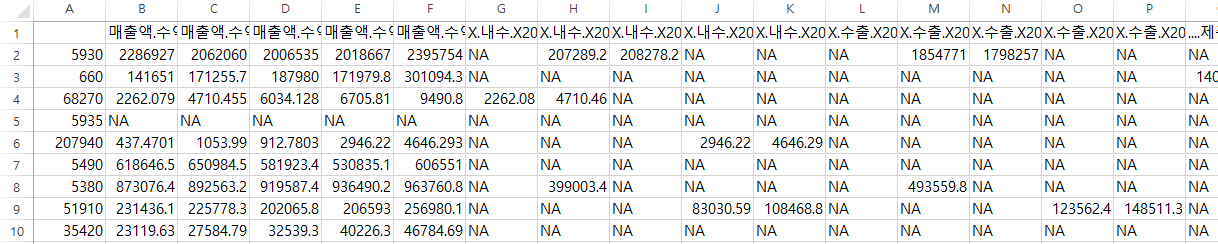
또한 fs\_list의 각 리스트 이름을 숫자가 아닌 계정 항목으로 변경해주도록 합니다. names(fs\_list) 코드를 통해 리스트의 이름을 item, 즉 계정 항목으로 변경해 줍니다. 아래 그림의 좌측이 변경 전 fs\_list 변수이며, 우측이 변경 후 fs\_list 변수입니다. 변경 후에는 각 리스트에 해당하는 정보가 어떠한 계정 항목인지 쉽게 확인할 수 있습니다.

**[그림] 리스트의 이름 변경 전과 후**



마지막으로 정리된 재무제표 데이터를 저장해주면 됩니다. 해당 데이터의 경우 리스트 형태로 저장되어 있으므로, 이전 예제들의 경우처럼 csv 파일로 저장할 경우 자동으로 열의 형태로 합쳐지며 데이터 형태가 깨지게 됩니다.

**[그림] 리스트 형태 데이터를 csv 파일로 저장했을 때의 문제**

****

따라서 리스트 형태의 데이터는 csv 파일로 저장하기 보다는 Rds 파일로 저장하는 것이 바람직 합니다.

|  |
| --- |
| saveRDS(fs\_list, "KOR\_fs.Rds")  fs = readRDS("KOR\_fs.Rds") |

saveRDS() 함수를 이용하면 R의 데이터를 손쉽게 Rds 형식으로 저장할 수 있습니다. 정리가 완료된 fs\_list 변수를 ‘KOR\_fs.Rds’ 파일로 저장하도록 합니다. 해당 파일이 저장된 폴더에서 파일을 더블 클릭한 후, 연결 프로그램을 RStudio로 설정해주면 변수를 입력한 후 파일을 불러올 수 있습니다. 혹은 readRDS() 함수를 이용하여 RDS 파일을 읽어올 수도 있습니다.

**[3] 가치지표 정리하기**

가치지표의 경우 페이지에 보이는 값을 그대로 사용하기 보다는, 현재 주가를 고려하여 가치지표를 계산하는 것이 훨씬 바람직함을 보았습니다. 가치지표 정리하는 방법의 경우 이전의 재무제표를 정리하는 방법과 거의 같으며, 몇줄의 코드만 변경해주면 됩니다.

|  |
| --- |
| ticker = read.csv("KOR\_ticker.csv", row.names = 1)  library(magrittr)  data\_csv = list()  for (i in 1 : nrow(ticker)) {    name = ticker[i, '종목코드'] %>% as.character()  data\_csv[[i]] = read.csv(paste0(name, "\_value.csv"), row.names = 1)    } |

티커 리스트를 불러온 후, magrittr 패키지를 불러옵니다. 재무제표 정리의 경우와 동일하게 csv 파일의 데이터들이 저장될 data\_csv 변수를 만들어 준 후, for loop 구문을 통해 파일을 순서대로 불러온 후 저장합니다.

|  |
| --- |
| > data\_csv[[1]]  x  PER 8.374484  PBR 1.492097  PCR 5.569950  PSR 1.445221  > data\_csv[[4]]  x  PER NA  PBR NA  PCR NA  PSR NA  > data\_csv[[2150]]  x  PER 131.531724  PBR 1.047968  PCR -173.908163  PSR NA |

보통의 경우 가치지표가 데이터 프레임 형태로 저장이 되어있습니다. 그러나 페이지가 존재하지 않는 경우는 모든 데이터가 NA로, 특정 가치지표를 계산할 수 없는 경우 해당값만 NA로 계산이 되어 있습니다.

|  |
| --- |
| item = data\_csv[[1]] %>% rownames()  value\_list = list()  temp\_data = c()  for (i in 1 : length(item)) {  value\_list[[i]] = lapply(data\_csv, function(x) {  if ( item[i] %in% rownames(x) ) {  x[which(rownames(x) == item[i]),]  } else {  **NA**  }  })  } |

이후 각 항목(PER, PBR, PCR, PSR)에 해당하는 값이 존재할 경우 해당 데이터를, 그렇지 않을 경우 NA 데이터를 반환합니다. 기존 재무제표의 정리에서는 NA로 이루어진 데이터 프레임을 반환한 반면, 이번 경우에는 하나의 NA 값만을 반환하는 것이 차이입니다.

해당 과정을 거치면 value\_list에는 총 네개의 리스트가 생성되며, 첫번째 리스트에는 전 종목의 PER 값이, 두번째 리스트에는 PBR 값이, 세번째 리스트에는 PCR 값이, 마지막 리스트에는 PSR 값이, 리스트 형식으로 저장되어 있습니다.

|  |
| --- |
| value\_list = lapply(value\_list, function(x) {do.call(rbind, x)}) |

각 항목의 하부 리스트 값들을 do.call() 함수와 rbind() 함수를 이용하여 행의 형태로 묶어줍니다. 해당 과정이 끝나면, value\_list 변수의 리스트에는 각 가치지표에 해당하는 값들이 매트릭스의 형태로 구성되어 있습니다

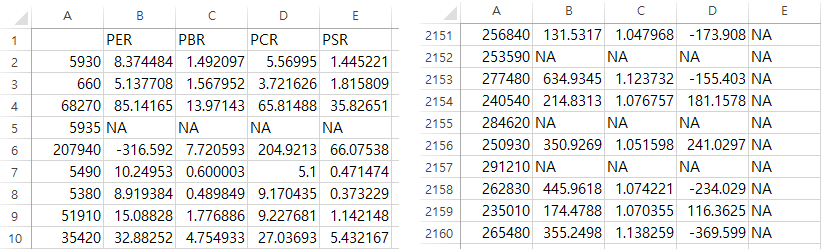
|  |
| --- |
| value\_list = do.call(cbind, value\_list) %>% data.frame()  rownames(value\_list) = ticker[, '종목코드']  colnames(value\_list) = item  write.csv(value\_list, "KOR\_value.csv") |

마지막으로 do.call() 함수와 cbind() 함수를 이용해 열의 형태로 묶은 후, data.frame() 함수를 이용하여 데이터 프레임 형식의 데이터로 만들어 줍니다. 그 후 행이름에는 각 종목들의 티커는, 열이름에는 각 가치지표의 항목을 설정하여 줍니다.

|  |
| --- |
| > head(value\_list)  PER PBR PCR PSR  005930 8.374484 1.4920973 5.569950 1.4452213  000660 5.137708 1.5679520 3.721626 1.8158089  068270 85.141654 13.9714268 65.814880 35.8265099  005935 NA NA NA NA  207940 -316.591627 7.7205933 204.921322 66.0753804  005490 10.249533 0.6000033 5.100000 0.4714737 |

정리된 테이블을 ‘KOR\_value.csv’ 파일로 저장하도록 합니다. 해당 데이터는 리스트 형태가 아니므로, csv 파일로 저장하여도 무방합니다.

**[그림] 가치지표 정리 파일**



**5장. 미국 데이터 다운로드 및 정리하기**

몇 년전까지만 해도 해외 주식시장에 투자를 한다는 것은 일부 고액자산가의 전유물처럼 여겨졌으며, 일부 일반투자자의 경우도 펀드를 통해 투자를 하는 경우가 대부분이었습니다. 그러나 국내 코스피가 2012년부터 2016년까지 박스권을 형성하던 것과는 대조적으로 미국 시장의 끊임없는 상승과 FAANG[[22]](#endnote-22) 주식의 폭발적인 성장은, 일반투자자들로 하여금 해외주식 직접 투자에 대한 관심을 불러 일으켰습니다.

아래 그림[[23]](#endnote-23)은 2012년 이후 국내 투자자의 해외주식 거래규모 추이를 나타낸 그래프입니다. 2017년 해외주식 거래규모는 전년대비 83%나 증가했으며, 최근 5년간은 무려 6배 이상 증가하였습니다.

**[그림] 해외주식 거래규모 추이**

그러나 투자자들의 관심에 비해 투자 행태는 아직까지 매우 단순한 것이 사실입니다. 해외에 상장된 ETF를 사거나 FAANG, Nvidia, Microsoft 등 대형주를 사는 경우가 대부분입니다. 이는 국내 주식과는 달리 해외 주식의 경우 일일이 뉴스를 접하기도 힘들며, 언어가 다르다는 문제에 직면하기 때문입니다. 또한 초대형주의 경우 그나마 어떠한 영업을 하는 기업인지 쉽게 알 수 있지만, 해외 중소형주 기업의 영업 활동을 일일이 조사하는 것은 사실상 불가능에 가깝습니다.

이러한 현실적인 문제에서 퀀트 투자가 바로 해답이 될 수 있습니다. 퀀트 투자는 개별 기업에 대한 분석과 예측이 아닌, 기업에 관련된 여러 데이터들을 고려하고 모형을 기반으로 투자하는 방법입니다. 따라서 우리가 잘 모르는 기업이나 국가를 대상으로도 데이터만 존재한다면 얼마든지 투자가 가능합니다.

그 중 미국 주식을 대상으로한 퀀트 투자는 매우 매력적이면서도 효율적입니다. 아래 표[[24]](#endnote-24)는 2017년 기준 세계 증권거래소의 시가총액 비중을 나타낸 것입니다. 한국의 경우 전 세계 시가총액에서 차지하는 비중이 겨우 2.1%에 불과하지만, 미국의 뉴욕증권거래소(NYSE)와 나스닥(NASDAQ)을 합친 비중은 무려 37.7%를 차지합니다. 미국 주식은 세계 주식 시장의 바로미터 역할을 하므로, 이곳에 투자한다는 것 만으로 세계 시장에 투자하는 역할을 합니다. 시장이 큰 만큼 이를 대상으로 하는 서비스 업체들도 다양하여, 퀀트 투자를 하기 위한 데이터 수집에도 매우 용이합니다.

**[표] 글로벌 거래소 시가총액 비중**

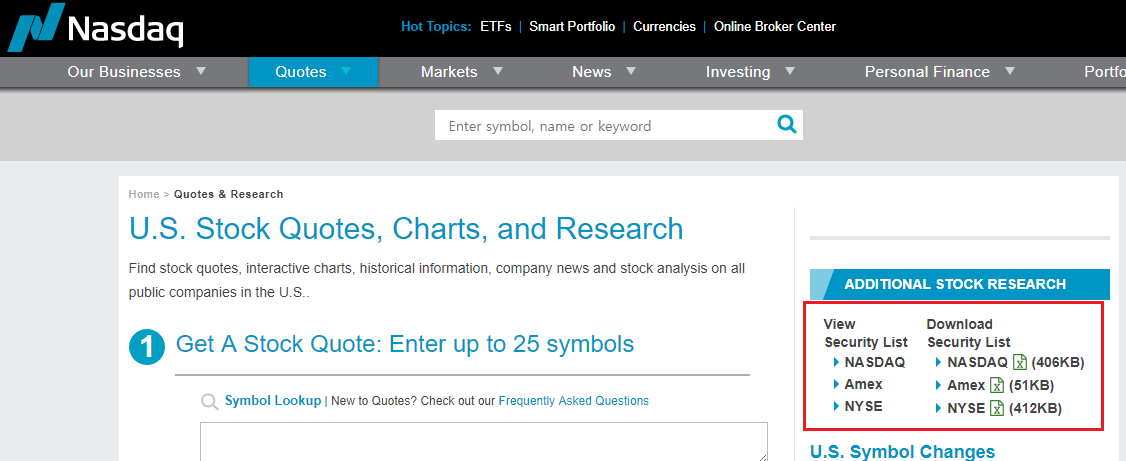
미국의 주식 거래시장 미국뿐 아니라 전세계에서 가장 큰 주식거래 시장인 뉴욕증권거래소(NYSE), 기술주 혹은 정보 산업주 위주로 상장되는 나스닥(NASDAQ), 이 두군데의 상장조건이 맞지 않는 소규모 업체들이 주로 상장하는 아멕스(AMEX), 이렇게 총 세가지로 나누어집니다.

해당 거래소에 상장된 종목들의 주가, 재무제표, 가치지표 등의 데이터를 다운로드 받는다면, 퀀트 투자 모형을 그대로 적용하여 미국 투자를 위한 포트폴리오를 만들 수도 있습니다. 즉 언어나 국가에 대한 장벽이 없이 미국 대형주뿐만 아니라 중소형주를 대상으로도 얼마든지 투자가 가능하며, 이러한 데이터 역시 API 혹은 크롤링을 통해 매우 쉽게 수집이 가능합니다. 미국이 아닌 유럽, 일본, 중국 등 전세계 모든 국가에 대한 투자도 데이터를 구할 수만 있다면 동일한 투자 방법으로 얼마든지 투자가 가능하다는 점, 그것이 바로 퀀트 투자의 매력입니다.

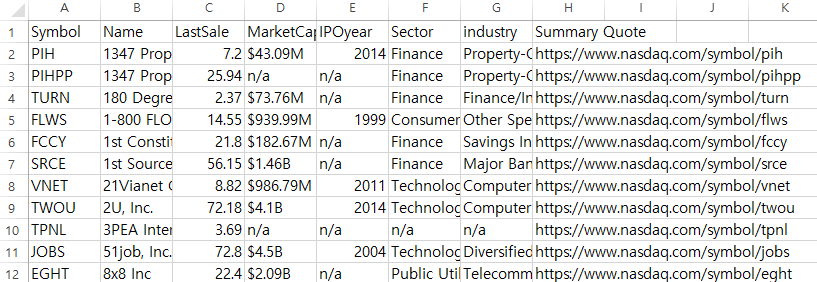
**[1] 주식티커 다운로드**

먼저 나스닥 홈페이지(<https://www.nasdaq.com/>)에 접속하여 Quotes 항목을 선택합니다. 우측에 NASDAQ, AMEX, NYSE의 상장종목을 확인할 수 있는 탭이 보이며, Download Security List의 파일을 클릭할 경우 csv 파일이 다운로드 되며, 각 거래소의 상장종목 리스트를 확인할 수 있습니다.

**[그림] 나스닥 홈페이지의 Quote 항목**

****

**[그림] 나스닥 시장의 상장종목 현황**

****

NASDAQ, AMEX, NYSE에 해당하는 csv 파일들을 다운로드 받은 후, 이를 R에서 읽어오는 작업을 할 수도 있지만 이는 매우 번거로운 작업입니다. 해당 파일의 url을 이용하여 R 내에서 직접 csv 파일을 다운로드 받은 후, 하나의 파일로 묶는 것이 훨씬 효율적인 작업니다.

|  |
| --- |
| url\_NYSE =  "http://www.nasdaq.com/screening/companies-by-name.aspx?letter=0&exchange=nyse&render=download"  url\_NASDAQ =  "http://www.nasdaq.com/screening/companies-by-name.aspx?letter=0&exchange=nasdaq&render=download"  url\_AMEX =  "http://www.nasdaq.com/screening/companies-by-name.aspx?letter=0&exchange=amexe&render=download"  download.file(url\_NYSE, destfile = "./url\_NYSE.csv")  download.file(url\_NASDAQ, destfile = "./url\_NASDAQ.csv")  download.file(url\_AMEX, destfile = "./url\_AMEX.csv")  NYSE = read.csv("./url\_NYSE.csv", stringsAsFactors = F)  NASDAQ = read.csv("./url\_NASDAQ.csv", stringsAsFactors = F)  AMEX = read.csv("./url\_AMEX.csv", stringsAsFactors = F)  us.ticker = rbind(NYSE, NASDAQ, AMEX)  us.ticker = us.ticker[us.ticker$MarketCap != "n/a", ]  us.ticker = us.ticker[us.ticker$Sector != "n/a", ]  us.ticker = us.ticker[!duplicated(us.ticker$Name), ]  us.ticker$Symbol = gsub(" ", "", us.ticker$Symbol)  rownames(us.ticker) = NULL  write.csv(us.ticker, "US\_ticker.csv")  file.remove("./url\_NYSE.csv")  file.remove("./url\_NASDAQ.csv")  file.remove("./url\_AMEX.csv") |

먼저 url\_NYSE, url\_NADAQ, url\_AMEX 변수에, 각각의 상장종목현황 csv의 url을 입력하도록 합니다. 그 후 인터넷으로부터 파일을 다운로드 받는 함수인 download.file()을 이용하여 csv 파일을 다운로드 받도록 합니다. read.csv() 함수를 이용하여 다운로드 받은 csv 파일들을 불러오며, 데이터가 요인형Factor이 아닌 문자형 그대로 사용하기 위해 stringsAsFactors 인자를 FALSE로 지정합니다. 불러온 데이터를 rbind() 함수를 이용해 행의 형태로 묶은 후, us.ticker 변수에 저장합니다.

us.ticker 변수에는 10,000 여개 종목의 정보가 저장되어 있지만, 일부 데이터를 클랜징 할 필요가 있습니다. 먼저, MarketCap(시가총액) 혹은 Sector(섹터)가 ‘n/a’로 표시되어 있는 종목의 경우 펀드인 경우가 대부분입니다. 따라서 해당 열이 ‘n/a’가 아닌 행들만 선택하여 저장하도록 합니다.

미국의 경우 동일한 종목이 여러 클래스로 상장되는 경우가 있으며, 구글의 모회사인 알파벳과 워런 버핏의 버크셔 해서웨이 주식이 대표적인 예입니다. 또한 우리나라 주식시장과는 다르게 NYSE와 NASDAQ에 동시 상장된 종목도 존재합니다. 이를 정리해 주기 위해 !duplicated() 함수를 이용하여 Name(이름)이 서로 다른 행들만 선택하여 다시 저장하도록 합니다. 해당 작업을 마치면 종목 수는, 처음의 절반 정도인 5천여개로 줄어들게 됩니다.

일부 종목의 Symbol(티커)에 “ “ 처럼 공백이 들어가는 경우가 있으며, gsub() 함수를 통해 공백 부분을 없애주도록 합니다. 마지막으로 행이름을 NULL로 설정하여 초기화 시켜주도록 합니다. 데이터 클랜징 작업이 완료된 후, ‘US\_ticker.csv’ 이름으로 데이터를 저장해주도록 하며, file.remove() 함수를 이용하여 다운로드 받았던 세개의 csv 파일도 삭제해주도록 합니다.

**[2] API 이용한 수정주가 다운로드**

미국 종목의 경우 quantmod 패키지의 getSymbols() 함수를 이용하여 매우 손쉽게 주가를 다운로드 받을 수 있습니다. 더군다가 국내 종목과는 달리 배당이 포함된 수정주가 데이터도 매우 안정적으로 제공이 됩니다.

|  |
| --- |
| ticker = read.csv("US\_ticker.csv", row.names = 1)  library(quantmod)  library(magrittr)  for (i in 1 : nrow(ticker)) {  name = ticker[i, 'Symbol'] %>% as.character  price = xts(NA, order.by = Sys.Date())    tryCatch({  price = Ad(getSymbols(name, auto.assign = FALSE))  }, error = function(e) {  print(paste0("Error in Ticker: ", name))  })    price = price[!duplicated(index(price))]    write.csv(data.frame(price), paste0(name, "\_price.csv"))  print(c(name, i / nrow(ticker)))  Sys.sleep(3)  } |

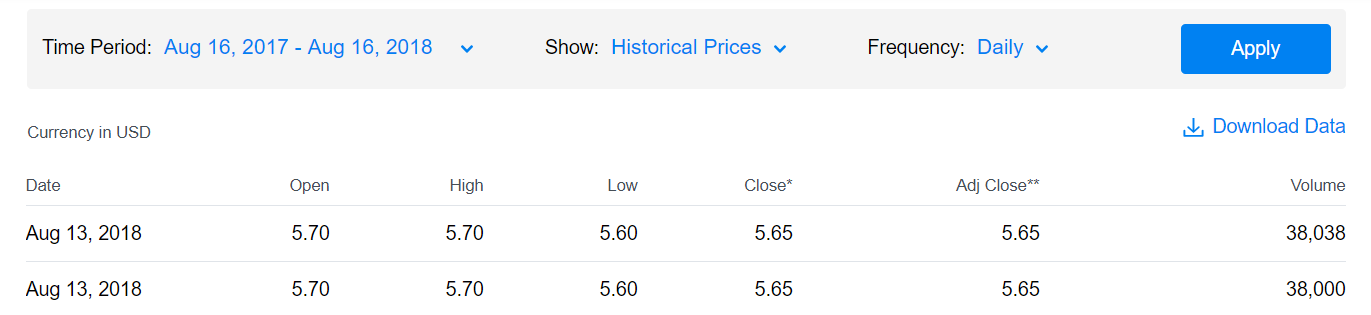
먼저 전 챕터에서 만든 미국 상장 전종목의 티커를 불러온 후, quantmod와 magrittr 패키지를 불러옵니다. for loop 구문에서 i=1인 예제를 살펴보면, 종목의 티커가 저장된 ‘Symbol’ 열을 선택한 후 문자열 형태로 변환한 후 name 변수에 저장해줍니다. 미국 종목의 경우 국내와 다르게 ‘.KS’ 혹은 ‘.KQ’를 붙이지 않아도 됩니다.

price 변수에 NA값으로만 이루어진 xts 데이터를 생성한 후, getSymbols() 함수에 name을 입력하여 주가 데이터를 다운로드 받습니다. 그 후, Ad() 함수를 통해 배당이 반영된 수정주가 만을 선택합니다. 이를 통해 price 변수에는 Adjusted 값만이 저장되게 됩니다.

|  |
| --- |
| DDD.Adjusted  2007-01-03 5.070000  2007-01-04 4.926667  2007-01-05 4.833333  2007-01-08 4.846667  2007-01-09 4.860000  2007-01-10 4.986667 |

tryCatch() 함수를 통해 오류가 발생했을 시에는 해당 종목의 티커를 출력합니다. 일부 종목의 경우 (시간외 거래량 반영 등으로 인해) 동일 일자의 주가가 두번 나오는 현상이 발생합니다. 이 중 중복된 일자 중 하나의 데이터만 선택하기 위해 !duplicated() 코드를 이용하여 줍니다.

**[그림] 동일 일자 데이터가 중복으로 존재하는 경우**



price 데이터를 데이터 프레임 형태로 변경 및 ‘티커\_price.csv’ 파일을 저장한 후, 마지막으로 다운로드 진행 상황을 출력합니다. 해당 과정이 끝나면 다음 loop가 진행되기 전 3초간 프로세스 지연을 주도록 합니다.

