회계가 바로 서야 경제가 바로 섭니다.

제2회 재무빅데이터분석사 자격시험

FDA 1급

Financial big-Data Analyst

▣ 시험시간: 180분
▣ 객관식배점: 문항별 배점 참조
▣ 주관식배점: 문항별 배점 참조

KICPA 한국공인회계사회

객관식/주관식

01 4개의 행과 6개의 열로 구성된 조합이 3개로 이루어지면서 모두 0으로 구성된 3차원 Array를 만드는 코드를 아래와 같이 작성하고자 한다. ㉮ ~ ㉯ 에 들어갈 내용으로 옳지 않은 것을 모두 고르시오(복수 선택). (3점)

```
>>> import numpy as ㉮
>>> eye_array = np.㉯((㉰,㉱,㉲))
>>> print (eye_array)
```

㉮ np	㉯ eye	㉰ 6
㉱ 4	㉲ 3	

① ㉮ ② ㉯ ③ ㉰
④ ㉱ ⑤ ㉲

해설
• 옳은 코드는 다음과 같다.

```
>>> import numpy as np
>>> eye_array = np.zeros((3, 4, 6))
>>> print(eye_array)
```

정답 ②, ③, ⑤

02 다음 코드는 a의 역행렬을 출력하는 코드이다. ㉮ ~ ㉯ 에 들어갈 내용으로 옳지 않은 것을 모두 고르시오(복수 선택). (3점)

```
>>> import numpy as np
>>> a = np.㉮([[5,6],[7,8]])
>>> a_역행렬 = np.㉯.㉰(㉱)
>>> print(㉲)
```

㉮ array	㉯ inv	㉰ linalg
㉱ a	㉲ a_역행렬	

① ㉮ ② ㉯ ③ ㉰
④ ㉱ ⑤ ㉲

해설
• 옳은 코드는 다음과 같다.

```
a = np.array([[5,6],[7,8]])
a_역행렬 = np.linalg.inv(a)
print(a_역행렬)
```

정답 ②, ③

03 print() 함수의 결과가 아래와 같다.

```
>>> import numpy as np
>>> 배열 = np.array([[ 24,  18, 166,  80],
                     [146, 118,  48,   4],
                     [ 88,   6,  36, 130],
                     [126,  56, 104, 178]])
>>> print(np.㉮(㉯ > ㉰))
(array([0, 1, 2, 3, 3], dtype=int64), array([2, 0, 3, 0, 3], dtype=int64))
```

상기에서 ㉰에 들어갈 내용으로 옳지 않은 것은(단수 선택)? (3점)

① 118 ② 120 ③ 122
④ 124 ⑤ 126

해설

• 옳은 코드는 다음과 같다.

```
>>> print(np.where(배열 > ㉰))  # 118 ~ 125까지는 결과가 상기와 동일하다.
```

```
>>> print(np.where(배열 > 126))
(array([0, 1, 2, 3], dtype=int64), array([2, 0, 3, 3], dtype=int64))
```

정답 ⑤

04 다음 Fraudit 테이블명은 '그룹' 이며, sub table로 구성되어 있다.

Filter Expression: Set 0 of 64

	입찰번호	프로젝트번호	입찰자번호	입찰일자	입찰금액	계산
0	5001	9001	8048	2009-03-07 00:00:00	188163.44	<N>
1	5002	9001	8063	2009-02-03 00:00:00	383786.19	<N>
2	5003	9001	8036	2009-02-03 00:00:00	593798.81	<N>
3	5004	9001	8059	2009-02-22 00:00:00	796503.19	<N>

상기 sub table의 '계산' 칼럼의 개별 레코드에 sub table의 '입찰금액' 칼럼의 값의 평균을 넣는 코드를 아래와 같이 작성하고자 한다. ㉮ ~ ㉲ 에 들어갈 내용으로 옳지 않은 것을 모두 고르시오(복수 선택). (3점)

```
>>> for i in range(len( ㉮ )) :
...     for ㉰ in range( len( ㉯ )) :
...         그룹[㉱]['계산'][j] = ㉲(그룹[i]['입찰금액'])
```

