KR-ESG
한국ESG연구소

유사도 기반 네트워크를 활용한 중요인물 선정

-인물 데이터 분석 아이디어 공모전-

2023. 7. 10.

신의 환수 팀

목차

Ⅰ. 서론

인물 네트워크 분석을 통해 회사의 인물들 간의 관계를 이해하는 것은 현대 사회에서 중요한 리더십과 조직 운영에 대한 이해를 높이는 데 도움이 된다. 인물 네트워크를 구축하고 분석함으로써, 회사 내부에서 리더와 팔로워들 간의 상호작용과 권력 형태를 파악할 수 있다.

이를 위해 본 프로젝트는 회사 임원진들의 인물 네트워크를 구축하여 분석하였다. 여러 회사보다는 한 회사를 중점적으로 살펴보아 연도 별로 인물 네트워크가 어떻게 달라지는지 분석하고 추가적인 인사이트를 얻고자 하였다. 회사 설정은 분석 용이성을 위해 시장 상황에 민감한 증권사로 채택하였고, 증권사 중에서 대신증권을 채택하였다.

인물 네트워크는 각 인물을 노드로, 인물 간의 관계를 에지로 연결한다. 관계를 네트워크로 표현하기 위해서는 관계에 대한 정의가 먼저 이루어져야 한다. 사회 심리학적으로 유사한 특성의 사람들과 관계를 맺는 유사성의 법칙, 서로 다른 사람들과 관계를 맺는 상보성의 법칙, 상호 간의 이익을 극대화하기 위해 관계를 맺는 사회적 교환 이론 등의 이론이 존재한다(김종운, 2017). 여기서 네트워크 형성에 보다 용이하게 적용할 수 있고, 다른 이론들보다 보편적으로 더 많은 영향을 끼친다고 알려진 유사성의 법칙에 기반해 인물 간의 관계를 인물 간의 유사성으로 치환해서 네트워크를 형성하고자 하였다(Byrne, 1997). 즉, 노드의 연결 기준으로 두 노드 간의 유사성을 사용하였다.

본 분석의 목적은 중요 인물들을 추출하고 임원들의 네트워크를 구축하여 미처 파악하지 못한 중요 인물들을 발굴하거나 기존에 중요하다고 여겨졌던 인물들의 중요도를 재평가하는 것이다. 이를 통해 시장 상황과 네트워크의 변화 속에서 실제로 중요한 역할을 수행하는 인물들을 도출해낼 수 있으며, 조직의 효율성과 성과 향상, 전략 등에 기여할 수 있다.

Ⅱ. 분석 방법

1. 진행순서

네트워크 구축과 분석은 다음과 같은 순서로 진행하였다.

1. 관심인물 선정
   상황 변화 속에서 인물의 특성이 변화한 인물을 관심인물로 선정
   a. 상황의 변화
      i. 매출액의 증가
   b. 인물의 변화
      i. 주식 증가
      ii. 승진

2. 관심인물과 다른 전체 인물들 사이의 네트워크 형성
   인물들을 노드로 삼고, 노드 간의 유사성을 가중치로 하여 엣지(edge)로 설정
   가중치(유사성, 연결강도)가 임계값을 넘으면 두 노드를 연결
   임계값은 가중치들의 삼사분위수로 설정

3. 네트워크 분석 및 중요인물 추출
  네트워크 시각화 및 분석
  관심인물과의 연결 횟수 지표와 네트워크 중심성 지표 4 개로 중요인물 추출

4. 중요인물 검증
 승진여부와 연속 중요인물 선정 여부로 선출된 중요인물이 의미 있는지 검증
    a. 이전 해 → 선정 당해 승진
    b. 선정 당해 → 다음해 승진
    c. 이전 해와 당해 2 년 연속 선정

2. 관심인물 선정

관심인물은 강한 영향력을 가지고 있어 네트워크의 구조나 연결성에 큰 영향을 미칠 수 있을 것으로 판단되는 인물이다. 관심인물을 선정한다면 네트워크 분석 시 네트워크의 구조와 효율성, 인물 간의 영향력, 관계성의 흐름 등을 파악하기 더욱 쉬워진다. 이러한 이유로 네트워크 형성에 앞서, 회사 네트워크 내에서 영향력과 중요도가 강할 것이라고 예상되는 인물을 다음과 같은 기준으로 선정하였다.

관심인물은 상황의 변화 속에서 개인의 변화가 일어난 인물로 설정하였다. 상황의 변화로는 매출이 증가하였을 때로, 연도별 누적 순이익의 평균을 기준으로 그 이상의 누적 순이익을 달성하는 연도를 기준으로 하였다. 개인의 변화로는 인물의 승진 또는 입사, 보유한 주식수(의결권 있는 주식수)가 증가한 인물을 기준으로 설정하였다. 즉, 기업의 매출이 증가한 시기에 승진 또는 입사를 하거나 보유 주식량이 증가한 인물을 관심인물로 선정하였다. 예를 들어, 회사의 매출이 증가한, 즉 회사가 좋은 실적을 내고 있는 상황에서 승진을 한 인물은 회사의 실적에 기여를 한 영향력 있는 인물이라고 추론해볼 수 있을 것이다.

대신증권 재무 정보(연도별)

Aa 연도	# 연도별 누적 순이익(단위: 천...	Σ 이전 연도 대비 증...	Σ 증가율
2014	₩4,549,569	₩4,549,569	
2015	₩96,421,291	₩91,871,722	2119%
2016	₩30,578,147	-₩65,843,144	32%
2017	₩61,411,920	₩30,833,773	201%
2018	₩114,808,879	₩53,396,959	187%
2019	₩87,928,648	-₩26,880,231	77%
2020	₩170,364,113	₩82,435,465	194%
2021	₩178,657,494	₩8,293,381	105%
2022	₩86,474,236	-₩92,183,258	48%
2023	₩59,459,519	-₩27,014,717	69%
+ New			
NOT EMPTY 10	AVERAGE ₩89,065,382	AVERAGE ₩5,945,952	

<표 1> 대신증권 연도별 재무정보 대신증권의 연도별 매출액 변화를 살펴보기 위해 정리한 표이다.
* 부록 - 1. 데이터 부분 참조

조건에 부합하는 인물에 대해 연도별로 각각 조건이 맞을 때마다 카운트하여 카운트 수가 2 이상인 인물을 관심인물로 추출하였다.

* 자세한 코드는 부록 - 2. 코드의 관심인물선정 부분 참조

위의 기준으로 선정된 관심인물은 다음과 같다.

선정된 관심인물: 양홍석, 이어룡, 홍대한, 송혁, 이순남, 김상원, 오익근, 김범철, 신인식, 조경순, 박성준, 정연규, 진승욱, 김성원, 권택현, 김호중, 이정화, 이재우, 박현식, 나유석, 신재범, 홍종국, 나재철, 박동현, 이동훈, 김병철, 정재중, 강윤기, 임민수, 정기동, 김수창, 최근영, 정연우, 강준규, 박규상, 하창룡, 장우철, 신재국, 김재중, 최명재, 배영훈, 이창세, 이지원, 문병식, 이성영, 김봉진, 길기모

단, 현재 절에서의 기준은 분석의 용이를 위해 설정한 임의의 기준이므로 이러한 판단기준으로 선정된 관심인물과 실제 중요인물의 차이는 Ⅲ절에서 더 살펴보고자 한다.

3. 가중치 함수

가중치(유사성) 판단 기준은 대학, 전공, 직위, 근속연수, 담당업무, 경력, 보유 주식 수 총 7 가지 피처를 활용하였다. 선정한 피처의 기준은 인물이 보통 비슷한 환경에서 자라거나 지내오면 유사한 성질을 가지게 될 가능성 존재한다는 가정 하에 환경적 요소에 중요한 영향을 미칠 것으로 판단되는 요소로 선정하였다. 예시로, 비슷한 전공의 사람들은 비슷한 분야에 대해 관심이 많고 비슷한 사고를 할 가능성이 높고, 비슷한 경력을 가진 사람들은 상황에 있어서 비슷한 대처 능력을 보일 가능성이 높다.

가중치 계산은 먼저, 각각의 피처에 대해 미리 정해 놓은 기준으로 노드 간의 거리를 구하고 α값으로 1 보다 작은 양수를 설정해 두 노드 간의 각 피처의 유사성은 $\alpha^{거리}$로 구해주었다. (박경미, 2013) 두 노드 간의 전체 유사성은 각 피처의 유사성을 합해준 값으로 설정하였다. $weight(X,Y) = \sum_{i=1}^{7} \alpha^{거리_i}$ 본 프로젝트에서는 α값은 0.7 로 설정해주었다.

각 피처의 거리에 대한 기준은 아래와 같이 설정하였다.

3.1. 대학 거리 기준

먼저 대학은 기준을 지역으로 나누어서 아래와 같이 거리를 설정하였다.

a. 같은 대학: 0
b. 같은 지역의 대학: 1
c. 다른 지역의 대학: 2

이때, 대학 지역의 기준은 서울경기, 광역시, 그 외 한국, 해외의 4 가지 범주로 나누었다.

3.2. 전공 거리 기준

전공도 마찬가지로 범주로 나누어서 아래와 같이 거리를 설정하였다.

a. 같은 전공: 0
b. 같은 계열의 전공: 1

c. 다른 계열의 전공: 2

이때, 전공의 범주 기준은 대한민국의 학과 계열 분류를 참고해 경영경제, 사회과학, 법률, 인문, 기타, 자연, 공학의 7 가지 범주로 나누었다.

3.3. 직위 거리 기준:

직위의 거리는 각 계급을 순서대로 나열해서 범주를 나누고 거리를 설정하였다.

a. 같은 직위: 0

b. 사외이사를 제외하고 나머지는 각 노드 간의 계급 차이만큼 거리를 설정

c: 사외이사: 사외이사는 회사 밖 인물이므로 모두와 거리를 4 로 조정하였다.
(최대 거리와 최소 거리의 평균)

계급의 순서는 다음과 같다.

회장->대표이사->부회장->사장->부사장->전무->상무->상무보->이사대우->사외이사

대신증권의 계급도를 참고하고 싶었으나 구하지 못해 계급 순서는 일반적으로 적용되는 기준으로 임의로 설정하였다.

3.4. 근속연수 거리 기준

먼저 근속연수를 구하고 근속 연수의 차이를 5 년씩으로 잘라서 범주를 생성

a. 같은 근속연수: 0

b. 근속연수에 차이가 있으면 그 차의 범주 만큼을 거리로 설정

구체적으로, 근속연수 차이를 5 년 기준으로 범주를 나누어 각각 거리는 1 씩 증가한다.
(차이 1~5/6~10/11~15/...같은 그룹)

3.5. 담당업무 거리 기준

담당업무도 마찬가지로 범주를 나누어서 거리를 구하였다. 다만, 여기서는 같은 계열로 나누고 그 안에서 또 비슷한 업무로 범주를 2 번 나누었다. 예를 들어, IT 계열의 업무에는 IT 업무, 정보보호업무가 존재하고 IT 업무에는 IT 부문장, IT 본부장 등이 해당되게 된다.

a. 같은 업무: 0

b. 상위, 하위 혹은 비슷한 업무: 1

c. 같은 계열 업무: 2

d. 다른 계열 업무: 3

이때, 각 업무를 나누는 범주의 기준은 아래 대신증권의 직무소개를 참고하였다. (대신증권, n.d.)
개략적인 범주만 존재하고 자세한 업무의 분류는 존재하지 않아 세부적인 범주 설정은 위 사이트의 내용을 토대로 임의로 설정하였다.

3.6. 경력 거리 기준

경력을 구하기에 앞서 각 인물이 종사했던 업무들의 범주를 나누었고 해당 인물의 경력 여부를 리스트로 만들어 각 업무에 종사했으면 1, 종사하지 않았으면 0 으로 설정하였다. 즉 리스트가 각 인물에 대한 경력 범주가 된다.

이때, 업무의 범주는 대신증권, 대신증권 자회사, 타증권사, 법·정책, 학교·교수·재단, 기타의 6 개의 범주로 나누었다.

a. 전체적으로 같은 업무 종사 (리스트 같음): 0

b. 법·정책, 학교·교수·재단, 기타에만 종사하였으면: 0.6*(1+같은 업무 종사한 개수)

+) 법·정책, 학교·교수·재단, 기타에만 종사하지 않았으면 이 세 범주는 기타로 통합

c. 두 인물이 종사했던 업종 중 다른 업종이 존재: 다른 업종끼리의 최대, 최소 거리를 구한 후 평균을 거리로 삼음. 만약 같은 업종이 있을 시 거리 - (0.5*같은 업종 개수)

+) 각 업종 간 거리는 대신자회사->대신->타증권->기타 순으로 순서의 차이를 거리로 지정. 단, 기타의 경우 모든 업종과 거리 4 로 고정

인물 리스트 예시

인물 1: 대신증권, 대신증권 자회사, 타증권사 종사 ->[1, 1, 1, 0, 0, 0]

인물 2: 타 증권사, 기타 종사 -> [0, 0, 1, 0, 0, 1]

3.7. 보유 주식수 거리 기준

먼저 보유 주식수를 구하고 주식수의 차이를 100 씩으로 잘라서 범주를 생성

a. 같은 주식수: 0

b. 주식수에 차이가 있으면 그 차의 범주 만큼을 거리로 설정

이때, 각 인물 간에 주식수 차의 편차가 심해 주식수 하나만으로 유사성 전체에 큰 영향을 미칠 가능성이 존재해 로그 10 으로 로그 스케일을 추가적으로 진행해주어 거리를 설정하였다.

추가적으로 전체 가중치를 생성 후 가중치 간의 차이를 더욱 크게 보기 위해 각각 구한 가중치에 제곱을 해 스케일을 조정해주었다.

* 자세한 내용은 부록 - 2 코드의 가중치함수 부분 참조

4. 네트워크 형성

네트워크 구축에 앞서 각 연도 별로 임원진 데이터를 살펴보았을 때, 각각 임원진이 대개 30 명 전후로 존재하였다. 모든 임원진에 대해서 연결성을 살펴보면 약 30 개 전후의 노드들이 각각과 관계를 맺으면 30x30 의 가중치 행렬이 생성되게 된다. 이렇게 네트워크를 형성하면, 네트워크의 복잡성이 올라가 시각화를 해도 인사이트를 얻기 매우 어려워진다. 따라서 네트워크 형성 시 임계값을 기준으로 임계값 아래의 가중치를 가진 에지는 지웠고, 임계값 이상의 에지만 남겨 연결해주었다. 여기서 임계값은 가중치의 삼분위수로 설정하였다. 즉, 상위 25%의 가중치의 엣지를 가지는 노드들만 연결되게 네트워크를 형성하였다. 추가적으로 앞서 선정한 관심인물과 비관심인물을 구별하기 위해 관심인물의 노드는 빨간색, 비관심인물의 노드는 파란색으로 설정해주었다. 네트워크 구축은 파이썬의 networkx 패키지를 사용하였다.

* 자세한 내용은 부록 - 2. 코드의 네트워크시각화 부분 참조

5. 분석 지표

분석에 사용한 지표는 총 5 가지이다(위키백과, 2022). 또한, 각 연도별 최종 중요인물을 선정하기 위해 5 가지 지표를 활용해 최종 중요인물 점수를 계산하였다.

5.1. 관심인물과 연결성

각 인물이 관심인물과 얼마나 연결되어 있는지 나타내는 지표이다. 중요인물과 연결된 수를 의미한다.

5.2. 연결정도 중심성(Degree Centrality)

한 노드가 얼마나 많은 노드들과 관계를 맺고 있는지 나타내는 지표이다. 연결정도가 높은 노드는 다른 노드들과의 관계를 통해 정보 획득에 용이하며, 네트워크 내에 핵심노드일 가능성이 높다.

5.3. 매개 중심성(Betweenness Centrality)

노드끼리 연결을 가장 빨리 해주는 경로 위에 있는 정도를 나타내며, 한 노드가 다른 노드들 사이에 위치하는 정도를 나타내는 지표이다. 매개 중심성이 높은 노드는 다른 핵심 노드들 사이에서의 중재자(broker) 역할을 수행하게 되며 핵심 노드로서 기능할 가능성이 높다.

5.4. 근접 중심성(Closeness Centrality)

각 노드 간의 거리를 바탕으로 중심성을 나타내는 지표이다. 다른 노드에 가장 빨리 연결하는 정도를 나타낸다. 연결정도 중심성은 직접 연결된 노드의 개수만을 파악한다면, 근접 중심성은 간접적으로 연결되어 있는 모든 노드 간의 거리를 바탕으로 중심성을 측정한다. 간접적인 중심성이 크다면, 핵심 노드와의 관계가 많다는 뜻으로 다수의 핵심 노드들과의 관계를 통해 해당 네트워크 내의 핵심 노드일 가능성이 높다.

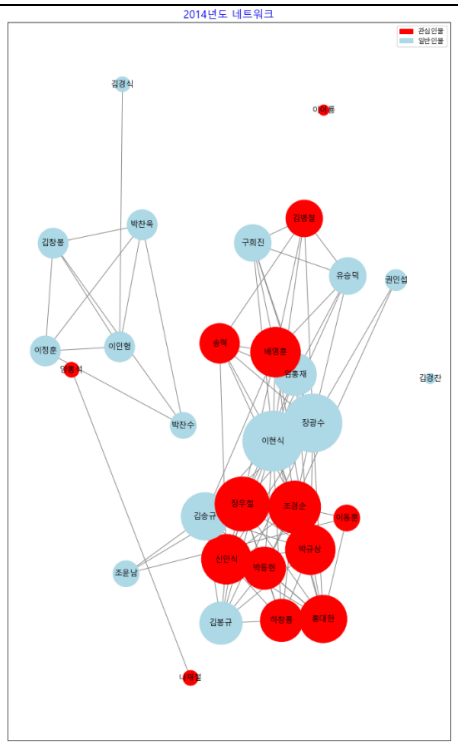
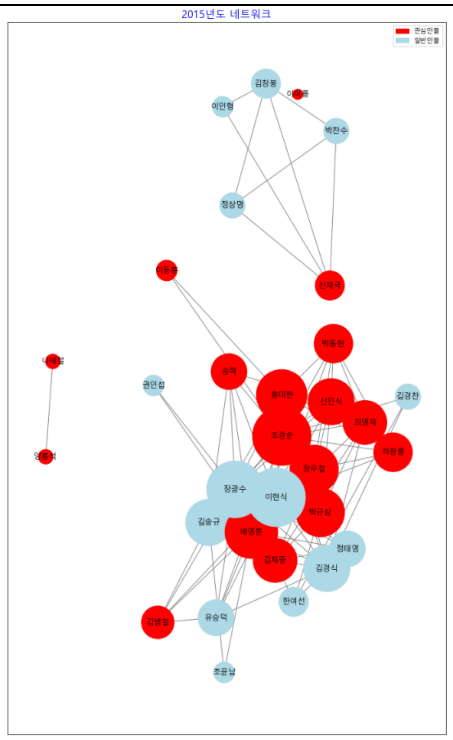
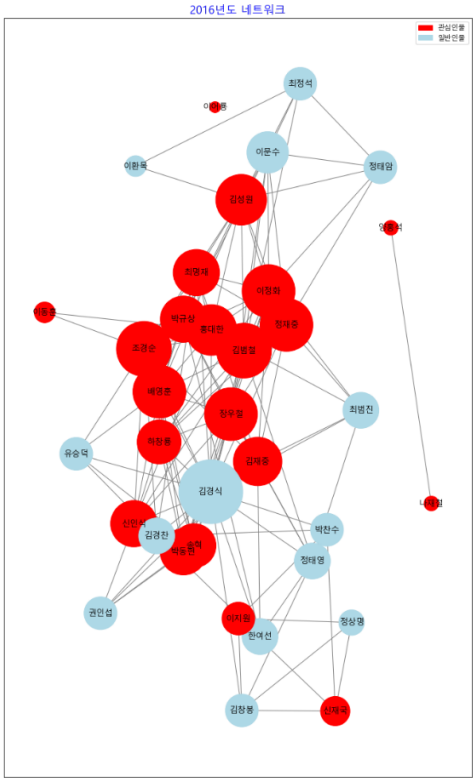
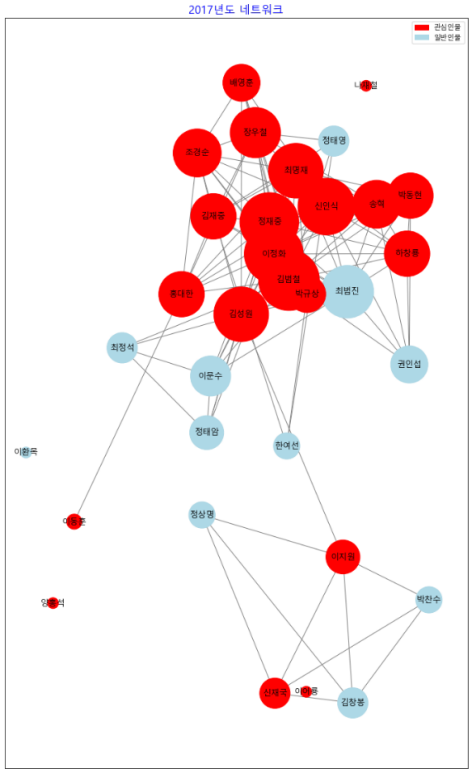
5.5. 아이겐벡터 중심성(Eigenvector Centrality)

한 노드와 연결된 다른 액터의 중심성을 가중치로 하여 계산하는 방식으로, 자신과 연결된 다른 노드들이 네트워크 내에서 얼마나 중요한지 파악하는 지표이다. 아이겐벡터 중심성이 높다면 연결정도가 적다 하더라도 높은 영향력을 가지며, 반대로 연결정도는 높지만 아이겐벡터 중심성이 낮다면 네트워크 내의 실질적인 영향력은 미미할 것으로 예측할 수 있다.

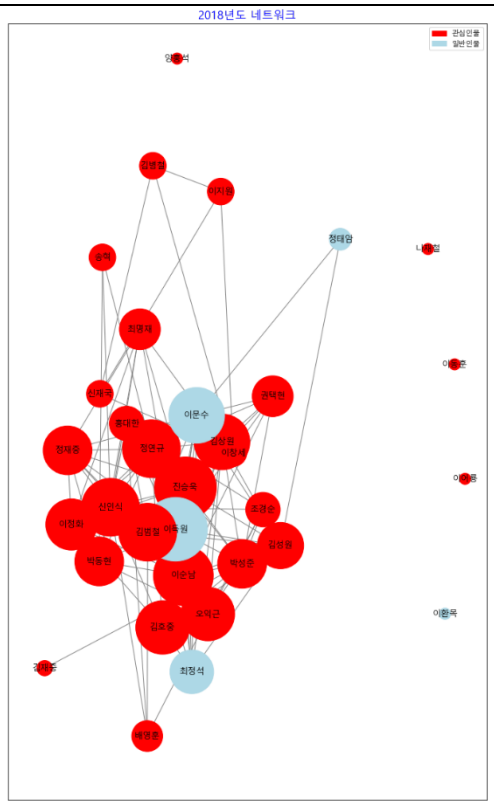
각 지표에 대해서 중요인물과 연결성은 직접 함수를 만들어 중요인물과 연결된 수를 count 해서 결과값으로 count 수를 나타내 주었고, 나머지 중심성에 대한 지표들은 networkx 패키지의 중심성 구하는 함수들을 사용하였다.