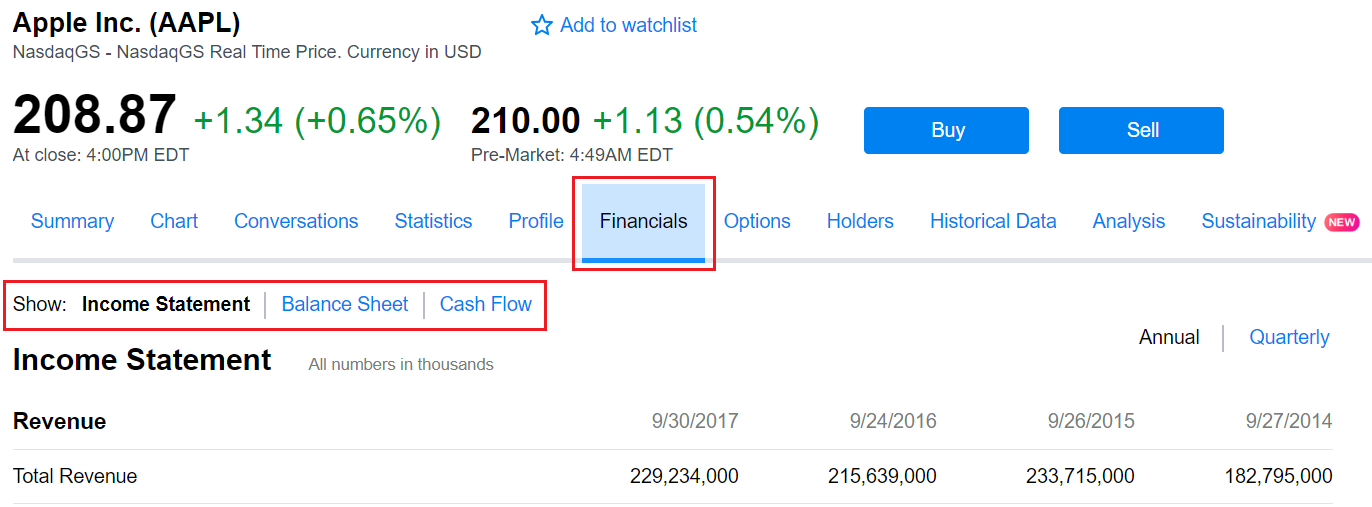
**[3] 재무제표 및 가치지표 다운로드**

미국 종목의 재무제표 및 가치지표는 야후 파이낸스, 마켓워치, CNN Money 등 다양한 금융 정보 사이트에서 제공하는 데이터를 크롤링하여 수집할 수 있습니다. 더욱이 가치지표는 quantmod 패키지의 getQuote() 함수를 통해 매우 쉽게 수집이 가능합니다.

**재무제표 다운로드**

먼저 야후 파이낸스(<https://finance.yahoo.com/>)에 접속하여 애플의 티커인 AAPL을 입력한 후, Financials 항목을 선택합니다. 해당 항목에는 재무제표의 각 항목인 Income Statement(포괄손익계산서), Balance Sheet(재무상태표), Cash Flow(현금흐름표) 항목이 존재합니다.

**[그림] 야후 파이낸스의 재무제표 항목**

****

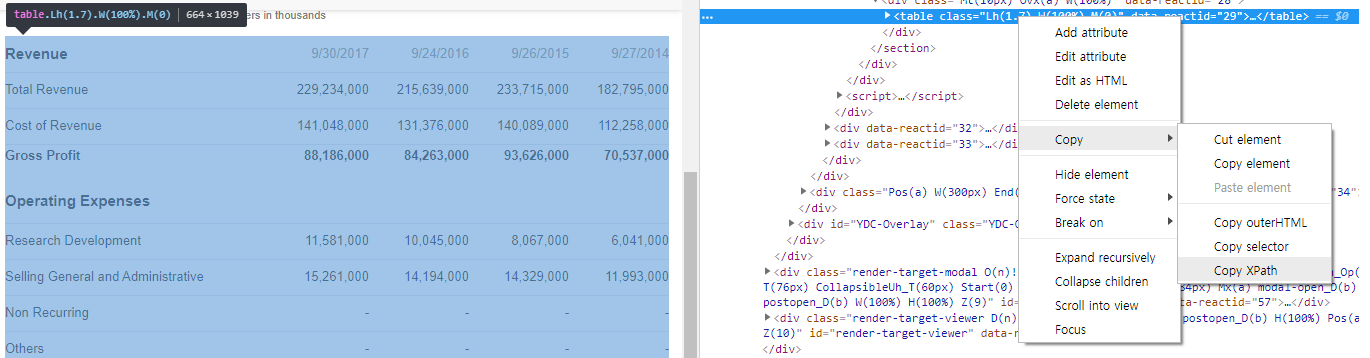
또한 각 페이지의 url을 살펴보면 GET 방식으로 데이터를 요청함을 알 수 있으며, 티커에 해당하는 AAPL 부분만 변경하여 원하는 종목의 재무제표 항목 페이지에 접속할 수 있습니다.

**[표] 야후 파이낸스 재무제표 항목의 url**

|  |  |
| --- | --- |
| **항목** | **url** |
| Income Statement | https://finance.yahoo.com/quote/AAPL/financials?p=AAPL |
| Balance Sheet | https://finance.yahoo.com/quote/AAPL/balance-sheet?p=AAPL |
| Cash Flow | https://finance.yahoo.com/quote/AAPL/cash-flow?p=AAPL |

각 항목 페이지의 재무제표 부분만 크롤링 하기 위해, 테이블에 해당하는 부분의 Xpath를 확인합니다. 테이블에 마우스를 둔 채 우클릭 후 검사 버튼을 눌러 개발자도구 화면을 활성화 한 후, HTML 문서 중 전체 테이블이 음영처리되는 부분을 찾도록 합니다. <table class="Lh(1.7) W(100%) M(0)" data-reactid="29"> 부분이 재무제표 테이블에 해당하는 부분임이 확인되며, 우클릭 → Copy → Copy XPath를 통해 XPath값을 복사하도록 합니다.

**[그림] 재무제표 테이블 항목 XPath 확인**



Income Statement, Balance Sheet, Cash Flow 각 페이지를 확인해보면 모두 다음의 동일한 XPath 값을 가짐이 확인됩니다.

|  |
| --- |
| //\*[@id="Col1-1-Financials-Proxy"]/section/div[3]/table |

각 종목의 url 및 XPath를 이용하여 재무제표에 해당하는 테이블을 크롤링한 후 데이터를 다운로드 하는 과정은 다음과 같습니다.

|  |
| --- |
| ticker = read.csv("US\_ticker.csv", row.names = 1)  library(rvest)  library(httr)  for (i in 1 : nrow(ticker)) {  name = ticker[i, 'Symbol'] %>% as.character  data\_fs = c()    tryCatch({  Sys.setlocale("LC\_ALL", "English")  yahoo.finance.xpath = '//\*[@id="Col1-1-Financials-Proxy"]/section/div[3]/table'    IS = paste0("https://finance.yahoo.com/quote/",name,"/financials?p=",name) %>%  GET() %>% read\_html() %>% html\_nodes(xpath = yahoo.finance.xpath) %>%  html\_table() %>% data.frame()  Sys.sleep(0.5)  BS = paste0("https://finance.yahoo.com/quote/",name,"/balance-sheet?p=",name) %>%  GET() %>% read\_html() %>% html\_nodes(xpath = yahoo.finance.xpath) %>%  html\_table() %>% data.frame()  Sys.sleep(0.5)  CF = paste0("https://finance.yahoo.com/quote/",name,"/cash-flow?p=",name) %>%  GET() %>% read\_html() %>% html\_nodes(xpath = yahoo.finance.xpath) %>%  html\_table() %>% data.frame()  data\_fs = rbind(IS, BS, CF)  data\_fs = data\_fs[!duplicated(data\_fs[, 1]), ]  colnames(data\_fs) = data\_fs[1,]  data\_fs = data\_fs[-1, ]  rownames(data\_fs) = data\_fs[,1]  data\_fs = data\_fs[,-1]  for (j in 1:ncol(data\_fs)) {  data\_fs[, j] = gsub(",", "", data\_fs[, j]) %>% as.numeric  }  colnames(data\_fs) = sapply(colnames(data\_fs), function(x) {  substring(x,nchar(x)-3, nchar(x))  })  }, error = function(e) {  data\_fs <<- NA  print(paste0("Error in Ticker: ", name))}  )  write.csv(data\_fs, paste0(name, "\_fs.csv"))  print(c(name, i / nrow(ticker)))  Sys.sleep(3)  } |

먼저 미국 상장 전종목의 티커를 불러온 후, 크롤링에 필요한 rvest와 httr 패키지를 불러옵니다. 첫번째 종목인 i=1의 예를 보면, 티커를 name 변수에 입력한 후, 재무제표가 저장될 data\_fs 변수를 빈 형태로 만들어 주도록 합니다. 그 후 재무제표의 3가지 항목에 해당하는 테이블을 다운로드 받도록 합니다.

먼저 위에서 확인한 XPath 값을 yahoo.finance.xpath 변수에 저장합니다. paste0을 통해 해당 티커의 Income Statement 페이지에 해당하는 url을 만들어 준 후, GET() 함수를 이용하여 데이터를 요청합니다. 그 후 read\_html() 함수를 통해 HTML 정보를 읽고, html\_nodes() 함수중 위에서 입력한 값을 XPath 인자로 받아 해당 데이터를 찾습니다. 그 후 html\_table()을 통해 테이블 데이터를 찾습니다. 해당 과정을 거치면 IS 변수에는 리스트의 형태로 데이터가 저장됩니다. 따라서 data.frame() 함수를 통해 데이터 프레임 형식으로 변경해주며, Balance Sheet과 Cash Flow 모두 동일한 과정을 거쳐줍니다. 각 항목의 다운로드 사이에는 0.5초의 프로세스 지연을 줍니다.

세가지 항목에 대한 데이터 수집이 끝난 후, rbind() 함수를 통해 이를 행의 형태로 묶어줍니다. 이 중 첫번째 행은 각 계정 항목을 의미하며, !duplicated() 함수를 통해 계정명이 중복되지 않는 행만 선택하여 data\_fs에 다시 저장하도록 합니다.

|  |
| --- |
| > head(data\_fs)  X1 X2 X3 X4 X5  1 Revenue 12/31/2017 12/31/2016 12/31/2015 12/31/2014  2 Total Revenue 31,657,000 30,109,000 30,274,000 31,821,000  3 Cost of Revenue 15,915,000 15,040,000 15,343,000 16,447,000  4 Gross Profit 15,742,000 15,069,000 14,931,000 15,374,000  5 Operating Expenses Operating Expenses Operating Expenses Operating Expenses Operating Expenses  6 Research Development 1,842,000 1,735,000 1,751,000 1,770,000 |

이 후 기존 국내 종목의 재무제표 다운로드와 동일한 형태의 데이터 클랜징 과정을 거칩니다. 먼저 첫번째 행을 열이름으로 설정한 후, 해당 열을 삭제해줍니다. 첫번째 열 또한 행이름으로 설정한 후, 해당 행을 삭제해줍니다. 또한 gsub() 함수를 통해 숫자 사이의 쉼표(,) 문자를 없애주도록 하며, as.numeric() 함수를 통해 숫자 형태로 변경하여 줍니다. Operating Expenses와 같은 문자는 숫자 형태로 바꾸는 과정에서 NA 데이터로 자동 변경되며, ‘NAs introduced by coercion’의 경고문구는 이를 나타내는 것입니다.

|  |
| --- |
| > head(data\_fs)  12/31/2017 12/31/2016 12/31/2015 12/31/2014  Total Revenue 31657000 30109000 30274000 31821000  Cost of Revenue 15915000 15040000 15343000 16447000  Gross Profit 15742000 15069000 14931000 15374000  Operating Expenses NA NA NA NA  Research Development 1842000 1735000 1751000 1770000  Selling General and Administrative 6567000 6222000 6167000 6469000 |

마지막으로 mm/dd/yyyy의 형태로 저장되어 있는 열이름에서 연도에 해당하는 숫자만 선택하도록 합니다. sapply() 함수의 function에 존재하는 substring()함수는 문자열 중 특정 부분만을 선택하는 함수입니다. substring(text, first, last)의 형태로 이루어지며 text는 문자열, first는 시작점, last는 종료점입니다. nchar() 함수는 문자열의 길이는 나타내는 것이며, 해당 데이터에서는 총 10개의 문자열로 이루어져 있습니다. 즉 substring(x,nchar(x)-3, nchar(x)) 코드는 열이름에서 10-3인 7에서 10번째 까지 문자열을 선택하며, 이는 yyyy 부분에 해당합니다.

|  |
| --- |
| > sapply(colnames(data\_fs), function(x) {  + substring(x,nchar(x)-3, nchar(x))  + })  12/31/2017 12/31/2016 12/31/2015 12/31/2014  "2017" "2016" "2015" "2014" |

데이터 다운로드 중 오류가 발생했을 때는 tryCatch() 함수를 통해 data\_fs 변수에 NA 데이터를 저장하도록 하며, 오류가 발생한 티커를 출력합니다. 그 후 ‘티커\_fs.csv’ 파일로 데이터를 저장한 후, 진행상황을 출력합니다.

미국의 재무제표 데이터가 한국 재무제표 데이터와 차이가 있는 점은, 한국의 경우 최근 연도의 데이터가 가장 오른쪽 열에 위치하였지만, 미국의 경우 가장 좌측에 위치한다는 점입니다.

**가치지표 다운로드**

가치지표의 경우 quantmod 패키지에서 제공하는 getQuote() 함수를 이용하면 야후 파이낸스 API에서 제공하는 지표를 쉽게 다운로드 받을 수 있습니다.

|  |
| --- |
| ticker = read.csv("US\_ticker.csv", row.names = 1)  library(quantmod)  for (i in 1 : nrow(ticker)) {  name = ticker[i, 'Symbol'] %>% as.character  data\_value = c()  tryCatch({  Ratios = yahooQF(c("P/E Ratio", "Price/Book", "Dividend Yield"))  data\_value = getQuote(name, what = Ratios)[Ratios[[2]]]  }, error = function(e) {  data\_value <<- NA  print(paste0("Error in Ticker: ", name))}  )  write.csv(data\_value, paste0(name, "\_value.csv"))  print(c(name, i / nrow(ticker)))  Sys.sleep(3)  } |

먼저 상장종목의 티커를 불러온 후, quantmod 패키지를 불러오도록 합니다. 그 후 티커를 name 변수에 저장한 후, 가치지표가 정리될 data\_value 변수를 만들어 주도록 합니다.

yahooQF() 함수는 야후 파이낸스로부터 제공되는 종목에 대한 쿼트의 리스트로써, 해당 함수를 입력하면 총 60개의 제공 쿼트가 출력됩니다.

**[표] yahooQF() 함수 제공 쿼트**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **번호** | **항목** | **번호** | **항목** | **번호** | **항목** |
| 1 | Symbol | 2 | Name | 3 | Name (Long) |
| 4 | Quote Type | 5 | Quote Source Name | 6 | Source Interval |
| 7 | Currency | 8 | Financial Currency | 9 | Market |
| 10 | Market State | 11 | Exchange | 12 | Exchange Full Name |
| 13 | Exchange Timezone | 14 | Exchange TZ | 15 | Exchange Data Delay |
| 16 | GMT Offset Millis | 17 | Tradeable | 18 | Ask |
| 19 | Bid | 20 | Ask Size | 21 | Bid Size |
| 22 | Last Trade (Price Only) | 23 | Open | 24 | Days High |
| 25 | Days Low | 26 | Volume | 27 | Days Low |
| 28 | Days High | 29 | Change in Percent | 30 | Previous Close |
| 31 | Change From 52-week Low | 32 | Percent Change From 52-week Low | 33 | Change From 52-week High |
| 34 | Percent Change From 52-week High | 35 | 52-week Low | 36 | 52-week High |
| 37 | 50-day Moving Average | 38 | Change From 50-day Moving Average | 39 | Percent Change From 50-day Moving Average |
| 40 | 200-day Moving Average | 41 | Change From 200-day Moving Average | 42 | Percent Change From 200-day Moving Average |
| 43 | Market Capitalization | 44 | P/E Ratio | 45 | Price/EPS Estimate Next Year |
| 46 | Price/Book | 47 | Book Value | 48 | Average Daily Volume |
| 49 | Shares Outstanding | 50 | Ex-Dividend Date | 51 | Dividend/Share |
| 52 | Dividend Yield | 53 | Earnings Timestamp | 54 | Earnings Start Time |
| 55 | Earnings End Time | 56 | Earnings/Share | 57 | EPS Forward |
| 58 | Language | 59 | Message Board ID | 60 | Price Hint |

이 중 가치지표에 해당하는 ‘P/E Ratio’, ‘Price/Book’, ‘Dividend Yield’를 선택한 후, Ratios 변수에 저장합니다. Ratios 변수를 확인해보면, 첫번째 리스트는 해당 쿼트의 상세내역, 두번째 리스트는 쿼트명이 입력됩니다.

|  |
| --- |
| > Ratios  [[1]]  [1] "trailingPE" "priceToBook" "trailingAnnualDividendYield"  [[2]]  [1] "P/E Ratio" "Price/Book" "Dividend Yield"  attr(,"class")  [1] "quoteFormat" |

getQuote() 함수는 입력된 쿼트에 대한 정보를 받아오는 함수입니다. getQuote(Symbols, what, ...)의 형태로 이루어지며, Symbols에는 종목의 티커를, what에는 받아오고자 하는 쿼트를 입력합니다. 쿼트에 해당하는 부분에 위에서 입력한 Ratios를 입력하면 다음과 같은 값이 데이터 프레임의 형태로 출력됩니다.

|  |
| --- |
| > getQuote(name, what = Ratios)  Trade Time P/E Ratio Price/Book Dividend Yield  DDD 2018-08-14 16:03:36 NA 3.693624 NA |

이 중 Trade Time은 필요하지 않은 데이터이므로, [-1]을 통해 첫번째 데이터를 삭제한 후, data\_value 변수에 저장하도록 합니다.

데이터 다운로드 중 오류가 발생했을 때는 tryCatch() 함수를 통해 data\_value 변수에 NA 데이터를 저장하도록 하며, 오류가 발생한 티커를 출력합니다. 그 후 ‘티커\_value.csv’ 파일로 데이터를 저장한 후, 진행상황을 출력합니다.

**[4] 주가, 재무제표 및 가치지표 정리하기**

API와 크롤링을 통하여 미국 퀀트 투자에 필요한 주가, 재무제표, 가치지표 역시 어렵지 않게 수집할 수 있습니다. 국내 데이터와 동일하게 미국 데이터 역시 각 종목 및 항목별로 나누어져 csv 파일로 나누어져 정리되어 있으므로, 각 항목 별로 정리할 필요가 있습니다.

미국 데이터를 정리하는 코드는 국내 데이터의 정리 과정에서 사용되었던 것과 매우 비슷하므로, 이전 내용을 제대로 학습했다면 어렵지 않게 이해할 수 있습니다.

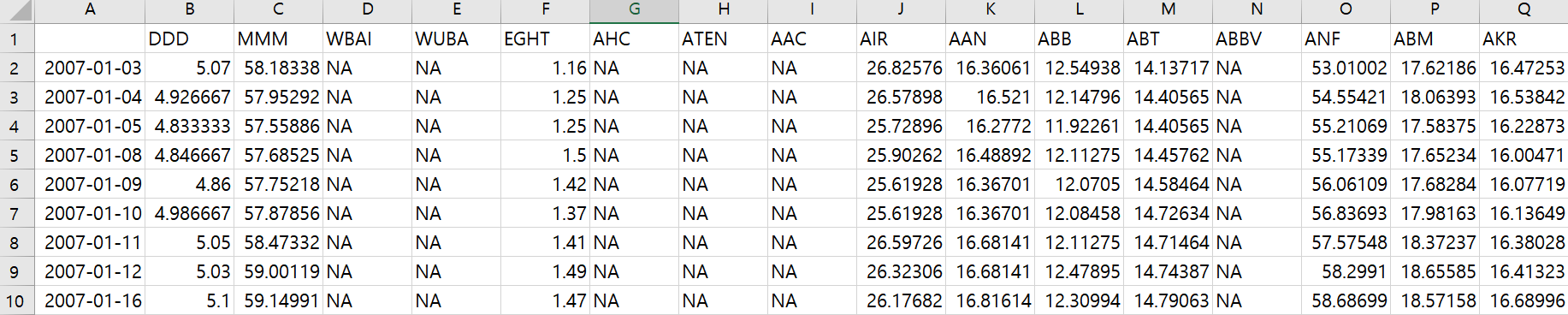
**주가 데이터 정리하기**

|  |
| --- |
| ticker = read.csv("US\_ticker.csv", row.names = 1)  library(quantmod)  library(magrittr)  price\_list = list()  for (i in 1 : nrow(ticker)) {    name = ticker[i, 'Symbol'] %>% as.character()  price\_list[[i]] = read.csv(paste0(name, "\_price.csv"), row.names = 1) %>%  as.xts()    }  price\_list = do.call(cbind, price\_list)  price\_list = na.locf(price\_list)  colnames(price\_list) = ticker$Symbol  write.csv(data.frame(price\_list), "US\_price.csv") |

국내 종목의 주가 정리 코드에서 '종목코드' 부분이 ‘Symbol'로 변경된 것 이외에는 모든 부분이 동일합니다. 티커 리스트와 quantmod 패키지, magrittr 패키지를 불러온 후, for loop 구문을 통해 csv 파일을 순서대로 불러온 후 xts 형태로 변경하여 price\_list의 i번째 리스트에 저장합니다.

그 후 do.call() 함수 내에서 cbind() 함수를 이용하여 열의 형태로 결합한 후, 결측치 보정을 위해 na.locf() 함수를 사용하도록 합니다. 열이름을 티커에 해당하는 ticker$Symbol로 변경하여, ‘US\_price.csv’ 파일로 저장해주도록 합니다.