㉮ : 그룹	㉯ : 그룹	㉰ : j
㉱ : i	㉲ : average	

① ㉮ ② ㉯ ③ ㉰
④ ㉱ ⑤ ㉲

해설

• ㉯ : 그룹[i]
• ㉲ : mean

정답 ②, ⑤

05 다음과 같이 데이터프레임 3개를 생성하였다.

```
>>> import pandas as pd
>>> df1 = pd.DataFrame({'칼럼1': ['현금', '보통예금' , '외상매출금', '미수금', '상품'],
...                     '칼럼2' : [10,20,30,40,50],  '칼럼3' : [15,25,35,45,55]})
>>> df2 = pd.DataFrame({'칼럼1': ['현금', '보통예금','외상매출금'],
...                      '칼럼4' : [70,60,10],  '칼럼5' : [40,90,40]})
>>> df3 = pd.DataFrame({'칼럼1': ['보통예금', '외상매출금','상품'],
...                      '칼럼6': [50,20,50], '칼럼7': [80,70,30]})
```

다음과 같이 결합한 결과를 나타내고자 한다.

```
>>> res1 = ㉮.㉯(㉰,㉱, how = '㉲' )
>>> res1
      칼럼1   칼럼4   칼럼5   칼럼6   칼럼7
0     현금    70.0    40.0    NaN     NaN
1     보통예금  60.0    90.0    50.0    80.0
2     외상매출금 10.0    40.0    20.0    70.0
3     상품    NaN     NaN     50.0    30.0
```

상기에서 ㉮, ㉯, ㉰, ㉱, ㉲에 들어갈 내용으로 옳지 않은 것을 고르시오(단수 선택). (3점)

㉮ : pd	㉯ : merge	㉰ : df2
㉱ : df3	㉲ : left	

① ㉮ ② ㉯ ③ ㉰
④ ㉱ ⑤ ㉲

해설

• 옳은 코드는 다음과 같다.

```
>>> res1 = pd.merge(df2,df3, how ='outer')
```

정답 : ⑤

06 다음과 같이 데이터프레임 2개를 생성하였다.

```
>>> import pandas as pd
>>> df1 = pd.DataFrame({'id': [128940, 130960, 138250, 139480, 145990],
...         'stock_name' : ['한미약품', 'CJ E&M', '엔에스쇼핑', '이마트', '삼양사'],
...         'price' : [421000, 98900, 13200, 254500, 82000],
...         'per' : [30.75, 15.69, 7.45, 13.93, 14.28],})
>>> df2 = pd.DataFrame({'id': [130960, 136480, 138040, 139480, 145990],
...         'name' : ['CJ E&M', '하림', '메리츠금융지주', '이마트', '삼양사'],
...         'eps' : [6301.33, 274.16, 2122.33, 18268.16, 5741.00],})
```

다음과 같이 결합한 결과를 나타내고자 한다.

```
>>> res2 = ㉮.㉯(㉰, ㉱ = ㉲)
>>> res2
```

	id	stock_name	price	per	name	eps
0	128940	한미약품	421000.0	30.75	NaN	NaN
1	130960	CJ E&M	98900.0	15.69	NaN	NaN
2	138250	엔에스쇼핑	13200.0	7.45	NaN	NaN
3	139480	이마트	254500.0	13.93	NaN	NaN
4	145990	삼양사	82000.0	14.28	NaN	NaN
0	130960	NaN	NaN	NaN	CJ E&M	6301.33
1	136480	NaN	NaN	NaN	하림	274.16
2	138040	NaN	NaN	NaN	메리츠금융지주	2122.33
3	139480	NaN	NaN	NaN	이마트	18268.16
4	145990	NaN	NaN	NaN	삼양사	5741.00

상기에서 ㉮, ㉯, ㉰, ㉱, ㉲에 들어갈 내용으로 옳지 않은 것을 모두 고르시오(복수 선택). (3점)

㉮ : pd	㉯ : merge	㉰ : [df1,df2]
㉱ : ignore_index	㉲ : True	

① ㉮ ② ㉯ ③ ㉰
④ ㉱ ⑤ ㉲

• 옳은 코드는 다음과 같다.

```
>>> res2 = pd.concat([df1,df2], ignore_index = False)
```

정답 : ②, ⑤

07 서울시는 결재문서, 정책연구보고서 등 시가 생산한 약 2000만 건 이상의 주요 행정정보 목록을 깃허브에 공개하고 있다. 여기서는 깃허브에 올린 2022년 2월 업무추진비 내역을 csv로 받아 Benford 분석을 하려고 한다. 2022년 2월 업무추진비 내역은 아래의 링크를 누르면 다운로드 받을 수 있다.

202202_expense_list.csv

상기 파일을 데이터프레임으로 불러온 후의 테이블 구조는 다음과 같다.