Ⅲ. 분석

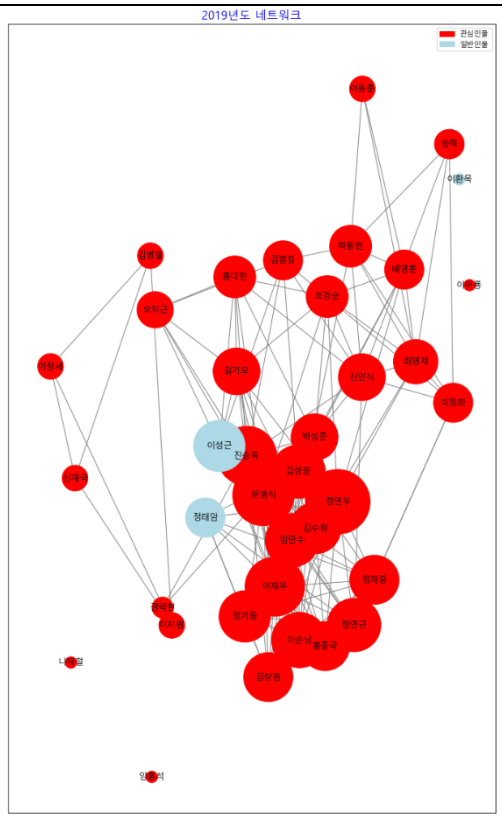
1. 인물 네트워크

2014	2015
	
2016	**2017**
	

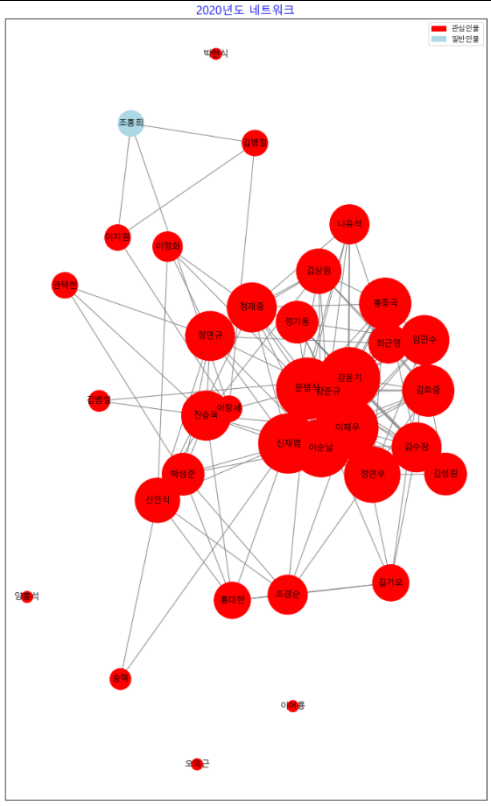
2018



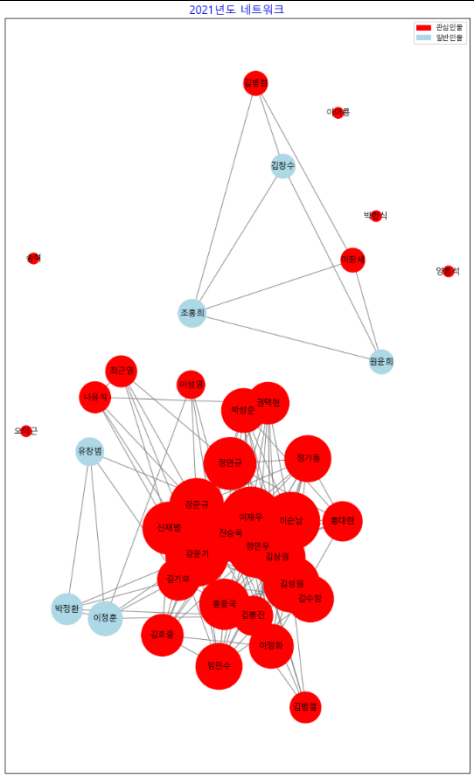
2019

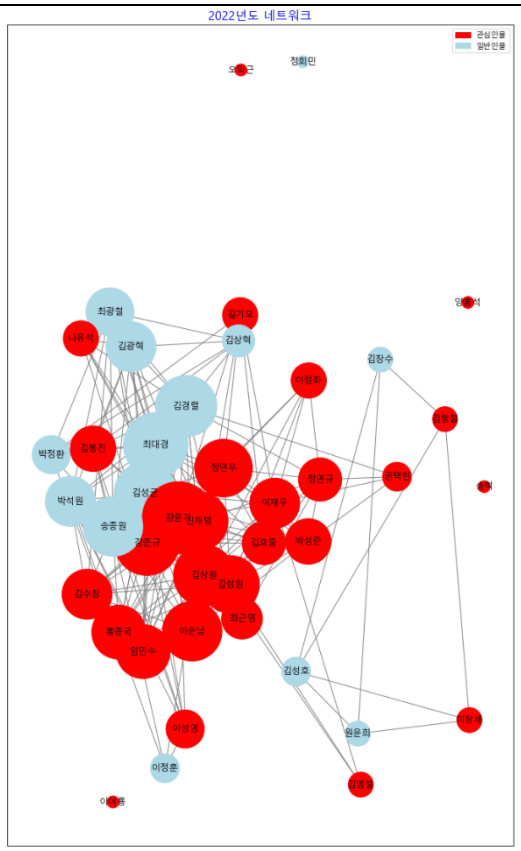
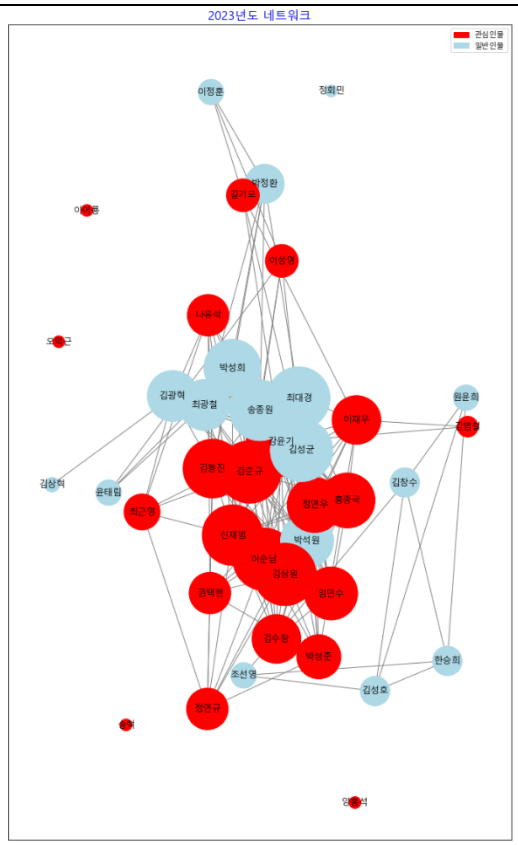


2020



2021



2022	2023
	

2. 연도별 인물의 중요도

2014

중요인물과 연결	Degree	Betweenness	Closeness	Eigenvector
이현식 10	이현식 0.552	장광수 0.064	이현식 0.566	이현식 0.351
조경순 9	장광수 0.517	이현식 0.056	장광수 0.541	장광수 0.320
장광수 8	장우철 0.448	장우철 0.034	장우철 0.498	장우철 0.305
홍대한 7	조경순 0.414	조경순 0.030	조경순 0.479	조경순 0.285
신인식 7	신인식 0.379	홍대한 0.017	신인식 0.461	박규상 0.283

* 각 수치는 소수점 4 번째 자리에서 반올림 진행

2015

중요인물과 연결	Degree	Betweenness	Closeness	Eigenvector
이현식 11	이현식 0.534	조경순 0.077	이현식 0.576	이현식 0.339
조경순 10	조경순 0.534	장광수 0.062	조경순 0.576	조경순 0.328
장광수 9	장광수 0.5	이현식 0.052	장광수 0.556	장광수 0.312
장우철 8	배영훈 0.434	배영훈 0.047	배영훈 0.520	홍대한 0.272
홍대한 8	홍대한 0.4	박규상 0.027	홍대한 0.504	배영훈 0.269

2016

중요인물과 연결	Degree	Betweenness	Closeness	Eigenvector
김경식 11	김경식 0.559	김경식 0.244	김경식 0.628	김경식 0.300
조경순 11	김범철 0.412	김성원 0.072	김범철 0.554	김범철 0.275
배영훈 11	조경순 0.412	조경순 0.068	조경순 0.554	배영훈 0.267
이정화 10	이정화 0.382	김창봉 0.066	배영훈 0.544	장우철 0.262
홍대한 10	장우철 0.382	김범철 0.064	홍대한 0.533	조경순 0.257

2017

중요인물과 연결	Degree	Betweenness	Closeness	Eigenvector
이정화 12	김범철 0.516	김성원 0.268	김성원 0.560	이정화 0.316
신인식 12	이정화 0.484	이지원 0.199	김범철 0.560	정재중 0.316
최명재 12	정재중 0.484	김범철 0.098	이정화 0.547	김범철 0.310
정재중 12	신인식 0.452	홍대한 0.057	정재중 0.547	신인식 0.303
김범철 11	김성원 0.419	이정화 0.045	신인식 0.523	최명재 0.285

2018

중요인물과 연결	Degree	Betweenness	Closeness	Eigenvector
이득원 15	이득원 0.531	박성준 0.055	이득원 0.570	이득원 0.321
진승욱 13	진승욱 0.5	신인식 0.050	진승욱 0.551	진승욱 0.307
신인식 13	이순남 0.469	김범철 0.045	이순남 0.533	이순남 0.293
이순남 12	정연규 0.438	이문수 0.034	정연규 0.517	정연규 0.267
정연규 12	신인식 0.438	정연규 0.032	김상원 0.501	김상원 0.255

2019

중요인물과 연결	Degree	Betweenness	Closeness	Eigenvector
정연우 17	정연우 0.528	정연우 0.058	정연우 0.589	정연우 0.314
문병식 16	문병식 0.472	진승욱 0.053	진승욱 0.531	문병식 0.296
진승욱 15	진승욱 0.445	신인식 0.036	문병식 0.531	이재우 0.294
이재우 14	이재우 0.445	최명재 0.031	이재우 0.519	이순남 0.270
이순남 13	이순남 0.389	문병식 0.031	정연규 0.495	진승욱 0.245

2020

중요인물과 연결	Degree	Betweenness	Closeness	Eigenvector
강윤기 18	강윤기 0.514	강윤기 0.062	강윤기 0.579	이재우 0.297
이재우 17	이재우 0.486	문병식 0.054	이재우 0.563	강윤기 0.296
문병식 17	문병식 0.486	신재범 0.048	강준규 0.548	강준규 0.280
강준규 16	강준규 0.457	진승욱 0.040	문병식 0.548	이순남 0.276
이순남 15	이순남 0.429	이재우 0.037	이순남 0.534	문병식 0.273

2021

중요인물과 연결	Degree	Betweenness	Closeness	Eigenvector
정연우 19	강윤기 0.541	강윤기 0.079	강윤기 0.579	정연우 0.292
이재우 18	정연우 0.514	이재우 0.048	정연우 0.563	이순남 0.280
강윤기 18	이재우 0.514	강준규 0.030	이재우 0.563	이재우 0.279
이순남 17	이순남 0.459	정연우 0.026	이순남 0.533	강윤기 0.277
김성원 16	김성원 0.432	길기모 0.018	김성원 0.518	김상원 0.268

2022

중요인물과 연결	Degree	Betweenness	Closeness	Eigenvector
강윤기 18	강윤기 0.610	강윤기 0.075	강윤기 0.633	강윤기 0.300
정연우 14	강준규 0.488	김경렬 0.047	강준규 0.558	최대경 0.267
김상원 14	최대경 0.488	최대경 0.031	최대경 0.558	김성균 0.261
강준규 13	김성균 0.463	강준규 0.030	김성균 0.545	강준규 0.259
이순남 13	김상원 0.439	김성균 0.020	김경렬 0.533	김상원 0.248

2023

중요인물과 연결	Degree	Betweenness	Closeness	Eigenvector
강윤기 15	강윤기 0.564	강윤기 0.072	강윤기 0.599	강윤기 0.287
이순남 13	강준규 0.462	김광혁 0.042	강준규 0.539	최대경 0.264
김상원 13	이순남 0.462	박성희 0.028	최대경 0.539	김성균 0.262
신재범 12	최대경 0.462	김성균 0.026	김성균 0.539	이순남 0.262
정연우 12	김상원 0.462	강준규 0.025	송종원 0.526	김상원 0.262

* 현재 절의 연도별 인물의 중요도는 상위 5 명만 추출한 것으로, 전체 인물에 대한 결과는 부록 - 2. 코드 실행결과를 참조

3. 연도별 네트워크 변화

Ⅱ-2절에서 선정한 관심인물의 기준이 연도별 회사 매출의 증가에 따른 개인의 변화였기 때문에, 연도가 지날수록 회사에 오래 남아있는 사람들은 회사의 매출의 증가를 더 많이 겪었기 때문에 자연히 관심인물이 될 수밖에 없다. 위 결과의 네트워크 그래프에서도 볼 수 있듯이 연도가 지날수록 관심인물(빨간색 노드)의 수가 많아지는 것을 확인할 수 있었다. 특히 18, 19, 20, 21년도에 관심인물이 많이 증가한 것을 확인할 수 있었는데, 이 해에 매출액이 증가한 것이 영향을 미치는 것으로 추정할 수 있다. 21년도 이후 다시 비관심인물의 수가 늘어난 것을 확인할 수 있었는데, 이는 21년도 이후에 매출액의 증가가 없어서 네트워크에 새로 유입된 사람은 관심인물로 선정되지 않은 것으로 추정된다. 또한 보통 전년도에 중요도 지표값이 높았던 사람들이 다음 해에도 높은 중요도 값을 유지하는 패턴을 발견할 수 있었다.

4. 분석 결과

Ⅲ-2절의 연도별 인물의 중요도를 살펴보면, 관심인물과 연결된 여부의 지표값이 높은 사람일수록 전체적으로 다른 지표의 값들이 높게 나와 네트워크상의 중요도가 높은 사람임을 확인할 수 있었다. 이를 통해 처음에 선정한 관심인물 선정 기준이 유의미하다고 판단할 수 있으며, 매출액의 증가에 따른 개인의 변화가 회사 내에서의 인물의 중요도를 판단할 수 있는 간략한 지표로서 작동할 수 있다는 점을 확인할 수 있었다.

관심인물들과 연결이 많이 된 인물들에 대해서도 추가로 분석을 진행하여 연도별로 관심인물과 연결된 여부 지표값이 높은 상위 5명을 뽑아서 그 인물의 특성을 살펴보았는데, 모두 대신증권에서 일한 경험이 있던 사람들이었으며, 그 중 거의 모든 사람들은 대신증권에서만 일했던 사람들임을 확인할 수 있었다. 겹치는 인물들을 제외하고 총 24명의 인물들 중 4명을 제외한 83%의 사람들은 전부 대신증권에서만 종사했던 사람들이며, 남은 4명 역시 대신증권에서 일했던 경험이 있고 김범철, 진승욱, 이득원과 같은 사람들은 대신증권과 대신증권의 자회사에서만 일했던 사람들이며 김경식의 경우만 대신증권과 타증권사에서 일했던 경험을 가지고 있는 사람임을 확인하였다. 이를 통해 대신증권에서만 일했던 요소가 인물네트워크에서의 영향력에 영향을 미칠 수 있음을 알 수 있다. 이와 같이 분석 결과로 도출된 중요 인물들의 공통점을 파악하면 인물의 영향력을 파악하는 데 활용할 수 있을 것으로 사료된다.

이어룡, 양홍석과 같이 직급이 높은 인물들은 다른 인물들과 연결된 노드가 적고, 연결이 많은 대부분의 인물들은 중간 직급에 속하는 사람들인 것을 확인할 수 있었다. 이는 최상위 계급은 다른 인물들과 유사성이 적으며, 이 유사성을 관계라 한다면 다른 인물들과 맺고 있는 관계가 적을 것이며, 중간 직급들은 활발한 관계를 갖고 있다고 볼 수 있으며 이러한 특징의 지배구조를 가지고 있을 것이라고 추측할 수 있을 것 같다.

분석결과의 연도별 네트워크 변화에서 전년도에 중요도 지표값이 높았던 사람들이 다음 해에도 높은 중요도 값을 유지하는 패턴을 발견할 수 있었는데, 이는 우리가 선정한 관심인물들이 영향력과 권력을 유지하여 기업 내에서 중요한 역할을 수행하는 것이라 판단할 수 있다.

5.최종 중요인물 선정

5.1. 최종 중요인물 점수 계산

각 연도별 네트워크 상 최종 중요인물을 선정하기 위해 Ⅱ-6절의 5가지 분석지표를 활용해 최종 중요인물 점수를 산정하였다. 5가지 지표를 같은 중요도로 생각하기 보다는, 기업의 입장에서 어떤 중심성이 더 중요한지 생각해 보다 중요하다고 생각되는 지표에 타 지표보다 높은 가중치를 주었다.

본 프로젝트에서는 기업입장에서 '중간관리자'의 역할에 주목하였다. ESG경제 기사에 따르면 중간관리자는 ESG의 전 과정을 다 꿰고 있어 조직의 버팀목으로서 역할을 한다고 한다. 기업에서 중간 관리자는 경영자와 조직원 중간에서 회사 발전의 열쇠를 쥔 리더로 중요성을 가진다(이후경, 2023).

중간관리자의 중요성을 바탕으로 노드들 사이에 중재자 역할을 수행하는 노드에 관한 지표인 매개 중심성(betweenness centrality)가 중요하다고 인식하였다. 또한 Cambridge intelligence의 마케팅 디렉터 Andrew Disney에 의하면 매개 중심성이 높은 노드는 시스템의 노드 사이의 브릿지(bridge) 역할을 해 준다고 한다(Andrew Disney, 2020). 따라서 매개 중심성을 가장 중요한 지표로 선정하였다.

다음으로 조직에서 '커뮤니케이터'의 역할에 주목해 다음으로 중요한 지표를 연결정도 중심성(Degree Centrality)로 선정하였다. 커뮤니케이터는 동료들과 잘 지내며 조직의 유대관계를 끈끈하게 만들어 조직에서 꼭 필요한 존재이다(jobplanet, 2021년). 연결정도 중심성이 높은 인물은 타 인물들과 많은 관계를 가지므로 조직 내 커뮤니케이터로서 활약할 수 있으므로 매개 중심성 다음으로 중요한 지표로 선정하였다.

위에서 언급한 Andrew Disney는 근접 중심성(Closeness Centrality)은 하나의 조직(군집)에서 유의미한 지표로 사용된다고 한다. 본 프로젝트의 네트워크는 다양한 군집이 형성되므로 근접 중심성을 가장 덜 중요한 지표로 선정하였다.

이를 바탕으로 최종 중요인물 점수는 다음과 같이 산정하였다. (최종 중요인물 점수 산정 전, 각 지표를 모두 정규화 해 스케일을 0~1 사이의 값으로 맞추어 주었다.)

최종 중요인물 점수 = ((관심인물과 연결성) x 1.5 + (연결정도 중심성) x 2.5 + (매개 중심성) x 3.5 + (근접 중심성) x 1 + (아이겐벡터 중심성) x 1.5) x 100

아래 표는 위의 점수 계산식에 바탕 한 연도별 최종 중요인물을 상위 5명까지 나타낸 결과이다.

*최종 중요인물 점수는 1000점 만점 기준

연도	인물	최종 점수
2014	이현식	955.1
	장광수	939.0
	장우철	713.4
	조경순	696.1
	홍대한	541.1
2015	조경순	983.3
	이현식	885.5
	장광수	872.4
	배영훈	714.3
	홍대한	626.7
2016	김경식	1000
	조경순	649.7
	김범철	636.6
	배영훈	578.1
	장우철	559.3
2017	김성원	890.5
	김범철	763.7
	이정화	689.6
	정재중	689.6
	신인식	654.0
2018	신인식	855.6
	이득원	846.0
	김범철	811.9

	진승욱	769.6
	박성준	761.3
2019	정연우	1000
	진승욱	874.3
	문병식	781.5
	이재우	653.4
	정연규	628.3
2020	강윤기	997.1
	문병식	990.6
	신재범	953.8
	이재우	890.0
	이순남	736.8
2021	강윤기	987.0
	이재우	832.5
	정연우	747.2
	이순남	650.1
	강준규	639.3
2022	강윤기	1000
	김강렬	689.5
	최대경	671.9
	강준규	668.2
	김성균	606.3
2023	강윤기	1000
	김성균	674.1
	강준규	659.1
	최대경	640.4
	김상원	625.4

5.2. 중요인물 검증

본 프로젝트에서 최종적으로 선정한 중요인물이 실제로 중요한 중요한 인물인지를 확인하기 위해 다음과 같은 기준으로 검증을 진행하였다.

1. 이전해에서 선정된 해로 넘어오는 시점에 승진 여부
2. 선정 당해에서 다음해로 넘어가는 시점에 승진 여부
3. 2 년 연속 선정 여부 : 이전해와 당해 선정

위의 기준에 하나라도 부합하는 인물은 True, 그렇지 않은 인물은 False하여 검증을 통과하는 인물의 비율을 살펴보았다.

2014	이현식 False	장광수 False	장우철 False	조경순 False	홍대한 False
2015	조경순 True	이현식 True	장광수 True	배영훈 False	홍대한 True
2016	김경식 False	조경순 True	김범철 False	배영훈 True	장우철 True
2017	김성원 False	김범철 True	이정화 False	정재중 False	신인식 False

2018	신인식 True	이득원 False	김범철 True	진승욱 True	박성준 True
2019	정연우 False	진승욱 True	문병식 False	이재우 False	정연규 True
2020	강윤기 False	문병식 True	이재우 True	신재범 False	강준규 False
2021	강윤기 True	이재우 True	정연우 True	이순남 False	강준규 True
2022	강윤기 True	김경렬 False	최대경 False	강준규 True	김성균 False
2023	강윤기 True	김성균 True	강준규 True	최대경 True	김상원 False

검증을 진행한 결과 각 인물별 통과여부는 위와 같고, 통과한 인물의 비율은 0.52가 나왔다. 이는 우리가 중요인물로 뽑은 인물들이 50%이상은 정확하다는 의미로 볼 수 있으며 정확도를 좀 더 높이기 위해 GridSearch를 이용한 최적화를 진행하였다.

⑴ 가능한 가중치 조합을 [0.1, 0.5, 1, 1.5, 2, 2.5, 3]으로 설정 후 GridSearch 진행

최적의 가중치의 조합 = {0.1, 0.1, 1, 2, 0.1}

* {관심인물과 연결성, 연결정도 중심성, 매개 중심성, 근접 중심성, 아이겐 벡터 중심성} 순

2014	장광수 False	이현식 False	장우철 False	조경순 False	배영훈 False
2015	조경순 True	장광수 True	이현식 True	배영훈 True	홍대한 False
2016	김경식 False	조경순 True	김범철 False	배영훈 True	홍대한 True
2017	김성원 False	김범철 True	이정화 False	정재중 False	신인식 False
2018	신인식 True	이득원 False	김범철 True	박성준 True	진승욱 True
2019	정연우 False	진승욱 True	문병식 False	신인식 True	정연규 True
2020	강윤기 False	문병식 True	이재우 True	신재범 False	강준규 False
2021	강윤기 True	이재우 True	정연우 True	이순남 False	강준규 True
2022	강윤기 True	김경렬 False	최대경 False	강준규 True	김성균 False
2023	강윤기 True	김성균 True	강준규 True	최대경 True	김봉진 True

최적화 진행결과, 정확도가 0.56으로 오른 것을 확인하였다.

⑵ 가능한 가중치 조합을 [0.1, 0.3, 0.5, 0.7, 0.9]으로 설정 후 GridSearch 진행

최적의 가중치의 조합 = {0.3, 0.3, 0.9, 0.7, 0.1}

2014	이현식 False	장광수 False	장우철 False	조경순 False	홍대한 False
2015	조경순 True	장광수 True	이현식 True	배영훈 False	홍대한 True
2016	김경식 False	조경순 True	김범철 False	배영훈 True	김성원 False
2017	김성원 True	김범철 True	이정화 False	정재중 False	신인식 False
2018	신인식 True	이득원 False	김범철 True	박성준 True	진승욱 True
2019	정연우 False	진승욱 True	문병식 False	신인식 True	정연규 True
2020	강윤기 False	문병식 True	신재범 False	이재우 True	강준규 False
2021	강윤기 True	이재우 True	정연우 True	강준규 True	이순남 False
2022	강윤기 True	김경렬 False	최대경 False	강준규 True	김성균 False
2023	강윤기 True	김성균 True	강준규 True	최대경 True	김봉진 True

최적화 진행결과, 정확도가 0.56임을 확인하였다.