**[그림] 미국 주가 정리 파일**



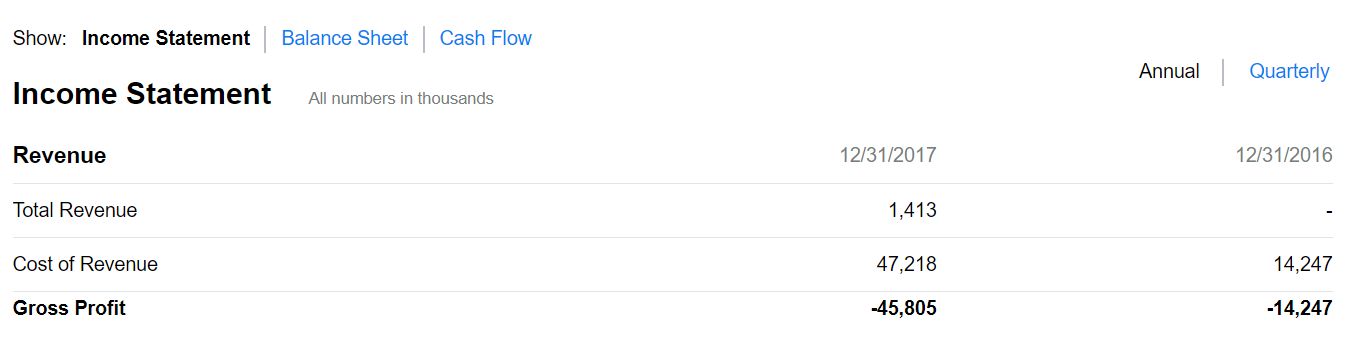
**재무제표 데이터 정리하기**

|  |
| --- |
| ticker = read.csv("US\_ticker.csv", row.names = 1)  library(magrittr)  library(data.table)  data\_csv = list()  for (i in 1 : nrow(ticker)) {    name = ticker[i, 'Symbol'] %>% as.character()  data\_csv[[i]] = read.csv(paste0(name, "\_fs.csv"), row.names = 1)    }  item = data\_csv[[1]] %>% rownames()  fs\_list = list()  for (i in 1 : length(item)) {  fs\_list[[i]] = lapply(data\_fs, function(x) {  if ( item[i] %in% rownames(x) ) {  cbind(x[which(rownames(x) == item[i]),],  matrix(NA, 1, 4 - ncol(x)) %>% data.frame())  } else {  matrix(NA, 1, 4) %>% data.frame()  }  })  } |

미국 재무제표 데이터의 정리 역시 국내 재무제표 데이터 정리의 코드와 거의 유사합니다. data\_csv 리스트에 재무제표 데이터를 저장한 후, lapply 함수를 통해 재무제표 항목별 종목들의 데이터를 저장합니다. 단, 정리과정에서 주의할 점이 있습니다. 첫째, 국내 재무제표 데이터의 경우와는 달리 야후 파이낸스에서는 최근 결산기준 4개 년도의 재무재표 데이터를 제공합니다.

둘째, 일부 종목의 경우 기업의 역사가 오래되지 않아 최근 2~3개 연도의 재무제표만이 존재하는 경우가 있으며, 이러한 경우 타 종목들과 열 갯수가 일치하지 않아 차후 rbindlist() 함수 실행시 오류가 발생합니다

**[그림] 재무제표 데이터가 4년이 되지 않는 경우 (RCUS)**



국내 재무제표 정리와 동일하게 item[i]에 해당하는 내역이 행이름에 존재할 경우, 해당 항목인 부분의 데이터를 찾은 후 반환하도록 합니다. 그 후, 4년간의 데이터가 부족한 항목을 보정해주기 위해 다음과 같은 과정을 추가하도록 합니다. matrix(NA, 1, 4-ncol(temp\_data)) 코드를 통해 행의 개수가 1, 열의 개수가 4에서 재무제표 열의 개수를 뺀만큼의 NA로 이루어진 매트릭스를 생성합니다. 만일 4개 년도 데이터가 모두 있을 경우 matrix(NA, 1, 0), 즉 빈 매트릭스가 생성되며, 2개 년도의 데이터만이 존재하는 경우 1행 2열의 매트릭스가 만들어져 cbind() 함수를 통해 기존의 데이터와 열의 형태로 결합이 됩니다. 즉, 모자라는 연도 수만큼 우측 데이터를 NA로 채워주는 결과가 됩니다.

**[표] 재무재표 데이터의 결측치 보정**

|  |  |  |
| --- | --- | --- |
| **재무재표 데이터** | **NA 생성** | **최종 반환 데이터** |
| |  |  |  | | --- | --- | --- | | 2017 | 2016 | 2015 | | |  | | --- | | NA | | |  |  |  |  | | --- | --- | --- | --- | | 2017 | 2016 | 2015 | NA | |
| |  |  | | --- | --- | | 2017 | 2016 | | |  |  | | --- | --- | | NA | NA | | |  |  |  |  | | --- | --- | --- | --- | | 2017 | 2016 | NA | NA | |

재무제표 데이터가 없는 경우는 matrix(NA, 1, 4) %>% data.frame() 코드를 통해 NA로 이루어진 1행, 4열의 데이터 프레임을 저장합니다.

|  |
| --- |
| > head(fs\_list[[1]], 2)  [[1]]  X2017 X2016 X2015 X2014  Total Revenue 646069 632965 666163 653652  [[2]]  X2017 X2016 X2015 X2014  Total Revenue 31657000 30109000 30274000 31821000 |

fs\_list의 첫번째 리스트에는 Total Revenue에 해당하는 정보들이 각 종목별 순서대로 리스트의 형태로 저장되어 있습니다.

|  |
| --- |
| fs\_list = lapply(fs\_list, function(x) {rbindlist(x) %>% data.frame()})  fs\_list = lapply(fs\_list, function(x) {  rownames(x) = ticker[,'Symbol'] %>% as.character()  return(x)  })  names(fs\_list) = item  saveRDS(fs\_list, "US\_fs.Rds") |

lapply() 함수와 rbindlist() 함수를 이용하여 fs\_list의 하부 리스트를 묶은 후 데이터 프레임 형식으로 저장하도록 합니다. 그 후, 행이름을 티커에 해당하는 ticker[,'Symbol']로 변경하며, 리스트의 이름은 재무제표 항목에 해당하는 item으로 변경하도록 합니다.

|  |
| --- |
| > head(fs\_list[[1]])  X2017 X2016 X2015 X2014  DDD 646069 632965 666163 653652  MMM 31657000 30109000 30274000 31821000  WBAI 131323 10928 99552 579717  WUBA 10068780 7592127 4478098 1628120  EGHT 296500 253388 209336 162413  AHC 248626 259984 272108 272788  > head(names(fs\_list))  [1] "Total Revenue" "Cost of Revenue"  [3] "Gross Profit" "Operating Expenses"  [5] "Research Development" "Selling General and Administrative" |

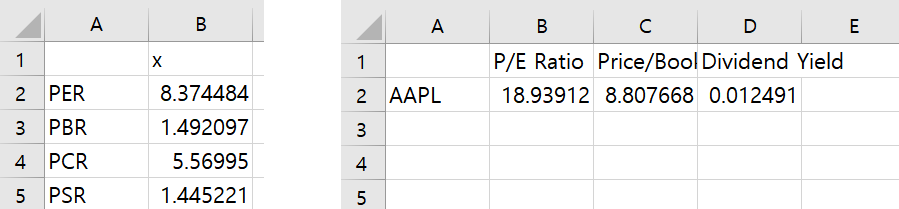
마지막으로 saveRDS() 함수를 이용하여 정리된 데이터를 ‘US\_fs.Rds’의 Rds 형식으로 저장해주도록 합니다.

**가치지표 데이터 정리하기**

|  |
| --- |
| ticker = read.csv("US\_ticker.csv", row.names = 1)  library(magrittr)  data\_csv = list()  for (i in 1 : nrow(ticker)) {    name = ticker[i, 'Symbol'] %>% as.character()  data\_csv[[i]] = read.csv(paste0(name, "\_value.csv"), row.names = 1)    }  item = data\_csv[[1]] %>% colnames()  value\_list = list()  for (i in 1 : length(item)) {  value\_list[[i]] = lapply(data\_value, function(x) {  if ( item[i] %in% colnames(x) ) {  x[which(colnames(x) == item[i])]  } else {  NA  }  })  } |

가치지표의 정리 역시 기존 국내 데이터와 동일하며, 일부분만을 변경하면 됩니다. 티커와 magrittr 패키지를 불러온 후, data\_csv 리스트에 가치지표 데이터를 저장합니다. 그 후 가치지표 항목들을 불러오기 위해 국내 데이터는 rownames() 함수를 이용했지만, 미국 데이터의 경우 colnames() 함수를 이용합니다. 이는 국내 데이터와 해외 데이터의 저장된 양식이 다르기 때문입니다. [그림]의 좌측은 국내 가치지표의 저장 형식이며, 우측은 미국 가치지표의 저장 형식입니다. 국내의 경우 크롤링을 통해 열의 형태로 저장되어 각 지표 항목이 행이름에 위치하지만, 미국의 경우 getQuote() 함수에서 데이터를 열의 형태로 제공하여 각 지표 항목이 열이름에 위치하고 있습니다.

**[그림] 국내와 미국 가치지표 저장 양식의 차이**

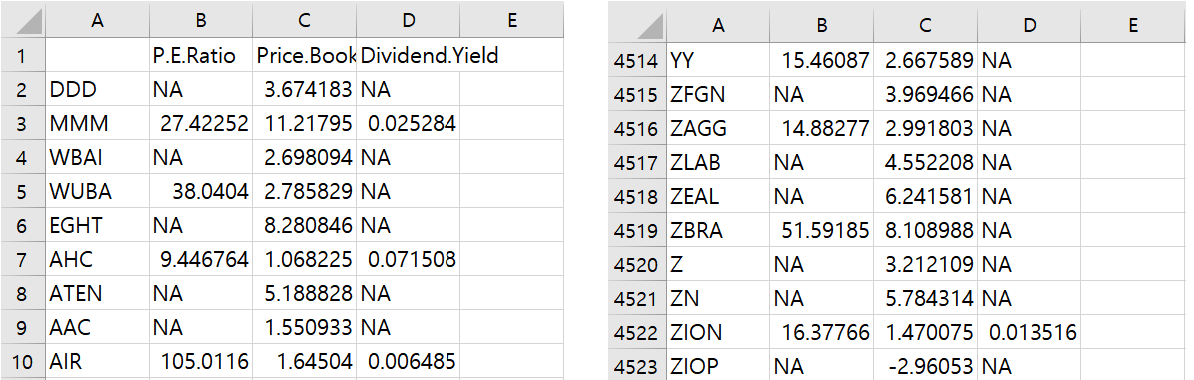


이러한 이유로 국내 데이터의 경우 각 항목에 해당하는 데이터를 x[which(rownames(x) == item[i]),] 코드를 통해 행의 형태로 불러왔지만, 미국 데이터의 경우 x[which(colnames(x) == item[i])] 코드를 통해 열의 형태로 불러오게 됩니다.

|  |
| --- |
| value\_list = lapply(value\_list, function(x) {do.call(rbind, x)})  value\_list = do.call(cbind, value\_list) %>% data.frame()  rownames(value\_list) = ticker[, 'Symbol']  colnames(value\_list) = item  write.csv(value\_list, "US\_value.csv") |

이후의 과정은 모두 동일하며, ‘US\_value.csv’ 파일로 저장해주도록 합니다.

**[그림] 미국 가치지표 정리 파일**



**Part 2 레시피를 이용한 종목 선정 및 포트폴리오 구성**

요리에 관련된 비밀이라면 응당 공유되어야 한다.

지식의 공유 덕에 미식문화가 이어지고 있다.

* 보 송비사바

**6장. 퀀트 투자 전략을 이용한 종목선정**

API와 크롤링을 이용하여 요리에 필요한 재료들을 모두 수집하고, 데이터 클랜징을 통해 이를 다듬고 모으는 과정을 끝마쳤습니다. 재료가 준비되었으니 이제 본격적으로 요리를 시작해볼 수 있습니다.

요리를 처음할때는 무작정 시도하기 보다는 기존에 나와있는 레시피를 이용하여 어떠한 재료를 얼마나 사용하고, 어떠한 순서로 사용하는지 따라하는 것이 좋습니다. 레시피를 따라 요리한다면, 유명 쉐프가 만든 환상적인 맛의 요리를 만들지는 못하지만 꽤나 맛있는 요리를 만들수는 있습니다.

퀀트 투자 역시 이와 동일합니다. 우리가 준비한 데이터를 바탕으로 기존에 나와있는 다양한 퀀트 전략을 이용하여 포트폴리오를 구성할 경우, 워런 버핏이나 제임스 사이먼스와 같이 엄청난 수익률을 얻을 수는 없지만 평균 이상의 꾸준한 수익률을 얻을 수 있습니다.

퀀트 투자는 크게 포트폴리오 운용 전략과 트레이딩 전략으로 나눌 수 있습니다. 포트폴트폴리오 운용 전략의 경우 과거 주식 시장을 분석하여 ‘좋은 주식’의 기준을 찾아낸 후 해당 기준에 만족하는 종목을 매수하거나, 이와 정반대에 있는 ‘나쁜 주식’을 공매도 하기도 합니다. 투자의 속도가 매우 느리며, 다수의 종목을 하나의 포트폴리오로 구성하여 운용하는 특징이 있습니다. 반면 트레이딩 전략의 경우, 단기간에 발생되는 주식의 움직임을 연구한 후 예측하여, 매수 혹은 매도하는 전략입니다. 투자의 속도가 매우 빠르며 매우 소수의 종목을 대상으로 합니다.

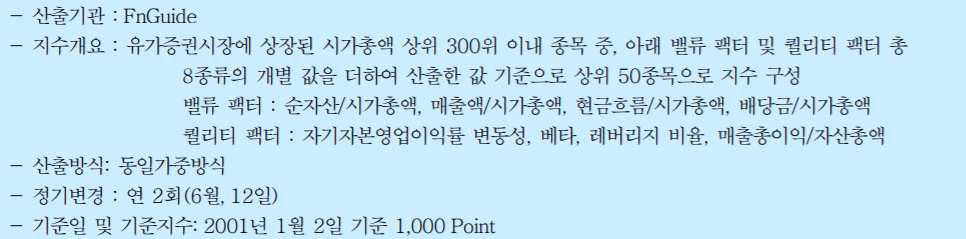
**[표] 퀀트 투자 종류의 비교**

|  |  |  |
| --- | --- | --- |
|  | **포트폴리오 운용 전략** | **트레이딩 전략** |
| 공통점 | 규칙에 기반한 투자 | |
| 투자 목적 | 좋은 주식을 매수 | 좋은 시점을 매수 |
| 학문적 기반 | 경제학, 통계학 등 | 통계학, 공학, 정보처리 등 |
| 투자의 속도 | 느림 | 빠름 |

이 중 본 책에서는 포트폴리오에 기반한 운용 전략에 대해 다루도록 합니다. 과거의 데이터를 바탕으로 주식의 수익률에 영향을 지표를 팩터Factor라 합니다. 즉 팩터의 강도가 양인 종목들로 구성한 포트폴리오의 경우 향후 수익률이 높을 것으로 예상되며, 팩터의 강도가 음인 종목들로 구성한 포트폴리오의 경우 반대로 향후 수익률이 낮을 것으로 예상됩니다.

팩터에 대한 연구는 학자들에 의해 수십년간 끊임없이 진행되어 왔지만, 일반 투자자들이 이러한 논문을 모두 찾아보고 연구하는 것은 사실상 불가능에 가깝습니다. 그러나 다행히 최근에는 ‘스마트베타’라는 이름으로 팩터 투자가 대중화되고 있습니다. 최근 유행하고 있는 스마트베타 ETF의 경우 팩터를 기준으로 포트폴리오를 구성한 상품으로써, 학계나 실무에서 검증된 팩터 전략을 기반으로 합니다. 해당 상품들의 홈페이지나 투자설명서에는 종목 선정 기준에 대해 자세히 나와있으므로 이는 매우 훌륭한 레시피이기도 합니다.

**[그림] 스마트베타 ETF 투자설명서 예시**



스마트베타 ETF에 나와있는 투자 전략을 자세히 분석하는 것만으로도 훌륭한 퀀트 투자 전략을 만들 수 있습니다. 그러나 좀더 특별한 전략을 만들거나, 혹은 단기간의 수익률 부진에 대한 인내심을 키우기 위해서는 팩터 투자에 대한 근본적 이해가 필요합니다.

저의 지난 책 **‘스마트베타(2017)’**에는 각종 팩터들에 대한 학문적 이론에 대해 자세히 설명하고 있으며, 저의 블로그(<http://henryquant.blogspot.com/>)에도 팩터에 대한 연구 결과를 업로드 하고 있습니다.

**[1] 주식의 베타 계산**

투자자들이라면 누구나 한번쯤 들어봤을만한 용어가 ‘베타Beta’ 입니다. 기본적으로 개별 주식의 수익률에 가장 크게 영향을 주는 요소는 주식시장의 움직임일수 밖에 없습니다. 아무리 좋은 주식도 주식시장이 폭락한다면 같이 떨어지며, 아무리 나쁜 주식도 주식시장이 급등한다면 대부분 같이 오르기 마련입니다.

개별 주식이 전체 주식시장의 변동에 반응하는 정도를 나타낸 값이 베타입니다. 베타가 1이라는 뜻은 주식시장과 움직임이 같다는 뜻으로써, 시장 그 자체를 나타냅니다. 베타가 1.5라는 뜻은 주식시장이 수익률이 +1% 일 때 개별 주식의 수익률은 +1.5% 이며, 반대로 주식시장의 수익률이 -1% 일 때 개별 주식의 수익률은 -1.5% 입니다. 반면 베타가 0.5라는 주식시장 수익률의 절반 정도만이 움직이게 됩니다.

**[표] 베타에 따른 개별 주식의 수익률 움직임**

|  |  |  |
| --- | --- | --- |
| **베타** | **주식시장이 +1%일 경우** | **주식시장이 -1%일 경우** |
| 0.5 | +0.5% | -0.5% |
| 1.0 | +1.0% | -1.0% |
| 1.5 | +1.5% | -1.5% |

이처럼 베타가 큰 주식은 주식시장보다 수익률의 움직임이 크며, 반대로 베타가 낮은 주식은 주식시장보다 수익률의 움직임이 작습니다. 따라서 일반적으로 상승장이 기대될 때는 베타가 큰 주식에, 하락장일이 기대될 때는 베타가 낮은 주식에 투자하는 것이 바람직합니다.

**베타 계산하기**

주식시장에서의 베타는 통계학의 회귀분석모형에서 기울기를 나타내는 베타와 정확히 의미가 같습니다. 회귀분석모형은 y = a + bx 형태로 나타나며, x의 변화에 따른 y의 변화의 기울기가 회귀계수인 b입니다. 이를 주식에 적용한 모형이 자산가격결정모형CAPM: Capital Asset Pricing Model’이며, 그 식은 다음과 같습니다.

|  |
| --- |
| **회귀분석모형:**  **자산가격결정모형:** |

먼저 회귀분석모형의 상수항인 a에 해당하는 부분은 무위험 수익률을 나타내는 입니다. 독립변수인 x에 해당하는 부분은 무위험 수익률 대비 주식 시장의 초과 수익률을 나태내는 입니다.종속변수인 y에 해당하는 부분은 개별주식의 수익률을 나타내는 이며, 최종적으로 회귀계수인 b에 해당하는 부분은 개별 주식의 베타인 입니다.

**[표] 회귀분석모형과 자산가격결정모형의 비교**

|  |  |  |
| --- | --- | --- |
|  | **회귀분석모형** | **자산가격결정모형** |
| 상수항 | a | (무위험 수익률) |
| 독립변수 | x | (초과 수익률) |
| 종속변수 | y | (개별주식의 수익률) |
| 회귀계수 | b | (개별주식의 베타) |

통계학에서 회귀계수는 형태로 구할 수 있으며, x와 y에 각각 시장수익률과 개별주식의 수익률을 대입할 경우 개별주식의 베타는 형태로 구할 수 있습니다. 그러나 R을 이용할 경우 회귀분석모형을 통해 간단히 베타를 구할 수 있습니다.

베타를 구하는 방법을 알아보기 위해 전통적 고베타주인 증권주, 저베타주인 유틸리티주를 이용하도록 하며, 주식시장에 대한 대용치로 KOSPI 200 ETF를 이용하도록 하겠습니다.

|  |
| --- |
| library(quantmod)  library(PerformanceAnalytics)  library(magrittr)  symbols = c("039490.KS", "015760.KS", "069500.KS")  getSymbols(symbols)  prices = do.call(cbind, lapply(symbols, function(x) Cl(get(x))))  ret = Return.calculate(prices)  ret.sub = ret["2016-01::2017-12"] |

먼저 패키지들을 불러온 후, symbols 변수에 증권주인 키움증권(039490.KS), 유틸리티주인 한국전력(015760.KS), KOSPI 200 ETF인 KODEX 200(069500.KS) 티커를 입력합니다. 그 후, getSymbols() 함수를 이용하면 해당 티커들의 데이터가 다운로드 되어 각각의 변수명에 저장됩니다. lapply() 함수 내에 Cl()과 get()함수를 사용하여 종가에 해당하는 데이터만 추출하며, 리스트 형태의 데이터를 열의 형태로 묶어주기 위해 do.call() 함수와 cbind() 함수를 사용해 줍니다.

get() 함수는 주어진 값에 해당하는 객체를 검색한 후 이에 해당하는 값을 반환합니다. 즉 get(symbols[1])의 경우, symbols의 첫번째 값인 "039490.KS"의 객체를 검색하여 이에 해당하는 데이터를 반환합니다. 반환된 값이 Cl() 함수를 적용하면 종가 부분만을 선택할 수 있습니다.

|  |
| --- |
| > head(Cl(get(symbols[1])))  039490.KS.Close  2007-01-02 25736.4  2007-01-03 25112.2  2007-01-04 24584.0  2007-01-05 24343.9  2007-01-08 23863.8  2007-01-09 24007.8 |

종가 부분만이 선택된 price 변수를 Return.calculate() 내에 입력하면 수익률이 계산됩니다. 이중 2016년 1월부터 2017년 12월까지의 데이터만 선택하여 ret.sub 함수에 저장해줍니다. xts 형식의 데이터는 대괄호 속에 ‘시작일자::종료일자’와 같은 형태로, 원하는 일자를 편리하게 선택할 수 있으며, 위의 기간은 ret["2016::2017"] 코드를 입력하여도 동일하게 2016년부터 2017년 까지의 데이터가 선택됩니다.

|  |
| --- |
| x = ret.sub[, 3] %>% as.numeric()  y = ret.sub[, 1] %>% as.numeric()  reg = lm(y ~ x)  summary(reg)  reg$coefficients[2] |

먼저 고베타주인 증권주를 대상으로 베타를 구하기 위한 회귀분석을 실시해 주도록 합니다. 자산가격결정모형의 수식인 에서 편의를 위해 무위험 수익률인 를 0으로 가정하면, 의 형태로 나타낼 수 있습니다. 이 중 는 독립변수인 주식시장의 수익률을, 는 종속변수인 개별주식의 수익률을 의미합니다.

독립변수인 x는 세번째 열인 KODEX 200 ETF의 수익률을 선택해주며, as.numeric() 함수를 통해 숫자 형태로 바꾸어 줍니다. 종속변수인 y는 첫번째 열인 키움증권의 수익률을 선택해주며, 이 역시 숫자 형태로 바꾸어줍니다. lm() 함수를 통해 손쉽게 선형회귀분석을 실시할 수 있으며, 회귀분석의 결과를 reg 변수에 저장해줍니다. summary() 함수는 데이터의 요약 정보를 나타내며, 해당 예시에서는 회귀분석결과에 대한 정보를 보여줍니다.