칼럼명	데이터 타입	설명
nid	int64	업무추진 거래 id
title	object	업무명
url	object	url 주소
dept_nm_lvl_1	object	사용부서 레벨 1
dept_nm_lvl_2	object	사용부서 레벨 2
dept_nm_lvl_3	object	사용부서 레벨 3
dept_nm_lvl_4	object	사용부서 레벨 4
dept_nm_lvl_5	object	사용부서 레벨 5
exec_yr	int64	사용년도
exec_month	int64	사용월
expense_budget	int64	비용예산
expense_execution	int64	비용실행
category	object	범주
dept_nm_full	object	사용부서 full_name
exec_dt	object	사용일시
exec_loc	object	사용장소
exec_purpose	object	사용목적
target_nm	object	사용자 및 인원
payment_method	object	결제방법
exec_amount	int64	사용금액

사용금액의 앞의 2단위(예: 사용금액 12345이면 12를 의미함)에 대해서 벤포드 분석을 하려고 한다. 앞의 2단위의 벤포드 평균이 Expected라고 가정할 경우 실제 발견된 비율(Found Distribution)의 Z값이 10을 초과하는 항목의 개수는 몇 개인가? 예를 들어 앞의 2단위가 11, 40이며 이것의 Z값이 10을 초과한다면 2개가 된다. (4점)

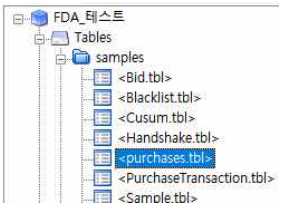
① 1 ② 2 ③ 3
④ 4 ⑤ 5

해설

	상세분석	First_2_Dig	Counts	Found	Expected	Z_score
0	0	50	210	0.0360	0.0086	22.5748
1	0	10	475	0.0813	0.0414	15.2949
2	0	90	91	0.0156	0.0048	11.8315

정답 ③

08 당신은 ㈜한공의 매입의 적정성을 테스트하기 위하여 MUS 샘플링을 수행하려고 한다. 관련 매입 자료는 `purchases.tbl`이다. purchases.tbl은 Fraudit 교육용 버전을 설치하면 Project Folder >> samples folder가 생기고 그 안에 있다.



MUS 샘플링을 위한 파라미터 설정은 다음과 같다.

구분	설명
모집단의 화폐금액(Population)	금액 칼럼의 합계액
허용왜곡표시(Tolerable Error)	5%
예상오차율(Expected Error)	0.5%
Confidence Level	99%

Random starting point는 10000을 입력하고 샘플을 추출한다.

한편, 샘플에 포함된 다음의 송장번호의 금액과 입증감사후 금액(금액_Audit)의 내용은 다음과 같아서 금액_Audit의 내용을 다음과 같이 수정한다.

송장번호	일자	거래처	거래처코드	제품코드	금액	금액_Audit
2	2009-01-01	Vijay	OJQ26	GNR	85351.00	70000
42	2009-01-11	Suzie	UOF02	CQZ	57975.00	40000
87	2009-01-09	Vijay	OVJ22	GLH	78402.00	60000

Fraudit을 이용하여 샘플 추출한 후 이를 분석한 결과가 다음과 같았다.

Reject the population as materially misstated

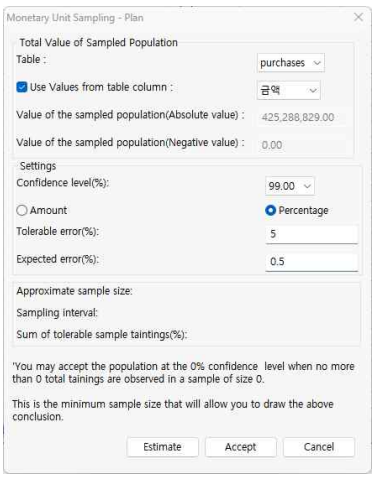
population amount(425,288,829.0) × Tolerable error(5.0%) = 21,264,441.0
UpperErrorLimit : ㉮
21,264,441.0 < ㉮

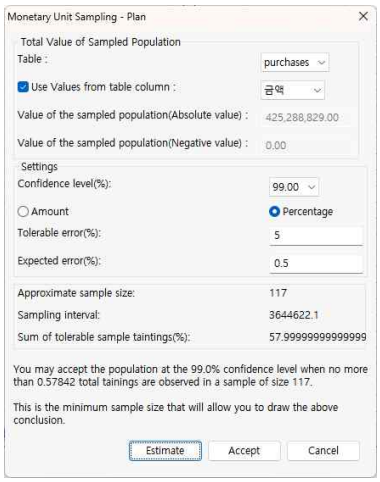
빈칸 ㉮에 들어갈 금액으로 옳은 것은? (4점)