⑶ 가능한 가중치 조합을 0.005부터 1까지 10개의 값으로 설정하고, 가중치의 합이 1이 되도록 정규화하여 진행

최적의 가중치의 조합 = {0.00402865, 0.00402865, 0.27126231, 0.71665175, 0.00402865}

2014	장광수 False	이현식 False	장우철 False	조경순 False	배영훈 False
2015	조경순 True	장광수 True	이현식 True	배영훈 True	홍대한 False
2016	김경식 False	조경순 True	김범철 False	홍대한 True	배영훈 True
2017	김성원 False	김범철 True	이정화 False	정재중 False	신인식 False
2018	이득원 False	신인식 True	김범철 True	박성준 True	진승욱 True
2019	정연우 False	진승욱 True	문병식 False	신인식 True	정연규 True
2020	강윤기 False	문병식 True	이재우 True	신재범 False	강준규 False
2021	강윤기 True	이재우 True	정연우 True	이순남 False	강준규 True
2022	강윤기 True	김경렬 False	최대경 False	강준규 True	김성균 False
2023	강윤기 True	김성균 True	강준규 True	최대경 True	김봉진 True

최적화 진행결과, 정확도가 0.56가 나오는 것을 확인하였다.

최적화까지 진행 후 최종 정확도는 0.56이 나오는 것을 확인하였고, 정확도를 높이고자 수행한 최적화에서 상대적으로 매개중심성과 근접 중심성에 가중치를 더 부여하는 것도 확인할 수 있었다. 이를 통해 네트워크 상에서 이 두개의 지표가 좀 더 중요한 역할을 한다고 추론할 수 있었고, 매개중심성은 앞서 5.1절에서 산정한 것과 같이 중요하다고 판단한 기준이 상당히 맞았던 것을 확인할 수 있었다. 반면, 근접중심성의 경우 가장 덜 중요하다고 판단하고 중요인물을 뽑았으나 GridSearch 결과 가장 큰 가중치를 부여받아 인물 간의 간접연결성의 정도가 중요인물 선정에 큰 영향을 미친다고 추론할 수 있었다.

한정적인 데이터로 인해 실제 중요한 역할을 하는 인물임을 검증할 기준이 적어 정확도가 높지 않아 보일 수 있으나, 더 많은 데이터를 이용해 다양한 검증 기준을 세워 진행한다면 좀 더 높은 정확도를 기대할 수 있을 것으로 보인다.

Ⅳ. 결론 및 고찰

본 프로젝트에서는 유사성을 기반으로 인물 간의 관계성을 분석하였기 때문에 상대적으로 유사성이 적은 핵심 수뇌부인 회장, 대표이사, 부사장 등과 같은 직위의 인물들 보다는 중간의 상무, 전무, 감사위원 위주의 분석을 진행하게 되었다.

설정한 지표에 따라 네트워크에서 영향력이 있는 중요인물을 선정할 수 있었으며, 분석 전 설정한 관심인물에 대한 지표가 유의미할 수도 있다는 점을 확인할 수 있었다. 또한 네트워크의 형태로 회사의 지배구조에 대해서도 생각해볼 수 있었으며, 연도별 지표값에 따른 네트워크의 패턴에 대해서도 살펴볼 수 있었다. 또한, 지표에 선정된 상위 사람들의 특성이 어느정도 유사할 수 있음을 확인해볼 수 있었다.

관심인물 선정을 토대로 관심인물과 비관심인물들 간의 관계를 파악하고, 네트워크 구조에서의 역할과 영향력을 분석하였다. 이를 통해 비관심인물들 중에서 관심인물들과 많이 연결되어

두각을 드러내는 인물을 찾아낼 수 있었으며, 또한 관심인물들 중에서 네트워크의 중심에 있는 인물들을 분석할 수 있었다.

관심인물들의 식별과 네트워크 분석을 통해 조직 내에서 영향력과 권력을 가진 인물들과 어느 시기에 어떤 네트워크가 활성화되는지 파악할 수 있으며 이는 리더들이 어떤 인물들을 중심으로 네트워크를 형성해야 하는지에 대한 결정에 도움을 줄 수 있다. 또한 조직의 변화와 동향을 파악하고 리더십의 효과를 평가를 통해 향후 전략과 의사결정에 반영할 수 있다.

또한 선정한 5가지의 중요성 지표들을 토대로 중요인물을 선정해 이 중요인물을 나름의 기준으로 검증해보았으며 0.56정도의 정확도를 얻을 수 있었다.

본 프로젝트를 진행하며 한계점도 많이 찾을 수 있었는데, 유사성에만 국한하여 네트워크를 형성하였기 때문에 인물 간의 연결성을 보기에는 다양성과 정보가 조금 부족하다는 점이 존재한다. 유사성이 연결성을 설명할 수 있다는 가정으로 시작하였기 때문에 유사성과 연결성 사이를 증명할 수 있는 정보와 데이터들이(인물 간의 연결성을 직접적으로 나타내는 정보 ex. 연락을 주고받는지 여부 등) 부족하였다. 추가적인 정보나 지표들에 관해서는 크롤링을 통해 기사에서 이름이 묶여 나오는 횟수로 인물 간의 관계에 대한 특성과 기사수를 인물의 사회적 영향력으로 잡아서 네트워크를 형성하고 분석해보는 방법도 고려해볼 수 있을 것이다. 즉, 더 많은 데이터를 활용하여 유사성 외 다른 연결 관계를 추가하고, 사회적 평판을 판단할 수 있는 데이터를 활용해 중요도 지표를 추가할 수 있을 것이다. 다양한 지표를 활용하면 보다 정확하고 포괄적인 네트워크 분석이 가능해질 것으로 예상한다.

또한 유사성 기반으로 형성한 네트워크에서 중요인물을 선정하였는데, 이러한 중요인물의 검증에 필요한 정보와 데이터들도 많이 부족하였다. 검증 역시 승진여부와 연속중요인물 선정을 기준으로 판단하였지만, 승진의 여부만으로는 기업에서의 인물의 중요성을 전부 파악하기에는 부족함이 있고, 연속중요인물 선정 여부 역시 본 프로젝트에서 고안한 지표를 사용한 것이기에 검증 기준에 대한 검증이 더 필요하다. 이에 관해서 인물의 실적이나 포상금 여부 등의 중요성을 나타낼 수 있는 데이터가 더 존재한다면 풍부한 분석을 할 수 있으리라 기대된다.

VI. 참고자료

김종운. (2017). *만남 그리고 성장을 위한 인간관계 심리학*. 학지사.

대신증권(n.d.). 직무소개. http://money2.daishin.co.kr/company/recruit/job_introduce.shtml

박경미, 김성환 and 조환규. (2013). 소설 등장인물의 텍스트 거리를 이용한 사회 구성망 분석. 한국콘텐츠학회 논문지, 13(4), 427-439.

위키백과(2022). 사회 연결망 분석. https://ko.wikipedia.org/wiki/사회_연결망_분석

이후경. (2023년 06월 11일). ESG의 내재화, 팀장급 중간 관리자 리더십이 열쇠. *ESG경제* https://www.esgeconomy.com/news/articleView.html?idxno=3841

jobplanet. (2021년. 12월. 29일). "말해봐 들어줄게" 분위기 커뮤니케이터. https://www.jobplanet.co.kr/contents/news-2489

Andrew Disney. (2020년 1월 2일). Social network analysis 101: centrality measures explained by Andrew Disney. *Cambridge Intelligence*. https://cambridge-intelligence.com/keylines-faqs-social-network-analysis/

Byrne, D. (1997). *An overview (and underview) of research and theory within the attraction paradigm*. Journal of Social and Personal Relationships, 14(3), 417-431.

VII. 부록

1. 데이터

데이터는 DART(전자 공시 시스템)에 기재된 데이터를 사용하였다. DART 에 OpenDartAPI 가 존재해 공시정보를 바로 파이썬으로 받아올 수 있지만, 이렇게 받아온 데이터에는 임원진의 '의결권 있는 주식수'가 누락 되어있고, 연도도 한정되어 있었다. 이러한 이유로 따로 노션(Notion)에 데이터베이스를 직접 만들어 파이썬으로 가져오는 방식을 사용하였다.

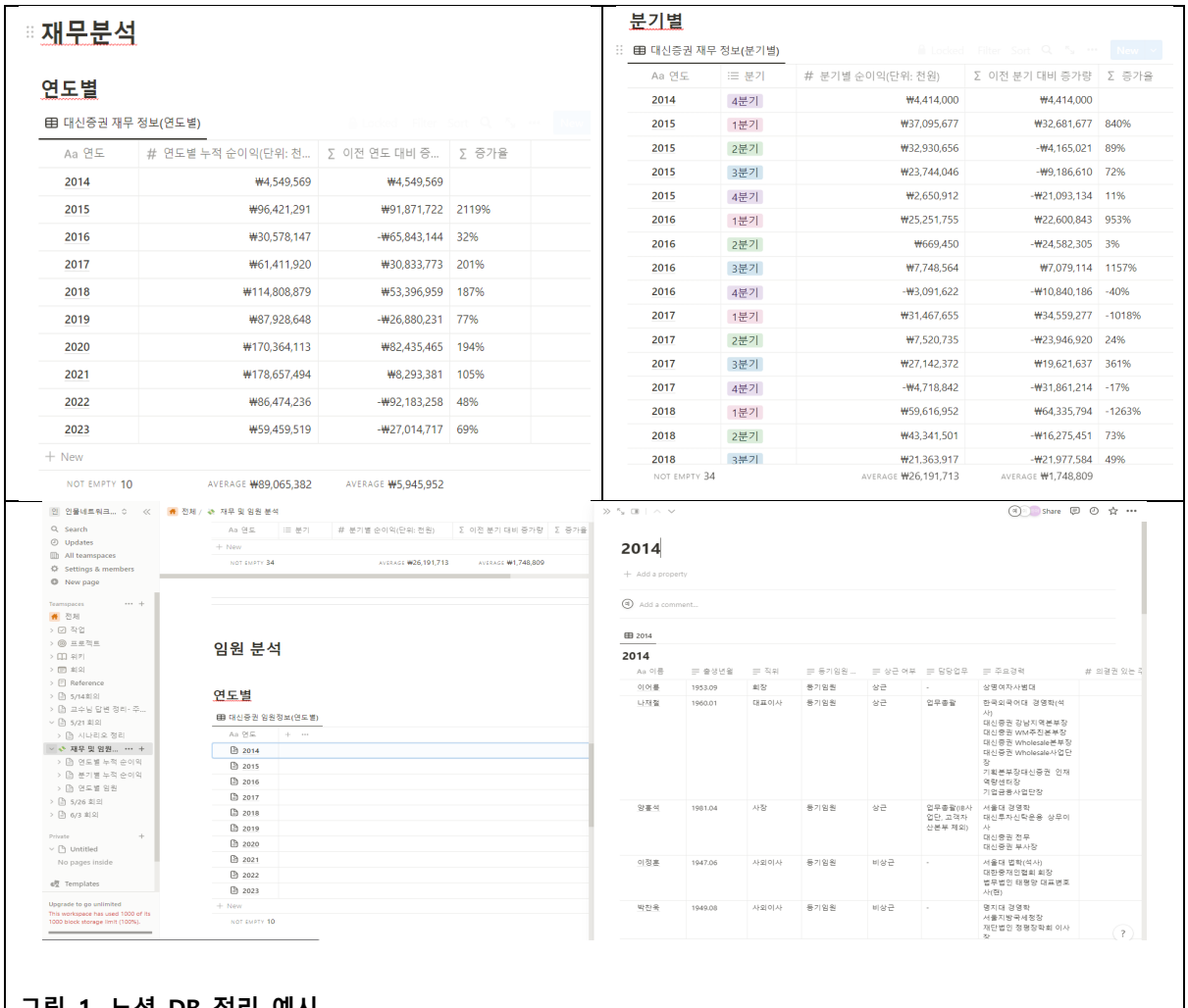


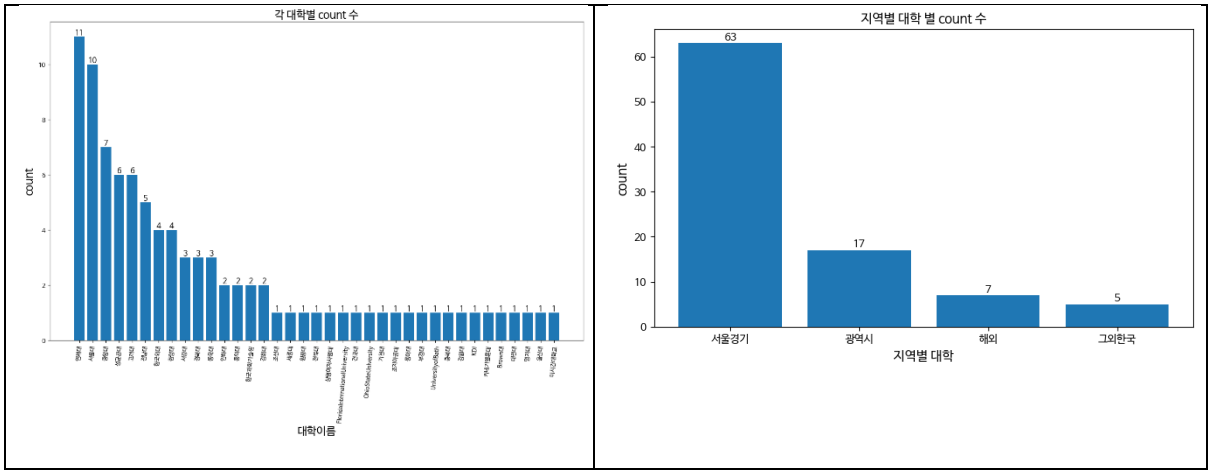
그림 1. 노션 DB 정리 예시

2. 데이터 분석

가중치 함수 설정과 함께 기본적인 데이터셋 분석과 가중치 함수에 쓰인 대학, 전공, 직위, 근속연수, 담당업무, 경력, 보유 주식 수 7개의 피처에 대한 분석도 진행하였다. 전체적인 데이터셋은 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023년도의 총 10년의 임원들에 대한 데이터셋이며 354개의 데이터가 존재한다. 14년도 임원은 총 30명, 15년도 임원은 총 31명, 16년도 임원은 총 35명, 17년도 임원은 총 32명, 18년도 임원은 총 33명, 19년도 임원은 총 37명, 20년도 임원은 총 36명, 21년도 임원은 총 38명, 22년도 임원은 총 42명, 23년도 임원은 총 42명이 존재한다. 각 년도에 중복없이 10년간 존재했던 임원은 총 92명이다. 피처에 대한 분석은 총 92명의 중복 없는 데이터에 대해서 진행하였으며, 연도별 중복 임원에 관해서는 가장 최신의 데이터를 사용하였다.

2.1. 대학

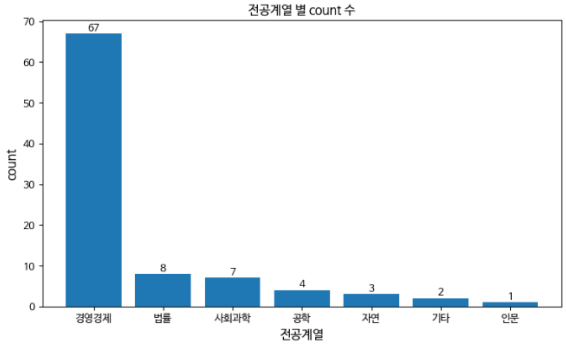
먼저 대학 피처에 대해 살펴보면 대학교는 총 37개가 존재하며 연세대가 11명으로 가장 많고, 그 다음은 서울대 10명, 중앙대 7명, 성균관대, 고려대가 6명 순으로 많았다. 3.1절에서 서술했듯이 지역별로 대학을 나누어 가중치 함수를 구성하였기 때문에 지역별 대학에 대해서도 살펴보았다. 마찬가지로 지역은 서울/경기, 광역시, 그 외 한국, 해외로 나누었으며 서울/경기 지역의 대학교에 63명, 광역시 17명, 해외에 7명, 그 외 한국에 5명이 존재하였다. 서울/경기 지역의 대학이 압도적으로 많은 수의 임원들이 졸업한 것을 확인하였고, 광역시, 해외, 그 외 한국 지역의 대학 순으로 졸업한 임원들이 존재하는 것을 확인할 수 있었다.



<표2> 대학별 임원의 수를 count한 막대그래프 왼쪽은 각 대학별 임원의 수, 오른쪽은 지역별 대학에 대한 임원의 수이다.

2.2. 전공

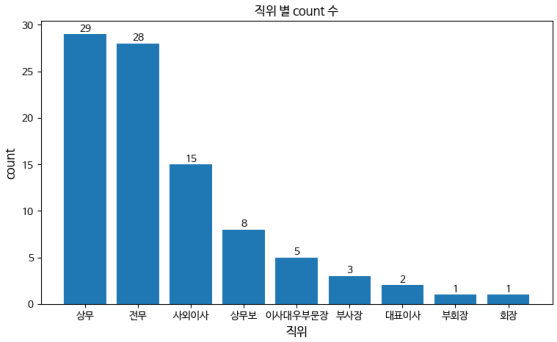
전공도 마찬가지로 가중치 함수에서 사용한 기준에 따라 전공 계열에 따라 임원의 수를 확인해보았다. 경영경제 계열인 전공의 임원들이 67명으로 압도적으로 많았으며, 법률, 사회과학이 각각 8명, 7명으로 존재하는 것을 확인할 수 있었다. 이 분석을 통해 임원들의 상당수는 경영이나 경제 관련된 전공을 전공한 것을 확인할 수 있었다.



<표3> 전공계열별 임원의 수를 count한 막대그래프

2.3. 직위

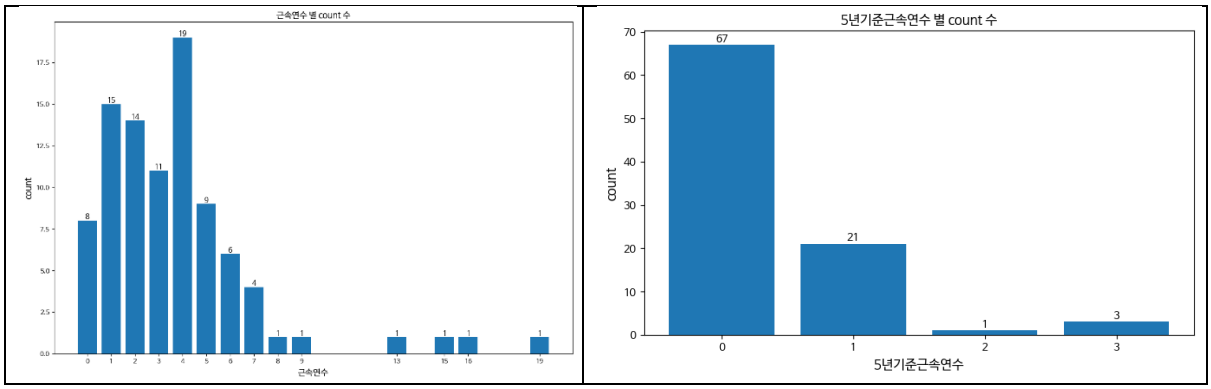
직위는 상무가 29명, 전무가 28명으로 많고 부회장, 회장은 1명씩 존재하는 것을 확인할 수 있다.



<표4> 직위별 임원의 수를 count한 막대그래프

2.4. 근속연수

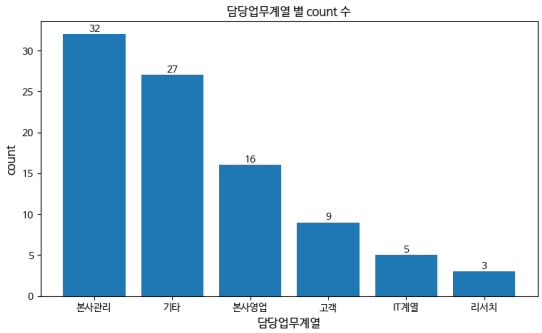
근속연수 별 임원은 4년일 때 19명으로 가장 많고, 그 다음은 1년일 때 15명, 2년일 때 14명 순으로 많았다. 근속연수는 주로 0~9년 사이에 몰려 있으며 13, 15, 16, 19년에 1명씩 존재한다. 가중치 함수와 마찬가지로 5년 기준으로 자른 근속연수에 대해서도 살펴보았는데, 0~9년에 몰려 있는 만큼 0~4년까지 67명, 5~9년까지 21명, 10~14에 1명, 15~19에 3명 존재하는 것을 확인할 수 있다. 추가로 근속연수는 최소가 0년, 최대가 19년, 평균이 3.7065년 정도 되며, 25% 사분위수가 1.75년, 50% 사분위수가 3년, 75%사분위수가 5년이 된다.



<표5> 근속연수별 임원의 수를 count한 막대그래프 왼쪽은 전체 근속연수별 임원의 수, 오른쪽은 5년 기준으로 자른 근속연수 별 임원의 수이다.

2.5. 담당업무

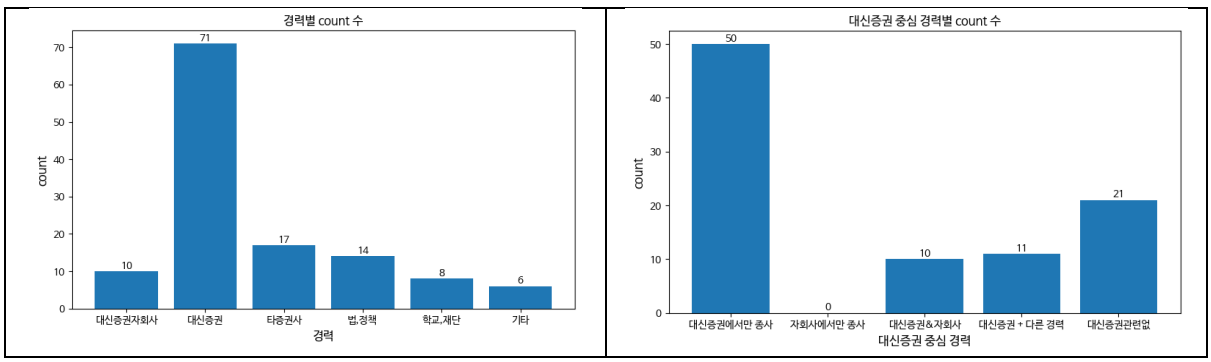
담당업무도 마찬가지로 가중치 함수에서의 계열로 나눈 것에 대해 담당업무계열별 임원의 수를 살펴보았다. 본사관리 계열이 32명으로 제일 많고, 기타가 27명, 본사 영업이 16명 순으로 존재하였으며 리서치 계열이 3명으로 가장 작았다. 다만, 이에 대해서는 데이터의 담당업무를 정확한 기준을 구할 수 없어 3.5절과 같이 전체적인 직무소개를 기준으로 임의로 나눈 것이기에 차이가 존재할 수도 있다.



<표6> 근속연수별 임원의 수를 count한 막대그래프

2.6. 경력

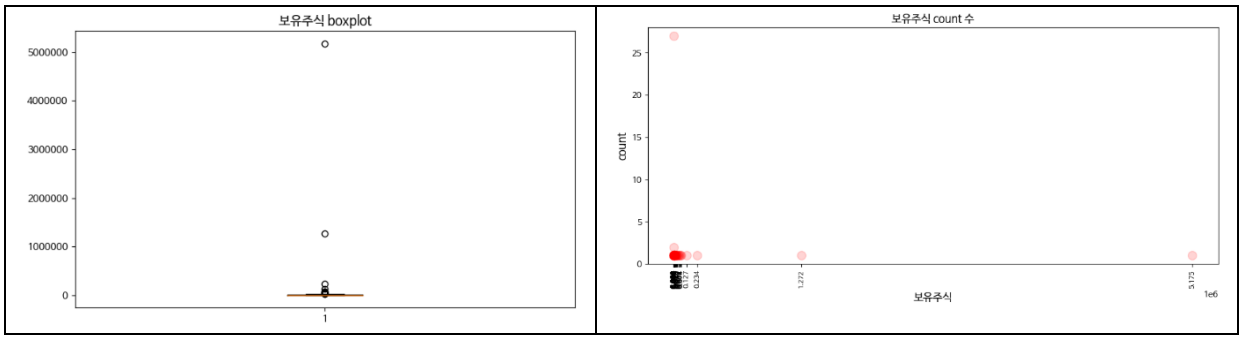
경력 별 임원의 수도 가중치 함수의 기준에 따라 나누어 대신증권자회사 종사, 대신증권 종사, 타증권사 종사, 법·정책 분야에 종사, 학교·재단 등의 분야에 종사, 기타 분야에 종사한 인물들로 나누었다. 단, 여기서 경력은 복수로 종사했던 경우가 포함되므로 총 count수는 총 임원의 수였던 92개를 넘는다. 대신증권 임원들인 만큼 대신증권에 종사한 임원들이 71명으로 가장 많았고, 타증권사가 17명, 법·정책 분야에 14명이 존재하였으며 기타 분야가 6명으로 가장 적었다. 대신증권 임원들이고 경력은 복수로 존재할 수 있어 대신증권 중심으로 경력을 한 번 더 살펴보았다. 대신증권에서만 종사, 자회사에서만 종사, 대신증권과 자회사에서 종사했었던 경우(두개 경력 동시에 갖고 있고 다른 경력은 가지고 있지 않은), 대신증권과 자회사를 제외한 다른 경력을 가지고 있는 경우, 대신증권과 아예 관련이 없는 경우의 5가지 경우로 나누어 살펴보았다. 역시 대신증권에서만 종사했던 임원들이 50명으로 가장 많았으며, 자회사에서만 종사했던 임원은 존재하지 않았다. 반면, 대신증권과 자회사에 종사했던 경우, 대신증권과 다른 경력을 가지고 있는 경우에 비해 대신증권과 관련이 없는 경력만을 가지고 있는 임원도 21명이나 존재하는 것을 확인할 수 있었다. 타증권사에서만 일했던 사람들이나 법조계, 학계에 있던 사외이사들이 이런 경우가 아닐까 추측된다.