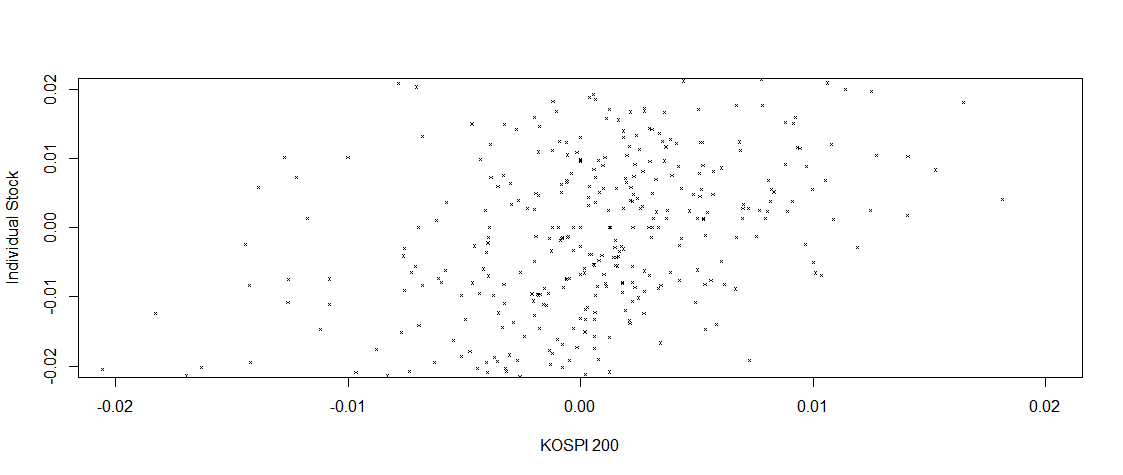
|  |
| --- |
| > summary(reg)  Call:  lm(formula = y ~ x)  Residuals:  Min 1Q Median 3Q Max  -0.070053 -0.011456 -0.001993 0.009743 0.094789  Coefficients:  Estimate Std. Error t value Pr(>|t|)  (Intercept) -2.606e-05 8.778e-04 -0.03 0.976  x 1.511e+00 1.198e-01 12.62 <2e-16 \*\*\*  ---  Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1  Residual standard error: 0.01904 on 472 degrees of freedom  (13 observations deleted due to missingness)  Multiple R-squared: 0.2523, Adjusted R-squared: 0.2507  F-statistic: 159.2 on 1 and 472 DF, p-value: < 2.2e-16 |

회귀분석의 결과 중 가장 중요한 부분은 계수를 나타내는 Coefficients 부분입니다. Intercept 부분은 회귀분석의 상수항에 해당하는 부분으로써, 값이 거의 0에 가깝고 t밸류 또한 매우 작아 유의하지 않음이 보입니다. 우리가 원하는 베타에 해당하는 부분은 x의 Estimate 부분으로써, 베타값이 1.511로 증권주의 특성인 고베타주임이 확인되며, t밸류 또한 12로 매우 유의한 결과입니다. 조정된 결정계수(R-square)는 25%를 보입니다. reg$coefficients[2] 코드를 통해 계수의 두번째 값, 즉 베타만을 선택할 수도 있습니다.

|  |
| --- |
| plot(x, y, pch = 4, cex = 0.3,  xlab = "KOSPI 200", ylab = "Individual Stock",  xlim = c(-0.02, 0.02), ylim = c(-0.02, 0.02))  grid()  abline(a = 0, b = 1, lty = 2)  abline(reg, col = 'red') |

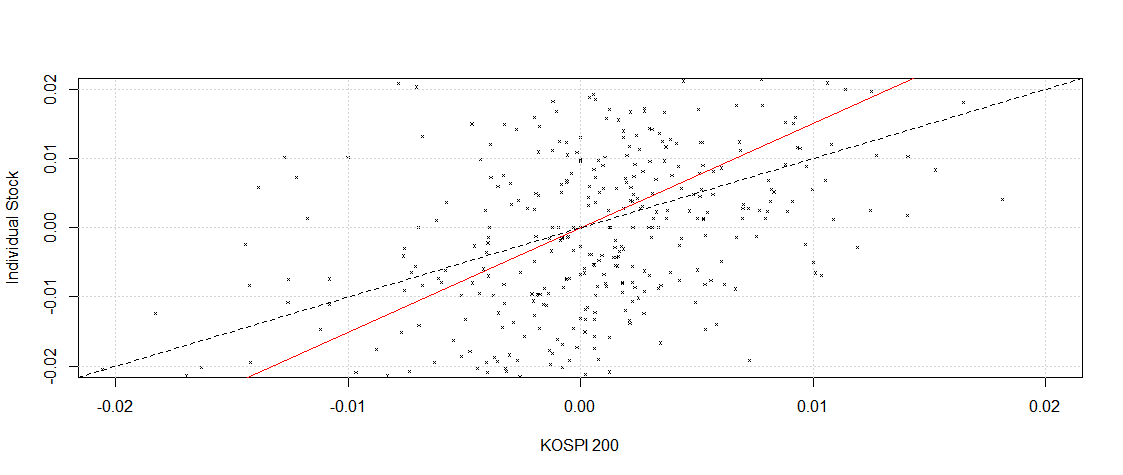
구해진 베타를 그래프로 표현해보도록 하겠습니다. 먼저 plot() 함수는 각종 그래프를 그리는 함수로써, x축과 y축에 각각의 데이터를 입력합니다. pch는 점들의 모양을, cex는 점들의 크기를 나타내며, xlab과 ylab은 각각 x축과 y축에 들어갈 문구를 나타냅니다. xlim과 ylim은 x축과 y축의 최소 및 최대 범위를 지정해줍니다. 해당 코드를 입력하면 다음과 같은 그래프가 나오며, 이는 증권주와 KOSPI 200의 일별 수익률을 나타낸 것입니다.

**[그림] 증권주와 KOSPI 200 일별 수익률 그래프**



grid() 함수는 그래프에 격자무늬를 추가해주는 함수이며, abline() 함수는 직선을 그려주는 함수입니다. 먼저 abline(a = 0, b = 1, lty = 2) 코드에서 a는 상수, b는 직선의 기울기, lty는 선의 유형을 나타냅니다. 이를 통해 기울기, 즉 베타가 1일 경우의 선을 그립니다. abline(reg, col = 'red') 코드의 입력값인 reg는 회귀분석의 결과이며, 이를 통해 회귀식을 그릴수 있습니다. col은 선의 색깔이며, 붉은 색으로 나타내도록 합니다.

**[그림] 증권주와 시장 베타의 비교**



먼저 검은색의 점선이 기울기가 1인 경우이며, 붉은색의 직선이 증권주의 회귀분석결과를 나타낸 그래프입니다. 기울기가 1보다 훨씬 가파름이 확인되며, 즉 베타가 1보다 크다는 사실을 알 수 있습니다.

|  |
| --- |
| x = ret.sub[, 3] %>% as.numeric()  y = ret.sub[, 2] %>% as.numeric()  reg = lm(y ~ x)  summary(reg)  reg$coefficients[2]  plot(x, y, pch = 4, cex = 0.3,  xlab = "KOSPI 200", ylab = "Individual Stock",  xlim = c(-0.02, 0.02), ylim = c(-0.02, 0.02))  grid()  abline(a = 0, b = 1, lty = 2)  abline(reg, col = 'red') |

이번에는 저베타주인 유틸리틸리주를 대상으로 베타를 구하기 위한 회귀분석을 실시해 주도록 합니다. 독립변수인 x는 KODEX 200 ETF의 수익률을 선택해주며, 종속변수인 y는 두번째 열인 한국전력의 수익률을 선택해줍니다. 회귀분석 후 결과를 확인하면 다음과 같습니다.

|  |
| --- |
| > summary(reg)  Call:  lm(formula = y ~ x)  Residuals:  Min 1Q Median 3Q Max  -0.054071 -0.008709 -0.000372 0.008523 0.059973  Coefficients:  Estimate Std. Error t value Pr(>|t|)  (Intercept) -0.0006013 0.0006666 -0.902 0.36756  x 0.2945330 0.0909517 3.238 0.00129 \*\*  ---  Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1  Residual standard error: 0.01446 on 472 degrees of freedom  (13 observations deleted due to missingness)  Multiple R-squared: 0.02174, Adjusted R-squared: 0.01966  F-statistic: 10.49 on 1 and 472 DF, p-value: 0.001287 |

Coefficients를 확인해보면, 이번에도 Intercept 부분은 거의 0에 가까우면 t밸류가 -0.902로써 유의하지도 않습니다. 베타의 경우 0.2945로써 매우 낮아 유틸리티주의 특성인 저베타주임이 확인되며, t밸류는 3.238로써 유의미함이 확인됩니다. 반면 조정된 결정계수는 2%로써 매우 낮습니다. 위에서 그래프를 그렸던 과정과 동일하게 유틸리티주의 베타를 그래프로 나타내면 다음과 같습니다.

**[그림] 유틸리티주와 시장 베타의 비교**



기울기가 1인 검은색의 직선과 비교해보면, 붉은색으로 표현된 유틸리티주의 경우 기울기가 매우 낮음이 확인됩니다.

**[2] 저변동성 전략**

금융 시장에서 변동성은 수익률이 움직이는 정도로써, 일반적으로 표준편차가 사용됩니다. 표준편차는 자료가 평균을 중심으로 얼마나 퍼져 있는지를 나타내는 수치로써, 수식은 다음과 같습니다.

|  |
| --- |
|  |

는 개별 관측값이며, 는관측값들의 평균입니다. 즉 개별 관측값들이 평균에 가까울수록 표준편차는 작은 값이 나오며, 평균에서 멀리 떨어질수록 표준편차는 큰 값이 나오게 됩니다. **85, 76, 73, 80, 72** 값의 표준편차를 계산하는 방법은 다음과 같습니다.

|  |
| --- |
|  |

관측값의 개수가 작을 경우에는 수식에 대입하여 계산하는 것이 가능하지만, 관측값이 수백 혹은 수천개로 늘어날 경우 컴퓨터를 이용하지 않고 계산하는 것은 사실상 불가능합니다. R에서는 복잡한 계산과정 없이 sd() 함수를 이용하여 간단하게 표준편차를 계산할 수 있습니다.

|  |
| --- |
| example = c(85,76,73,80,72)  > sd(example)  [1] 5.357238 |

개별 주식의 표준편차를 측정할 때는 주식의 가격이 아닌 수익률을 관측값으로 사용합니다. 수익률의 표준편차가 크다는 의미는 수익률의 위 아래로 많이 움직여 위험한 종목으로 받아들여 집니다. 반면, 표준편차가 작다는 의미는 수익률의 움직임이 적어 상대적으로 안전한 종목으로 받아들여 집니다.

전통적 금융 이론에서는 수익률의 변동성이 클수록 위험이 크고, 이런 위험에 대한 보상으로 기대수익률이 높아야 한다고 보았습니다. 따라서 고변동성 종목의 기대수익률이 크고, 저변동성 종목의 기대수익률이 낮은 고위험 고수익High Risk High Return이 당연한 믿음이었습니다. 그러나 현실에서는 오히려 변동성이 낮은 종목들의 수익률이 변동성이 높은 종목들의 수익률 보다 높은, 저변동성 효과가 발견[[25]](#endnote-25)되고 있습니다.

이러한 저변동성 효과가 발생하는 원인으로는 여러 가설이 있습니다. 첫째, 투자자들은 대체로 자신의 능력을 과신하는 경향이 있으며, 복권과 같이 큰 수익을 가져다 주는 고변동성 주식을 선호하는 경향이 있습니다. 이러한 결과로 고변동성 주식은 과대 평가가 되어 수익률이 낮은 반면, 과소 평가된 저변동성 주식들은 높은 수익률을 보이게 됩니다.[[26]](#endnote-26) 둘째, 대부분 기관투자가들이 레버리지 투자가 되지 않는 상황에서, 벤치마크 대비 높은 성과를 얻기 위해 고변동성 주식에 투자하는 경향이 있으며, 이 또한 고변동성 주식이 과대 평가되는 결과로 이어집니다.[[27]](#endnote-27) 마지막으로 시장의 상승과 하락이 반복됨에 따라 고변동성 주식이 변동성 손실Volatility Drag로 인해 수익률이 하락하게 되는 이유도 있습니다.

주식의 위험은 변동성뿐만 아니라 베타와 같은 지표로도 측정할 수 있습니다. 저변동성 효과와 비슷하게 고유변동성이 낮은 주식의 수익률이 높은 저고유변동성 효과[[28]](#endnote-28), 베타가 낮은 주식의 수익률이 오히려 높은 저베타 효과[[29]](#endnote-29)도 발견되고 있으며, 이러한 효과들을 합쳐 ‘저위험 효과’로 부르기도 합니다.

**저변동성 포트폴리오 구현**

**7장. 포트폴리오 구성**

포트폴리오 구성에 관한 내용은, 금융에 관한 내용이기는 하지만 수식을 기반으로 컴퓨터 프로그래밍을 통한 최적화된 해를 찾습니다. 물론 엑셀의 해찾기와 같은 기능을 사용하여 간단한 형태의 최적화 구현은 가능하지만, 데이터가 방대해질 경우에는 속도가 지나치게 느려져 효율적이지 않습니다. 동일한 최적화 방법을 지속적으로 사용할 경우에도 프로그래밍을 통해 함수를 만들고, 입력 변수만 변경하는 것이 훨씬 효율적인 방법입니다.

최적화 프로그래밍을 처음 해보시는 분들은 해당 내용들이 다소 어려울 수 있습니다. 그러나 이는 익숙하지 않은 것이지 결코 어려운 내용이 아닙니다. 코드를 한줄씩 직접 입력하고 각 입력변수와 결과물들이 의미하는 바를 이해한다면, 금방 익숙해질 것입니다. 또한 포트폴리오 최적화에 관한 좋은 패키지들이 이미 많이 나와 있으므로, 대략적인 내용만 이해하고 실제 구현은 패키지를 이용하는 것도 유용한 방법이기에 이에 대해서도 다룰 것입니다.

본 장에서는 일반적으로 많이 쓰이는 **최소분산 포트폴리오Minimum Variance Portfolio, 최대분산효과 포트폴리오Maximum Diversification portfolio, 위험균형 포트폴리오Risk Parity Portfolio**를 구현해보도록 하며, 예제로는 글로벌 자산들의 일별 수익률을 사용[[30]](#endnote-30)합니다. 포트폴리오 최적화를 위해서는 분산-공분산 행렬이 필요하며, 이는 R에서 기본적으로 제공되는 stats 패키지의 **cov()** 함수를 통해 손쉽게 구할 수 있습니다. csv 파일에서 수익률 데이터를 불러온 후 이를 구하는 방법은 아래와 같습니다.

|  |
| --- |
| ret = read.csv("ret\_allocation.csv", row.names = 1)  covmat = cov(ret) |

**[1] 최적화 기본: 뉴튼 방법**

**[2] 최소분산 포트폴리오**

최소분산 포트폴리오Minimum Variance Portfolio는 포트폴리오의 변동성이 최소가 되는 지점을 찾아줍니다. 포트폴리오의 변동성은 일반적으로 의 형태로 표현되지만, 최적화 작업을 위해서는 행렬의 형태인 로 표현하는 것이 더욱 편리합니다. 이 중 는 각 자산들의 비중을 행렬의 형태로 나타낸 것이며, 는 각 자산 별 분산-공분산 행렬을 나타낸 것입니다. 분산-공분산 행렬은 사전에 고정되어 있는 값이므로, 각 자산들의 비중인 를 변화시킴으로써 포트폴리오의 변동성이 최소인 지점을 찾을 수 있습니다.

최소분산 포트폴리오의 목적함수는 아래의 수식으로 표현할 수 있습니다. 이 중 은 단지 미분했을 때 계산을 용이하게 하기 위한 장치일 뿐 결과에는 영향을 미치지 않습니다.

|  |
| --- |
| 최소분산 포트폴리오의 목적함수: |

다만 단순히 위의 목적함수를 찾는 해를 구할 경우 결과 값이 음수가 나오기도 하며, 이는 공매도를 의미합니다. 일반적으로 공매도가 불가능하다는 점과, 투자 비중의 합이 100%가 되어야 한다는 점을 고려하면 아래와 같은 제약조건을 추가해야 합니다.

|  |
| --- |
|  |

물론 이 외에도 각 섹터의 투자비중 합에 대한 제약, 회전율에 대한 제약 등도 추가할 수 있습니다. 그러나 이는 코드가 지나치게 복잡해지며, 정확한 계산을 위해서는 좀더 전문화된 프로그램을 사용할 필요가 있으므로, 해당 책에서는 다루지 않도록 하겠습니다.

**slsqp() 함수를 이용한 최소분산 포트폴리오 구현**

R에서 가장 손쉽게 최적화 작업을 수행하는 방법은 nloptr 패키지에서 제공하는 **slsqp()** 함수를 이용하는 법입니다. 해당 함수는 순차적 이차 계획Sequential quadratic programming을 이용하여 해를 찾으며, 목적함수와 제약조건은 아래와 같습니다.

|  |  |
| --- | --- |
| **목적함수** | **제약조건** |
|  |  |

목적함수에서 는 최소화 하고자 하는 값, 즉 포트폴리오의 변동성입니다. 제약조건은 크게 개별 자산의 투자 비중이 0 이상인 것과, 투자 비중의 합이 1이 되도록 하는 것입니다. 첫번째 제약조건은 자연스럽게 개별 자산의 투자 비중이 0 이상인 것을 의미합니다. 두 번째 제약조건은 약간의 변형을 통해 투자 비중의 합이 1이 되는 제약조건을 만들 수 있습니다. 를 ‘투자 비중의 합 – 1’ 로 변형할 경우, -1을 우변으로 넘기면 결국 ‘투자 비중의 합 = 1’ 의 형태로 나타낼 수 있습니다. **slsqp()** 함수의 사용법은 아래와 같습니다.

|  |
| --- |
| slsqp(x0, fn, gr = NULL, lower = NULL, upper = NULL,  hin = NULL, hinjac = NULL, heq = NULL, heqjac = NULL,  nl.info = FALSE, control = list(), ...) |

이 중 우리가 구체적으로 입력해야 할 값은 x0, fn, hin, heq 항목 입니다. x0는 초기 시작값으로써, 일반적으로 모든 x에 대해 동일한 값을 입력합니다. fn은 최소화 하고자 하는 목적함수로써, 포트폴리오의 변동성이 해당됩니다. hin은 부등위 제약조건inequality constraints을 의미하며, 프로그래밍 내에서는 hin>=0 로 인식합니다. 각 자산의 비중이 0보다 크다는 제약조건과 연결됩니다. 마지막으로 heq는 등위 제약조건equality constraints을 의미하며 프로그래밍 내에서는 heq==0을 의미합니다. ‘투자 비중의 합 - 1’의 형태를 입력 할 경우, 투자 비중의 합이 1이 라는 제약조건과 연결됩니다.

해당 함수에 대한 더욱 자세한 내용은 nloptr 패키지의 설명서를 통해 확인할 수 있습니다. 아래의 표는 최소분산 포트폴리오를 구할 때 필요한 주요 변수에 대한 내용입니다.

|  |  |  |
| --- | --- | --- |
| **변수명** | **내용** | **포트폴리오 내 변수** |
| x0 | 초기값 |  |
| fn | 목적함수 | 포트폴리오 변동성 |
| hin | 부등위 제약조건 | 각 자산의 비중이 0 보다 큰 제약조건 |
| heq | 등위 제약조건 | 투자 비중의 합이 1인 제약조건 |

**slsqp()** 함수를 이용하여 최소분산 포트폴리오를 만족하는 자산의 투자 비중을 구하는 과정은 다음과 같습니다. 먼저 fn, hin, heq에 해당하는 함수들을 각각 만들어 준 후, 이를 **slsqp()** 함수와 결합하여 최적화된 결과값을 얻을 수 있습니다. 구체적인 과정은 아래와 같습니다.

|  |
| --- |
| objective = function(w) {  obj = t(w) %\*% covmat %\*% w  return(obj)  } |

먼저 목적함수에 해당하는 부분입니다. covmat은 사전에 계산된 분산-공분산 행렬이며, w는 각 자산의 투자 비중입니다. obj는 포트폴리오의 변동성인 를 계산한 것입니다. 즉, 해당 함수는 계산된 w를 바탕으로 포트폴리오의 변동성을 반환하고, 우리의 목적은 해당 값이 최소가 되도록 하는 것입니다.

|  |
| --- |
| hin.objective = function(w) {  return(w)  } |

각 자산의 비중이 0 보다 큰 제약조건에 해당하는 부등위 제약조건입니다. hin>=0 의 형태로 인식을 하므로, 계산된 비중인 w를 단순히 입력하기만 하면 됩니다.

|  |
| --- |
| heq.objective = function(w) {  sum\_w = sum(w)  return( sum\_w - 1 )  } |

투자 비중의 합이 1인 제약조건에 해당하는 등위 제약조건입니다. 먼저 계산된 비중인 w들의 합계를 구한 후, 해당 값에서 1을 빼주는 값을 반환하도록 합니다. 프로그래밍 내에서는 heq==0 의 형태로 인식을 하므로, 결국 (sum\_w – 1) == 0, 즉 sum\_w == 1 의 제약조건과 동일하게 됩니다.

|  |
| --- |
| library(nloptr)  result = slsqp( x0 = x0 = rep(0.1, 10),  fn = objective,  hin = hin.objective,  heq = heq.objective) |

먼저 **slsqp()** 함수를 실행하기 위해 nloptr 패키지를 연 후, 위에 입력된 내역들을 바탕으로 최적화 작업을 실행합니다. 초기값인 x0에는 먼저 동일한 비중들을 입력합니다. 예제에서는 종목이 10개 이므로, x0값에는 rep(0.1, 10)의 결과값인 (0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1) 값이 입력됩니다. 최소화 하고자 하는 목적함수에는 위에서 구성한 objective 함수를 입력합니다. 부등위 제약조건과 등위 제약조건에도 각각 위에서 구성한 hin.objective 함수와 heq.objective를 입력합니다.