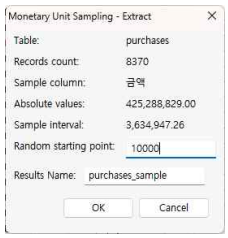
① 21,369,238.1 ② 21,564,561.9 ③ 22,784,816.4
④ 22,964,381.3 ⑤ 23,148,473.2

해설

• Sampling >> MUS(PPS) >> Plan, Extract 메뉴를 누른 후 다음과 같이 설정한 후 Estimate 버튼을 누르면 샘플 개수, 인터벌이 생성된다. 그후 Accept 버튼을 누르고 Random starting point에 10000을 입력한 후 OK 버튼을 누른다.







• 다음과 같이 금액_Audit 칼럼을 수정한 후 Sampling >> MUS(PPS) >> Evaluation 메뉴를 누른다.

Filter Expression: Set [Count : 3, Sum : 170000.]

	Row	송장번호	일자	거래처	거래처코드	제품코드	금액	금액_Audit
0	0	2	2009-01-01	Vijay	OJQ26	GNR	85351.00	70000.
1	75	42	2009-01-11	Suzie	UOF02	CQZ	57975.00	40000.
2	144	87	2009-01-09	Vijay	OVJ22	GLH	78402.00	60000.
3	208	127	2009-01-02	Vijay	OUW27	GNR	89856.00	89856.00

정답 ②

09 다음은 풋옵션과 관련된 내용이다.

• 아메리칸 풋옵션	• 현재주가 : 100	• 행사가격 : 100
• 무위험이자율 : 6%	• 배당률 : 0%	• 만기 : 5년
• 변동성 : 20%	• 스텝(노드) : 200	• dt = 만기/스텝(노드)

풋옵션 가격을 소수점 4째 자리까지 반올림한 금액을 고르시오(예 : 0.12345→0.1235). 단, 소수점 처리를 위해서 코드 상단에 np.set_printoptions(precision=4, suppress =True)를 입력해서 처리하라. (4점)

① 8.9765 ② 9.1247 ③ 9.3726
④ 9.8602 ⑤ 9.9285

해설

• 마지막 옵션 가격 노드는 다음과 같이 나온다.

```
[[ 8.9765  7.9223  6.9715 ...  0.      0.      0.    ]
 [ 0.     10.1276  8.9597 ...  0.      0.      0.    ]
```

```
[ 0.       0.      11.4032 ...  0.       0.       0.     ]
...
[ 0.       0.       0.      ... 99.8091 99.803  99.7967]
[ 0.       0.       0.      ...  0.     99.8151 99.8091]
[ 0.       0.       0.      ...  0.      0.     99.8208]]
```

정답 : ①

10 다음 코드를 실행한 결과는 아래와 같다. (3점)

```
>>> import numpy as np
>>> a = np.array([1, 2, 3])
>>> b = np.array([4, 5])
>>> c = np.array([[1, 2, 3], [4, 5, 6]])
>>> print(c + np.reshape(㉮, (㉯, ㉰)))
[[ 5  6  7]
 [ 9 10 11]]
```

상기에서 ㉮, ㉯, ㉰에 들어갈 내용을 토대로 '㉮'+'㉯'+'㉰'의 결과(print 함수를 사용하지 않는다)를 입력하라. 예를 들어, ㉮에 들어갈 내용이 A이고 ㉯에 들어갈 내용이 B라면 '㉮'+'㉯'의 결과는 'AB'가 된다.

해설

• 옳은 코드는 다음과 같다.

```
print(c + np.reshape(b, (2, 1)))
```

정답 'b21'

11 print() 함수의 결과가 아래와 같다.

```
>>> import numpy as np
>>> a = np.array([89, 34, 56, 87, 90, 23, 45, 12, 65, 78, 9, 34,
...               12, 11, 2, 65, 78, 82,28, 78, 33, 50, 19, 70,
...               92, 44, 67, 29, 49, 36, 80, 40, 18, 86, 22, 59,
...                53, 16, 27, 61])
>>> b = np.㉮(㉯, bins = ㉰)
>>> print(b)
(array([㉱, 10,  9, ㉲], dtype=int64), array([ 2. , 24.5, 47. , 69.5, 92. ]))
```