<표7> 경력별 임원의 수를 count한 막대그래프 왼쪽은 경력별 임원의 수, 오른쪽은 대신증권 기준으로 본 경력별 임원의 수이다.

2.7. 보유주식

임원별 보유주식에 대해서도 살펴보았는데, boxplot과 각 보유주식별 count 수를 산점도로 나타내 살펴보았다. 아래 boxplot과 같이 주식수의 편차가 매우 심하며 굉장히 많은 주를 가진 소수의 인물들이 이상치와 같이 존재하는 것을 살펴볼 수 있었다. 대부분의 임원들은 소수에 비해 굉장히 적은 주식을 가지고 있었으며, 보유주식수의 최대는 5,175,034주, 최소는 0주이며 평균은 82,078.54주이다. 하지만 산점도에서 살펴볼 수 있듯이 대부분은 0~234,000주 사이에 존재하는 것을 확인할 수 있다.



<표8> 보유주식별 임원의 수를 count한 boxplot과 산점도 왼쪽은 boxplot, 오른쪽은 산점도이다.

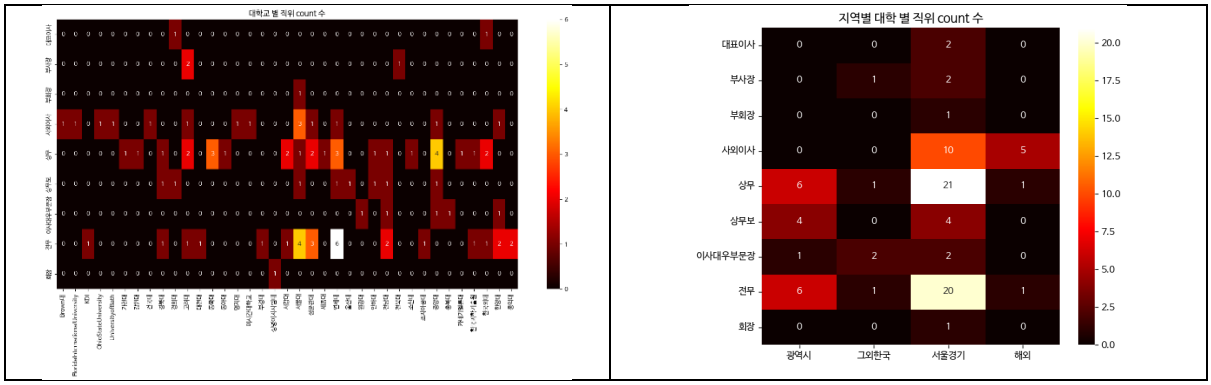
2.8. 피처 간 분석

기본적인 피처 분석 후 피처들 사이에 어떤 영향이 있는지 살펴보고자 피처 간 분석을 진행하였다. 인물의 중요도에 가장 많은 정보를 보여주리라 판단되는 직위, 보유 주식수와 다른 피처들과의 관계를 탐구하였다. 기본적으로 그래프는 각 피처 사이의 임원의 수는 히트맵으로, 보유주식의 경우 데이터의 분포를 산점도로 추가적으로 더 살펴보았다.

* 다른 피처들 사이의 관계는 보고서의 분량 상 생략하였으나, 채택한 7 개의 피처 모두에 대해서 관계에 대한 코드를 구현해 놓았으니 더 자세한 사항은 부록 - 2 코드의 피처분석 부분의 실행결과를 참고하기를 바란다.

2.8.1. 직위별 대학

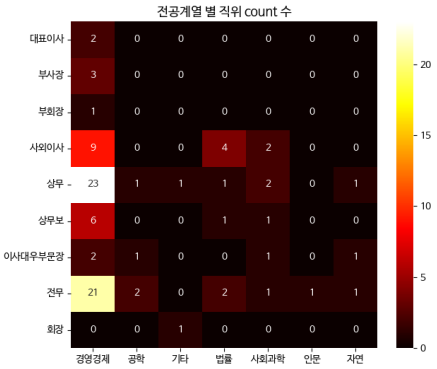
직위별로 대학, 지역별 대학에 대한 임원 수를 살펴보면 전반적으로 상무와 상무보, 사외이사는 골고루 분포해 있으며 사외이사는 서울대가 3명으로 가장 많은 것을 볼 수 잇고, 상무는 중앙대, 연세대, 동국대가 4, 3, 3명으로 그나마 조금 많은 것을 볼 수 있으나 그렇게 큰 차이는 없는 것으로 판단된다. 전무의 경우 연세대에 6명으로 제일 많고 그 다음이 서울대 4명으로 고학력자가 많은 것을 확인할 수 있었으나 전반적으로 고루 분포해 있는 것을 확인할 수 있었다. 그 외에는 직책에 사람이 적어 해당 직책의 인물이 나온 대학정도의 정보만 파악할 수 있었다. 지역별 대학 같은 경우에는 대부분 서울경기권에 많이 몰려 있으며 상무, 상무보, 전무의 경우 광역시의 경우도 조금 존재하는 것을 확인할 수 있었다. 추가로 사외이사의 경우 해외 파도 많은 것을 확인할 수 있었다.



<표9> 직위별 대학의 임원의 수를 count한 히트맵 왼쪽은 대학교별, 오른쪽은 대학지역별 히트맵이다.

2.8.2. 직위별 전공

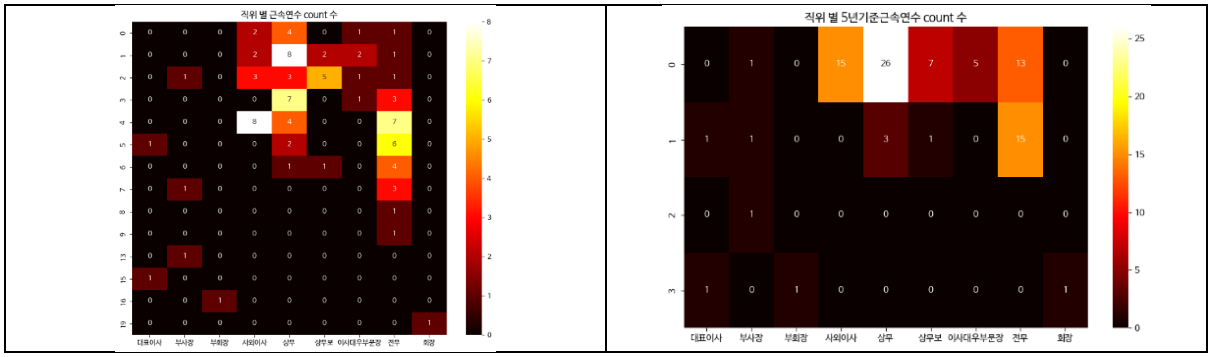
직위별 전공에 대해서는 대부분 경영경제 분야를 전공한 것을 확인할 수 있으며 사외이사의 경우 법률계열을 전공자도 4명정도 있는 것을 확인할 수 있었다. 사외이사, 상무의 경우 사회과학계열도 2명정도 존재하며, 전무의 경우 공학, 법률의 경우도 각각 2명씩 존재하는 것을 확인할 수 있었다.



<표10> 직위별 전공의 임원의 수를 count한 히트맵

2.8.3. 직위별 근속연수

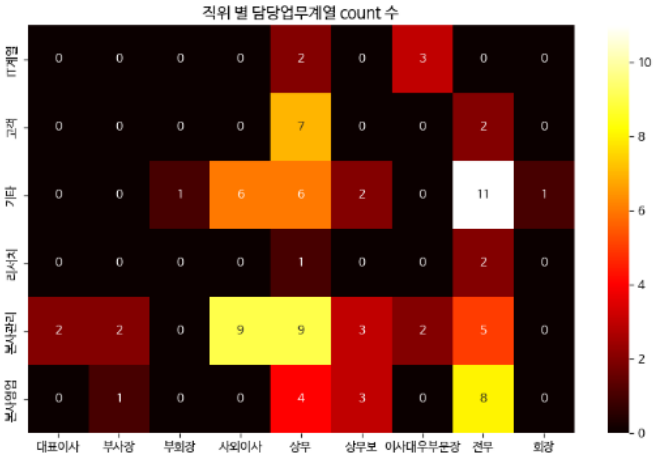
직위별 근속연수에 관해서는 사외이사, 상무, 상무보, 이사대우 부문장의 경우 근속연수가 0~4년 사이에 많이 존재하며 전무의 경우 9년까지 많이 존재하는 것을 확인할 수 있었다. 높은 직급은 인물이 적어 큰 경향성은 파악할 수 없으나 대체적으로 직급이 높을수록 더 오래 일한 것으로 추정되며 사외이사의 경우 사외의 인물이기 때문에 근속연수가 굉장히 짧은 것을 확인할 수 있었다.



<표11> 직위별 근속연수의 임원의 수를 count한 히트맵 왼쪽은 근속연수, 오른쪽은 5년기준근속연수

2.8.4. 직위별 담당업무

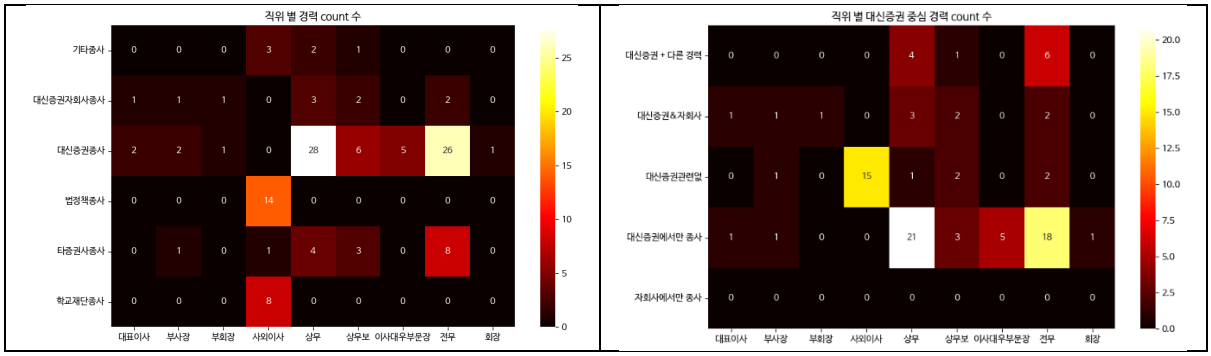
직위별 담당업무는 4.5절에서도 보았듯이 본사관리, 기타에 쏠려 있으며 상무의 경우 본사관리에, 전무의 경우 기타 계열을 제일 많은 인물이 담당하고 있는 것을 확인할 수 있었다.



<표12> 직위별 담당업무의 임원의 수를 count한 히트맵

2.8.5. 직위별 경력

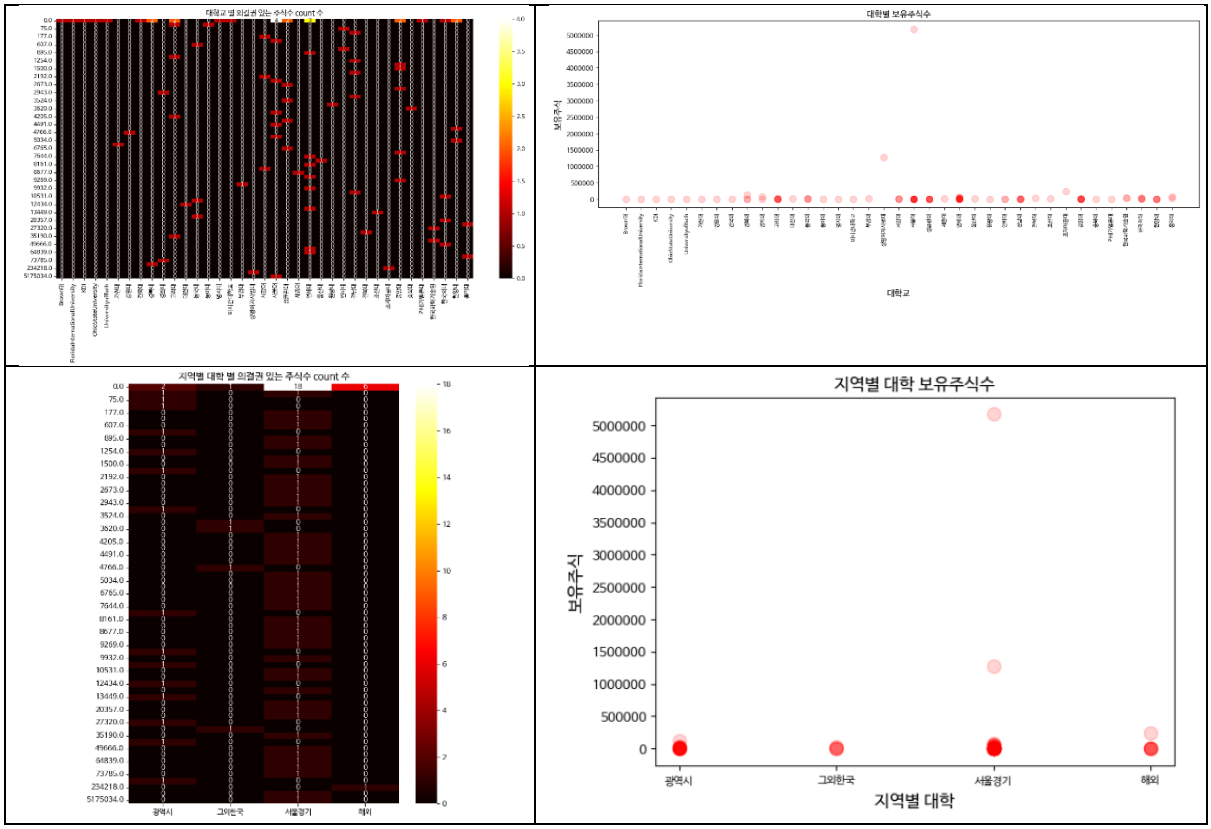
직위별 경력에 관해서는 대부분 대신증권에 종사한 사람들이 많았으며 전무의 경우는 타증권사에 종사한 경우도 8번으로 조금 있는 것을 확인할 수 있었다. 또한, 사외이사의 경우 확실히 법·정책 분야나 학교·재단 분야에 많이 있는 것을 확인할 수 있었으며 대신증권 중심으로 다시 살펴본 경력에서도 마찬가지로 사외이사는 대신증권과 관련이 없는 분야에만 종사했던 것을 확인할 수 있었다. 특히, 전무의 경우 대신증권에서만 종사했던 경우가 가장 많았으며, 회장은 대신증권에서만 종사하였으며, 대표이사와 부회장의 경우 자회사에 같이 있었던 경우도 관찰할 수 있었다.



<표13> 직위별 경력의 임원의 수를 count한 히트맵 왼쪽은 경력별, 오른쪽은 대신증권 중심 경력별

2.8.6. 보유주식별 대학

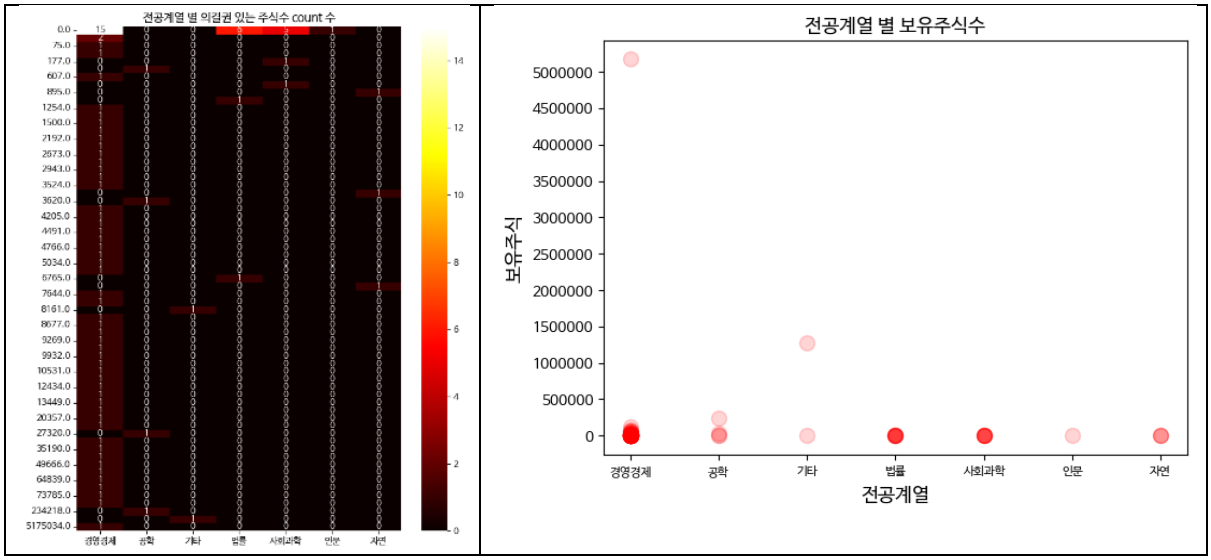
보유주식별 대학을 살펴보면 전반적으로 모든 대학에 대해 보유주식수가 0인 경우가 가장 많았으며 회장, 부회장 등의 경우가 속한 대학에서 많은 보유주식수를 가지고 있는 것을 확인할 수 있었다. 지역별 대학에서도 마찬가지로 뚜렷한 경향성은 찾지 못하였다.



<표14> 보유주식별 대학의 임원의 수를 count한 히트맵과 산점도 왼쪽 위는 대학교별 히트맵, 오른쪽 위는 대학교별 산점도, 왼쪽 아래는 지역별 대학에 대한 히트맵, 오른쪽 아래는 지역별 대학에 대한 산점도이다.

2.8.7. 보유주식별 전공

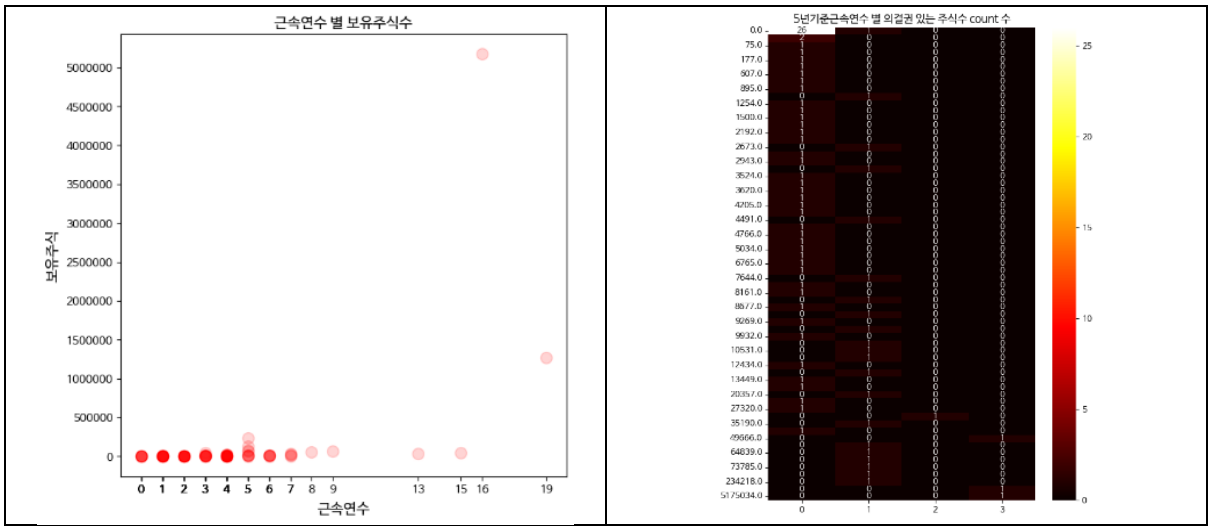
보유주식별 전공도 보유주식수가 0인 경우가 가장 많았으며, 경영경제에 속한 임원들이 압도적으로 많은 만큼 경영경제 전공에서 보유주식수가 고루 분포한 것을 확인할 수 있었다. 추가로, 경영경제 이외에도 공학과 기타 계열에서 소수의 임원들이 비교적 높은 보유주식수를 가지고 있는 것을 확인할 수 있었다.



<표15> 보유주식별 전공의 임원의 수를 count한 히트맵과 산점도 왼쪽은 히트맵, 오른쪽은 산점도이다.

2.8.8. 보유주식별 근속연수

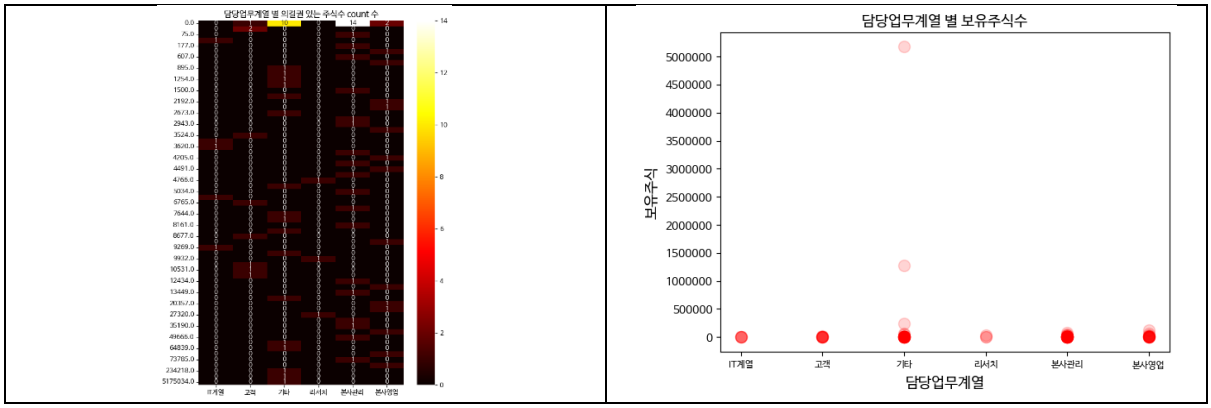
보유주식별 근속연수에 관해서는 높은 근속연수의 경우(10년 이상) 비교적 많은 보유주식을 가지고 있는 것을 확인할 수 있었으며, 0~9년 사이에는 비슷하거나 근속연수가 올라감에 따라 더 많은 주식을 보유하고 있는 것을 확인할 수 있었다. 다만, 근속연수 5년의 경우 특이적으로 0~9년 사이에서 보유주식수들의 분포가 조금 높은 것을 확인할 수 있었다.



<표16> 보유주식별 근속연수의 임원의 수를 count한 히트맵과 산점도 왼쪽은 근속연수별 산점도, 오른쪽은 5년 기준으로 자른 근속연수별 히트맵이다.

2.8.9. 보유주식별 담당업무

보유주식별 담당업무에 대해서는 기타 계열에 있는 임원들이 많은 보유주식수를 가지고 있는 것을 확인할 수 있었다. 본사관리, 본사영업의 경우 많은 임원들이 분포하고 있기에 보유주식수도 고루 분포하고 있으나 비교적 많은 보유주식수를 보유하고 있는 임원들이 소수 존재하고 있는 것을 확인할 수 있었다.