즉, 해당 함수는 초기값을 시작점으로 하여 주어진 제약조건을 만족하는 해를 찾기 위해 w값들을 조정하는 작업을 반복한 후, 목적함수가 최소가 되는 지점의 w를 반환합니다. 위의 코드를 실행한 후 result를 입력하면 다음과 같은 결과값을 확인할 수 있습니다.

|  |
| --- |
| > result  $par  [1] 9.041922e-02 1.003046e-02 1.270211e-02 -2.674359e-18 6.492206e-18 7.565467e-01 2.032515e-18  [8] 4.019395e-02 3.474087e-02 5.536672e-02  $value  [1] 7.011663e-06  $iter  [1] 113  $convergence  [1] 4  $message  [1] "NLOPT\_XTOL\_REACHED: Optimization stopped because xtol\_rel or xtol\_abs (above) was reached." |

먼저 $par는 최적화된 지점의 해를 의미하며, 여기서는 최소분산 포트폴리오를 구성하는 자산들의 투자 비중을 의미합니다. $value는 $par에서 산출된 값을 목적함수 fn에 입력하였을 때 나오는 결과값으로써, 여기서는 포트폴리오의 분산을 의미합니다.

|  |
| --- |
| w = result$par  w = round(w, 4) |

가장 중요한 자산들의 투자비중은 result$par 를 통해 추출할 수 있습니다. 또한, 결과값이 9.041922e-02처럼 가독성이 떨어지는 지수 표시형식이므로, 가독성을 위해 round() 함수를 이용하여 반올림을 하여 줍니다. 결과적으로 w에는 다음과 같은 값이 입력됩니다.

|  |
| --- |
| > w  [1] 0.0904 0.0100 0.0127 0.0000 0.0000 0.7565 0.0000 0.0402 0.0347 0.0554 |

**solve.QP() 함수를 이용한 최소분산 포트폴리오 구현**

다음으로는 quadprog 패키지 내의 solve.QP() 함수를 이용하여 포트폴리오 최적화를 하는 방법이 있습니다. 해당 함수는 쌍대기법Dual Method를 이용하여 제약조건 내에서 목적함수가 최소화 되는 해를 구합니다.[[31]](#endnote-31) 해당 함수의 목적함수와 제약조건은 아래와 같이 표현할 수 있습니다.

|  |  |
| --- | --- |
| **목적함수** | **제약조건** |
|  |  |

최소분산 포트폴리오의 목적함수가 로 표시된다는 점을 생각하면, 해당 함수는 매우 이해하기 쉽게 구성되어 있습니다. b를 각 개별 자산의 투자 비중인 w, D를 분산-공분산 행렬인 라 생각하면, 목적함수인 는 최소분산 포트폴리오의 목적함수와 정확히 동일합니다. d를 0으로 생각할 경우 또한 0이 되어 목적함수에 아무런 영향도 미치지 않습니다. 제약조건 역시 항목을 적절하게 수정하여 준다면, 개별 자산의 투자 비중이 0 이상인 것과, 투자 비중의 합이 1이 되도록 만들 수 있습니다. 이에 대해서는 뒤에서 구체적으로 다루도록 합니다. solve.QP() 함수의 사용법은 아래와 같습니다.

|  |
| --- |
| solve.QP(Dmat, dvec, Amat, bvec, meq=0, factorized=FALSE) |

먼저 Dmat은 목적함수 중 D에 해당하는 행렬부분으로써, dvec은 목적함수 중 에 해당하는 벡터부분입니다. Amat은 제약조건 중 에 해당하는 부분으로써, 제약조건 중 좌변에 위치하는 항목입니다. 주의할 점은 제약조건에서 보듯이 제약조건 행렬을 구한 후 이의 전치Transpose행렬을 입력하여야 합니다. bvec은 제약조건 중 에 해당하는 부분으로써, 부등위 제약조건 중 우변에 위치하는 항목입니다. meq는 bvec의 몇번째 까지를 등위 제약조건으로 설정할지에 대한 부분입니다. 각 변수를 입력한 후 함수를 실행하면, 위의 목적함수와 제약조건을 만족하는 b값을 찾습니다.

solve.QP() 함수의 경우 slsqp()함수 보다 직관적으로 이해하기는 어려운 점이 있지만, Amat과 bvec에 해당하는 부분만 변경하여 주면 복잡한 형태의 제약조건도 다룰 수 있다는 장점이 있습니다. 해당 함수에 대한 더욱 자세한 내용은 quadprog 패키지의 설명서를 통해 확인할 수 있으며. 아래의 표는 최소분산 포트폴리오를 구할 때 필요한 주요 변수에 대한 내용입니다.

|  |  |  |
| --- | --- | --- |
| 변수명 | 내용 | 포트폴리오 내 변수 |
| Dmat | 목적함수 중 D | 분산-공분산 행렬 |
| dvec | 목적함수 중 d | 해당사항 없음 |
| Amat | 제약조건 (좌변) | , |
| bvec | 제약조건 (우변) | 각 비중의 비중이 0 보다 큼, 비중의 합이 1 |
| meq | 등위 제약조건 개수 | 1개 |

solve.QP() 함수를 이용하여 최소분산 포트폴리오 비중을 구할 때는 Amat 항목을 제대로 입력하는 것이 가장 중요하며, 나머지 항목은 매우 손쉽게 입력이 가능합니다. 설명된 내용에 해당하는 행렬을 손으로 직접 써가며 계산해 보신다면 훨씬 이해하기가 쉬울 것입니다. 구체적인 과정은 아래와 같습니다.

|  |
| --- |
| Dmat = covmat  dvec = rep(0, 10) |

위에서 언급한 대로 Dmat에는 분산-공분산 행렬을 입력하며, dvec은 최소분산 포트폴리오를 구하는데는 필요한 값이 아니므로 0벡터를 입력[[32]](#endnote-32)합니다.

|  |
| --- |
| Amat = t(rbind(rep(1, 10), diag(10), -diag(10)))  bvec = c(1, rep(0, 10), -rep(1, 10))  meq = 1 |

먼저, 등위 제약조건과 부등위 제약조건을 행렬의 형태로 표현하면 다음과 같습니다.

이 중, 의 전치행렬 값이 제약조건의 좌변인 Amat, 부분이 제약조건의 우변인 bvec을 의미합니다. 해당 부분은 크게 투자 비중의 합이 1인 제약조건을 의미하는 최소 투자 비중이 0보다 크거나 같은 제약조건을 의미하는 , 최대 투자 비중이 1보다 작거나 같은 제약조건을 의미하는 , 총 3부분으로 나눌 수 있습니다. solve.QP() 함수의 제약조건은 항상 좌변이 큰 형태이므로, 최대 투자 비중에 대한 제약 조건의 경우 에서 양변에 (-)을 곱하여 부등호를 맞춰주었습니다.

첫번째 제약조건인 은 부등호가 아닌 등호, 즉 투자 비중의 합이 1인 조건을 의미하므로 meq는 1, 즉 첫번째 제약조건은 등식 제약조건임을 나타내었습니다.

주의해야 할 점은, 제약조건이 로 나타나 있으므로, Amat은 위에서 보여지는 행렬의 전치행렬의 형태를 입력해야 한다는 점입니다.

먼저 형태의 행렬을 만들어 주어야 합니다. 행렬은 rep()함수를 통해 생성이 가능하며, 과 행렬은 diag() 함수를 이용하여 쉽게 만들 수 있습니다. diag(10) 명령어를 입력하면 10 X 10에 해당하는 대각행렬을 만들어 줍니다. 이를 rbind(), 즉 행의 형태로 묶는다면 제약조건의 좌변과 동일한 형태의 행렬을 만들 수 있으며, 전치행렬을 만들어주는 t() 함수를 이용하여 행렬을 Amat에 입력해 줍니다.

|  |
| --- |
| library(quadprog)  result = solve.QP(Dmat, dvec, Amat, bvec, meq) |

먼저 solve.QP() 함수를 실행하기 위해 quadprog 패키지를 연 후, 위에 입력된 내역들을 바탕으로 최적화 작업을 실행합니다. 위의 코드를 실행한 후 result를 입력하면 다음과 같은 결과값을 확인할 수 있습니다.

|  |
| --- |
| > result  $solution  [1] 9.041922e-02 1.003046e-02 1.270211e-02 -8.728935e-22 -5.838166e-18 7.565467e-01  [7] 0.000000e+00 4.019395e-02 3.474087e-02 5.536672e-02  $value  [1] 3.505831e-06  $unconstrained.solution  [1] 0 0 0 0 0 0 0 0 0 0  $iterations  [1] 5 0  $Lagrangian  [1] 7.011663e-06 0.000000e+00 0.000000e+00 0.000000e+00 8.989427e-07 4.337309e-06  [7] 0.000000e+00 1.294355e-07 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00  [13] 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00  [19] 0.000000e+00 0.000000e+00 0.000000e+00  $iact  [1] 1 6 5 8 |

$solution은 최적화된 지점의 해, 즉 최소분산 포트폴리오를 구성하는 자산들의 투자 비중을 의미합니다. $value는 $solution에서 산출된 값을 목적함수에 입력하였을 때 나오는 결과값으로써, 포트폴리오의 분산을 의미합니다.

|  |
| --- |
| w = result$solution  w = round(w, 4) |

result$solution 명령어를 통해 자산들의 투자비중을 추출합니다. 또한, 가독성을 위해 round() 함수를 이용하여 반올림을 하여 줍니다. 결과적으로 w에는 다음과 같은 값이 입력됩니다.

|  |
| --- |
| > w  [1] 0.0904 0.0100 0.0127 0.0000 0.0000 0.7565 0.0000 0.0402 0.0347 0.0554 |

**optimalPortfolio 함수를 이용한 최소분산 포트폴리오 구현**

RiskPortfolios 패키지 내 optimalPortfolio() 함수를 이용하여 매우 간단하게 최적화 포트폴리오를 구현할 수도 있습니다. 해당 함수의 사용법은 아래와 같습니다.

|  |
| --- |
| optimalPortfolio(Sigma, mu = NULL, semiDev = NULL, control = list()) |

Sigma는 분산-공분산 행렬입니다. mu와 semiDev는 각각 기대수익률과 세미 편차semi deviation로써, 입력하지 않아도 됩니다. control은 포트폴리오 종류 및 제약조건에 해당하는 변수입니다. control에 해당하는 요소는 다음과 같습니다.

|  |  |  |
| --- | --- | --- |
| type | ‘minvol’ | 최소분산 포트폴리오 |
| ‘invvol’ | 역변동성 포트폴리오 |
| ‘erc’ | 위험 균형 포트폴리오 |
| ‘maxdiv’ | 최대 분산효과 포트폴리오 |
| ‘riskeff’ | 위험-효율적 포트폴리오 |
| constraint | ‘lo’ | 최소 투자 비중이 0보다 클 것 |
| ‘user’ | 최소 투자 비중(LB) 및 최대 투자 비중(UB) 설정 |

control 항목에서 원하는 포트폴리오 타입과 제약조건을 입력해주면, 매우 손쉽게 최적화 포트폴리오를 구현할 수 있습니다.

|  |
| --- |
| library(RiskPortfolios)  w = optimalPortfolio(covmat, control = list(type = 'minvol', constraint = 'lo')) |

먼저 optimalPortfolio() 함수를 실행히기 위해 RiskPortfolios 패키지를 열어줍니다. type에는 최소분산 포트폴레오에 해당하는 ‘minvol’을 입력합니다. constraint에는 각 자산의 비중이 0보다 큰 제약조건인 ‘lo(Long Only)’를 입력하며, 비중의 합이 1인 제약조건은 자동적으로 적용이 됩니다. w에는 최소분산 포트폴리오를 만족하는 자산들의 투자비중이 입력되어 있습니다.

|  |
| --- |
| > w  [1] 0.09041922 0.01003046 0.01270211 0.00000000 0.00000000 0.75654666 0.00000000  [8] 0.04019395 0.03474087 0.05536672 |

이처럼 패키지를 이용할 경우 매우 훨씬 간단하게 원하는 값을 얻을 수 있습니다. 해당 함수의 코드[[33]](#endnote-33)를 살펴보면 solve.QP()를 이용하여 작성한 방법과 거의 동일합니다. 따라서 위의 과정들을 대략적으로 이해한 후, 패키지를 사용하여 포트폴리오 최적화를 구현하는 것이 현명한 방법이 될 수도 있습니다.

**결과값들의 비교**

아래 표는 (1) slsqp(), (2) solve.QP(), (3) optimalPortfolio()를 이용하여 구한 해들의 비교입니다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| (1) | 0.0904 | 0.0100 | 0.0127 | 0.0000 | 0.0000 | 0.7565 | 0.0000 | 0.0402 | 0.0347 | 0.0554 |
| (2) | 0.0904 | 0.0100 | 0.0127 | 0.0000 | 0.0000 | 0.7565 | 0.0000 | 0.0402 | 0.0347 | 0.0554 |
| (3) | 0.0904 | 0.0100 | 0.0127 | 0.0000 | 0.0000 | 0.7565 | 0.0000 | 0.0402 | 0.0347 | 0.0554 |

3가지 방법 모두 동일한 해를 가지는 것이 확인됩니다. 그러나 여기서 나온 결과를 이용하여 그대로 투자하기에는 문제가 있습니다. 일부 자산은 투자비중이 0%, 즉 전혀 투자를 하지 않으며, 특정 자산에 75%를 투자를 하는, 일부 종목에 지나치게 편중된 결과가 나옵니다. 이러한 구석해Corner Solution 문제를 해결하기 위해 각 자산의 최소 및 최대 투자 비중 제약조건을 추가해 주기도 합니다.

**최소 및 최대 투자비중 제약조건의 추가**

구석해 문제를 방지하고, 모든 자산에 골고루 투자하기 위해 개별 투자비중을 최소 5%, 최대 20%로 하는 제약조건을 추가해 주도록 하겠습니다. 먼저 slsqp()에서 제약조건을 추가하는 방법은 다음과 같습니다.

|  |
| --- |
| result = slsqp( x0 = rep(0.1, 10),  fn = objective,  hin = hin.objective,  heq = heq.objective,  lower = rep(0.05, 10),  upper = rep(0.20, 10))  w = result$par  w = round(w, 4)  > w  [1] 0.0632 0.0500 0.0500 0.0500 0.2000 0.2000 0.0500 0.0500 0.1920 0.0948 |

입력값의 마지막에 lower와 upper 항목을 추가로 입력합니다. lower와 upper를 입력할 경우, 해당 값 사이에서 최적화를 만족하는 해를 찾게 되며, 해당 예에서는 5%와 20% 사이에서 해를 찾게 됩니다. 결과적으로 w에는 다음과 같은 값이 입력됩니다. 추가로 입력한 제약조건에 맞게, 최소 투자비중은 5%이며, 최대 투자비중은 20%임을 확인할 수 있습니다.

다음은 solve.QP() 함수 내에서 제약조건을 추가하는 방법입니다. 해당 함수 역시 다른 입력변수는 모두 동일하며, 제약조건의 우변에 해당하는 bvec 항목만 수정하여 주면 됩니다. 기존 제약조건은 최소, 최대 투자비중이 (0, 1)이며, 행렬의 형태로는 입니다. 변경된 제약조건인 최소, 최대 투자비중인 (0.05, 0.20)을 적용하면 로 표현됩니다. 변경된 제약조건을 적용한 식은 다음과 같습니다.

|  |
| --- |
|  |

|  |
| --- |
| Dmat = covmat  dvec = rep(0, 10)  Amat = t(rbind(rep(1, 10), diag(10), -diag(10)))  bvec = c(1, rep(0.05, 10), -rep(0.20, 10))  meq = 1  result = solve.QP(Dmat, dvec, Amat, bvec, meq)  w = result$solution  w = round(w, 4) |

bvec 항목을 제외한 모든 코드는 기존과 동일하며, 조건함수의 우변인 bvec만 각각 최소 투자비중과 최대 투자비중이 (0, 1)에서 (0.05, 0.20)으로 변경되었습니다. 결과적으로 w에는 다음과 같은 값이 입력되며, 추가적인 투자비중 제약이 잘 적용되었음이 확인됩니다. solve.QP() 함수를 이용하여 최적화를 수행할 경우, 위의 예시처럼 bvec 값을 변경하여 다양한 제약조건을 추가할 수 있습니다.

|  |
| --- |
| > w  [1] 0.0632 0.0500 0.0500 0.0500 0.2000 0.2000 0.0500 0.0500 0.1920 0.0948 |

마지막으로 optimalPortfolio() 함수 내에서 최소 및 최대 투자비중을 추가하는 방법입니다. 입력변수의 control 항목 중 constraint 부분을 간단하게 수정합니다.

|  |
| --- |
| w = optimalPortfolio(covmat, control = list(type = 'minvol', constraint = 'user',  LB = rep(0.05, 10), UB = rep(0.20, 10))) |

기존 constraint의 경우 constraint 부분에 롱온리 제약조건에 해당하는 ‘lo’를 입력하였다면, 이번에는 직접 제약값들을 입력할 수 있는 ‘user’를 입력합니다. 그리고 LB에는 최소 투자비중 벡터를, UB에는 최대 투자비중 벡터를 입력합니다. w에는 역시나 추가된 제약조건이 적용된 투자비중이 입력되어 있습니다.

|  |
| --- |
| > w  [1] 0.06319557 0.05000000 0.05000000 0.05000000 0.20000000 0.20000000 0.05000000  [8] 0.05000000 0.19204934 0.09475509 |

아래 표는 제약조건을 추가하여 (1) slsqp(), (2) solve.QP(), (3) optimalPortfolio() 함수로 구한 해들의 비교입니다. 3가지 방법 모두 동일한 해를 가지며 최소 및 최대 투자비중도 각각 5%와 20%로 제한되어 구석해 문제 또한 해결되었음이 확인됩니다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| (1) | 0.0632 | 0.0500 | 0.0500 | 0.0500 | 0.2000 | 0.2000 | 0.0500 | 0.0500 | 0.1920 | 0.0948 |
| (2) | 0.0632 | 0.0500 | 0.0500 | 0.0500 | 0.2000 | 0.2000 | 0.0500 | 0.0500 | 0.1920 | 0.0948 |
| (3) | 0.0632 | 0.0500 | 0.0500 | 0.0500 | 0.2000 | 0.2000 | 0.0500 | 0.0500 | 0.1920 | 0.0948 |

**자산 별 서로 다른 제약조건 추가**

투자 규모가 크지 않을 경우에는 위에서 추가한 제약조건 만으로도 충분히 훌륭한 포트폴리오가 구성됩니다. 그러나 투자 규모가 커질 경우 추가적인 제약조건들을 고려해야 할 경우가 생깁니다. 벤치마크 비중과의 괴리로 인한 추적오차Tracking Error을 고려해야 할 수도 있고, 투자 대상 별 거래량을 고려한 제약조건을 추가해야 할 경우도 있습니다.

기존 제약조건에는 자산 별로 동일한 최소(5%) 및 최대(20%) 투자비중 제약조건을 다루었지만, 자산 별로 상이한 제약조건이 필요할 경우도 있습니다. solve.QP() 함수의 bvec 부분을 간단하게 수정하여 주면 이 또한 어렵지 않게 구현이 가능합니다.

아래 표는 새롭게 설정하고자 하는 각 자산 별 최소 및 최대 투자비중 값과, 변경된 제약조건을 행렬의 형태로 나타낸 것입니다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| 최소 | 0.10 | 0.10 | 0.05 | 0.05 | 0.10 | 0.10 | 0.05 | 0.05 | 0.03 | 0.03 |
| 최대 | 0.25 | 0.25 | 0.20 | 0.20 | 0.20 | 0.20 | 0.10 | 0.10 | 0.08 | 0.08 |

|  |
| --- |
|  |

|  |
| --- |
| Dmat = covmat  dvec = rep(0, 10)  Amat = t(rbind(rep(1, 10), diag(10), -diag(10)))  bvec = c(1, c(0.10, 0.10, 0.05, 0.05, 0.10, 0.10, 0.05, 0.05, 0.03, 0.03),  -c(0.25, 0.25, 0.20, 0.20, 0.20, 0.20, 0.10, 0.10, 0.08, 0.08))  meq = 1  result = solve.QP(Dmat, dvec, Amat, bvec, meq)  w = result$solution  w = round(w, 4) |

역시나 bvec 항목을 제외한 모든 코드는 기존과 동일하며, 조건함수의 우변인 bvec만 자산 별 제약조건에 해당하는 값을 입력하였습니다. w를 확인해 보면, 결과 값들이 모두 최소 및 최대 투자비중 제약조건 내에 위치함을 확인할 수 있습니다.

|  |
| --- |
| > w  [1] 0.102 0.100 0.050 0.050 0.200 0.200 0.050 0.088 0.080 0.080 |

**그룹 별 제약조건**

자산 별로 다른 제약조건이 아닌, 자산 그룹별로 다른 제약조건이 추가될 경우도 있습니다. 예를 들어 주식 자산에의 투자 한도 제한, 채권 자산에의 투자 한도 제한과 같은 조건이 있을 수 있습니다. 주식 포트폴리오의 경우에서 각 섹터에 대한 투자 한도 제한은 이미 펀드 내에서 일반적으로 사용되고 있습니다.

자산 별 최소 및 최대 투자 비중 제약조건인 5%와 20%에 다음과 같은 그룹 제약조건을 추가해 보도록 합니다. 이는 1~4번 자산 비중의 합은 40% 이상과 60% 이하, 5~6번 자산 비중의 합은 20% 이상과 40% 이하, 7~10번 자산 비중의 합은 10% 이상과 30% 이하인 조건을 의미합니다.

|  |
| --- |
|  |

그룹 별 제약조건은 solve.QP() 함수의 제약조건의 좌변에 해당하는 Amat 항목을 먼저 수정해 주어야 합니다. 그 후 그룹별 투자비중의 제약조건은 개별 투자비중의 제약조건과 동일하게 bvec 항목을 수정하면 됩니다. (1)번 그룹조건의 는 다음과 같은 형태로 나타낼 수도 있습니다.

|  |
| --- |
|  |

이 중 행렬을 라 두면, (1)번 그룹의 최소 및 최대 비중 제약조건은 다음과 같이 표현할 수 있습니다.

|  |  |
| --- | --- |
| **최소 비중 제약조건** | **최대 비중 제약조건** |
|  |  |

는 i번째 그룹 제약조건에서 그룹에 해당하는 열은 1, 아닌 곳은 0으로 이루어진 행벡터 입니다. 따라서 그룹 제약조건은 조건문으로 이루어진 행백터를 최소 제약조건에는 (+)형태로, 최대 제약조건에는 (-)형태로 와 곱한 형태입니다. 기존 최소 및 최대 제약조건에 추가로 입력할 그룹 제약조건을 행렬로 표현하면 다음과 같습니다.

|  |
| --- |
|  |

부분은 기존 Amat 행렬에, 부분은 기존 bvec 부분에 각각 추가로 입력해 주면 그룹 제약조건이 추가됩니다. 주의해야 할 사항은 Amat 부분에 추가로 입력할 경우 전치 행렬의 형태로 입력해야 한다는 점입니다. 이를 코드로 나타내면 다음과 같습니다.

|  |
| --- |
| Dmat = covmat  dvec = rep(0, 10)  Amat = t(rbind(  rep(1, 10), diag(10), -diag(10),  rbind(c(1,1,1,1,0,0,0,0,0,0), c(0,0,0,0,1,1,0,0,0,0), c(0,0,0,0,0,0,1,1,1,1)),  rbind(c(-1,-1,-1,-1,0,0,0,0,0,0), c(0,0,0,0,-1,-1,0,0,0,0), c(0,0,0,0,0,0,-1,-1,-1,-1))  )  )  bvec = c(1, rep(0.05, 10), -rep(0.20, 10),  0.3, 0.3, 0.0, -0.7, -0.5, -0.4)  result = solve.QP(Dmat, dvec, Amat, bvec, meq)  w = result$solution  w = round(w, 4) |

기존 Amat과 bvec 부분에 그룹 제약조건 부분을 추가로 입력해 줍니다. w를 확인해 보면, 결과 값들이 모두 최소 및 최대 투자비중 제약조건 내에 위치하며, 각 그룹별 제약조건 역시 만족함을 확인할 수 있습니다.

|  |
| --- |
| > w  [1] 0.15 0.05 0.05 0.05 0.20 0.20 0.05 0.05 0.15 0.05 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| 비중 | 0.15 | 0.05 | 0.05 | 0.05 | 0.20 | 0.20 | 0.05 | 0.05 | 0.05 | 0.05 |
| 제약 | (0.3, 0.7) | | | | (0.3, 0.5) | | (0.0, 0.4) | | | |
| 합계 | 0.4 | | | | 0.4 | | 0.2 | | | |

Amat 부분과 bvec 부분의 수정을 통해, 자산 별 비중과 그룹 별 비중 제약조건을 동시에 입력할 수도 있습니다. 실제 펀드에서 운용되는 최소분산 포트폴리오는 해당 제약조건이 반영된 결과값을 사용하고 있습니다. FTSE에서 제공하는 ‘Korea Minimum Variance Index’[[34]](#endnote-34)의 제약조건을 살펴보면 이를 알 수 있습니다.

|  |
| --- |
| 1. 종목 별 최소 및 최대 투자비중: 0.00%, 0.05% 2. 투자비중 합계: 100% 3. 최대 투자비중(시가총액 대비): 종목 별 벤치마크 시가총액 대비 20배 4. 섹터 투자비중: 벤치마크 대비 ±20% |

(1)번과 (2)번 제약조건의 경우 가장 기본적인 투자비중 제약조건 입니다. (3)번의 경우 종목 별로 시가총액의 비중이 다르므로, 서로 다른 제약조건과 동일합니다. (4)번의 경우 각 섹터 별 투자비중의 최소 및 최대 비중에 대한 조건이므로, 이는 그룹 제약조건과 동일합니다.