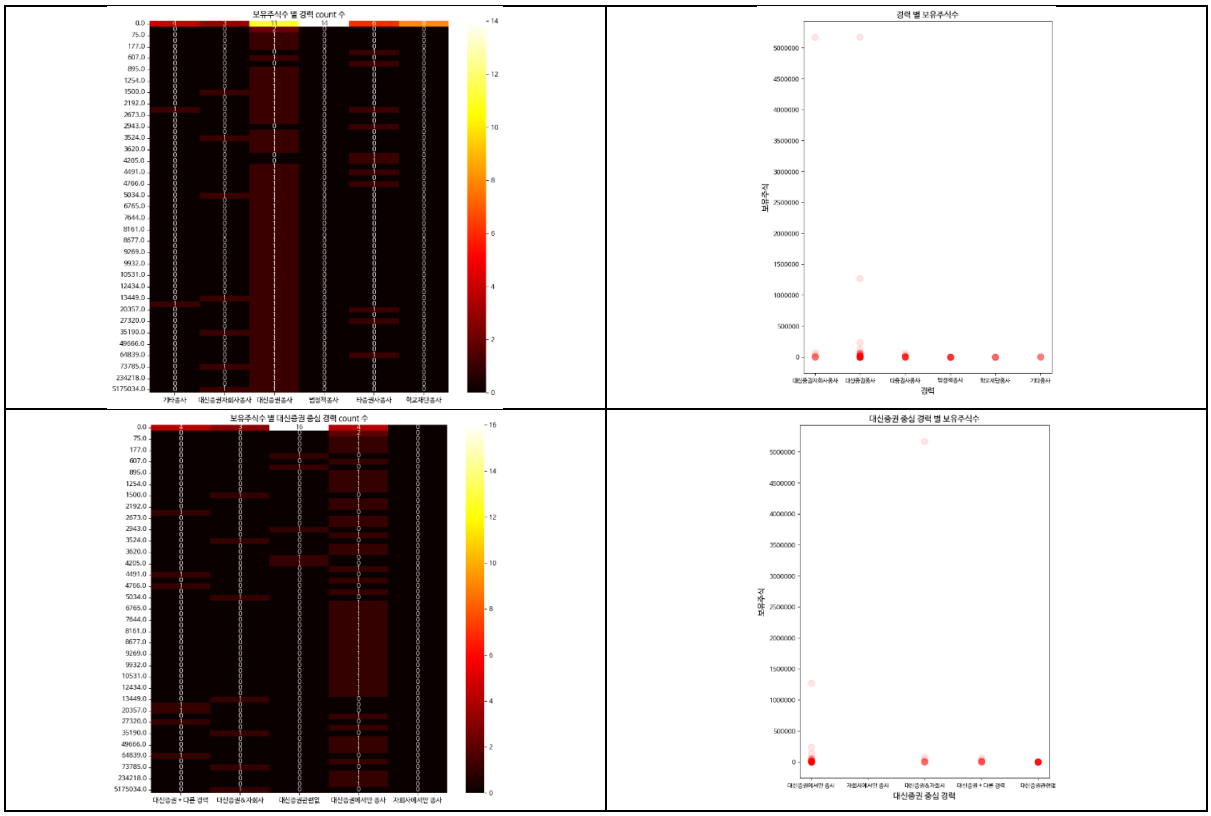


<표17> 보유주식별 담당업무의 임원의 수를 count한 히트맵과 산점도 왼쪽은 히트맵, 오른쪽은 산점도이다.

2.8.10. 보유주식별 경력

먼저 경력별 보유주식에 관해서는 앞서 살펴봤듯이 대신증권에 종사했었던 임원들이 압도적으로 많아 대신증권에 종사했던 경력에 다양한 보유주식수들이 고루 분포한 것을 확인할 수 있었으며, 마찬가지로 많은 보유주식을 가지고 있는 임원들은 대신증권에 종사했었던 것을 알 수 있다. 이를 대신증권을 중심 경력에 대해 다시 살펴보았을 때, 대신증권에서만 종사했던 임원들이 제일

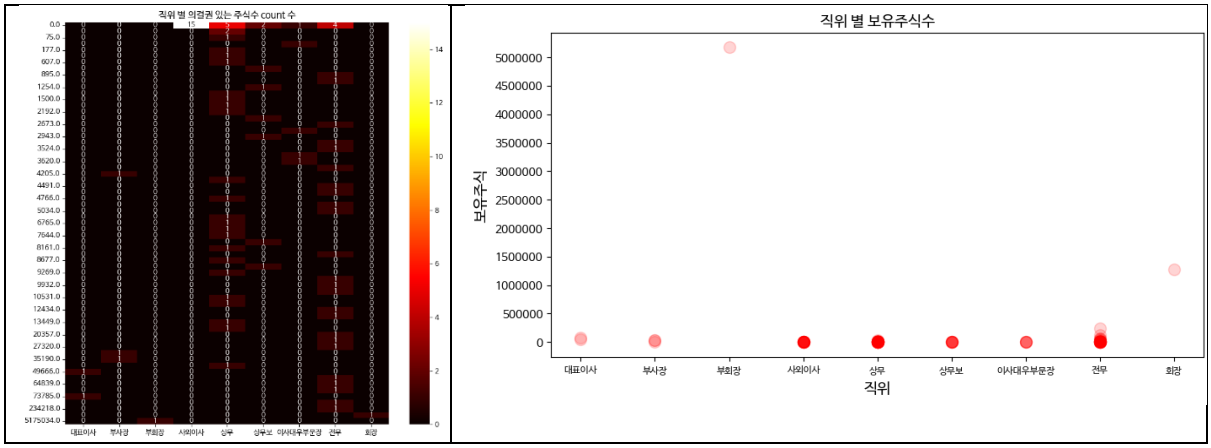
많았던 만큼 전반적으로 고루 분포하고, 많은 보유주식수를 가진 임원들도 다수 존재하는 것을 확인할 수 있었다. 대신증권과 관련 없는 경력을 가지고 있는 임원들에 비해 전반적으로 대신증권과 관련이 있을수록 많은 보유주식수를 가지고 있는 것을 확인할 수 있었다.



<표18> 보유주식수별 경력의 임원의 수를 count한 히트맵과 산점도 왼쪽 위는 경력별 히트맵, 오른쪽 위는 경력별 산점도, 왼쪽 아래는 대신증권 중심 경력별 대학에 대한 히트맵, 오른쪽 아래는 대신증권 중심 경력별 대학에 대한 산점도이다.

2.8.11. 직위별 보유주식

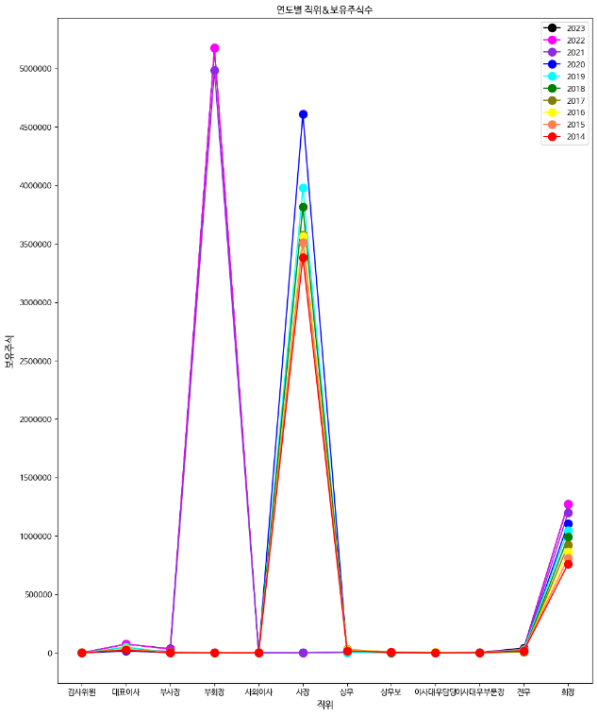
직위별로 보유주식수는 높은 직위일수록 많은 것을 확인할 수 있었으며 상무와 전무가 많은 만큼 이 직위에 대해서는 많은 분포를 가지고 있는 것을 확인할 수 있었다.



<표19> 보유주식별 직위의 임원의 수를 count한 히트맵과 산점도 왼쪽은 히트맵, 오른쪽은 산점도이다.

2.8.12. 연도별 직위별 보유주식

연도별로 직위별 보유주식이 어떻게 변하는지에 대해서도 살펴보았다. 전체적으로 모든 직위에 대해서 해가 지날수록 평균 보유주식량이 늘어나거나 거의 비슷한 것을 확인할 수 있었으며, 부회장의 경우 21년도부터 급격하게 증가한 것을 확인할 수 있고, 사장의 경우 점점 증가하다 20년에 큰 폭으로 증가하였다가 21년도에 굉장히 0으로 감소한 것을 확인할 수 있다. (다만, 이는 데이터 누락의 가능성도 있어 이러한 경향이 있다고 판단할 수 있을 것이다.) 사장과 대표이사의 경우 꾸준히 해를 거듭할수록 평균 보유주식량이 늘어난 것을 확인할 수 있다.



<표20> 연도별로 직위별 평균 보유주식수를 나타낸 꺾은선 그래프

* 연도별 직위별 보유주식수에 대한 각각 임원 count 수를 나타낸 히트맵은 마찬가지로 분량상 부록 - 2 코드의 피처분석 부분의 실행결과를 참조.

3. 코드

```
import pandas as pd
import numpy as np
import re
import math
import requests, json
import pandas as pd

NOTION_TOKEN = "노션 인증키"
```

##데이터 수집

```
headers = {
    "Authorization": "Bearer " + NOTION_TOKEN,
    "Content-Type": "application/json",
```

```
    "Notion-Version": "2022-06-28",
}

DATABASE_ID_LIST = ["db2b026a03f0442f828bc88dcdbf143c", "c238334399d54b10aad0bcb84cb77d5c",
"bf7efa466c624f1ca200d05c5cf34f58", "2f0e230f82974dd09841ae871cb558f9",
"b708c8576ea34bb385d24aedc5b1277b", "d6211c2113914ecc86332d4bb305a115",
"a8b0289c72eb4daaa1063152eb5e787a", "73e5dc61f97f4f2095c0265f2dae70d8",
"1a91276509b246dda4c170a248d096c2", "6a374753947e42f7aa57fac3a474b78f"]
YEAR_LIST = [2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023]

def readDatabase(DATABASE_ID_LIST, YEAR_LIST, headers):
    for id in zip(DATABASE_ID_LIST, YEAR_LIST):
        readUrl = f"https://api.notion.com/v1/databases/{id[0]}/query"
        res = requests.request("POST", readUrl, headers=headers)
        data = res.json()
        print(res.status_code)

        if res.status_code == 200:
            try:
                with open(f"./data/{id[1]}.json", 'w', encoding='utf8') as f:
                    json.dump(data, f, ensure_ascii=False)
            except:
                continue

readDatabase(DATABASE_ID_LIST, YEAR_LIST, headers)
```

##데이터 정제

```
YEAR_LIST = [2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023]


def jsonToDataFrame(YEAR_LIST):
    temp = {}
    year_list = []
    name = []
    birth = []
    registered = []
    fullTime = []
    responsibilities = []
    career = []
    stock = []
    time1 = []
    time2 = []
    position = []

    for year in YEAR_LIST:
        with open(f"./data/{year}.json", 'r') as f:
            json_data = json.load(f)
            for i in range(len(json_data['results'])):
```

```
                properties = json_data['results'][i]['properties']
                year_list.append(year)
                for key in properties:
                    if key == '이름':
                        name.append(properties[key]['title'][0]['text']['content'])
                    elif key == '출생년월':
                        birth.append(properties[key]['rich_text'][0]['text']['content'])
                    elif key == '등기임원 여부':
                        registered.append(properties[key]['rich_text'][0]['text']['content'])
                    elif key == '상근 여부':
                        fullTime.append(properties[key]['rich_text'][0]['text']['content'])
                    elif key == '담당업무':
                        responsibilities.append(properties[key]['rich_text'][0]['text']['content'])
                    elif key == '주요경력':
                        career.append(properties[key]['rich_text'][0]['text']['content'])
                    elif key == '의결권 있는 주식수':
                        stock.append(properties[key]['number'])
                    elif key == '재직기간':
                        time1.append(properties[key]['rich_text'][0]['text']['content'])
                    elif key == '임기만료일':
                        time2.append(properties[key]['rich_text'][0]['text']['content'])
                    elif key == '직위':
                        position.append(properties[key]['rich_text'][0]['text']['content'])

    temp['이름'] = name
    temp['연도'] = year_list
    temp['출생년월'] = birth
    temp['등기임원 여부'] = registered
    temp['상근 여부'] = fullTime
    temp['담당업무'] = responsibilities
    temp['주요경력'] = career
    temp['의결권 있는 주식수'] = stock
    temp['재직기간'] = time1
    temp['임기만료일'] = time2
    temp['직위'] = position
    return temp

dict = jsonToDataFrame(YEAR_LIST)

df = pd.DataFrame.from_dict(data=dict, orient='columns')

df['의결권 있는 주식수'].fillna(0.0, inplace=True)
#'\xa0' 처리
df['주요경력']=df['주요경력'].str.replace('\xa0', '')
#대학, 전공 피처 생성
temp=df['주요경력'].str.split('\n')
colleges=temp.str.get(0)
```

```
#추가로 처리해줘야될 대학 form 맞춰주기
colleges[colleges== 'University of Bath 사회과학대학원(석사)'] ='UniversityofBath 사회과학대학원(석사)'
colleges[colleges== 'Florida International University
경영대학원(박사)']='FloridaInternationalUniversity 경영대학원(박사)'
colleges[colleges== 'Ohio State University 정책대학원(박사)'] = 'OhioStateUniversity 정책대학원(박사)'
colleges[colleges== 'KDI 국제정책대학원(석사)'] = 'KDI 국제정책대학원(석사)'
colleges[colleges== '상명여자사범대'] = '상명여자사범대 교육학'
colleges[colleges== '한국외국어대경영학(석사)'] = '한국외국어대 경영학(석사)'
colleges[colleges== '미시간대학교 경영대학원 MBA'] = '미시간대학교 경영대학원 MBA'

#대학, 전공으로 나누기
ttemp = colleges.str.split()

#대학 피처 생성
df['대학교']=ttemp.str.get(0)

#전공 피처 생성
tttemp = ttemp.str.get(1)
tttemp=tttemp.replace(r'\([^)]*\)', '', regex=True)
df['전공']=tttemp

#근속연수 생성
def create_work_years(x):
  temp = re.split(r'~|~\n', x)
  temp[0] = temp[0].strip()
  temp[1] = temp[1].strip()
  return int(temp[1][:4]) - int(temp[0][:4])

df['근속연수'] = df['재직기간'].apply(lambda x : create_work_years(x))

#주요경력 전처리
def replace_career(x):
  temp = [string.split() for string in re.split(r'\n |\n', x)]
  if len(temp)==1 :
    return {'대신증권'}
  else:
    return set(list(zip(*temp[1:]))[0])

df['주요경력요약'] = df['주요경력'].apply(lambda x : replace_career(x))

df_2014 = df[df['연도']==2014].reset_index(drop=True)
df_2015 = df[df['연도']==2015].reset_index(drop=True)
df_2016 = df[df['연도']==2016].reset_index(drop=True)
df_2017 = df[df['연도']==2017].reset_index(drop=True)
df_2018 = df[df['연도']==2018].reset_index(drop=True)
df_2019 = df[df['연도']==2019].reset_index(drop=True)
df_2020 = df[df['연도']==2020].reset_index(drop=True)
df_2021 = df[df['연도']==2021].reset_index(drop=True)
```

```
df_2022 = df[df['연도']==2022].reset_index(drop=True)
df_2023 = df[df['연도']==2023].reset_index(drop=True)
```

##가중치(유사도) 계산

```
#학교 거리 계산
def cal_college(df1, df2, alpha = 0.7):
  #지역별로 나눔
  college_dict = {'서울경기':['상명여자사범대', '서울대', '홍익대', '성균관대', '연세대', '연세대학교',
'동국대', '중앙대', '한국외대', '한국외국어대', '세종대학교', '세종대', '성균관대학교', '경희대',
'서강대', '고려대', '건국대', '가천대', '가천대학교', '한양대', '명지대'],
                  '광역시':['부경대', '경북대', '전남대', '조선대', '한국과학기술원', '가천대',
'인하대', '동아대', '대전대', '울산대', '영남공업전문대'],
                  '그외한국':['전북대', '원광대', '충북대', '충북대학교', '강원대', 'KDI'],
                  '해외':['FloridaInternationalUniversity', 'OhioStateUniversity',
'UniversityofBath', '조지아공대', 'Brown 대', '카네기멜론대', '미시간대학교']}

  #키값 찾기
  for key, val in college_dict.items() :
    if df1['대학교'] in val :
      df1_key = key
    if df2['대학교'] in val :
      df2_key = key

  #k 값 구하기
  if df1['대학교'] == df2['대학교'] :
    k=0
  elif df1_key == df2_key :
    k=1
  elif df1_key != df2_key :
    k=2

  return alpha ** k

#학과 거리 계산
def cal_department(df1, df2, alpha = 0.7):
  department_dict={'경영경제':['회계학', '세무학', '경제학', '국제경영학', '경영학', '무역학', 'MBA',
'경제통상학', '경영대학원', '경영대학원 MBA', '산업정보학', '금융공학', '국제금융학'],
                  '사회과학':['사회학', '사회과학대학원', '행정학', '정치외교학', '정치학',
'신문방송학'],
                  '법률':['법학', '사법학', '정책대학원', '국제정책대학원'],
                  '인문':['불어불문학'],
                  '기타':['교육학', '인적자원개발학'],
                  '자연':['응용통계학', '통계학', '수학'],
                  '공학':['기계공학', '화학공학', '전자계산학', '전자전산학', '전산학', '항공우주학']}

  #키값 찾기
  for key, val in department_dict.items() :
    if df1['전공'] in val :
```

```
      df1_key = key
    if df2['전공'] in val :
      df2_key = key

  #k 값 구하기
  if df1['전공'] == df2['전공'] :
    k=0
  elif df1_key == df2_key :
    k=1
  elif df1_key != df2_key :
    k=2

  return alpha ** k

#직위 거리 계산
def cal_position(df1, df2, alpha = 0.7):
  position_dict={'회장':0, '대표이사':1, '부회장':2, '사장':3, '부사장':4, '전무':5, '상무': 6,
'상무보':7, '이사대우부문장':8, '이사대우담당':8, '사외이사':99, '감사위원':99}

  #value 값 찾기
  df1_value = position_dict[df1['직위']]
  df2_value = position_dict[df2['직위']]

  #k 값 구하기
  if df1['직위'] == df2['직위'] :
    k=0
  elif df1_value == 99 or df2_value == 99 : #사외이사 혹은 감사인원(회사밖 인물)은 모두와 거리
4(최대차이의 중간값// 회장, 이사대우담당과의 거리)
    k=4
  else :
    k=abs(df1_value - df2_value) #값 차이만큼 거리

  return alpha ** k

# 근속연수 거리 계산
def cal_work_years(df1, df2, alpha = 0.7):
  #k 값 구하기
  if df1['근속연수'] == df2['근속연수'] :
    k=0
  else :
    if abs(df1['근속연수'] - df2['근속연수']) % 5 ==0:    #근속연수 차이 5 년 기준으로 1 씩 증가 (차이
1~5/6~10/11~15/...같은그룹)
      k= abs(df1['근속연수'] - df2['근속연수']) // 5
    else:
      k= 1 + abs(df1['근속연수'] - df2['근속연수']) // 5

  return alpha ** k
```

```
#당당업무 거리 계산
def cal_task(df1, df2, alpha = 0.7):
  task_dict={'IT 계열': {'IT':['IT 부문장', 'IT 본부장', 'IT 서비스본부장', '디지털부문장',
'스마트 Biz 본부장', 'Operation&Technology 본부장'], '정보보호':['정보보호부문장', '정보보호담당']},
            '본사영업':{'Wolesale':['Wholesale 부문장','Wholesale 영업본부장', 'Wholesale 사업단장',
'Trading 부문장', 'Trading Center 장', 'Sales&Trading 총괄', '대외협력담당'],
                    'IB':['IB 사업단장', 'IB 부문장', 'IB 부부문장', 'IB1 부문장', 'IB2 부문장',
'IPO 담당',  'PF 부문장', 'PF1 본부장', '구조화상품본부장'],
                    '영업':['영업부장']},
            '본사관리':{'리테일':['리테일총괄', 'Club1962 센터장','WM 추진부문장', 'WM 추진본부장',
'WM 사업단장'],
                    '리스크':['리스크관리부문장', '리스크관리담당'] ,
                    '경영':['경영전략총괄', '전략지원부문장', '전략지원담당', '경영지원본부장',
'경영지원부문장', '경영기획부문장', '경영기획본부장', '기획본부장', '기획본부장 / 투자금융담당',
'전략지원부문장/프라이빗라운지 부문장'],
                    '상품':['Product 부문장', 'Solution&Product 사업단장'],
                    '업무':['업무총괄',  '업무총괄(IB 사업단, 고객자산본부 제외)',
'업무총괄(IB 사업단,고객자산본부 제외)', '업무총괄 (IB 사업단,고객자산본부 제외)', '업무총괄(IB 사업단,
고객자산본부, 경영지원본부, 정보보호부문, 준법지원부문, 감사부문 제외)'],
                    '금융':['기업금융담당', '스마트금융본부장', '금융주치의추진본부장',
'금융주치의사업단장'],
                    '인사,인프라':['인재역량센터장', '인프라관리본부장'],
                    '법률':['준법감시인\n 준법지원부문장', '준법감시인 준법지원부문장',
'준법감시인\n 준법지원부문장', '준법감시인\n 준법지원담당', '준법감시인/준법지원부문장', '준법감시인 /
준법지원담당', 'Advisory 본부장'],
                    '감사':['감사위원장', '감사부문장', '감사위원', '상근감사위원', '감사담당'],
                    '언론홍보':['홍보부문장', 'Coverage 본부장']},
            '리서치': {'리서치센터':['Research Center 장', '리서치센터장', 'Market\n  Solution 부문장',
'Research&Strategy 본부장', 'Market Solution 부문장', 'Market Solution 부문장']},
            '고객':{'고객자산':['고객자산부문장', '고객자산부문 부부문장', '고객자산본부장',
'고객자산본부장 / 홍보담당', '고객자산본부장/ 홍보담당', '고객자산본부장/ 홍보부문장',
'고객자산본부장/홍보부문장', '고객자산부문장, 홍보부문장', '고객자산부문장\n 홍보부문장'],
                  'WM':['서부 WM 부문장', '서부 WM 본부장', '재경 1WM 부문장', '재경 1WM 본부장',
'재경 2WM 본부장', '재경 2WM 부문장', '동부 WM 부문장', '동부 WM 본부장'],
                  '프라이빗':['프라이빗부문장,\n 나인원프라이빗라운지장', '프라이빗부문장,
나인원프라이빗라운지장', '대신나인원 \n 프라이빗라운지장', '프라이빗라운지 부문장 /대신나인원
프라이빗라운지장', '프라이빗부문장, 대신나인원프라이빗라운지장', '프라이빗부문장\n 나인원프라이빗라운지장',],
                  '소비자보호':['금융소비자보호부문장\n(COO)', '금융소비자보호부문장(COO)',
'금융소비자보호부문장 (COO)', '금융소비자보호 총괄', '금융소비자보호총괄']},
            '기타':{'지점':['강북지역본부장', '강남지역본부장', '강남선릉센터장', '서부지역본부장',
'동부지역본부장', '울산지점장'],
                  '비서':['비서실장', '비서/브랜드본부', '비서/브랜드담당'], '공란':['-']}}

   #키값 찾기
  for outerKey in task_dict.keys() :
    for key, val in task_dict[outerKey].items() :
      if df1['당당업무'] in val :
        df1_outerKey = outerKey
```

```
        df1_innerKey = key
      if df2['담당업무'] in val :
        df2_outerKey = outerKey
        df2_innerKey = key


  #k 값 구하기
  if df1['담당업무'] == df2['담당업무'] : #같은업무
    k=0
  elif df1_outerKey == df2_outerKey and df1_innerKey == df2_innerKey: #상위하위 혹은 비슷한업무
    k=1
  elif df1_outerKey == df2_outerKey : #같은계열업무
    k=2
  elif df1_outerKey != df2_outerKey :
    k=3 #다른계열업무
  return alpha ** k

#경력 거리 계산
def cal_career(df1, df2, alpha = 0.7):
  career_dict={1:['대신증권', '대신증권 WM 사업단장', 'WM 추진부문장', '재경 1WM 부문장',
'대신증권중부지역본부장', '기획본부장대신증권', '대신증권분당지점장', '대신증권무거동지점장',
'대신증권 Capital',
                '이사대우부본부장', '기획본부장대신증권인재역량센터장', '대신증권인재전략부',
'대신증권파생상품운용부장', '대신증권자산운용본부', '대신증권영업기획부장', '대신증권기획실장',
                '대신증권홍보실', '대신증권브랜드전략실', '대신증권 IT 개발부장',
'대신증권트레이딩시스템부장', '대신증권 Global 사업본부', '신증권', '기업금융사업단장'], #대신증권
              0:['대신에프앤아이', '대신투자신탁운용', '대신투자신탁운용상무이사', '대신에이엠씨',
'대신자산운용', '대신저축은행'], #대신증권 자회사
              2:['메리츠종금증권', '메리츠증권', '미래에셋증권', 'KTB 투자증권', 'SC 제일은행',
'하나금융투자', 'IBK 투자증권', 'DB 금융투자', 'NH 투자증권', 'KB 증권', 'LIG 손해보험', '굿모닝신한증권',
'메릴린치증권',
                '삼성증권', '동양증권', '한국투자증권', '한국투자증권평촌지점', '대우증권',
'대우본부장', '대우증권전략기획본부', '대우증권해외사업부문', '대우증권 IB 사업부문',
                '우리투자증권', '우리 CS 자산운용', '우리투자증권기업금융 2 팀장', '우리프라이빗에쿼티',
'푸르덴셜투자증권', '하이투자증권주식인수팀',  '하이투자증권주식인수팀', 'SBC', 'Bank', '도이치은행'],
#타증권사
              3:['금융감독원', '금융위원회', '現)금융위원회', '감사원', '대검찰청', '법무법인',
'現)법무법인', '세무법인', '국세청', '중부지방국세청', '서울북부지방검찰청','서울지방국세청장',
'서울지방국세청',
                '제 58 대', '제 22 대', '기획재정부', '국세청,관세청,산업통상자원부', '자본시장연구원',
'한국조세연구원', '국가청렴위원회', '대한중재인협회', '한국회계정보학회',
                '금융위원회금융발전심의위원(현)', '금융위원회적극행정심의위원(現)',
'안진회계법인',  'L&C 세무회계사무소', '변호사정상명법률사무소', '피앤비세무컨설팅', '現)법무법인(유)'],
#법,정책
              4:['중앙대', '중앙대학교', '現)중앙대', '연세대학교', 'KAIST', '수원대', '수원대학교',
'서울시립대', '現)서울시립대', '학교법인', '現)학교법인', '재단법인', '現)재단법인'], #학교, 교수, 재단
              5:['하비스트', '한국물산', '우리선물', '부영주택', 'LG 경제연구원', 'Arthur'] #기타
              }
```

```
  #경력 여부 빈리스트 생성
  df1_list = [0,0,0,0,0,0] #0 번인덱스: 대신증권, 1 번인덱스: 대신증권 자회사, 2 번인덱스: 타증권사,
3 번인덱스: 기타
  df2_list = [0,0,0,0,0,0]

  #리스트에 각 업종에 종사했었으면 1, 아니면 0
  for career in df1['주요경력요약'] :
    for key, val in career_dict.items() :
      if career in val :
        df1_list[key]=1

  for career in df2['주요경력요약'] :
    for key, val in career_dict.items() :
      if career in val :
        df2_list[key]=1

  #df1_list 와 df2_list 합 구하기(요소가 0-> 둘다 종사 안했음, 1-> 한쪽만 종사함, 2-> 둘다 종사함)
  sum_list = [x + y for x, y in zip(df1_list, df2_list)]

  #k 값 구하기
  if df1_list == df2_list : #전체 같으면 0
    k=0

  elif (sum_list[0]==0 and sum_list[1]==0 and sum_list[2]==0): #기타만 있을 때 (법률, 학교, 기타)
    count2 = sum_list.count(2) #2 인 요소 개수 찾기(df1, df2 같은거 개수)
    k = 0.6 * (1 + count2)

  else : #기타만 있는거 아니고,
    #법률, 학교, 기타 전부 기타로 통합. 범주는 3 으로.
    if sum(df1_list[3:]) > 0 :
      df1_list[3] = 1
      df1_list[4] = 0
      df1_list[5] = 0
    if sum(df2_list[3:]) > 0 :
      df2_list[3] = 1
      df2_list[4] = 0
      df2_list[5] = 0

    #수정한 df1_list, df2_list 로 sum_list 다시 생성
    sum_list = [x + y for x, y in zip(df1_list, df2_list)]

    #df1_list, df2_list 에서 1 인 인덱스 뽑기
    index1 = [i for i in range(len(df1_list)) if df1_list[i] == 1]
    index2 = [i for i in range(len(df2_list)) if df2_list[i] == 1]

    min = 10
    max = 0
```

```
    for i in index1 :
      for j in index2 :
        if i != j: #두 사람 간에, 다른 업종끼리 최대거리 최소 거리 구하기

          if (i==3 and j != 3) or (j==3 and i != 3) : #기타와 다른 업종
            max = 4 #기타는 모두와 거리 4 고정 (최대거리 구하기)
            temp = 4
          else : #최대거리 구하기
            temp = abs(i-j)
            if temp > max:
              max = temp
          if temp < min: #최소거리 구하기
            min = temp

    k=float((max+min)/2) #거리는 최대거리, 최소거리의 평균
    k= k- (0.5 * sum_list.count(2)) # 같은 업종 있으면, 있는 만큼 거리 -0.5

  return alpha ** k

#보유 주식수별 거리 계산
def cal_stock(df1, df2, alpha = 0.7):
  #k 값 구하기
  if df1['의결권 있는 주식수'] == df2['의결권 있는 주식수'] :
    k=0
  else :
    k= abs(df1['의결권 있는 주식수'] - df2['의결권 있는 주식수'])//100 #차이로 보기=> 차이가 거리가
된(너무 차이 크면 거리 멀어짐), 100 개 단위로 끊어서 보기
    k=math.log10(1+k) #로그스케일
  return alpha ** k

#가중치 함수
def weight_sum(df1, df2):
  wt_college = cal_college(df1, df2)
  wt_department = cal_department(df1, df2)
  wt_position = cal_position(df1, df2)
  wt_task = cal_task(df1, df2)
  wt_work_years = cal_work_years(df1, df2)
  wt_career = cal_career(df1, df2)
  wt_stock = cal_stock(df1, df2)
  return wt_college + wt_department + wt_position + wt_task + wt_work_years + wt_career + wt_stock

# 인물 가중치행렬
col = []
row = []
matrixs = []
dataFrames = [df_2014, df_2015, df_2016, df_2017, df_2018, df_2019, df_2020, df_2021, df_2022,
df_2023]
```

```
for dataframe in dataFrames:
  for i in range(len(dataframe)):
      col.append(dataframe.loc[i, '이름'])
      row.append(dataframe.loc[i, '이름'])
  weight_matrix = pd.DataFrame(columns=col, index=row)
  matrixs.append(weight_matrix)
  col=[]
  row=[]

#가중치 계산
for t, dataframe in enumerate(dataFrames):
  for i in range(0, len(dataframe)):
    for j in range(0, len(dataframe)):
      matrixs[t].iloc[i, j] = round(weight_sum(dataframe.iloc[i], dataframe.iloc[j])**2, 2)
```

##관심 인물 선정

```
# 대신증권 연간 재무 정보
daishin_Annual_Financial_Info = pd.read_csv('data/daishin_Annual_Financial_Info.csv', sep=",",
encoding='UTF-8')
profit_avg = daishin_Annual_Financial_Info['연도별 누적 순이익 (단위:천원)'].mean()

def find_promoted_employees(df, daishin_Annual_Financial_Info):
    increased_years = []  # 매출이 증가한 연도를 담을 리스트
    promoted_employees = []  # 승진 또는 입사한 인물을 담을 리스트

    # 매출액이 증가한 연도 찾기
    for index, row in daishin_Annual_Financial_Info.iterrows():
        if row['연도별 누적 순이익 (단위:천원)'] > profit_avg:
            increased_years.append(row['연도'])

    for year in increased_years:
        for index, row in df.iterrows():
            if row['연도'] == year:
                previous_year = year - 1

                previous_row = df.loc[df['연도'] == previous_year]
                start_year = int(row['재직기간'].split('.')[0])

                # 입사 여부 확인
                if start_year == year:
                    promoted_employees.append(row['이름'])

                # 직위 변화 확인
                elif len(previous_row) > 0 and row['직위'] != previous_row['직위'].values[0]:
                    promoted_employees.append(row['이름'])

    return promoted_employees
```

```
promoted_employees = find_promoted_employees(df, daishin_Annual_Financial_Info)

def find_increased_stockholders(df, daishin_Annual_Financial_Info):
    increased_years = []  # 매출이 증가한 연도를 담을 리스트
    increased_stockholders = []  # 주식보유량이 증가한 인물을 담을 리스트

    # 매출액이 증가한 연도 찾기
    for index, row in daishin_Annual_Financial_Info.iterrows():
        if row['연도별 누적 순이익 (단위:천원)'] > profit_avg:
            increased_years.append(row['연도'])

    for year in increased_years:
        for index, row in df.iterrows():
            if row['연도'] == year:
                previous_year = year - 1

                # 주식보유량 변화 확인
                previous_row = df.loc[df['연도'] == previous_year]
                if len(previous_row) > 0 and row['의결권 있는 주식수'] > previous_row['의결권 있는
주식수'].values[0]:
                    increased_stockholders.append(row['이름'])

    return increased_stockholders

increased_stockholders = find_increased_stockholders(df, daishin_Annual_Financial_Info)

important_person = promoted_employees + increased_stockholders

from collections import Counter

counter = Counter(important_person)

# 등장 횟수가 많은 순서대로 정렬된 튜플 리스트 생성
sorted_elements = sorted(counter.items(), key=lambda x: x[1], reverse=True)

# 정렬된 리스트에서 요소와 등장 횟수 출력
for element, count in sorted_elements:
    print(f'{element}: {count}')

def select_important_people(sorted_elements, threshold=2):   # threshold 이상 등장한 인물을 리스트로
반환
    import_p = []
    for element, count in sorted_elements:
        if count >= threshold:
            import_p.append(element)
    return import_p
```

```
major_p = select_important_people(sorted_elements, 2)
print(major_p)
```

###네트워크 시각화

```
import networkx as nx
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

#중심성 정규화 함수
def normalize_centrality(centrality_dict):        # 중심성들 0 ~ 1 값으로 정규화
    max_value = max(centrality_dict.values())
    min_value = min(centrality_dict.values())
    normalized_centrality = {node: (value - min_value) / (max_value - min_value) for node, value in
centrality_dict.items()}
    return normalized_centrality


def print_network(matrixs):
    #연도별로 중심값 저장할 dic
    result_dict_all_year ={}
    year = 2013
    for k in range(len(matrixs)):
        year += 1
        node = matrixs[k].columns.tolist()
        # 특정 임계값보다 작은 가중치를 가진 간선은 표시하지 않음 (centrality 계산을 위해 다시 아래 코드를
삽입한것임)
        m_w_sum = []
        for i in range(len(matrixs[k])):
            for j in range(len(matrixs[k])):
                m_w_sum.append(matrixs[k].iloc[i, j])

        threshold = np.percentile(m_w_sum, 75)
        node = matrixs[k].columns.tolist()
        # 그래프에 노드와 간선 삽입
        G = nx.Graph()
        G.add_nodes_from(node)
        for i in range(len(node)):
            for j in range(i + 1, len(node)):
                weight = matrixs[k].iloc[i, j]
                if weight >= threshold:  # 특정 임계값 이상인 경우에만 add_edge
                    G.add_edge(node[i], node[j], weight=matrixs[k].iloc[i,j])
        plt.figure(figsize=(12, 20))
        plt.rcParams['font.family'] = 'AppleGothic'
        pos = nx.spring_layout(G, k=3.0)  # 레이아웃 설정
        node_colors = ['red' if n in major_p else 'lightblue' for n in G.nodes()]  # major_p 에
해당하는 노드는 빨간색, 나머지는 파란색으로 설정
        weights = [G[u][v]['weight'] for u, v in G.edges()]  # 엣지의 가중치 리스트
```

```
        degree_centrality = nx.degree_centrality(G)
        node_sizes = [250 if degree_centrality[n] * 10000 <= 50 else degree_centrality[n] * 15000
for n in G.nodes()]   # degree centrality 에 따라 노드 크기 조정

        nx.draw_networkx(G, pos, with_labels=True, node_color=node_colors, node_size=node_sizes,
edge_color='gray', width=1.0, font_family='AppleGothic')

        # 가중치에 따른 엣지 라벨 추가
        # labels = nx.get_edge_attributes(G, 'weight')
        # nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)              가중치를 표시하려면
주석 처리 x



        # Create legend handles and labels
        legend_handles = [
            mpatches.Patch(color='red', label='중요인물'),
            mpatches.Patch(color='lightblue', label='일반인물')
        ]
        plt.legend(handles=legend_handles)
        plt.title('{0}년도 네트워크'.format(2014+k), color='blue', fontsize=16)
        plt.show()

        # 각 인물들이 중요 인물과 얼마나 연결되어있는지 출력
        sorted_nodes = sorted(G.nodes(), key=lambda n: sum(1 for neighbor in G.neighbors(n) if
neighbor in major_p), reverse=True)

        # 노드와 중요인물과의 연결 개수를 포함하는 딕셔너리 생성
        node_connections = {n: sum(1 for neighbor in G.neighbors(n) if neighbor in major_p) for n in
sorted_nodes}

        # 딕셔너리 출력
        # for node, connections in node_connections.items():
        #     print(f"{node}은 중요인물과 {connections}개 연결됨")

        # 중심성 계산
        degree_centrality = nx.degree_centrality(G)
        betweenness_centrality = nx.betweenness_centrality(G)
        closeness_centrality = nx.closeness_centrality(G)
        eigenvector_centrality = nx.eigenvector_centrality(G)

        #정규화
        normalized_node_connections = normalize_centrality(node_connections)
        normalized_degree_centrality = normalize_centrality(degree_centrality)
        normalized_betweenness_centrality = normalize_centrality(betweenness_centrality)
        normalized_closeness_centrality = normalize_centrality(closeness_centrality)
        normalized_eigenvector_centrality = normalize_centrality(eigenvector_centrality)
```

```
        # 딕셔너리 키 순서를 정렬하기 위한 공통 키 리스트 생성
        common_keys = sorted(node_connections.keys())

        result_dict = {}
        for node in common_keys:
            result_dict[node] = [normalized_node_connections[node],
normalized_degree_centrality[node], normalized_betweenness_centrality[node],
normalized_closeness_centrality[node], normalized_eigenvector_centrality[node]]


        #해당 연도의 중심성 값 저장
        result_dict_all_year[year] = result_dict

        # 결과 출력
        print("{0}년도 네트워크 점수 계산".format(2014+k))
        print()
        print("중요인물 연결 수")
        for node, centrality in sorted(node_connections.items(), key=lambda x: x[1], reverse=True):
            print(node, centrality)
        print()
        print("Degree Centrality:")
        for node, centrality in sorted(degree_centrality.items(), key=lambda x: x[1], reverse=True):
            print(node, centrality)
        print()
        print("Betweenness Centrality:")
        for node, centrality in sorted(betweenness_centrality.items(), key=lambda x: x[1],
reverse=True):
            print(node, centrality)
        print()
        print("Closeness Centrality:")
        for node, centrality in sorted(closeness_centrality.items(), key=lambda x: x[1],
reverse=True):
            print(node, centrality)
        print()
        print("Eigenvector Centrality:")
        for node, centrality in sorted(eigenvector_centrality.items(), key=lambda x: x[1],
reverse=True):
            print(node, centrality)
        print()

    return result_dict_all_year

result_dict_all_year = print_network(matrixs)
```

###가중합 선정

```
#weights 값 임의로 설정
def weighted_average_values(result_dict, weights = [1.5, 2.5, 3.5, 1, 1.5], rank = 5):
    # 각 키에 해당하는 값들을 가중 평균하여 하나의 값으로 계산
```

```
    weighted_values = []
    for values in result_dict.values():
        weighted_value = sum(value * weight for value, weight in zip(values, weights))
        weighted_values.append(weighted_value)

    weighted_dict = {}
    keys = list(result_dict.keys())
    for i, key in enumerate(keys):
        weighted_dict[key] = weighted_values[i]

    sorted_items = sorted(weighted_dict.items(), key=lambda x: x[1], reverse=True)
    top_values = sorted_items[:rank]
    top_dict = {key: value for key, value in top_values}

    bottom_values = sorted_items[-rank:]
    bottom_dict = {key: value for key, value in bottom_values}

    for key in top_dict:
        top_dict[key] *= 100

    for key in bottom_dict:
        bottom_dict[key] *= 100

    return top_dict, bottom_dict

def cal_weight(weights):
    top = {}
    bottom = {}
    for year in YEAR_LIST:
        # 속성별 가중치 값
        top_dict, bottom_dict = weighted_average_values(result_dict_all_year[year], weights)
        top[year] = top_dict
        bottom[year] = bottom_dict

    return top, bottom

top, bottom = cal_weight([1.5, 2.5, 3.5, 1, 1.5])
```

###중요인물 검증

```
top

bottom

def verification(top):
    IP = {}
    num = 0

    for year in top.keys():
```

```
        dict = {}
        for name in top[year]:

            temp = df.loc[df['이름'] == name]
            promoted_this_year = False
            promoted_next_year = False
            last_this_IP_select = False
            verify = False

            now_position = temp.loc[temp['연도']==year,'직위'].values[0]

            #이전년도 데이터 존재
            if(year-1 in temp['연도'].values):
                last_position =temp.loc[temp['연도']==year-1,'직위'].values[0]
                if(name in list(IP[year-1].keys())):
                    last_this_IP_select = True
                if(last_position != now_position):
                    promoted_this_year = True

            #다음년도 데이터 존재
            if(year+1 in temp['연도'].values):
                next_position = temp.loc[temp['연도']==year+1,'직위'].values[0]
                if(next_position != now_position):
                    promoted_next_year = True

            if(promoted_next_year | promoted_this_year | last_this_IP_select):
                verify = True
                num += 1

            dict[name] = verify

        IP[year] = dict

    score = num / (len(top) * 5)
    return IP, score

IP, score = verification(top)

IP

score
```

###최적화

```
# 1
import itertools

YEAR_LIST = [2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023]
```

```
def grid_search(grid_weights):
    best_weights = None
    best_score = float('-inf')

    # 모든 가중치 조합 탐색
    for weights in itertools.product(*grid_weights):
        top, bottom = cal_weight(weights)

        IP, score = verification(top)

        if score > best_score:
            best_weights = weights
            best_score = score

    return best_weights


grid_weights = [[0.1, 0.5, 1, 1.5, 2, 2.5, 3.0], [0.1, 0.5, 1, 1.5, 2, 2.5, 3.0], [0.1, 0.5, 1, 1.5,
2, 2.5, 3.0], [0.1, 0.5, 1, 1.5, 2, 2.5, 3.0], [0.1, 0.5, 1, 1.5, 2, 2.5, 3.0,]]
best_weights = grid_search(grid_weights)

best_weights

top, bottom = cal_weight(best_weights)

IP, score = verification(top)

IP

score

# 2
YEAR_LIST = [2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023]

grid_weights = [[0.1, 0.3, 0.5, 0.7, 0.9], [0.1, 0.3, 0.5, 0.7, 0.9], [0.1, 0.3, 0.5, 0.7, 0.9],
[0.1, 0.3, 0.5, 0.7, 0.9], [0.1, 0.3, 0.5, 0.7, 0.9]]
best_weights = grid_search(grid_weights)

best_weights

top, bottom = cal_weight(best_weights)

IP, score = verification(top)

IP

score
# 3
```

```
num_weights = 5
weights = np.linspace(0.005, 1, num=10)  # 0.005 부터 1 까지 10 개의 가중치 생성
weight_combinations = itertools.product(weights, repeat=num_weights)  # 가중치 조합 생성

best_score = 0
best_weights = None

# 가중치 조합 시도
for w in weight_combinations:
    # 가중치의 합이 1 이 되도록 정규화
    normalized_weights = np.array(w) / sum(w)


    top, bottom = cal_weight(w)

    IP, score = verification(top)


    # 최적 성능 갱신
    if score > best_score:
        best_score = score
        best_weights = normalized_weights

print("Best weights:", best_weights)
print("Best score:", best_score)

top, bottom = cal_weight([0.00402865, 0.00402865, 0.27126231, 0.71665175, 0.00402865])

IP, score = verification(top)

IP

score

# init 1, 2, 3
IP_init = {2014: {'이현식': False, '장광수': False, '장우철': False, '조경순': False, '홍대한':
False},
 2015: {'조경순': True, '이현식': True, '장광수': True, '배영훈': False, '홍대한': True},
 2016: {'김경식': False, '조경순': True, '김범철': False, '배영훈': True, '장우철': True},
 2017: {'김성원': False, '김범철': True, '이정화': False, '정재중': False, '신인식': False},
 2018: {'신인식': True, '이득원': False, '김범철': True, '진승욱': True, '박성준': True},
 2019: {'정연우': False, '진승욱': True, '문병식': False, '이재우': False, '정연규': True},
 2020: {'강윤기': False, '문병식': True, '이재우': True, '신재범': False, '강준규': False},
 2021: {'강윤기': True, '이재우': True, '정연우': True, '이순남': False, '강준규': True},
 2022: {'강윤기': True, '김경렬': False, '최대경': False, '강준규': True, '김성균': False},
 2023: {'강윤기': True, '김성균': True, '강준규': True, '최대경': True, '김상원': False}}

IP1 = {2014: {'장광수': False, '이현식': False, '장우철': False, '조경순': False, '배영훈': False},
```

```
 2015: {'조경순': True, '장광수': True, '이현식': True, '배영훈': True, '홍대한': False},
 2016: {'김경식': False, '조경순': True, '김범철': False, '배영훈': True, '홍대한': True},
 2017: {'김성원': False, '김범철': True, '이정화': False, '정재중': False, '신인식': False},
 2018: {'신인식': True, '이득원': False, '김범철': True, '박성준': True, '진승욱': True},
 2019: {'정연우': False, '진승욱': True, '문병식': False, '신인식': True, '정연규': True},
 2020: {'강윤기': False, '문병식': True, '이재우': True, '신재범': False, '강준규': False},
 2021: {'강윤기': True, '이재우': True, '정연우': True, '이순남': False, '강준규': True},
 2022: {'강윤기': True, '김경렬': False, '최대경': False, '강준규': True, '김성균': False},
 2023: {'강윤기': True, '김성균': True, '강준규': True, '최대경': True, '김봉진': True}}

IP2 = {2014: {'이현식': False, '장광수': False, '장우철': False, '조경순': False, '홍대한': False},
 2015: {'조경순': True, '장광수': True, '이현식': True, '배영훈': False, '홍대한': True},
 2016: {'김경식': False, '조경순': True, '김범철': False, '배영훈': True, '김성원': False},
 2017: {'김성원': True, '김범철': True, '이정화': False, '정재중': False, '신인식': False},
 2018: {'신인식': True, '이득원': False, '김범철': True, '박성준': True, '진승욱': True},
 2019: {'정연우': False, '진승욱': True, '문병식': False, '신인식': True, '정연규': True},
 2020: {'강윤기': False, '문병식': True, '신재범': False, '이재우': True, '강준규': False},
 2021: {'강윤기': True, '이재우': True, '정연우': True, '강준규': True, '이순남': False},
 2022: {'강윤기': True, '김경렬': False, '최대경': False, '강준규': True, '김성균': False},
 2023: {'강윤기': True, '김성균': True, '강준규': True, '최대경': True, '김봉진': True}}

IP3 = {2014: {'장광수': False, '이현식': False, '장우철': False, '조경순': False, '배영훈': False},
 2015: {'조경순': True, '장광수': True, '이현식': True, '배영훈': True, '홍대한': False},
 2016: {'김경식': False, '조경순': True, '김범철': False, '홍대한': True, '배영훈': True},
 2017: {'김성원': False, '김범철': True, '이정화': False, '정재중': False, '신인식': False},
 2018: {'이득원': False, '신인식': True, '김범철': True, '박성준': True, '진승욱': True},
 2019: {'정연우': False, '진승욱': True, '문병식': False, '신인식': True, '정연규': True},
 2020: {'강윤기': False, '문병식': True, '이재우': True, '신재범': False, '강준규': False},
 2021: {'강윤기': True, '이재우': True, '정연우': True, '이순남': False, '강준규': True},
 2022: {'강윤기': True, '김경렬': False, '최대경': False, '강준규': True, '김성균': False},
 2023: {'강윤기': True, '김성균': True, '강준규': True, '최대경': True, '김봉진': True}}
```