**[3] 최대분산효과 포트폴리오**

앞서 설명했듯이 포트폴리오의 변동성은 형태로 나타나며, 이는 다음과 같이 표현할 수도 있습니다.

|  |
| --- |
|  |

이 중 부분에는 자산 간 상관관계가 포함되어 있습니다. 상관관계는 -1과 1 사이(에 위치하며, 자산 간 상관관계가 낮을수록 포트폴리오의 변동성 역시 낮아지게 됩니다.

아래 예제는 비중과 변동성이 각각 50%, 9%로 동일할 두 자산을 대상으로, 상관관계에 따른 포트폴리오의 변동성입니다. 상관관계가 1, 즉 두 자산이 완벽하게 동일한 경우에는 포트폴리오의 변동성은 개별 자산 변동성의 가중합과 같습니다. 그러나 상관관계가 낮아질수록 포트폴리오의 변동성 또한 점차 낮아집니다. 이러한 효과를 투자에서는 ‘분산효과’라 합니다.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | **A** | | | **B** | |
| 비중 | | | 50% | | | 50% | |
| 변동성 | | | 9% | | | 9% | |
|  | | | | | | | |
| 상관관계 | 1.0 | 0.5 | | 0.0 | -0.5 | | -1.0 |
| 포트폴리오 변동성 | 0.09 | 0.0675 | | 0.045 | 0.0225 | | 0 |

이러한 분산효과의 정도를 측정하는 지표가 분산 비율DR(Diversification Ratio) 입니다. 분산 비율의 분자는 개별 변동성의 가중합이며, 분모는 포트폴리오의 변동성입니다. 이를 수식으로 나타내면 다음과 같습니다.

|  |
| --- |
|  |

모든 자산 간의 상관관계가 1일 경우, 위의 예시에서 살펴본 것과 같이 포트폴리오의 변동성은 개별 자산 변동성의 가중합과 같아지게 됩니다. 즉, 조건이 되어, 분산 비율은 1이 됩니다. 그러나 대부분의 경우에서 자산 간의 상관관계는 1보다 낮으며, 이로 인해 포트폴리오의 분산은 단순 가중합 보다 작아지게 되고, 이로 인해 분산 비율은 1보다 커지게 됩니다.

자산 간 상관관계가 낮은 종목을 위주로 포트폴리오를 구성할 수록 분산효과로 인해 포트폴리오의 변동성은 낮아지게 되고, 분산 비율은 점점 커지게 됩니다. 최대분산 포트폴리오Most Diversified Portfolio는 이를 이용하여 분산효과가 최대가 되는, 즉 분산 비율이 최대가 되는 포트폴리오를 구성하는 방법입니다. 이에 대한 목적함수와 제약조건은 다음과 같습니다.

|  |
| --- |
| 목적함수:  제약조건: |

최대분산효과 포트폴리오의 목적함수는 분산비율을 최대화하는데 있는 반면, 대부분의 최적화 프로그래밍은 목적함수를 최소화 하는 형태로 이루어집니다. 따라서 목적함수인 을 최소화 형태로 바꿀 필요가 있으며 이는 크게 Choueifaty Synthetic Asset Back-Transformation[[35]](#endnote-35)을 이용하는 방법, Duality[[36]](#endnote-36)를 이용하는 방법, Min (-)DR 방법, 3가지가 있습니다.

먼저 Choueifaty Synthetic Asset Back-Transformation 방법은 목적함수 와 제약조건 을 만족하는 자산 별 비중을 구합니다. 그 후, 구해진 비중을 각각의 표준편차로 나누어 주며, 비중의 합이 1이 되도록 표준화를 해줍니다. 이 중 주의할 점은 목적함수의 c가 우리가 지금까지 사용하던 분산-공분산 행렬이 아닌, 상관관계 행렬이라는 점입니다.

Duality 방법을 이용할 경우 목적함수는 최소분산 포트폴리오와 동일한 이며, 제약조건만 **,** 즉 개별 자산의 비중이 0보다 크고 개별 표준편차의 가중합이 1인 조건으로 바뀝니다. 그 후, 비중의 합이 1이 되도록 표준화를 해줍니다.

기존 두 방법이 수학적 증명에 의해 을 최소화의 형태로 풀어준 반면 간단하게 목적함수를 의 형태로 바꾸어 풀 수도 있습니다. 3가지 방법을 요약하면 다음과 같습니다.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **목적함수** | **제약조건** | **표준화** |
| Transformation |  |  | 비중을 각각의 표준편차로 나눈 후  비중의 합으로 표준화 |
| Duality |  |  | 비중의 합으로 표준화 |
| -DR |  |  | 불필요 |

**slsqp() 함수를 이용한 최대분산효과 포트폴리오 구현 (Transformation 방법)**

slsqp() 함수를 이용하여 Transformation 방법으로 최대분산효과 포트폴리오를 구현할 경우, 먼저 기존에 작성한 최소분산 포트폴리오의 코드에서 목적함수 부분만 새롭게 작성하며, 부등위 제약조건과 등위 제약조건은 동일합니다.

|  |
| --- |
| objective = function(w) {  corr = cov2cor(covmat)  obj = t(w) %\*% corr %\*% w  return(obj)  } |

Transformation 방법의 목적함수는 이므로 이에 맞게 코드를 작성해 줍니다.기존 최소분산 포트폴리오의 목적함수인에서 분산-공분산 행렬인 를 상관관계 행렬인로 바꾸어야 합니다. 상관관계 행렬은 R의 기본함수인 cov2cor()를 통해 간단히 구할 수 있습니다.

|  |
| --- |
| hin.objective = function(w) {  return(w)  }  heq.objective = function(w) {  sum\_w = sum(w)  return( sum\_w - 1 )  }  library(nloptr)  result = slsqp( x0 = rep(0.1, 10),  fn = objective,  hin = hin.objective,  heq = heq.objective) |

각 자산의 비중이 0 보다 큰 부등위 제약조건과, 투자 비중의 합이 1인 등위 제약조건은 기존 최소분산 포트폴리오와 동일합니다.

|  |
| --- |
| w = result$par  w = w / sqrt(diag(covmat)) |

result$par를 통해 최적 조건을 만족하는 해를 찾습니다. 그 후, 자산 별 비중을 각각의 표준편차로 나누어주어야 합니다. diag 함수는 행렬의 대각선 부분만을 추출합니다.

|  |
| --- |
|  |

의 경우 형태로 쓸 수 있으며, 을 적용하면 와 같습니다. 따라서 diag(covmat) 값에 제곱근을 계산하는 sqrt() 함수를 적용할 경우, 각각의 표준편차인 만 남게 됩니다. 이를 w로 나누어 줄 경우 분산 비율을 최대로 만드는 자산의 비중이 계산됩니다.

|  |
| --- |
| > w  US.Large Europe Japan Emerging  11.2718 2.2091 8.1096 0.0000  US.Long.Term US.Mid.Term US.REITs International.REITs  37.3717 28.9748 6.7867 7.7296  Gold Commodities  14.2968 14.5365 |

그러나 비중을 확인해 보면, 합계가 1을 훨씬 초과하게 됩니다. 따라서 비중의 합이 1이 되도록 표준화를 해줄 필요가 있습니다.

|  |
| --- |
| w = w / sum(w)  w = round(w, 4) |

자산 별 비중을 비중의 합으로 나누어 주도록 합니다. 가독성을 위해 반올림을 해준 후 w를 확인해 보면, 비중의 합이 1이 되었음이 확인됩니다.

|  |
| --- |
| > w  US.Large Europe Japan Emerging  0.0859 0.0168 0.0618 0.0000  US.Long.Term US.Mid.Term US.REITs International.REITs  0.2847 0.2207 0.0517 0.0589  Gold Commodities  0.1089 0.1107 |

**slsqp() 함수를 이용한 최대분산효과 포트폴리오 구현 (Duality 방법)**

slsqp() 함수를 이용하여 Duality방법으로 최대분산효과 포트폴리오를 구현할 경우, 목적함수는 최소분산 포트폴리오와 동일하며 등위 제약조건만 의 형태로 변경하여 주면 됩니다.

|  |
| --- |
| objective = function(w) {  obj = t(w) %\*% covmat %\*% w  return(obj)  }  hin.objective = function(w) {  return(w)  }  heq.objective = function(w) {  sum.prod = (w %\*% sqrt(diag(covmat))) - 1  return(sum.prod)  } |

목적함수인 objective 함수와, 부등위 제약조건인 hin.objective 함수는 최소분산 포트폴리오의 코드를 그대로 사용하며, 등위 제약조건을 Duality 방법의 제약조건인 에 맞추어 변경하여 줍니다. w %\*% sqrt(diag(covmat))를 통해 좌변을 계산할 수 있으며, 프로그래밍 내에서는 heq==0 형태로 인식하므로, 우변의 1을 좌변으로 이항해 줍니다.

|  |
| --- |
| result = slsqp( x0 = rep(0.1, 10),  fn = objective,  hin = hin.objective,  heq = heq.objective)  w = result$par  w = round(w, 4)  > w  [1] 11.2718 2.2091 8.1096 0.0000 37.3717 28.9748 6.7867 7.7296 14.2968 14.5365 |

slsqp() 함수를 통해 최적화를 실행하면, Duality 방법에 의해 계산된 최대분산효과 포트폴리오를 만족하는 해가 구해집니다. 이번 경우에도 비중의 합이 1을 초과하게 됩니다. 비중의 합이 1이 되도록 표준화를 해주도록 합니다.

|  |
| --- |
| w = w / sum(w)  w = round(w, 4) |

자산 별 비중을 비중의 합으로 나누어 주도록 합니다. 가독성을 위해 반올림을 해준 후 w를 확인해 보면, 비중의 합이 1이 되었음이 확인됩니다.

|  |
| --- |
| > w  [1] 0.0859 0.0168 0.0618 0.0000 0.2847 0.2207 0.0517 0.0589 0.1089 0.1107 |

**slsqp() 함수를 이용한 최대분산효과 포트폴리오 구현 (-DR 방법)**

slsqp() 함수를 이용하여 -DR방법으로 최대분산효과 포트폴리오를 구현할 경우, 먼저 기존에 작성한 최소분산 포트폴리오의 코드에서 목적함수 부분만 새롭게 작성하며, 부등위 제약조건과 등위 제약조건은 동일합니다.

|  |
| --- |
| objective = function(w) {  nom = w %\*% sqrt(diag(covmat))  denom = sqrt(t(w) %\*% covmat %\*% w)  dr = nom / denom  return(-dr)  } |

분산비율은 표준편차의 가중합을 포트폴리오의 표준편차로 나눈 값이며, 수식으로는 입니다. 분자인 nom에는 표준편차의 가중합을, 분모인 denom에는 포트폴리오의 표준편차를 입력하며, 이를 나눈 값을 dr에 입력합니다. 마지막으로 return을 통해 objective 함수는 –dr을 반환하도록 합니다.

|  |
| --- |
| hin.objective = function(w) {  return(w)  }  heq.objective = function(w) {  return( sum(w) - 1 )  }  result = slsqp( x0 = rep(0.1, 10),  fn = objective,  hin = hin.objective,  heq = heq.objective) |

모든 제약조건은 최소분산 포트폴리오와 동일하며, 해당 코드를 실행하면 –DR이 최소가 되는, 즉 DR이 최대가 되는 포트폴리오의 비중을 계산하게 됩니다.

|  |
| --- |
| w = result$par  w = round(w, 4) |

해당 방법에서는 등위 제약조건을 통해 비중의 합이 1인 조건을 입력하였으므로 w의 합은 1이 되며, 결과값을 조정할 필요가 없습니다.

|  |
| --- |
| > w  [1] 0.0859 0.0168 0.0618 0.0000 0.2847 0.2207 0.0517 0.0589 0.1089 0.1107 |

**solve.QP() 함수를 이용한 최대분산효과 포트폴리오 구현 (Duality 방법)**

최대분산효과 포트폴리오도 solve.QP() 함수를 통해 최적의 해를 구할 수 있습니다. Duality 방법에서 목적함수는 로 최소분산 포트폴리오와 동일하며, 제약함수는 **,** 입니다. 제약조건 부분인 Amat과 bvec 부분을 입력할 때 이 부분을 고려하여 입력해야 합니다.

|  |
| --- |
| Dmat = covmat  dvec = rep(0, 10) |

목적함수에 해당하는 Dmat과 dvec은 최소분산 포트폴리오와 동일합니다.

|  |
| --- |
| Amat = t(rbind(sqrt(diag(covmat)), diag(10)))  bvec = c(1, rep(0, 10))  meq = 1 |

제약조건에 해당하는 Amat 부분과 bvec 부분은 최소분산 포트폴리오와 다소 다릅니다. Duality 방법의 제약조건을 행렬의 형태로 표현하면 다음과 같습니다.

|  |
| --- |
|  |

1행의 는 과 같으며, 등위 제약조건의 개수를 의미하는 meq가 1이므로, 해당 식만이 등위제약조건, 즉 인 조건을 의미합니다. 2행부터 마지막 행까지는 모두 로써, 개별 자산의 투자 비중이 0보다 큰 조건을 의미합니다. 개별 자산의 투자 비중이 1보다 작은 조건은 Duality 방법에서는 입력하지 않아야 합니다.

|  |
| --- |
| library(quadprog)  result = solve.QP(Dmat, dvec, Amat, bvec, meq)  w = result$solution |

입력된 목적함수와 제약조건들을 바탕으로 solve.QP()를 통해 최적화를 수행한 후, 최대분산효과를 만족하는 해를 w에 입력합니다.

|  |
| --- |
| > w  [1] 11.271751 2.209125 8.109576 0.000000 37.371656 28.974813 6.786705 7.729628 14.296778 14.536527 |

slsqp()의 예시와 같이 비중의 합이 1을 초과하게 됩니다. 비중의 합이 1이 되도록 표준화를 해주도록 합니다.

|  |
| --- |
| w = w / sum(w)  w = round(w, 4) |

자산 별 비중을 비중의 합으로 나누어 주도록 합니다. 가독성을 위해 반올림을 해준 후 w를 확인해 보면, 비중의 합이 1이 되었음이 확인됩니다. 또한 결과값이 기존 slsqp()에서 구한 값과 동일한 것도 확인할 수 있습니다.

|  |
| --- |
| > w  [1] 0.0859 0.0168 0.0618 0.0000 0.2847 0.2207 0.0517 0.0589 0.1089 0.1107 |

**optimalPortfolio() 함수를 이용한 최대분산효과 포트폴리오 구현**

최소분산 포트폴리오와 동일하게 optimalPortfolio() 함수를 이용하여 매우 간단하게 최대분산효과 포트폴리오를 구현할 수 있습니다.

|  |
| --- |
| library(RiskPortfolios)  w = optimalPortfolio(covmat, control = list(type = 'maxdiv', constraint = 'lo'))  w = round(w, 4) |

control 항목의 type에 maximum diversification을 의미하는 ‘maxdiv’를 입력해주며, 제약조건에는 투자 비중이 0보다 큰 ‘lo’(long only) 조건을 입력해 줍니다. 그 후 가독성을 위해 반올림을 해줍니다.

|  |
| --- |
| > w  [1] 0.0859 0.0168 0.0618 0.0000 0.2847 0.2207 0.0517 0.0589 0.1089 0.1107 |

패키지를 활용하여 매우 간단하게 최대분산효과 포트폴리오를 구현할 수 있으며, 그 결과값 또한 앞에서 계산한 것들과 동일합니다. 해당 패키지의 코드를 확인해보면, 해당함수는 최대분산효과 포트폴리오 계산시 -DR 방법 방법을 사용하며, 코드 또한 위에서 작성한 것과 거의 동일합니다.

**결과값들의 비교**

아래 표는 (1) slsqp() (Transformation 방법), (2) slsqp() (Duality 방법) (3) slsqp() (-DR 방법), (4) solve.QP() (Duality) 방법), (5) optimalPortfolio()를 이용하여 구한 해들의 비교입니다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| (1) | 0.0859 | 0.0168 | 0.0618 | 0.0000 | 0.2847 | 0.2207 | 0.0517 | 0.0589 | 0.1089 | 0.1107 |
| (2) | 0.0859 | 0.0168 | 0.0618 | 0.0000 | 0.2847 | 0.2207 | 0.0517 | 0.0589 | 0.1089 | 0.1107 |
| (3) | 0.0859 | 0.0168 | 0.0618 | 0.0000 | 0.2847 | 0.2207 | 0.0517 | 0.0589 | 0.1089 | 0.1107 |
| (4) | 0.0859 | 0.0168 | 0.0618 | 0.0000 | 0.2847 | 0.2207 | 0.0517 | 0.0589 | 0.1089 | 0.1107 |
| (5) | 0.0859 | 0.0168 | 0.0618 | 0.0000 | 0.2847 | 0.2207 | 0.0517 | 0.0589 | 0.1089 | 0.1107 |

5가지 방법 모두 동일한 해를 가지는 것이 확인됩니다. 그러나 최소분산 포트폴리오와 동일하게 구석해Corner Solution 문제가 발행하며, 이를 해결하기 위해 각 자산의 최소 및 최대 투자 비중 제약조건을 추가하도록 합니다.

**최소 및 최대 투자비중 제약조건의 추가**

구석해 문제를 방지하고, 모든 자산에 골고루 투자하기 위해 개별 투자비중을 최소 5%, 최대 20%로 하는 제약조건을 추가해 주도록 하겠습니다. 먼저 slsqp() 함수에서 해당 제약조건을 추가하며, Transformation 방법을 이용하여 해를 구하도록 하겠습니다.

|  |
| --- |
| objective = function(w) {  corr = cov2cor(covmat)  obj = t(w) %\*% corr %\*% w  return(obj)  }  hin.objective = function(w) {  return(w)  }  heq.objective = function(w) {  sum\_w = sum(w)  return( sum\_w - 1 )  }  result = slsqp( x0 = rep(0.1, 10),  fn = objective,  hin = hin.objective,  heq = heq.objective,  lower = rep(0.05, 10),  upper = rep(0.20, 10))  w = result$par  w = w / sqrt(diag(covmat))  w = round(w, 4)  w = w / sum(w)  w = round(w, 4) |

앞의 예제와 모든 코드는 동일하며, 최소 및 최대 제약조건만 각각 5%와 20%를 입력하였습니다. 해당 코드를 실행할 경우 다음과 같은 결과가 나옵니다.

|  |
| --- |
| > w  US.Large Europe Japan Emerging  0.0625 0.0330 0.0519 0.0272  US.Long.Term US.Mid.Term US.REITs International.REITs  0.2324 0.3008 0.0426 0.0446  Gold Commodities  0.1020 0.1030 |

최소 제약조건을 5%로 정하였음에도 불구하고 일부 종목의 최적해는 5%보다 작은 결과가 나오며, 최대 제약조건을 20%로 정하였음에도 불구하고 그 이상의 해가 나오는 종목 또한 존재합니다.

아래 표는 각 단계별 최적해와 비중의 합을 나타낸 것입니다. slsqp() 함수를 통한 result$par 단계에서는 비중의 제약조건이 제대로 반영된 모습이지만, 각각의 표준편차로 나눈 후 비중의 합이 1이 되도록 표준화 해주는 과정을 거치면서 이러한 제약조건을 위배하는 해가 나오는 것이 확인됩니다.

|  |  |  |
| --- | --- | --- |
| **단계** | **최적해** | **비중의 합** |
| result$par | 0.0816 0.0500 0.0856 0.0500 0.2000  0.1356 0.0720 0.0617 0.1204 0.1432 | 1 |
| w / sqrt(diag(covmat)) | 8.4946 4.4881 7.0576 3.6980 31.5796  40.8694 5.7903 6.0581 13.8561 13.9992 | 135.891 |
| w / sum(w) | 0.0625 0.0330 0.0519 0.0272 0.2324  0.3008 0.0426 0.0446 0.1020 0.1030 | 1 |

이러한 문제는 표준화 과정이 필요한 Duality 방법에서도 역시나 발생합니다. 반면에 비중의 표준화 과정이 필요하지 않은 –DR 방법에서는 비중의 제약조건이 제대로 적용이 됩니다.

|  |
| --- |
| w = optimalPortfolio(covmat, control = list(type = 'maxdiv', constraint = 'user',  LB = rep(0.05, 10), UB = rep(0.20, 10)))  w = round(w, 4) |

optimalPortfolio() 함수의 경우 –DR 이용하므로, 최소 및 최대비중 제약조건이 제대로 적용되는 것이 확인합니다.[[37]](#endnote-37)

|  |
| --- |
| > w  [1] 0.0609 0.0500 0.0539 0.0500 0.2000 0.2000 0.0500 0.0500 0.1668 0.1183 |

최소분산효과 포트폴리오와 마찬가지로, solve.QP() 함수를 이용하여 Duality 방법으로 최대분산효과 포트폴리오를 구현할 경우에도 다양한 제약조건을 추가할 수 있습니다. 먼저 Duality 방법의 목적함수는 , 제약조건은 이며, 여기서 나온 값을 비중의 합이 1이 되도록 표준화하는 과정을 거칩니다. 따라서 비중의 최소 및 최대 제약조건은 단순히 가 아닌 표준화 과정()까지 고려하여 적용해 주어야 합니다. 이를 수식으로 나타내면 다음과 같습니다.

|  |  |
| --- | --- |
| **최소 비중 제약조건** | **최대 비중 제약조건** |
|  |  |

의 경우 행렬로 표현할 경우 즉 -lb로 이루어진 n × n 행렬입니다. I는 즉 대각선 부분이 1, 나머지가 0인 항등행렬을 의미합니다. 결과적으로 최소분산 포트폴리오와는 다르게 Duality 방법으로 최대분산효과 포트폴리오를 구현할 경우 최소 및 최대 제약조건이 좌변에 들어가며, 해당 제약조건을 행렬로 표현하면 다음과 같습니다.

|  |
| --- |
|  |

1번째 행은 제약조건을 의미합니다. 2번째 행을 정리하면,즉 비중의표준화가 고려된 최소비중 제약조건입니다. 마지막 행 역시 정리하면 가 되어 비중의 표준화가 고려된 최대비중 제약조건을 의미합니다. 제약조건 좌변식의 전치행렬을 고려한 코드는 다음과 같습니다.

|  |
| --- |
| Dmat = covmat  dvec = rep(0, 10)  Alb = -rep(0.05, 10) %\*% matrix(1, 1, 10) + diag(10)  Aub = rep(0.20, 10) %\*% matrix(1, 1, 10) - diag(10)  Amat = t(rbind(sqrt(diag(covmat)), Alb, Aub))  bvec = c(1, rep(0, 10), rep(0, 10))  meq = 1  result = solve.QP(Dmat, dvec, Amat, bvec, meq)  w = result$solution  w = w / sum(w)  w = round(w, 4) |