데이터분석

```
df.info()


print("2014년 총 임원 수: ", len(df_2014))
print("2015년 총 임원 수: ", len(df_2015))
print("2016년 총 임원 수: ", len(df_2016))
print("2017년 총 임원 수: ", len(df_2017))
print("2018년 총 임원 수: ", len(df_2018))
print("2019년 총 임원 수: ", len(df_2019))
print("2020년 총 임원 수: ", len(df_2020))
print("2021년 총 임원 수: ", len(df_2021))
print("2022년 총 임원 수: ", len(df_2022))
print("2023년 총 임원 수: ", len(df_2022))


#2014년도 요약통계
```

```
df_2014.describe()

#2015 년도 요약통계
df_2015.describe()

#2016 년도 요약통계
df_2016.describe()

#2017 년도 요약통계
df_2017.describe()

#2018 년도 요약통계
df_2018.describe()

#2019 년도 요약통계
df_2019.describe()

#2020 년도 요약통계
df_2020.describe()

#2021 년도 요약통계
df_2021.describe()

#2022 년도 요약통계
df_2022.describe()

#2023 년도 요약통계
df_2023.describe()

#임원 이름 중복없이 추출
names= []
for i in range(len(df)):
  names.append(df.loc[i, '이름'])
#print(len(names))

names = set(names)

print(len(names))

#중복 없는 데이터셋 생성
df_a = df.copy()
for name in names:
  indexes = list(df[df['이름']==name].index)
  indexes.pop(-1) #가장 최신꺼 제외
  if len(indexes) > 0 :
    df_a.drop(indexes, inplace=True) #최신꺼만 남기고 삭제
df_a.reset_index(drop=True, inplace=True)
```

##<중복없이 임원들에 대해서 각 피처 살펴보기>

```
#피처별 count 수 그리는 막대그래프 함수
def graph_with_bar(col, fsize1 = 9, fsize2 = 5, xlabels=12, ylabels=12, titles = 12, rsize=10,
xtick=False):
  x = df_a[col].value_counts().index
  y = df_a[col].value_counts().values
  plt.figure(figsize=(fsize1,fsize2))
  if xtick :
    plt.xticks(x)
  plt.xlabel(col, size=xlabels)
  plt.ylabel('count', size=ylabels)
  plt.title(f'{col} 별 count 수', size=titles)
  bar = plt.bar(x, y)

  for rect in bar:
      height = rect.get_height()
      plt.text(rect.get_x() + rect.get_width()/2.0, height, '%.f' % height, ha='center',
va='bottom', size = rsize)

## 대학 중복 통합
df_a.loc[df_a['대학교']=='연세대학교', '대학교'] = '연세대'
df_a.loc[df_a['대학교']=='성균관대학교', '대학교'] = '성균관대'
df_a.loc[df_a['대학교']=='한국외국어대', '대학교'] = '한국외대'
df_a.loc[df_a['대학교']=='세종대학교', '대학교'] = '세종대'
df_a.loc[df_a['대학교']=='가천대학교', '대학교'] = '가천대'
df_a.loc[df_a['대학교']=='충북대학교', '대학교'] = '충북대'

#각 대학별 count 막대그래프
x = df_a['대학교'].value_counts().index
y = df_a['대학교'].value_counts().values
plt.figure(figsize=(15,9))
plt.xticks(rotation=85, size= 9)
plt.xlabel('대학이름', size=17)
plt.ylabel('count', size=17)
plt.title('각 대학별 count 수', size=17)
bar = plt.bar(x, y)

for rect in bar:
    height = rect.get_height()
    plt.text(rect.get_x() + rect.get_width()/2.0, height, '%.f' % height, ha='center', va='bottom',
size = 12)

# 대학 지역별 count 막대그래프
def find_college_location(x):
  #지역별로 대학 딕셔너리
  college_dict = {'서울경기':['상명여자사범대', '서울대', '홍익대', '성균관대', '연세대', '연세대학교',
'동국대', '중앙대', '한국외대', '한국외국어대', '세종대학교', '세종대', '성균관대학교', '경희대',
'서강대', '고려대', '건국대', '가천대', '가천대학교', '한양대', '명지대'],
```

```
                  '광역시':['부경대', '경북대', '전남대', '조선대', '한국과학기술원', '가천대',
'인하대', '동아대', '대전대', '울산대', '영남공업전문대'],
                  '그외한국':['전북대', '원광대', '충북대', '충북대학교', '강원대', 'KDI'],
                  '해외':['FloridaInternationalUniversity', 'OhioStateUniversity',
'UniversityofBath', '조지아공대', 'Brown 대', '카네기멜론대', '미시간대학교']}

  #키값 찾기
  for key, val in college_dict.items() :
    if x in val :
      return key

#지역별 대학 피처 생성
df_a['지역별 대학'] = df_a['대학교'].apply(find_college_location)

#그래프 그리기
graph_with_bar('지역별 대학')

# 전공 계열별 count 막대그래프
def find_department_location(x):
  #전공별 계열 딕셔너리
  department_dict={'경영경제':['회계학', '세무학', '경제학', '국제경영학', '경영학', '무역학', 'MBA',
'경제통상학', '경영대학원', '경영대학원 MBA', '산업정보학', '금융공학', '국제금융학'],
                  '사회과학':['사회학', '사회과학대학원', '행정학', '정치외교학', '정치학',
'신문방송학'],
                  '법률':['법학', '사법학', '정책대학원', '국제정책대학원'],
                  '인문':['불어불문학'],
                  '기타':['교육학', '인적자원개발학'],
                  '자연':['응용통계학', '통계학', '수학'],
                  '공학':['기계공학', '화학공학', '전자계산학', '전자전산학', '전산학', '항공우주학']}

  #키값 찾기
  for key, val in department_dict.items() :
    if x in val :
      return key

#전공 계열 피처 생성
df_a['전공계열'] = df_a['전공'].apply(find_department_location)

#그래프 그리기
graph_with_bar('전공계열')

#직위별 count 막대그래프
graph_with_bar('직위')

#근속연수 count 막대그래프
graph_with_bar('근속연수', fsize1=14, fsize2=9, xlabels=13, ylabels=13, titles=13, rsize=11, xtick =
True)
```

```
#근속연수 boxplot
plt.figure(figsize=(5,5))
plt.boxplot(df_a['근속연수'])
plt.title('근속연수 boxplot', size=13)

df_a['근속연수'].describe()

df_a['5 년기준근속연수'] = df_a['근속연수'].apply(lambda x : x//5)
graph_with_bar('5 년기준근속연수', xtick=True)

#담당업무 같은 계열 별 count 막대그래프
def find_task_location(x):
  #담당업무 딕셔너리
  task_dict={'IT 계열': {'IT':['IT 부문장', 'IT 본부장', 'IT 서비스본부장', '디지털부문장',
'스마트 Biz 본부장', 'Operation&Technology 본부장'], '정보보호':['정보보호부문장', '정보보호담당']},
            '본사영업':{'Wolesale':['Wholesale 부문장','Wholesale 영업본부장', 'Wholesale 사업단장',
'Trading 부문장', 'Trading Center 장', 'Sales&Trading 총괄', '대외협력담당'],
                    'IB':['IB 사업단장', 'IB 부문장', 'IB 부부문장', 'IB1 부문장', 'IB2 부문장',
'IPO 담당',  'PF 부문장', 'PF1 본부장', '구조화상품본부장'],
                    '영업':['영업부장']},
            '본사관리':{'리테일':['리테일총괄', 'Club1962 센터장','WM 추진부문장', 'WM 추진본부장',
'WM 사업단장'],
                    '리스크':['리스크관리부문장', '리스크관리담당'] ,
                    '경영':['경영전략총괄', '전략지원부문장', '전략지원담당', '경영지원본부장',
'경영지원부문장', '경영기획부문장', '경영기획본부장', '기획본부장', '기획본부장 / 투자금융담당',
'전략지원부문장/프라이빗라운지 부문장'],
                    '상품':['Product 부문장', 'Solution&Product 사업단장'],
                    '업무':['업무총괄',  '업무총괄(IB 사업단, 고객자산본부 제외)',
'업무총괄(IB 사업단,고객자산본부 제외)', '업무총괄 (IB 사업단,고객자산본부 제외)', '업무총괄(IB 사업단,
고객자산본부, 경영지원본부, 정보보호부문, 준법지원부문, 감사부문 제외)'],
                    '금융':['기업금융담당', '스마트금융본부장', '금융주치의추진본부장',
'금융주치의사업단장'],
                    '인사,인프라':['인재역량센터장', '인프라관리본부장'],
                    '법률':['준법감시인\n 준법지원부문장', '준법감시인 준법지원부문장',
'준법감시인\n 준법지원부문장', '준법감시인\n 준법지원담당', '준법감시인/준법지원부문장', '준법감시인 /
준법지원담당', 'Advisory 본부장'],
                    '감사':['감사위원장', '감사부문장', '감사위원', '상근감사위원', '감사담당'],
                    '언론홍보':['홍보부문장', 'Coverage 본부장']},
            '리서치': {'리서치센터':['Research Center 장', '리서치센터장', 'Market\n  Solution 부문장',
'Research&Strategy 본부장', 'Market Solution 부문장', 'Market Solution 부문장']},
            '고객':{'고객자산':['고객자산부문장', '고객자산부문 부부문장', '고객자산본부장',
'고객자산본부장 / 홍보담당', '고객자산본부장/ 홍보담당', '고객자산본부장/ 홍보부문장',
'고객자산본부장/홍보부문장', '고객자산부문장, 홍보부문장', '고객자산부문장\n 홍보부문장'],
                  'WM':['서부 WM 부문장', '서부 WM 본부장', '재경 1WM 부문장', '재경 1WM 본부장',
'재경 2WM 본부장', '재경 2WM 부문장', '동부 WM 부문장', '동부 WM 본부장'],
                  '프라이빗':['프라이빗부문장,\n 나인원프라이빗라운지장', '프라이빗부문장,
나인원프라이빗라운지장', '대신나인원 \n 프라이빗라운지장', '프라이빗라운지 부문장 /대신나인원
프라이빗라운지장', '프라이빗부문장, 대신나인원프라이빗라운지장', '프라이빗부문장\n 나인원프라이빗라운지장',],
```

```
                '소비자보호':['금융소비자보호부문장\n(COO)', '금융소비자보호부문장(COO)',
'금융소비자보호부문장 (COO)', '금융소비자보호 총괄', '금융소비자보호총괄']},
            '기타':{'지점':['강북지역본부장', '강남지역본부장', '강남선릉센터장', '서부지역본부장',
'동부지역본부장', '울산지점장'],
                '비서':['비서실장', '비서/브랜드본부', '비서/브랜드담당'], '공란':['-']}}

   #키값 찾기
  for outerKey in task_dict.keys() :
    for key, val in task_dict[outerKey].items() :
      if x in val :
        return outerKey

#담당업무계열 피처 생성
df_a['담당업무계열'] = df_a['담당업무'].apply(find_task_location)

#그래프 그리기
graph_with_bar('담당업무계열')

def find_career_location(x):
  # 경력 딕셔너리
  career_dict={1:['대신증권', '대신증권 WM 사업단장', 'WM 추진부문장', '재경 1WM 부문장',
'대신증권중부지역본부장', '기획본부장대신증권', '대신증권분당지점장', '대신증권무거동지점장',
'대신증권 Capital',
                '이사대우부본부장', '기획본부장대신증권인재역량센터장', '대신증권인재전략부',
'대신증권파생상품운용부장', '대신증권자산운용본부', '대신증권영업기획부장', '대신증권기획실장',
                '대신증권홍보실', '대신증권브랜드전략실', '대신증권 IT 개발부장',
'대신증권트레이딩시스템부장', '대신증권 Global 사업본부', '신증권', '기업금융사업단장'], #대신증권
              0:['대신에프앤아이', '대신투자신탁운용', '대신투자신탁운용상무이사', '대신에이엠씨',
'대신자산운용', '대신저축은행'], #대신증권 자회사
              2:['메리츠종금증권', '메리츠증권', '미래에셋증권', 'KTB 투자증권', 'SC 제일은행',
'하나금융투자', 'IBK 투자증권', 'DB 금융투자', 'NH 투자증권', 'KB 증권', 'LIG 손해보험', '굿모닝신한증권',
'메릴린치증권',
                '삼성증권', '동양증권', '한국투자증권', '한국투자증권평촌지점', '대우증권',
'대우본부장', '대우증권전략기획본부', '대우증권해외사업부문', '대우증권 IB 사업부문',
                '우리투자증권', '우리 CS 자산운용', '우리투자증권기업금융 2 팀장', '우리프라이빗에쿼티',
'푸르덴셜투자증권', '하이투자증권주식인수팀',  '하이투자증권주식인수팀', 'SBC', 'Bank', '도이치은행'],
#타증권사
              3:['금융감독원', '금융위원회', '現)금융위원회', '감사원', '대검찰청', '법무법인',
'現)법무법인', '세무법인', '국세청', '중부지방국세청', '서울북부지방검찰청','서울지방국세청장',
'서울지방국세청',
                '제 58 대', '제 22 대', '기획재정부', '국세청,관세청,산업통상자원부', '자본시장연구원',
'한국조세연구원', '국가청렴위원회', '대한중재인협회', '한국회계정보학회',
                '금융위원회금융발전심의위원(현)', '금융위원회적극행정심의위원(現)',
'안진회계법인',  'L&C 세무회계사무소', '변호사정상명법률사무소', '피앤비세무컨설팅', '現)법무법인(유)'],
#법,정책
              4:['중앙대', '중앙대학교', '現)중앙대', '연세대학교', 'KAIST', '수원대', '수원대학교',
'서울시립대', '現)서울시립대', '학교법인', '現)학교법인', '재단법인', '現)재단법인'], #학교, 교수, 재단
              5:['하비스트', '한국물산', '우리선물', '부영주택', 'LG 경제연구원', 'Arthur'] #기타
```

```
            }

  #경력 여부 빈리스트 생성
  x_list = [0,0,0,0,0,0] #1 번인덱스: 대신증권, 0 번인덱스: 대신증권 자회사, 2 번인덱스: 타증권사,
3 번인덱스: 기타

  #리스트에 각 업종에 종사했었으면 1, 아니면 0
  for career in x :
    for key, val in career_dict.items() :
      if career in val :
        x_list[key]=1
  return x_list

# 리스트의 각 요소에 접근하는 함수
def element_0(lst):
  return lst[0]

def element_1(lst):
  return lst[1]

def element_2(lst):
  return lst[2]

def element_3(lst):
  return lst[3]

def element_4(lst):
  return lst[4]

def element_5(lst):
  return lst[5]

df_a['주요경력리스트'] = df_a['주요경력요약'].apply(find_career_location)

#경력 원핫인코딩
df_a['대신증권자회사종사'] = df_a['주요경력리스트'].apply(lambda x: element_0(x))
df_a['대신증권종사'] = df_a['주요경력리스트'].apply(lambda x: element_1(x))
df_a['타증권사종사'] = df_a['주요경력리스트'].apply(lambda x: element_2(x))
df_a['법정책종사'] = df_a['주요경력리스트'].apply(lambda x: element_3(x))
df_a['학교재단종사'] = df_a['주요경력리스트'].apply(lambda x: element_4(x))
df_a['기타종사'] = df_a['주요경력리스트'].apply(lambda x: element_5(x))

#경력 별 count 막대그래프
x = ['대신증권자회사', '대신증권', '타증권사', '법,정책', '학교,재단', '기타']

#경력별 count 수
career_lists = np.array([0,0,0,0,0,0])
for i in range(len(df_a)):
```

```
  career_lists += np.array(df_a['주요경력리스트'].iloc[i])

plt.figure(figsize=(9,5))
plt.xlabel('경력', size=12)
plt.ylabel('count', size=12)
plt.title('경력별 count 수', size=12)
bar = plt.bar(x, career_lists)

for rect in bar:
    height = rect.get_height()
    plt.text(rect.get_x() + rect.get_width()/2.0, height, '%.f' % height, ha='center', va='bottom',
size = 10)

#대신증권 중심 경력 count 막대그래프
#피처 생성
df_a['대신증권에서만 종사'] = df_a['주요경력리스트'].apply(lambda x: x==[0,1,0,0,0,0]).astype(int)
df_a['자회사에서만 종사'] = df_a['주요경력리스트'].apply(lambda x: x==[1,0,0,0,0,0]).astype(int)
df_a['대신증권&자회사'] = df_a['주요경력리스트'].apply(lambda x: x==[1,1,0,0,0,0]).astype(int)
df_a['대신증권 + 다른 경력'] = df_a['주요경력리스트'].apply(lambda x: (x[1]==1) and (x!=[0,1,0,0,0,0])
and (x!=[1,1,0,0,0,0])).astype(int)
df_a['대신증권관련없'] = df_a['주요경력리스트'].apply(lambda x: (x[0]!=1) and (x[1]!=1)).astype(int)

#경력 별 count 막대그래프
x = ['대신증권에서만 종사', '자회사에서만 종사', '대신증권&자회사', '대신증권 + 다른 경력',
'대신증권관련없']
y = [df_a['대신증권에서만 종사'].sum(), df_a['자회사에서만 종사'].sum(), df_a['대신증권&자회사'].sum(),
df_a['대신증권 + 다른 경력'].sum(), df_a['대신증권관련없'].sum()]

plt.figure(figsize=(9,5))
plt.xlabel('대신증권 중심 경력', size=12)
plt.ylabel('count', size=12)
plt.title('대신증권 중심 경력별 count 수', size=12)
bar = plt.bar(x, y)

for rect in bar:
    height = rect.get_height()
    plt.text(rect.get_x() + rect.get_width()/2.0, height, '%.f' % height, ha='center', va='bottom',
size = 10)

#보유주식 boxplot
plt.figure(figsize=(9,5))
plt.boxplot(df_a['의결권 있는 주식수'])
plt.ticklabel_format(axis='y',useOffset=False, style='plain')
plt.title('보유주식 boxplot', size=13)

df_a['의결권 있는 주식수'].describe()

#보유주식 count 산점도그래프
```

```
x = df_a['의결권 있는 주식수'].value_counts().index
y = df_a['의결권 있는 주식수'].value_counts().values
plt.figure(figsize=(12,5))
plt.ylim(y.min()-1, y.max()+1)
plt.xticks(x, rotation=90, size=8)
plt.xlabel('보유주식', size=13)
plt.ylabel('count', size=13)
plt.title('보유주식 count 수', size=13)
plt.scatter(x, y, alpha = 0.17, s= 100, color = 'red')

df_a['의결권 있는 주식수'].value_counts()

#count 최댓값 주식수, count 수
print(df_a['의결권 있는 주식수'].value_counts().idxmax(), ',', df_a['의결권 있는
주식수'].value_counts().max())
```

##<2개 결합>

```
def graph_with_heatmap(col1, col2, fsize1=11, fsize2=8, titles=13):
  #히트맵 그릴 데이터프레임 생성
  a= df_a.groupby([col1,col2])[col1].agg(['count'])
  la = list(a.index)
  rows = list(df_a[col1].unique())
  cols = list(df_a[col2].unique())
  data = pd.DataFrame(index =rows, columns =  cols)
  #count 수 채우기
  for i in la:
    data.loc[i[0], i[1]] = a.loc[i[0], i[1]].item()
  #count 수 없는 None 0 으로 채우기
  data.fillna(0, inplace=True)

  #정렬
  data.sort_index(axis = 1, inplace=True)
  data.sort_index(axis = 0, inplace=True)

  #히트맵 그리기
  plt.figure(figsize=(fsize1,fsize2))
  plt.title(f'{col1} 별 {col2} count 수', fontsize=titles)
  sns.heatmap(data.transpose(), cmap='hot', annot=True, fmt='d')

#대학별 전공 count 수 히트맵
graph_with_heatmap('대학교', '전공계열', fsize1=17)

#대학별 직위 count 수 히트맵
graph_with_heatmap('대학교', '직위', fsize1=17)

#대학별 근속연수 count 수 히트맵
graph_with_heatmap('대학교', '근속연수', fsize1=17)
```

```
#대학별 5년 기준으로 자른 근속연수 count 수 히트맵
graph_with_heatmap('대학교', '5년기준근속연수', fsize1=17, fsize2 = 5)

#대학별 담당업무계열 count 수 히트맵
graph_with_heatmap('대학교', '담당업무계열', fsize1=16)

#대학별 경력 count 수 히트맵

#대학별 경력 데이터프레임 생성
careers = ['대신증권자회사종사', '대신증권종사', '타증권사종사', '법정책종사', '학교재단종사', '기타종사']
df_b = pd.DataFrame(index = df_a['대학교'].unique(), columns = careers)
for i in df_a['대학교'].unique():
  for career in careers:
    df_b.loc[i, career] = np.sum(df_a[df_a['대학교']==i][career])
df_b = df_b.astype(float)

#정렬
df_b.sort_index(axis = 1, inplace=True)
df_b.sort_index(axis = 0, inplace=True)

#히트맵 그리기
plt.figure(figsize=(17,8))
plt.title(f'대학교 별 경력 count 수', fontsize=12)
sns.heatmap(df_b.transpose(), cmap='hot', annot=True)

#대학별 대신증권 중심 경력 count 수 히트맵

#대학별 대신증권 중심 경력 데이터프레임 생성
careers = ['대신증권에서만 종사', '자회사에서만 종사', '대신증권&자회사', '대신증권 + 다른 경력',
'대신증권관련없']
df_b = pd.DataFrame(index = df_a['대학교'].unique(), columns = careers)
for i in df_a['대학교'].unique():
  for career in careers:
    df_b.loc[i, career] = np.sum(df_a[df_a['대학교']==i][career])
df_b = df_b.astype(float)

#정렬
df_b.sort_index(axis = 1, inplace=True)
df_b.sort_index(axis = 0, inplace=True)

#히트맵 그리기
plt.figure(figsize=(17,8))
plt.title(f'대학교 별 대신증권 중심 경력 count 수', fontsize=12)
sns.heatmap(df_b.transpose(), cmap='hot', annot=True)

#대학별 보유주식수 count 수 히트맵
graph_with_heatmap('대학교', '의결권 있는 주식수', fsize1=16)
```

```
#대학별 보유주식수 산점도

#산점도 그리기 위해 문자형 라벨인코딩
uni = df_a['대학교'].unique()
x = df_a['대학교'].values
encoder = LabelEncoder()
encoder.fit(uni)
label1 = encoder.transform(uni)
label2 = encoder.transform(x)
y = df_a['의결권 있는 주식수'].values

#산점도 그리기
plt.figure(figsize=(16,5))
plt.ticklabel_format(axis='y',useOffset=False, style='plain')
plt.xticks(label1, labels = uni, rotation=90, size=8)
plt.yticks(np.arange(np.array(y).min(), np.array(y).max(), 500000))
plt.xlabel('대학교', size=13)
plt.ylabel('보유주식', size=13)
plt.title('대학별 보유주식수', size=13)
plt.scatter(label2, y, alpha = 0.17, s= 100, color = 'red')

#지역대학별 전공 count 수 히트맵
graph_with_heatmap('지역별 대학', '전공계열', fsize1 = 7, fsize2 = 6)

#지역대학별 직위 count 수 히트맵
graph_with_heatmap('지역별 대학', '직위', fsize1 = 7, fsize2 = 6)

#지역대학별 근속연수 count 수 히트맵
graph_with_heatmap('지역별 대학', '근속연수', fsize1 = 7)

#지역대학별 5년 기준으로 자른 근속연수 count 수 히트맵
graph_with_heatmap('지역별 대학', '5년기준근속연수', fsize1 = 7, fsize2 = 6)

#지역대학별 담당업무계열 count 수 히트맵
graph_with_heatmap('지역별 대학', '담당업무계열',  fsize1 = 7, fsize2 = 6)

#지역대학별 경력 count 수 히트맵

#대학별 경력 데이터프레임 생성
careers = ['대신증권자회사종사', '대신증권종사', '타증권사종사', '법정책종사', '학교재단종사', '기타종사']
df_b = pd.DataFrame(index = df_a['지역별 대학'].unique(), columns = careers)
for i in df_a['지역별 대학'].unique():
  for career in careers:
    df_b.loc[i, career] = np.sum(df_a[df_a['지역별 대학']==i][career])
df_b = df_b.astype(float)

#정렬
df_b.sort_index(axis = 1, inplace=True)
```

```
df_b.sort_index(axis = 0, inplace=True)

#히트맵 그리기
plt.figure(figsize=(7,6))
plt.title(f'지역별대학 별 경력 count 수', fontsize=12)
sns.heatmap(df_b.transpose(), cmap='hot', annot=True)

#지역대학별 대신증권 중심 경력 count 수 히트맵

#대학별 대신증권 중심 경력 데이터프레임 생성
careers = ['대신증권에서만 종사', '자회사에서만 종사', '대신증권&자회사', '대신증권 + 다른 경력',
'대신증권관련없']
df_b = pd.DataFrame(index = df_a['지역별 대학'].unique(), columns = careers)
for i in df_a['지역별 대학'].unique():
  for career in careers:
    df_b.loc[i, career] = np.sum(df_a[df_a['지역별 대학']==i][career])
df_b = df_b.astype(float)

#정렬
df_b.sort_index(axis = 1, inplace=True)
df_b.sort_index(axis = 0, inplace=True)

#히트맵 그리기
plt.figure(figsize=(7, 6))
plt.title(f'지역별대학 별 대신증권 중심 경력 count 수', fontsize=12)
sns.heatmap(df_b.transpose(), cmap='hot', annot=True)

#지역대학별 보유주식수 count 수 히트맵
graph_with_heatmap('지역별 대학', '의결권 있는 주식수', fsize1 = 8, fsize2 = 10)

#지역대학별 보유주식수 산점도

#산점도 그리기 위해 문자형 라벨인코딩
uni = df_a['지역별 대학'].unique()
x = df_a['지역별 대학'].values
encoder = LabelEncoder()
encoder.fit(uni)
label1 = encoder.transform(uni)
label2 = encoder.transform(x)
y = df_a['의결권 있는 주식수'].values

#산점도 그리기
plt.figure(figsize=(7,5))
plt.ticklabel_format(axis='y',useOffset=False, style='plain')
plt.xticks(label1, labels = uni, size=8)
plt.yticks(np.arange(np.array(y).min(), np.array(y).max(), 500000))
plt.xlabel('지역별 대학', size=13)
plt.ylabel('보유주식', size=13)
```

```
plt.title('지역별 대학 보유주식수', size=13)
plt.scatter(label2, y, alpha = 0.17, s= 100, color = 'red')

#전공계열별 직위 count 수 히트맵
graph_with_heatmap('전공계열', '직위', fsize1 = 7, fsize2 = 6)

#전공계열별 근속연수 count 수 히트맵
graph_with_heatmap('전공계열', '근속연수', fsize1 = 7)

#전공계열별 5년 기준으로 자른 근속연수 count 수 히트맵
graph_with_heatmap('전공계열', '5년기준근속연수', fsize1 = 7, fsize2 = 6)

#전공계열별 담당업무계열 count 수 히트맵
graph_with_heatmap('전공계열', '담당업무계열',  fsize1 = 7, fsize2 = 6)

#전공계열별 경력 count 수 히트맵

#경력 데이터프레임 생성
careers = ['대신증권자회사종사', '대신증권종사', '타증권사종사', '법정책종사', '학교재단종사', '기타종사']
df_b = pd.DataFrame(index = df_a['전공계열'].unique(), columns = careers)
for i in df_a['전공계열'].unique():
  for career in careers:
    df_b.loc[i, career] = np.sum(df_a[df_a['전공계열']==i][career])
df_b = df_b.astype(float)

#정렬
df_b.sort_index(axis = 1, inplace=True)
df_b.sort_index(axis = 0, inplace=True)

#히트맵 그리기
plt.figure(figsize=(7,6))
plt.title(f'전공계열 별 경력 count 수', fontsize=12)
sns.heatmap(df_b.transpose(), cmap='hot', annot=True)

#전공계열별 대신증권 중심 경력 count 수 히트맵

#대신증권 중심 경력 데이터프레임 생성
careers = ['대신증권에서만 종사', '자회사에서만 종사', '대신증권&자회사', '대신증권 + 다른 경력',
'대신증권관련없']
df_b = pd.DataFrame(index = df_a['전공계열'].unique(), columns = careers)
for i in df_a['전공계열'].unique():
  for career in careers:
    df_b.loc[i, career] = np.sum(df_a[df_a['전공계열']==i][career])
df_b = df_b.astype(float)

#정렬
df_b.sort_index(axis = 1, inplace=True)
df_b.sort_index(axis = 0, inplace=True)
```

```
#히트맵 그리기
plt.figure(figsize=(7, 6))
plt.title(f'전공계열 별 대신증권 중심 경력 count 수', fontsize=12)
sns.heatmap(df_b.transpose(), cmap='hot', annot=True)

#전공계열별 보유주식수 count 수 히트맵
graph_with_heatmap('전공계열', '의결권 있는 주식수', fsize1 = 8, fsize2 = 10)

#전공계열별 보유주식수 산점도

#산점도 그리기 위해 문자형 라벨인코딩
uni = df_a['전공계열'].unique()
x = df_a['전공계열'].values
encoder = LabelEncoder()
encoder.fit(uni)
label1 = encoder.transform(uni)
label2 = encoder.transform(x)
y = df_a['의결권 있는 주식수'].values

#산점도 그리기
plt.figure(figsize=(7,5))
plt.ticklabel_format(axis='y',useOffset=False, style='plain')
plt.xticks(label1, labels = uni, size=8)
plt.yticks(np.arange(np.array(y).min(), np.array(y).max(), 500000))
plt.xlabel('전공계열', size=13)
plt.ylabel('보유주식', size=13)
plt.title('전공계열 별 보유주식수', size=13)
plt.scatter(label2, y, alpha = 0.17, s= 100, color = 'red')

#직위별 근속연수 count 수 히트맵
graph_with_heatmap('직위', '근속연수', fsize1 = 10)

#직위별 5년 기준으로 자른 근속연수 count 수 히트맵
graph_with_heatmap('직위', '5년기준근속연수', fsize1 = 10, fsize2 = 6)

#직위별 담당업무계열 count 수 히트맵
graph_with_heatmap('직위', '담당업무계열',  fsize1 = 10, fsize2 = 6)

#직위별 경력 count 수 히트맵

#경력 데이터프레임 생성
careers = ['대신증권자회사종사', '대신증권종사', '타증권사종사', '법정책종사', '학교재단종사', '기타종사']
df_b = pd.DataFrame(index = df_a['직위'].unique(), columns = careers)
for i in df_a['직위'].unique():
  for career in careers:
    df_b.loc[i, career] = np.sum(df_a[df_a['직위']==i][career])
df_b = df_b.astype(float)
```

```
#정렬
df_b.sort_index(axis = 1, inplace=True)
df_b.sort_index(axis = 0, inplace=True)

#히트맵 그리기
plt.figure(figsize=(10,6))
plt.title(f'직위 별 경력 count 수', fontsize=12)
sns.heatmap(df_b.transpose(), cmap='hot', annot=True)

#직위별 대신증권 중심 경력 count 수 히트맵

#대신증권 중심 경력 데이터프레임 생성
careers = ['대신증권에서만 종사', '자회사에서만 종사', '대신증권&자회사', '대신증권 + 다른 경력',
'대신증권관련없']
df_b = pd.DataFrame(index = df_a['직위'].unique(), columns = careers)
for i in df_a['직위'].unique():
  for career in careers:
    df_b.loc[i, career] = np.sum(df_a[df_a['직위']==i][career])
df_b = df_b.astype(float)

#정렬
df_b.sort_index(axis = 1, inplace=True)
df_b.sort_index(axis = 0, inplace=True)

#히트맵 그리기
plt.figure(figsize=(10, 6))
plt.title(f'직위 별 대신증권 중심 경력 count 수', fontsize=12)
sns.heatmap(df_b.transpose(), cmap='hot', annot=True)

#직위별 보유주식수 count 수 히트맵
graph_with_heatmap('직위', '의결권 있는 주식수', fsize1 = 10, fsize2 = 10)

#직위별 보유주식수 산점도

#산점도 그리기 위해 문자형 라벨인코딩
uni = df_a['직위'].unique()
x = df_a['직위'].values
encoder = LabelEncoder()
encoder.fit(uni)
label1 = encoder.transform(uni)
label2 = encoder.transform(x)
y = df_a['의결권 있는 주식수'].values

#산점도 그리기
plt.figure(figsize=(10,5))
plt.ticklabel_format(axis='y',useOffset=False, style='plain')
plt.xticks(label1, labels = uni, size=8)
```

```
plt.yticks(np.arange(np.array(y).min(), np.array(y).max(), 500000))
plt.xlabel('직위', size=13)
plt.ylabel('보유주식', size=13)
plt.title('직위 별 보유주식수', size=13)
plt.scatter(label2, y, alpha = 0.17, s= 100, color = 'red')

#담당업무계열별 근속연수 count 수 히트맵
graph_with_heatmap('담당업무계열', '근속연수', fsize1 = 7)

#담당업무계열별 5년 기준으로 자른 근속연수 count 수 히트맵
graph_with_heatmap('담당업무계열', '5년기준근속연수', fsize1 = 7, fsize2 = 8)

#담당업무계열별 경력 count 수 히트맵

#경력 데이터프레임 생성
careers = ['대신증권자회사종사', '대신증권종사', '타증권사종사', '법정책종사', '학교재단종사', '기타종사']
df_b = pd.DataFrame(index = df_a['담당업무계열'].unique(), columns = careers)
for i in df_a['담당업무계열'].unique():
  for career in careers:
    df_b.loc[i, career] = np.sum(df_a[df_a['담당업무계열']==i][career])
df_b = df_b.astype(float)

#정렬
df_b.sort_index(axis = 1, inplace=True)
df_b.sort_index(axis = 0, inplace=True)

#히트맵 그리기
plt.figure(figsize=(7,6))
plt.title(f'담당업무계열 별 경력 count 수', fontsize=12)
sns.heatmap(df_b.transpose(), cmap='hot', annot=True)

#담당업무계열별 대신증권 중심 경력 count 수 히트맵

#대신증권 중심 경력 데이터프레임 생성
careers = ['대신증권에서만 종사', '자회사에서만 종사', '대신증권&자회사', '대신증권 + 다른 경력',
'대신증권관련없']
df_b = pd.DataFrame(index = df_a['담당업무계열'].unique(), columns = careers)
for i in df_a['담당업무계열'].unique():
  for career in careers:
    df_b.loc[i, career] = np.sum(df_a[df_a['담당업무계열']==i][career])
df_b = df_b.astype(float)

#정렬
df_b.sort_index(axis = 1, inplace=True)
df_b.sort_index(axis = 0, inplace=True)

#히트맵 그리기
plt.figure(figsize=(7, 6))
```

```
plt.title(f'담당업무계열 별 대신증권 중심 경력 count 수', fontsize=12)
sns.heatmap(df_b.transpose(), cmap='hot', annot=True)

#담당업무계열별 보유주식수 count 수 히트맵
graph_with_heatmap('담당업무계열', '의결권 있는 주식수', fsize1 = 7, fsize2 = 10)
#담당업무계열별 보유주식수 산점도

#산점도 그리기 위해 문자형 라벨인코딩
uni = df_a['담당업무계열'].unique()
x = df_a['담당업무계열'].values
encoder = LabelEncoder()
encoder.fit(uni)
label1 = encoder.transform(uni)
label2 = encoder.transform(x)
y = df_a['의결권 있는 주식수'].values

#산점도 그리기
plt.figure(figsize=(7,5))
plt.ticklabel_format(axis='y',useOffset=False, style='plain')
plt.xticks(label1, labels = uni, size=8)
plt.yticks(np.arange(np.array(y).min(), np.array(y).max(), 500000))
plt.xlabel('담당업무계열', size=13)
plt.ylabel('보유주식', size=13)
plt.title('담당업무계열 별 보유주식수', size=13)
plt.scatter(label2, y, alpha = 0.17, s= 100, color = 'red')

#5 년기준근속연수별 경력 count 수 히트맵

#경력 데이터프레임 생성
careers = ['대신증권자회사종사', '대신증권종사', '타증권사종사', '법정책종사', '학교재단종사', '기타종사']
df_b = pd.DataFrame(index = df_a['5 년기준근속연수'].unique(), columns = careers)
for i in df_a['5 년기준근속연수'].unique():
  for career in careers:
    df_b.loc[i, career] = np.sum(df_a[df_a['5 년기준근속연수']==i][career])
df_b = df_b.astype(float)

#정렬
df_b.sort_index(axis = 1, inplace=True)
df_b.sort_index(axis = 0, inplace=True)

#히트맵 그리기
plt.figure(figsize=(7,6))
plt.title(f'5 년기준근속연수 별 경력 count 수', fontsize=12)
sns.heatmap(df_b.transpose(), cmap='hot', annot=True)

#5 년기준근속연수별 대신증권 중심 경력 count 수 히트맵

#대신증권 중심 경력 데이터프레임 생성
```

```
careers = ['대신증권에서만 종사', '자회사에서만 종사', '대신증권&자회사', '대신증권 + 다른 경력',
'대신증권관련없']
df_b = pd.DataFrame(index = df_a['5 년기준근속연수'].unique(), columns = careers)
for i in df_a['5 년기준근속연수'].unique():
  for career in careers:
    df_b.loc[i, career] = np.sum(df_a[df_a['5 년기준근속연수']==i][career])
df_b = df_b.astype(float)

#정렬
df_b.sort_index(axis = 1, inplace=True)
df_b.sort_index(axis = 0, inplace=True)

#히트맵 그리기
plt.figure(figsize=(7, 6))
plt.title(f'5 년기준근속연수 별 대신증권 중심 경력 count 수', fontsize=12)
sns.heatmap(df_b.transpose(), cmap='hot', annot=True)

#근속연수별 보유주식수 산점도
x = df_a['근속연수'].values
y = df_a['의결권 있는 주식수'].values

plt.figure(figsize=(7,7))
plt.ticklabel_format(axis='y',useOffset=False, style='plain')
plt.xticks(x)
plt.yticks(np.arange(np.array(y).min(), np.array(y).max(), 500000))
plt.xlabel('근속연수', size=13)
plt.ylabel('보유주식', size=13)
plt.title('근속연수 별 보유주식수', size=13)
plt.scatter(x, y, alpha = 0.17, s= 100, color = 'red')

#5 년기준근속연수별 보유주식수 count 수 히트맵
graph_with_heatmap('5 년기준근속연수', '의결권 있는 주식수', fsize1 = 7, fsize2 = 10)

#5 년기준근속연수별 보유주식수 산점도
x = df_a['5 년기준근속연수'].values
y = df_a['의결권 있는 주식수'].values

plt.figure(figsize=(7,7))
plt.ticklabel_format(axis='y',useOffset=False, style='plain')
plt.xticks(x)
plt.yticks(np.arange(np.array(y).min(), np.array(y).max(), 500000))
plt.xlabel('5 년기준근속연수', size=13)
plt.ylabel('보유주식', size=13)
plt.title('5 년기준근속연수 별 보유주식수', size=13)
plt.scatter(x, y, alpha = 0.17, s= 100, color = 'red')

#보유주식수별 경력 count 수 히트맵
```

```
#경력 데이터프레임 생성
careers = ['대신증권자회사종사', '대신증권종사', '타증권사종사', '법정책종사', '학교재단종사', '기타종사']
df_b = pd.DataFrame(index = df_a['의결권 있는 주식수'].unique(), columns = careers)
for i in df_a['의결권 있는 주식수'].unique():
  for career in careers:
    df_b.loc[i, career] = np.sum(df_a[df_a['의결권 있는 주식수']==i][career])
df_b = df_b.astype(float)

#정렬
df_b.sort_index(axis = 1, inplace=True)
df_b.sort_index(axis = 0, inplace=True)

#히트맵 그리기
plt.figure(figsize=(10,10))
plt.title(f'보유주식수 별 경력 count 수', fontsize=12)
sns.heatmap(df_b, cmap='hot', annot=True)

#보유주식수별 경력 산점도

#y 데이터 생성
y1 = df_a[df_a['대신증권자회사종사']==1]['의결권 있는 주식수'].values.tolist()
y2 = df_a[df_a['대신증권종사']==1]['의결권 있는 주식수'].values.tolist()
y3 = df_a[df_a['타증권사종사']==1]['의결권 있는 주식수'].values.tolist()
y4 = df_a[df_a['법정책종사']==1]['의결권 있는 주식수'].values.tolist()
y5 = df_a[df_a['학교재단종사']==1]['의결권 있는 주식수'].values.tolist()
y6 = df_a[df_a['기타종사']==1]['의결권 있는 주식수'].values.tolist()
y = y1+y2+y3+y4+y5+y6

#x 데이터 생성
x1 = [0]*len(y1)
x2 = [1]*len(y2)
x3 = [2]*len(y3)
x4 = [3]*len(y4)
x5 = [4]*len(y5)
x6 = [5]*len(y6)
x = x1+x2+x3+x4+x5+x6

#산점도 그리기
plt.figure(figsize=(7,10))
plt.ticklabel_format(axis='y',useOffset=False, style='plain')
plt.xticks([0,1,2,3,4,5], labels = careers, size=9)
plt.yticks(np.arange(np.array(y).min(), np.array(y).max(), 500000))
plt.xlabel('경력', size=13)
plt.ylabel('보유주식', size=13)
plt.title('경력 별 보유주식수', size=13)
plt.scatter(x, y, alpha = 0.1, s= 100, color = 'red')

#보유주식수별 대신증권 중심 경력 count 수 히트맵
```

```
#대신증권 중심 경력 데이터프레임 생성
careers = ['대신증권에서만 종사', '자회사에서만 종사', '대신증권&자회사', '대신증권 + 다른 경력',
'대신증권관련없']
df_b = pd.DataFrame(index = df_a['의결권 있는 주식수'].unique(), columns = careers)
for i in df_a['의결권 있는 주식수'].unique():
  for career in careers:
    df_b.loc[i, career] = np.sum(df_a[df_a['의결권 있는 주식수']==i][career])
df_b = df_b.astype(float)

#정렬
df_b.sort_index(axis = 1, inplace=True)
df_b.sort_index(axis = 0, inplace=True)

#히트맵 그리기
plt.figure(figsize=(10, 10))
plt.title(f'보유주식수 별 대신증권 중심 경력 count 수', fontsize=12)
sns.heatmap(df_b, cmap='hot', annot=True)

#보유주식수별 대신증권 중심 경력 산점도

#y 데이터 생성
y1 = df_a[df_a['대신증권에서만 종사']==1]['의결권 있는 주식수'].values.tolist()
y2 = df_a[df_a['자회사에서만 종사']==1]['의결권 있는 주식수'].values.tolist()
y3 = df_a[df_a['대신증권&자회사']==1]['의결권 있는 주식수'].values.tolist()
y4 = df_a[df_a['대신증권 + 다른 경력']==1]['의결권 있는 주식수'].values.tolist()
y5 = df_a[df_a['대신증권관련없']==1]['의결권 있는 주식수'].values.tolist()
y = y1+y2+y3+y4+y5

#x 데이터 생성
x1 = [0]*len(y1)
x2 = [1]*len(y2)
x3 = [2]*len(y3)
x4 = [3]*len(y4)
x5 = [4]*len(y5)
x = x1+x2+x3+x4+x5

#산점도 그리기
plt.figure(figsize=(7,10))
plt.ticklabel_format(axis='y',useOffset=False, style='plain')
plt.xticks([0,1,2,3,4], labels = careers, size=9)
plt.yticks(np.arange(np.array(y).min(), np.array(y).max(), 500000))
plt.xlabel('대신증권 중심 경력', size=13)
plt.ylabel('보유주식', size=13)
plt.title('대신증권 중심 경력 별 보유주식수', size=13)
plt.scatter(x, y, alpha = 0.1, s= 100, color = 'red')
```

##연도별 직위/보유주식수

```
#연도별 직위&보유주식수 히트맵
#연도리스트
YEAR_LIST = [2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023]

#subplot 생성
fig, axes = plt.subplots(5, 2, figsize=(24,50))

for i in range(len(YEAR_LIST)):
  #히트맵 그릴 데이터프레임 생성
  b=df[df['연도']==YEAR_LIST[i]]
  a = b.groupby(['직위','의결권 있는 주식수'])['직위'].agg(['count'])
  la = list(a.index)
  rows = list(b['직위'].unique())
  cols = list(b['의결권 있는 주식수'].unique())
  data = pd.DataFrame(index =rows, columns =  cols)
  #count 수 채우기
  for j in la:
    data.loc[j[0], j[1]] = a.loc[j[0], j[1]].item()
  #count 수 없는 None 0 으로 채우기
  data.fillna(0, inplace=True)

  #정렬
  data.sort_index(axis = 1, inplace=True)
  data.sort_index(axis = 0, inplace=True)

  axes[i//2,i%2].set_title(YEAR_LIST[i], fontsize=14)
  sns.heatmap(data.transpose(), cmap='hot', annot=True, fmt='d', ax = axes[i//2,i%2])
fig.tight_layout()

#연도별 직위&보유주식수 산점도
#연도리스트
YEAR_LIST = [2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023]

#subplot 생성
fig, axes = plt.subplots(5, 2, figsize=(17,30))


for i in range(len(YEAR_LIST)):
  #산점도 그리기 위해 문자형 라벨인코딩
  b=df[df['연도']==YEAR_LIST[i]]
  uni = b['직위'].unique()
  x = b['직위'].values
  encoder = LabelEncoder()
  encoder.fit(uni)
  label1 = encoder.transform(uni)
  label2 = encoder.transform(x)
  y = b['의결권 있는 주식수'].values
```

```
  #산점도 그리기
  axes[i//2,i%2].set_title(YEAR_LIST[i])
  axes[i//2,i%2].set_xlabel('직위')
  axes[i//2,i%2].set_ylabel('보유주식')
  axes[i//2,i%2].ticklabel_format(axis='y',useOffset=False, style='plain')
  axes[i//2,i%2].scatter(label2, y, alpha = 0.17, s= 100, color = 'red')
  axes[i//2,i%2].set_xticks(label1, labels = uni, size=8)
  axes[i//2,i%2].set_yticks(np.arange(np.array(y).min(), np.array(y).max(), 500000))


#연도별 직위&평균보유주식수 평균
#연도리스트
YEAR_LIST = [2023, 2022, 2021, 2020, 2019, 2018, 2017, 2016, 2015, 2014]
plt.figure(figsize=(12,15))
plt.title(f'연도별 직위&보유주식수', fontsize=12)
plt.xlabel('직위', size=12)
plt.ylabel('보유주식', size=12)
plt.ticklabel_format(axis='y',useOffset=False, style='plain')
plt.yticks(np.arange(0, 6000000, 500000))

for i in range(len(YEAR_LIST)):
  #그래프 그릴 데이터 생성
  #딕셔너리 생성
  p_of_df = df['직위'].unique()
  dictp = {}
  for p in p_of_df:
    dictp[p]=0.0
  dictp = {key: dictp[key] for key in sorted(dictp)}#딕셔너리 정렬

  #직위 별 각 연도의 평균 주식수
  b=df[df['연도']==YEAR_LIST[i]]
  positions = list(b['직위'].unique())
  means = [b[b['직위']==i]['의결권 있는 주식수'].mean() for i in positions]

  for j in range(len(positions)):
    dictp[positions[j]] = means[j]

  #x, y 데이터 생성
  x = list(dictp.keys())
  y=list(dictp.values())

  colors = ['black','magenta','blueviolet','blue','cyan','green','olive','yellow','coral','red']

  #꺾은선 그래프
  plt.plot(x, y, 'o-', color = colors[i], label = YEAR_LIST[i], markersize=10)

plt.legend()
```