Alb의 -rep(0.05, 10)는 -lb 부분, matrix(1, 1, 10)는 부분, diag(10)는 I 부분을 의미하며, 이는 최소비중 제약조건의 좌변부분()과 같습니다. 동일하게 Aub는 최대비중 제약조건의 좌변부분()과 같으며, 해당 코드를 실행하면 최소 및 최대비중 제약조건인 (5%, 20%)가 제대로 반영되었음이 확인됩니다.

|  |
| --- |
| > w  [1] 0.0609 0.0500 0.0539 0.0500 0.2000 0.2000 0.0500 0.0500 0.1668 0.1183 |

**자산 별 서로 다른 제약조건 추가**

최소분산 포트폴리오의 경우와 동일하게 자산 별로 다른 제약조건을 추가하여 포트폴리오를 구성하도록 하겠습니다. 아래 표는 각 자산 별 최소 및 최대 투자비중 값과, 변경된 제약조건을 행렬의 형태로 나타낸 것입니다. 주의해야 할 점은 최소비중과 최대비중의 제약조건 추가 시, 형태로 고려해 주어야 한다는 점입니다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| 최소 | 0.10 | 0.10 | 0.05 | 0.05 | 0.10 | 0.10 | 0.05 | 0.05 | 0.03 | 0.03 |
| 최대 | 0.25 | 0.25 | 0.20 | 0.20 | 0.20 | 0.20 | 0.10 | 0.10 | 0.08 | 0.08 |

|  |
| --- |
|  |

좌변의 경우 와 연산을 통해 생성할 수 있습니다. 이를 코드와 나타내면 아래와 같습니다.

|  |
| --- |
| Dmat = covmat  dvec = rep(0, 10)  Alb = -c(0.10, 0.10, 0.05, 0.05, 0.10, 0.10, 0.05, 0.05, 0.03, 0.03) %\*% matrix(1, 1, 10) + diag(10)  Aub = c(0.25, 0.25, 0.20, 0.20, 0.20, 0.20, 0.10, 0.10, 0.08, 0.08) %\*% matrix(1, 1, 10) - diag(10)  Amat = t(rbind(sqrt(diag(covmat)), Alb, Aub))  bvec = c(1, rep(0, 10), rep(0, 10))  meq = 1  result = solve.QP(Dmat, dvec, Amat, bvec, meq)  w = result$solution  w = w / sum(w)  w = round(w, 4) |

최소 및 최대투자비중 제약조건을 나타내는 Alb와 Aub 부분이 자산 별 각각의 제약비중으로 변경되었으며, 나머지 부분은 모두 동일합니다. w를 확인해 보면, 결과 값들이 모두 최소 및 최대 투자비중 제약조건 내에 위치함을 확인할 수 있습니다.

|  |
| --- |
| > w  [1] 0.1000 0.1000 0.0611 0.0500 0.2000 0.2000 0.0500 0.0789 0.0800 0.0800 |

**그룹 별 제약조건**

solve.QP() 함수를 사용하여 최대분산효과 포트폴리오에서도 그룹 별 제약조건을 실행할 수 있습니다. 자산 별 최소 및 최대 투자 비중 제약조건인 5%와 20%에 다음과 같은 그룹 제약조건을 추가해 보도록 합니다. 이는 1~4번 자산 비중의 합은 40% 이상과 60% 이하, 5~6번 자산 비중의 합은 20% 이상과 40% 이하, 7~10번 자산 비중의 합은 10% 이상과 30% 이하인 조건을 의미합니다.

|  |
| --- |
|  |

비중의 표준화를 고려하면 는 형태로 변형해야 합니다. 분모 부분인 은 행렬로 표현하면 와 같으며, 분자 부분은 다음과 같이 나타낼 수도 있습니다.

|  |
| --- |
|  |

이 중 행렬을 라 두면, (1)번 그룹의 최소 및 최대 비중 제약조건은 다음과 같이 표현할 수 있습니다.

|  |  |
| --- | --- |
| **최소 비중 제약조건** | **최대 비중 제약조건** |
|  |  |

와는 i번째 그룹 제약조건의최소 및 최대 비중 제약조건 이며, 와를 행렬로 나타내면 다음과 같으며, 결과적으로 각각 와로 이루어진 행벡터가 됩니다.

|  |
| --- |
| **=**  **=** |

는 i번째 그룹 제약조건에서 그룹에 해당하는 열은 1, 아닌 곳은 0으로 이루어진 행벡터 입니다. 따라서 그룹 제약조건은 와로 이루어진 행벡터에, 조건문으로 이루어진 행백터를더하거나 빼는 형태로 구성됩니다. 기존 최소 및 최대 제약조건에 추가로 입력할 그룹 제약조건을 행렬로 표현하면 다음과 같습니다.

|  |
| --- |
|  |

첫번째 행인 을 정리하면 , 즉 우리가 원하는 형태의 제약조건이 제대로 입력되었습니다.

와 부분은 각각 Agroup을 통해 Amat행렬에, 부분은 기존 bvec 부분에 각각 추가로 입력해 주면 그룹 제약조건이 추가됩니다. 이를 코드로 나타내면 다음과 같습니다.

|  |
| --- |
| Dmat = covmat  dvec = rep(0, 10)  Alb = -rep(0.05, 10) %\*% matrix(1, 1, 10) + diag(10)  Aub = rep(0.20, 10) %\*% matrix(1, 1, 10) - diag(10)  Agroup = rbind(  -0.3 %\*% matrix(1, 1, 10) + c(1,1,1,1,0,0,0,0,0,0),  -0.3 %\*% matrix(1, 1, 10) + c(0,0,0,0,1,1,0,0,0,0),  -0.0 %\*% matrix(1, 1, 10) + c(0,0,0,0,0,0,1,1,1,1),  0.7 %\*% matrix(1, 1, 10) - c(1,1,1,1,0,0,0,0,0,0),  0.5 %\*% matrix(1, 1, 10) - c(1,1,1,1,0,0,0,0,0,0),  0.4 %\*% matrix(1, 1, 10) - c(1,1,1,1,0,0,0,0,0,0)  )  Amat = t(rbind(sqrt(diag(covmat)), Alb, Aub, Agroup))  bvec = c(1, rep(0, 10), rep(0, 10), rep(0, 6))  meq = 1  result = solve.QP(Dmat, dvec, Amat, bvec, meq)  w = result$solution  w = w / sum(w)  w = round(w, 4) |

w를 확인해 보면, 결과 값들이 모두 최소 및 최대 투자비중 제약조건 내에 위치하며, 각 그룹별 제약조건 역시 만족함을 확인할 수 있습니다.

|  |
| --- |
| > w  [1] 0.1184 0.0500 0.0816 0.0500 0.2000 0.2000 0.0500 0.0500 0.1262 0.0738 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| 비중 | 0.1184 | 0.05 | 0.0816 | 0.05 | 0.2 | 0.2 | 0.05 | 0.05 | 0.1262 | 0.0738 |
| 제약 | (0.3, 0.7) | | | | (0.3, 0.5) | | (0.0, 0.4) | | | |
| 합계 | 0.3 | | | | 0.4 | | 0.3 | | | |

**[4] 위험균형 포트폴리오**

포트폴리오를 구성하는 자산들과 전체 위험의 관계를 이해하기 위해서는, 먼저 **한계 위험기여도Marginal Risk Contribution**와 **위험기여도Risk Contribution**에 대해 알아야 합니다. 한계 위험기여도는 특정 자산의 비중을 한 단위 증가시켰을 때 전체 포트폴리오의 위험의 증가를 나타내는 단위로써, 수학의 편미분과 같은 개념입니다. i번째 자산의 한계 위험기여도는 아래와 같이 나타낼 수 있습니다.

|  |
| --- |
|  |

인 사실을 이용하면, 한계 위험기여도는 아래와 같이 풀 수 있습니다. 결과적으로 분자는 분산-공분산 행렬과 각 자산의 비중의 곱, 분모는 포트폴리오의 표준편차 형태로 나타납니다.

|  |
| --- |
|  |

위험기여도는 특정 자산이 포트폴리오 내에서 차지하는 위험의 비중입니다. 한계 위험기여도가 큰 자산도 포트폴리오 내에서 비중이 작다면, 포트폴리오 내에서 차지하는 위험의 비중은 작을 것입니다. 반면에, 한계 위험기여도가 작은 자산일지라도 비중이 압도적으로 많다면, 포트폴리오 내에서 차지하는 위험의 비중은 클 것입니다.

결과적으로 i번째 자산의 위험기여도는, i번째 자산의 한계 위험기여도와 포트폴리오 내 비중의 곱으로 이루어집니다.

|  |
| --- |
|  |

위험기여도를 코드로 나타내면 다음과 같습니다. 먼저 포트폴리오 비중인 w와 분산-공분산 행렬인 covmat을 이용하여 한계 위험기여도를 계산하여 줍니다. 그 후, 비중 w를 곱하여 위험기여도를 계산해 준 후, 합계가 1이 되도록 표준화를 해줍니다.

|  |
| --- |
| get\_RC = function(w, covmat) {  port\_vol = t(w) %\*% covmat %\*% w  port\_std = sqrt(port\_vol)    MRC = (covmat %\*% w) / as.numeric(port\_std)  RC = MRC \* w  RC = c(RC / sum(RC))    return(RC)  } |

**전통적 자산배분, 주식 60% & 채권 40%**

자산배분에서 가장 많이 사용되는 투자방법은 주식에 60%, 채권에 40% 가량의 비율로 투자하는 것입니다. 주식과 채권이 서로 상관관계가 낮아 분산효과가 있다는 점, 장기적으로 주식이 채권에 비해 장기적으로 수익률이 높다는 점을 감안하면 이는 꽤나 합리적인 방법으로 보입니다.

그러나 눈에 보이는 비중은 60대 40의 형태일지라도, 포트폴리오 내에서 각 자산이 가지고 있는 위험기여도 60대 40의 비중이 아닌 전혀 다른 비중을 갖고 있으므로 이를 확인해 보도록 합니다.

|  |
| --- |
| ret = read.csv("ret\_allocation.csv", row.names = 1)  ret\_stock\_bond = ret[, c(1,5)]  cov\_stock\_bond = cov(ret\_stock\_bond)  RC\_stock\_bond = get\_RC(c(0.6, 0.4), cov\_stock\_bond) |

ret 데이터에서 첫번째 행은 미국 주식 수익률을, 다섯번째 행은 미국 장기채를 의미하므로, 해당 부분만 ret\_stock\_bond 변수에 지정합니다. 그 후 cov() 함수를 이용해 두 자산의 분산-공분산 행렬을 만들어 주며, 위에서 만든 get\_RC 함수를 통해 자산 별 위험기여도를 계산하도록 합니다. RC\_stock\_bond 변수에는 각 자산이 가지는 위험기여도가 저장되어 있습니다.

|  |
| --- |
| > RC\_stock\_bond  [1] 0.92004276 0.07995724 |

주식과 채권이 가지는 위험기여도는 각각 92%, 8%로써 투자 비중인 60%, 40% 와는 전혀 다른 위험 비중을 보입니다. 즉, 주식이 포트폴리오 위험의 대부분을 차지하는 모습을 보입니다.

|  |  |
| --- | --- |
| **자산 배분 비중** | **위험 배분 비중** |
|  |  |

**slsqp() 함수를 이용한 위험균형 포트폴리오 구현**

이처럼 특정 자산이 포트폴리오의 위험을 대부분 차지하는 문제를 막고, 모든 자산이 동일한 위험기여도를 가지는 포트폴리오가 **위험균형 포트폴리오Risk Parity Portfolio** 혹은 **동일 위험기여도 포트폴리오Equal Risk Contribution Portfolio** 입니다. 이를 수식으로 쓰면 다음과 같습니다.

|  |
| --- |
|  |

위험균형 포트폴리오 역시 slsqp() 함수를 이용하여 어렵지 않게 구현할 수 있습니다.

|  |
| --- |
| objective = function(w) {  RC = get\_RC(w, covmat)  Target = rep(0.1, 10)    diff = sum((RC - Target)^2)  return(diff)  } |

위험균형 포트폴리오를 구현하기 위한 목적함수는 형태입니다. 먼저 는 현재 비중을 기준으로 계산된 위험기여도로써, 기존에 만든 get\_RC 함수를 통해 구할 수 있습니다. 는 자산 별 배분되고자 하는 위험기여도 이며, 위험균형 포트폴리오에서는 모든 자산의 위험기여도가 동일하므로 10개 자산 모두에 10%씩 입력합니다.

get\_RC 함수를 통해 구한 현재의 위험기여도와, Target에서 설정한 목표 위험기여도 차이의 제곱의 합을 diff에 입력합니다. 만일 자산 별 비중이 모두 동일하게 10%라면 diff 값은 다음과 같습니다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **자산** | **비중** | **위험기여도** | **목표 위험기여도** | **차이** | **차이의 제곱** |
| 1 | 0.1 | 0.125867642 | 0.1 | 0.025868 | 0.000669 |
| 2 | 0.1 | 0.155317001 | 0.1 | 0.055317 | 0.00306 |
| 3 | 0.1 | 0.144782241 | 0.1 | 0.044782 | 0.002005 |
| 4 | 0.1 | 0.193275417 | 0.1 | 0.093275 | 0.0087 |
| 5 | 0.1 | -0.01435383 | 0.1 | -0.11435 | 0.013077 |
| 6 | 0.1 | -0.00612275 | 0.1 | -0.10612 | 0.011262 |
| 7 | 0.1 | 0.15383421 | 0.1 | 0.053834 | 0.002898 |
| 8 | 0.1 | 0.123713278 | 0.1 | 0.023713 | 0.000562 |
| 9 | 0.1 | 0.043701236 | 0.1 | -0.0563 | 0.00317 |
| 10 | 0.1 | 0.079985556 | 0.1 | -0.02001 | 0.000401 |
| **합계** | | | | | **0.045804** |

모든 자산의 비중이 동일할 경우 값은 0.045804로 계산됩니다. 이는 특정 자산의 위험기여도가 10%를 훨씬 초과하는 반면, 다른 자산의 위험기여도는 음수를 나타내는 등 자산 별 위험기여도의 차이가 크기 때문입니다.

위의 조건에서 위험기여도가 큰 자산의 비중을 줄이고 낮은 자산의 비중을 늘린다면, 차이의 제곱의 합은 기존 값보다 낮아질 것입니다. 이러한 방법을 무수히 반복해 나가면 모든 자산의 위험기여도가 동일한 해를 찾을 것이며, 차이의 제곱은 0이 될 것입니다. 따라서 값을 최소화 하는 조건은 동일 위험기여도를 갖는 해를 찾는 조건과 동일합니다.

|  |
| --- |
| hin.objective = function(w) {  return(w)  }  heq.objective = function(w) {  sum\_w = sum(w)  return( sum\_w - 1 )  }  library(nloptr)  result = slsqp( x0 = rep(0.1, 10),  fn = objective,  hin = hin.objective,  heq = heq.objective)  w = result$par |

각 자산의 비중이 0 보다 큰 부등위 제약조건과, 투자 비중의 합이 1인 등위 제약조건을 입력해 준 후, slsqp() 함수를 통해 목적함수를 최소화 시켜주는 최적해를 구해줍니다. w에는 각 자산의 위험기여도를 동일하게 만들어주는 자산 별 비중이 계산됩니다.

|  |
| --- |
| > w  [1] 0.06472758 0.05186119 0.05380107 0.04164183 0.18155855 0.31708673  [7] 0.05078415 0.06154508 0.09712090 0.07987293 |

get\_RC 함수를 통해 위험기여도를 확인해보면, 모든 자산이 동일한 위험기여도를 가지는 것이 확인됩니다. 결과적으로 w가 위험균형 포트폴리오를 만족하는 해가 됩니다.

|  |
| --- |
| > get\_RC(w, covmat)  [1] 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 |

**위험예산 포트폴리오**

모든 자산의 위험기여도가 동일한 값이 아닌, 자산 별로 다른 위험기여도를 가져가는 포트폴리오를 구성해야 할 경우도 있습니다. 이러한 포트폴리오를 **위험예산 포트폴리오Risk Budget Portfolio**라 합니다. 위험균형 포트폴리오 역시 각 자산의 위험예산이 로 동일한 특수 형태이며, slsqp() 함수를 이용하면 원하는 위험예산 포트폴리오를 손쉽게 구현할 수 있습니다.

먼저 각 자산 별 위험예산을 아래와 같이 정합니다. 1~4번 자산은 각각 15%씩, 5~6번 자산은 각각 10%씩, 7~10번 자산은 각각 5%씩 위험예산을 부여하고자 합니다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **자산** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| 예산 | 0.15 | 0.15 | 0.15 | 0.15 | 0.10 | 0.10 | 0.05 | 0.05 | 0.05 | 0.05 |

|  |
| --- |
| objective = function(w) {  RC = get\_RC(w, covmat)  Target = c(0.15, 0.15, 0.15, 0.15, 0.10, 0.10, 0.05, 0.05, 0.05, 0.05)    diff = sum((RC - Target)^2)  return(diff)  } |

위험예산 포트폴리오 역시 목적함수는 로 위험균형 포트폴리오와 같으며, Target 부분만 부여하고자 하는 위험예산으로 변경합니다. 자산 별 비중이 모두 동일한 경우 diff 값은 다음과 같습니다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **자산** | **비중** | **위험기여도** | **목표 위험기여도** | **차이** | **차이의 제곱** |
| 1 | 0.1 | 0.125867642 | 0.15 | -0.024132 | 0.000582 |
| 2 | 0.1 | 0.155317001 | 0.15 | 0.005317 | 0.000028 |
| 3 | 0.1 | 0.144782241 | 0.15 | -0.005218 | 0.000027 |
| 4 | 0.1 | 0.193275417 | 0.15 | 0.043275 | 0.001873 |
| 5 | 0.1 | -0.01435383 | 0.10 | -0.114354 | 0.013077 |
| 6 | 0.1 | -0.00612275 | 0.10 | -0.106123 | 0.011262 |
| 7 | 0.1 | 0.15383421 | 0.05 | 0.103834 | 0.010782 |
| 8 | 0.1 | 0.123713278 | 0.05 | 0.073713 | 0.005434 |
| 9 | 0.1 | 0.043701236 | 0.05 | -0.006299 | 0.000040 |
| 10 | 0.1 | 0.079985556 | 0.05 | 0.029986 | 0.000899 |
| **합계** | | | | | **0.044003** |

위의 조건에서 위험기여도가 목표 위험기여도 보다 큰 자산의 비중을 줄이고, 목표 위험기여도보다 낮은 자산의 비중을 늘린다면, 차이의 제곱의 합은 기존 값보다 낮아질 것입니다. 이러한 방법을 무수히 반복해 나가면 자산 별 위험기여도가 목표 위험기여도와 동일한 해를 찾을 것이며, 차이의 제곱은 0이 될 것입니다. 따라서 위험예산 포트폴리오 역시 값을 최소화 하는 조건을 목적함수로 합니다.

|  |
| --- |
| hin.objective = function(w) {  return(w)  }  heq.objective = function(w) {  sum\_w = sum(w)  return( sum\_w - 1 )  }  result = slsqp( x0 = rep(0.1, 10),  fn = objective,  hin = hin.objective,  heq = heq.objective)  w = result$par |

부등위 제약조건과, 등위 제약조건을 입력해 준 후, slsqp() 함수를 통해 목적함수를 최소화 시켜주는 최적해를 구해줍니다. w에는 각 자산의 위험기여도가 위험예산과 동일하게 만들어주는 자산 별 비중이 계산됩니다.

|  |
| --- |
| > w  [1] 0.08937943 0.07403733 0.07363406 0.06028851 0.19068213 0.33604491  [7] 0.02651103 0.03333502 0.06487509 0.05121250 |

get\_RC 함수를 통해 위험기여도를 확인해보면, 각 자산의 위험기여도가 위험예산과 동일한 값을 가지는 것이 확인됩니다. 결과적으로 w가 위험예산 포트폴리오를 만족하는 해가 됩니다.

|  |
| --- |
| > get\_RC(w, covmat)  [1] 0.15 0.15 0.15 0.15 0.10 0.10 0.05 0.05 0.05 0.05 |

**8장. 포트폴리오 백테스트**

백테스트란 현재 생각하는 전략을 과거부터 실행하였을 시, 어느 정도의 손익이 발생하는지 테스트해보는 과정입니다. 과거의 데이터를 기반으로 전략을 실행하는 퀀트 투자에 있어서, 이는 핵심 단계이기도 합니다. 백테스트 결과를 통해 해당 전략의 손익뿐만 아니라 각종 위험을 대략적으로 판단할 수 있으며, 어떤 구간에서 전략이 좋았는지 혹은 나빴는지에 대한 이해도 키울 수 있습니다. 이러한 이해를 바탕으로 퀀트 투자를 지속한다면 단기적으로 수익이 나쁜 구간에서도 그 이유에 대한 객관적인 안목을 키울 수 있으며, 확신을 가지고 전략을 지속할 수 있습니다.

그러나 백테스트를 아무리 보수적으로 혹은 엄밀하게 진행하더라도 이미 일어난 결과를 대상으로 한다는 점에서 ***‘정답을 보고 답지를 쓰는 격’*** 이라는 점을 항상 명심해야 합니다. 백테스트 수익률만을 보고 투자에 대한 판단을 하거나, 혹은 동일한 수익률이 미래에도 반복될 것이라 믿는다면 이는 ***‘백미러를 보고 운전을 하는’*** 매우 위험한 결과를 초래할 수도 있습니다.

R에서 백테스트는 PerformanceAnalytics 패키지[[38]](#endnote-38) 내의 Return.portfolio() 함수를 사용하여 매우 간단하게 수행할 수 있습니다. 이번 장에서는 해당 함수에 대한 이해와 더불어, 구체적인 사용 방법에 대한 예시로써 ‘전통적인 주식 60% & 채권 40% 포트폴리오’, ‘시점 선택 전략’, ‘동적 자산배분’에 대한 백테스트를 실행합니다. 이와 더불어 해당 패키지 내의 각종 함수를 통한 포트폴리오의 성과측정방법도 알아보도록 하겠습니다.

**[1] Return.Portfolio 함수**

프로그래밍을 이용하여 백테스트를 할 경우, 전략이 단순하다면 단 몇 줄 만으로도 테스트가 가능합니다. 그러나 전략이 복잡해지거나 적용해야 할 요소가 많아질 경우, 패키지를 이용하는 것이 효율적인 방법입니다.

PerformanceAnalytics 패키지의 Return.portfolio() 함수는 백테스트 과정에서 가장 대중적으로 사용되는 함수이며, 해당 책에서는 해당 함수의 상세한 사용법과 여러 예시를 살펴보도록 하겠습니다. 해당 함수의 가장 큰 장점은 각 자산의 수익률과 리밸런싱 비중만 있으면 백테스트 수익률, 턴오버 등을 쉽게 계산할 수 있다는 점이며, 리밸런싱 시점과 수익률의 시점이 일치하지 않아도 된다는 점입니다. 즉, 수익률 데이터는 일간, 리밸런싱 시점은 분기 혹은 연간으로 된 경우에도 매우 쉽게 백테스트를 수행할 수 있습니다. 해당 함수의 입력 변수들과 그 내용은 아래와 같습니다.

|  |
| --- |
| Return.portfolio(R, weights = NULL, wealth.index = FALSE,  contribution = FALSE, geometric = TRUE, rebalance\_on = c(NA, "years",  "quarters", "months", "weeks", "days"), value = 1, verbose = FALSE, ...) |

|  |  |
| --- | --- |
| **변수명** | **내용** |
| R | 개별 자산의 수익률입니다. |
| weights | 각 리밸런싱 시기의 자산 별 비중입니다. 미 입력시 동일비중 포트폴리오를 가정하여 백테스트가 이루어집니다. |
| wealth.index | 포트폴리오 시작점이 1인 wealth index에 대한 생성여부이며, 디폴트는 FALSE로 설정되어 있습니다. |
| contribution | 포트폴리오 내에서 각 자산 별 성과기여를 나타내는지에 대한 여부이며, 디폴트는 FALSE로 설정되어 있습니다. |
| geometric | 포트폴리오 수익률 계산시 복리(기하)수익률 적용 여부이며, 디폴트는 TRUE로써 복리수익률을 계산합니다. FALSE 입력 시 단리수익률을 사용합니다. |
| rebalance\_on | weight 값이 미입력 혹은 시계열 형식이 아닐 경우, 리밸런싱 주기를 선택할 수 있습니다. |
| value | 초기 포트폴리오 가치를 의미하며, 디폴트는 1입니다. |
| verbose | 부가적인 결과를 표시할지에 대한 여부입니다. 디폴트인 FALSE를 입력할 경우 포트폴리오 수익률만이 시계열 형태로 표시되며, TRUE를 입력할 경우 수익률 외에 자산 별 성과기여, 비중, 성과 등이 리스트 형태로 표현됩니다. |

이 중 가장 중요한 입력변수는 개별 자산의 수익률인 **R**과 리밸런싱 시기의 자산 별 비중인 **weights** 입니다. **weights**의 경우 입력하지 않을 동일비중 포트폴리오를 구성하며, 만일 입력할 경우 R과 열의 숫자 및 순서가 동일해야 합니다. 매 리밸런싱 시점마다 자산 별 비중이 동일할 경우 상수 형태로 입력하여도 되지만, 시점마다 자산 별 비중이 다를 경우에는 **weights**는 시계열 형태로 입력되어야 합니다.

**[2] 전통적인 60대 40 포트폴리오 백테스트**

Return.portfolio()의 가장 간단한 예제로써 전통적인 60대 40 포트폴리오를 백테스트 하도록 합니다. 해당 포트폴리오는 주식과 채권에 각각 60%와 40%를 투자하며, 특정 시점 마다 해당 비중을 맞춰주기 위해 리밸런싱을 수행합니다. 해당 예제는 매해 말 리밸런싱을 가정합니다.

|  |
| --- |
| library(quantmod)  library(PerformanceAnalytics)  ticker = c("SPY","TLT")  getSymbols(ticker, from = "2003-01-01", to = "2017-12-31")  prices = do.call(cbind, lapply(ticker, function(x) Ad(get(x))))  rets = Return.calculate(prices)[-1,] |

먼저 주식과 채권에 해당하는 ETF 데이터를 다운로드 받습니다. 주식에 해당하는 ETF로는 S&P 500 수익률을 추종하는SPY(SPDR S&P 500 ETF Trust)를 사용하며, 채권에 해당하는 ETF로는 미국 장기채 수익률을 추종하는 TLT(iShares 20+ Year Treasury Bond)를 사용합니다.

getSymbols() 함수를 이용하여 해당 ETF의 데이터를 다운로드 받으며, 기간은 2003년부터 2017년 까지를 대상으로 합니다. 그 후 lapply()와Ad(get(X)) 함수를 이용하여 ticker에 해당하는 종목들의 수정주가를 list 형태로 저장하며, do.call()을 이용하여 cbind(), 즉 열의 형태로 저장합니다. 그 후 Return.calculate() 함수를 이용하여 수익률을 계산해주며, 1행은 수익률을 계산할 수 없어 NA로 표시되므로, [-1, ]을 통해 1행을 삭제해주도록 합니다.

|  |
| --- |
| > cor(rets)  SPY.Adjusted TLT.Adjusted  SPY.Adjusted 1.0000000 -0.4009437  TLT.Adjusted -0.4009437 1.0000000 |

cor 함수를 통해 두 자산간의 상관관계를 확인해보면 -0.40로써 매우 낮은 상관관계를 보입니다. 따라서 강한 분산효과를 기대해볼 수 있습니다.

|  |
| --- |
| portfolio = Return.portfolio(R = rets,  weights = c(0.6, 0.4),  rebalance\_on = "years")  portfolios = cbind(rets, portfolio) |

본격적으로 **Return.portfolio()** 함수를 이용하여 백테스트를 실행합니다. 자산의 수익률인 R에는 ret를, 리밸런싱 비중인 weights에는 60%와 40%를 의미하는 c(0.6, 0.4)를, 리밸런싱 시기인 rebalance\_on에는 연간 리밸런싱에 해당하는 “years”를 입력합니다. 리밸런싱 주기는 이 외에도 “quarters”, “months”, “weeks”, “days”도 입력이 가능합니다. 결과적으로 portfolio 변수에는 주식과 채권 매해 말60%대 40%로 리밸런싱하는 포트폴리오의 수익률이 저장됩니다.

portfolios 변수에는 기존 주식과 채권에 해당하는 rets와 portfolio를 열의 형태로 묶어주도록 합니다.

**PerformanceAnalytics** 패키지의 **charts.PerformanceSummary()** 함수를 이용하면 다음과 같은 그래프가 그려짐이 확인됩니다.

|  |
| --- |
| > charts.PerformanceSummary(portfolios, main = "Portfolios") |

|  |
| --- |
| **[그림] 주식, 채권 및 60대 40 포트폴리오 수익률** |
|  |

검은색 그래프는 주식 수익률(SPY), 붉은색 그래프는 채권 수익률(TLT), 초록색 그래프는 60대 40 포트폴리오 수익률을 의미합니다. 주식과 채권은 상반되는 움직임을 보이며 상승하며, 분산투자 포트폴리오는 각 개별 자산에 비해 훨씬 안정적인 수익률을 보입니다.

**[3] 시점 선택 전략 백테스트**

이전 테스트가 리밸런싱 시점 별 비중이 60%와 40%로 고정되어 있었다면, 이번에는 시점 별 비중이 다른 형태의 예제를 살펴보도록 하겠습니다.

메브 파버Meb Faber는 본인의 논문[[39]](#endnote-39)을 통해, 시점 선택Market Timing 전략을 사용할 경우 단순 매수 후 보유Buy & Hold 대비 극심한 하락장에서 낙폭을 줄일 수 있으며, 이로 인해 위험 대비 수익률을 올릴 수 있다고 설명합니다. 논문에서 말하는 시점 선택의 투자 규칙은 다음과 같습니다.

|  |
| --- |
| 주가 > 10개월 이동평균(SMA) → 매수  주가 < 10개월 이동평균(SMA) → 매도 및 현금 보유 |

해당 규칙을 미국 S&P 500에 적용하는 예제를 살펴보도록 하겠습니다. 현재 주식 가격이 과거 10개월 주식 가격의 단순 평균 대비 이상이면 매수, 그렇지 않으면 전량 매도 후 현금을 보유하는 전략이며, 리밸런싱은 매월 말에 실행하도록 합니다.

|  |
| --- |
| library(quantmod)  library(PerformanceAnalytics)  symbols = c("SPY","SHY")  getSymbols(symbols, src="yahoo", from = "2003-01-01", to = "2017-12-31")  prices = do.call(cbind, lapply(symbols, function(x) Ad(get(x))))  rets = Return.calculate(prices)[-1,] |

먼저 주식과 채권에 해당하는 ETF 데이터를 다운로드 받습니다. 주식에 해당하는 ETF로는 S&P 500 수익률을 추종하는SPY(SPDR S&P 500 ETF Trust)를 사용하며, 현금에 해당하는 ETF로는 미국 단기채 수익률을 추종하는 SHY(iShares 1-3 Year Treasury Bond ETF)를 사용합니다.

symbols 부분을 제외하고는 이전 예제와 동일합니다. getSymbols() 함수를 이용하여 해당 ETF의 데이터를 다운로드 받으며, 수정주가를 묶은 후 수익률을 계산합니다.

|  |
| --- |
| ep = endpoints(rets, on = "months")  # index(rets)[ep]  wts = list()  lookback = 10 |

먼저 **xts** 패키지의 **endpoints()** 함수를 이용해 월 말일의 위치를 구합니다. 해당 함수는 **endpoints(x, on="months", k=1)**의 형태로 이루어지며 x는 시계열 데이터, on은 원하는 기간, k는 구간 길이를 의미합니다. 즉, 시계열 데이터에서 월말에 해당하는 부분의 위치를 반환하며, 매 월이 아닌 “weeks”, “quarters”, “years” 등도 입력이 가능합니다.

**rets** 데이터의 인덱스 중 **ep**에 해당하는 값들을 살펴보면, 모두 월말 데이터임이 확인됩니다.

|  |
| --- |
| > index(rets)[ep]  [1] "2003-01-31" "2003-02-28" "2003-03-31" "2003-04-30" "2003-05-30" "2003-06-30"  [7] "2003-07-31" "2003-08-29" "2003-09-30" "2003-10-31" "2003-11-28" "2003-12-31"  ………………  [169] "2017-01-31" "2017-02-28" "2017-03-31" "2017-04-28" "2017-05-31" "2017-06-30"  [175] "2017-07-31" "2017-08-31" "2017-09-29" "2017-10-31" "2017-11-30" "2017-12-29" |

각 시점 별 비중이 입력될 **wts** 변수를 공백 리스트 형식으로 저장해주며, n개월 이동평균 값에 해당하는 **lookback** 변수에는 10을 입력해 줍니다.

|  |
| --- |
| for (i in (lookback+1) : length(ep)) {  sub.price = prices[ep[i-lookback] : ep[i] , 1]  sma = mean(sub.price)  wt = rep(0, 2)  wt[1] = ifelse(last(sub.price) > sma, 1, 0)  wt[2] = 1 - wt[1]    wts[[i]] = xts(t(wt), order.by = index(rets[ep[i]]))  } |

for 구문을 이용하여 매 월별 투자 규칙에 따른 투자 비중을 결정합니다. i의 시작점과 끝점은 **(lookback+i)**과 **length(ep)**, 즉 11부터 마지막 개월까지로 합니다. 현재부터 과거 10개월 간, prices의 첫번째 열인 SPY 가격을 **sub.price** 변수에 저장해 줍니다. 시작점인 i=11를 예시로 보면 sub.price의 데이터는 다음과 같습니다.

|  |
| --- |
| > sub.price  SPY.Adjusted  2003-01-02 67.31068  2003-01-03 67.51765  2003-01-06 68.70761  ………………  2003-10-28 78.53096  2003-10-29 78.63563  2003-10-30 78.80015 |

mean 함수를 통해 해당 가격들의 평균을 구한 후, 이를 sma 변수에 저장합니다. 그 후, 주식과 현금의 투자비중이 될 wt 변수를 만들어 줍니다.

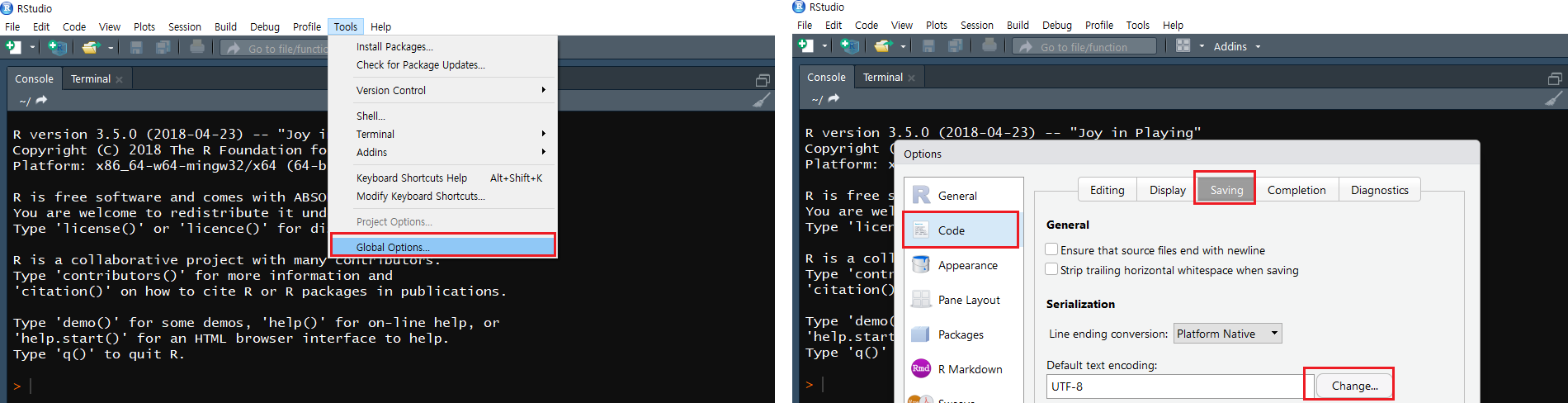
Ifelse() 함수는 **ifelse(test, yes, no)**의 형태로 이루어지며, 해당 예제에서는 last(sub.price), 즉 현재기준 주가가 10개월 간의 이동평균인 sma보다 클 경우 1, 그렇지 않을 경우 0을 반환하며 이는 주식의 투자비중과 같습니다. 이를 wt[1]에 저장해주며, 현금의 투자 비중은 1-주식비중 이므로 wt[2]에 1-wt[1]를 저장해 줍니다. 즉 해당 규칙은 아래와 같이 나타낼 수 있습니다.

|  |  |
| --- | --- |
| **현재 주가 > 10개월 이동평균** | **현재 주가 < 10개월 이동평균** |
| 주식비중: wt[1] = 1  현금비중: wt[2] = 0 | 주식비중: wt[1] = 0  현금비중: wt[2] = 1 |

**부록 1. 인코딩 문제 해결**

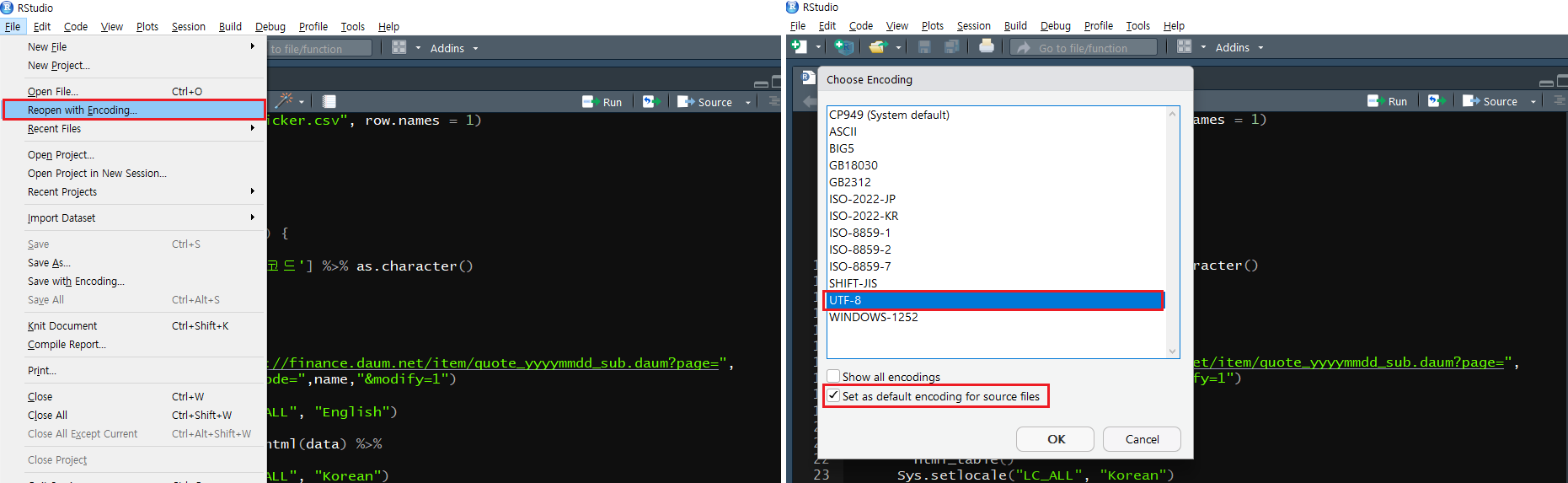
예시로 제공된 R 스크립트의 한글 부분이 깨져 나오는 경우가 있으며, 이는 대부분 인코딩 설정으로 인해 생기는 문제입니다. R Studio의 Tools → Global Options 메뉴에서 Code → Saving 항목 중 Default text encodings의 항목을 UTF-8로 변경해줄 경우 한글 부분이 정상적으로 출력됩니다.

**[그림] 인코딩 설정 항목**



해당 방법으로도 해결되지 않을 경우 File → Reopen with Encoding 메뉴에서 UTF-8 항목을 선택, Set as default encoding for source files 항목을 선택한 후 OK를 누르면 해결이 됩니다.

**[그림] 인코딩 설정 항목 2**



1. 2018년 6월 말 기준 [↑](#endnote-ref-1)
2. https://finance.yahoo.com/ [↑](#endnote-ref-2)
3. https://www.datacamp.com/ [↑](#endnote-ref-3)
4. https://en.wikipedia.org/wiki/List\_of\_programming\_languages [↑](#endnote-ref-4)
5. https://cran.r-project.org/web/packages/quantmod/index.html [↑](#endnote-ref-5)
6. 자세한 내용은 <https://docs.quandl.com/> 에서 확인할 수 있습니다. [↑](#endnote-ref-6)
7. 야후 파이낸스에서 제공하는 데이터의 경우 기본적으로 시가, 고가, 저가, 종가 모두 주식 유·무상증자나 액면분할 등을 고려한 수정주가를 제공합니다. [↑](#endnote-ref-7)
8. https://fred.stlouisfed.org/ [↑](#endnote-ref-8)
9. 파이프 오퍼레이터에 대한 더욱 자세한 설명은 Datacamp 페이지에서 확인할 수 있습니다.

   https://www.datacamp.com/community/tutorials/pipe-r-tutorial [↑](#endnote-ref-9)
10. 해당 HTML 코드를 메모장에 입력한 후, example.html 파일로 저장하면 웹페이지 형식으로 저장됩니다. [↑](#endnote-ref-10)
11. 크롬 및 인터넷 익스플로러에서 F12키를 누르면 개발자도구 화면이 열리며, 효율성을 위해 크롬을 사용할 것을 권장합니다. [↑](#endnote-ref-11)
12. tbl\_df 타입은 기존 데이터프레임을 보완한 타입이며, 형태 자체는 데이터프레임과 거의 유사합니다. 또한 csv를 읽을 때 발생하는 오류(Warning: 2 parsing failures.)는 무시하여도 되는 오류입니다. [↑](#endnote-ref-12)
13. 비상장기업 인수합병을 목적으로 하는 페이퍼컴퍼니 [↑](#endnote-ref-13)
14. 해당 과정에서 포스코대우, 미래에셋대우와 같이 단순히 종목명이 ‘우’로 끝나는 종목이 함께 삭제되는 단점도 존재합니다. [↑](#endnote-ref-14)
15. 해당 함수에 대한 자세한 설명은 2장의 **‘getSymbols() 함수를 이용한 API 다운로드’** 챕터를 다시 참조하시기 바랍니다. [↑](#endnote-ref-15)
16. 티커가 바뀜에 따라 ‘티커\_price.csv’의 값으로 저장됩니다. [↑](#endnote-ref-16)
17. <https://finance.naver.com/> [↑](#endnote-ref-17)
18. <http://finance.daum.net/> [↑](#endnote-ref-18)
19. 이는 삼성전자 한 종목에 대한 경우이며, 각 종목별로 재무제표 항목이 상이하므로 개수는 달라질 수 있습니다. 특히 금융업의 경우 제조업과는 전혀 다른 재무제표 항목을 가지고 있습니다. [↑](#endnote-ref-19)
20. Asness, C., & Frazzini, A. (2013). The devil in HML’s details. Journal of Portfolio Management, 39(4), 49-68. [↑](#endnote-ref-20)
21. 기존에는 do.call(rbind, data)의 형태를 통해 리스트 형태를 데이터 프레임으로 묶어 주었지만, 해당 데이터는 각 리스트별 열이름이 다르므로 해당 함수를 사용할 수 없습니다. [↑](#endnote-ref-21)
22. 미국 IT 산업을 선도하는 Facebook, Amazon, Apple, Netflix, Google의 앞글자를 따서 만든 용어 [↑](#endnote-ref-22)
23. 출처: NH투자증권(2018). 100세시대 행복 리포트 Vol. 45. [↑](#endnote-ref-23)
24. 출처: World Federation of Exchanges(2017), 한국거래소 [↑](#endnote-ref-24)
25. Blitz, D., & Van Vliet, P. (2007). The volatility effect: Lower risk without lower return. [↑](#endnote-ref-25)
26. Brunnermeier, M. K., & Parker, J. A. (2005). Optimal expectations. *American Economic Review, 95(4),* 1092-1118. [↑](#endnote-ref-26)
27. Baker, M., Bradley, B., & Wurgler, J. (2011). Benchmarks as limits to arbitrage: Understanding the low-volatility anomaly. *Financial Analysts Journal*, *67*(1), 40-54. [↑](#endnote-ref-27)
28. Ang, A., Hodrick, R. J., Xing, Y., & Zhang, X. (2009). High idiosyncratic volatility and low returns: International and further US evidence. *Journal of Financial Economics, 91(1),* 1-23. [↑](#endnote-ref-28)
29. Baker, M., Bradley, B., & Taliaferro, R. (2014). The low-risk anomaly: A decomposition into micro and macro effects. *Financial Analysts Journal, 70(2),* 43-58. [↑](#endnote-ref-29)
30. 해당 데이터는 본 책의 github 자료실에서도 다운로드 받을 수 있으며, HenryQuant 패키지의 asset\_data 함수를 이용하면 R에서 직접 사용이 가능합니다. [↑](#endnote-ref-30)
31. Goldfarb, D., & Idnani, A. (1983). A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical programming*, *27*(1), 1-33. [↑](#endnote-ref-31)
32. d에 0벡터 값을 입력할 경우 이 되어 목적함수에서 사라지게 됩니다. [↑](#endnote-ref-32)
33. 패키지 내 함수들의 코드는 github를 통해 확인할 수 있으며, RiskPortfolios 패키지의 github 주소는 [https://github.com/ArdiaD/RiskPortfolios](https://github.com/ArdiaD/RiskPortfolios/) 입니다. [↑](#endnote-ref-33)
34. http://www.ftse.com/products/downloads/FTSE\_Global\_Minimum\_Variance\_Index\_Series.pdf [↑](#endnote-ref-34)
35. Choueifaty, Y., & Coignard, Y. (2008). Toward maximum diversification. *Journal of Portfolio Management*, *35*(1), 40. [↑](#endnote-ref-35)
36. Choueifaty, Y., Froidure, T., & Reynier, J. (2011). Properties of the most diversified portfolio. [↑](#endnote-ref-36)
37. slsqp 함수를 이용하여 –DR 방법으로 최대분산 포트폴리오를 구현하여도 동일한 결과가 나옵니다. [↑](#endnote-ref-37)
38. https://cran.r-project.org/web/packages/PerformanceAnalytics/ [↑](#endnote-ref-38)
39. Faber, M. (2013). A quantitative approach to tactical asset allocation. [↑](#endnote-ref-39)