상기에서 ㉮, ㉯, ㉰, ㉱, ㉲에 들어갈 내용을 토대로 '㉮'+'㉯'+'㉰'+'㉱'+'㉲'의 결과(print 함수를 사용하지 않는다)를 입력하라. 예를 들어 ㉮에 들어갈 내용이 A이고 ㉯에 들어갈 내용이 B라면 '㉮'+'㉯'의 결과는 'AB'가 된다. (3점)

해설

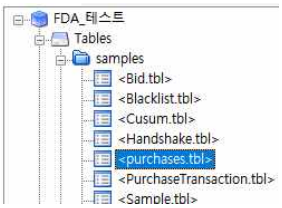
• 옳은 코드는 다음과 같다.

```
>>> b = np.histogram(a, bins = 4)
(array([10, 10,  9, 11], dtype=int64), array([ 2. , 24.5, 47. , 69.5, 92. ]))
>>> 'histogram'+'a'+'4'+'10'+'11'
'histograma41011'
```

정답 'histograma41011'

12~14

samples 폴더에 있는 `purchases.tbl`을 연다. `purchases.tbl`은 Fraudit 교육용 버전을 설치하면 Project Folder >> samples folder가 생기고 그 안에 있다.



그리고 Fraudit shell에서 다음과 같이 `purchases.tbl`을 데이터프레임으로 변환하고자 한다.

```
>>> import pandas as pd
>>> from pandas import *
>>> df = ㉮
```

12 ㉮ 에 들어갈 코드를 입력하라.(1점)

해설

• df = purchases.toDF()

정답 : purchases.toDF()

13 다음과 같이 df 데이터프레임을 피봇테이블로 나타내고자 한다.

```
>>> import pandas as pd
>>> from pandas import *
>>> pd.㉮(df, index = ㉯)
                    금액
송장번호 거래처
2    Vijay  65311.666667
3    Vijay  23756.000000
4    Adam   24162.000000
5    Adam   38250.400000
6    Suzie  17105.000000
                ...
4993 Vijay  72904.000000
4994 Vijay  55403.000000
4997 Adam    8343.000000
4998 Vijay  69742.500000
4999 Suzie  59376.000000
[4069 rows x 1 columns]
```

㉯ 에 들어가야할 코드를 입력하라. (2점)

해설

• 옳은 코드는 다음과 같다.

```
>>> pd.pivot_table(df, index = ['송장번호', '거래처'])
```

정답 : ['송장번호', '거래처']

14 다음과 같이 피봇테이블을 나타내고자 한다.

```
>>> import pandas as pd
>>> from pandas import *
>>> a = pd.pivot_table(df, ㉮ = [㉯],
...                    values = '금액',
...                    ㉰ ='제품코드',
...                    ㉱ = [㉲,'count'],
...                    ㉳ = 0,
...                    ㉴ = True)
>>> print(a)
                sum                                ... count
제품코드              COK      CQZ      DEG      EVR  ...   YEQ  ZIE  ZVX   All
송장번호 거래처                                        ...
2    Vijay        0        0        0        0  ...     0    0    0     3
3    Vijay     1215        0        0        0  ...     0    0    0     2
4    Adam         0        0        0        0  ...     0    0    0     1
5    Adam         0        0        0        0  ...     1    0    0     5
6    Suzie        0        0        0    13098  ...     0    0    0     2
...             ...      ...      ...      ...  ...   ...  ...  ...   ...
4994 Vijay        0        0        0        0  ...     0    0    0     1
4997 Adam         0        0        0        0  ...     0    0    0     1
4998 Vijay        0        0        0        0  ...     0    0    0     2
4999 Suzie        0        0        0        0  ...     0    0    0     1
All        27115268  28665251  27959855  28946037  ...   581  555  559  8370

[4070 rows x 32 columns]
```

상기에서 ㉯, ㉱, ㉳에 들어갈 내용을 토대로 '㉯'+'㉱'+'㉳'의 결과(print 함수를 사용하지 않는다)를 입력하라. 예를 들어, ㉯에 들어갈 내용이 A이고 ㉱에 들어갈 내용이 B라면 '㉯'+'㉱'의 결과는 'AB'가 된다. (3점)

해설

• 옳은 코드는 다음과 같다.

```
>>> a = pd.pivot_table(df, index = ['송장번호','거래처'],
...                    values = '금액',
...                    columns ='제품코드',
...                    aggfunc =['sum','count'],
...                    fill_value = 0,
...                    margins = True)
```

정답 : 'columnsaggfuncfill_value'

[분개장분석]

15~22

당신은 (주)한공의 외부감사인이다. (주)한공의 전산팀에 의뢰하여 차변과 대변합계가 일치하는 (주)한공의 분개장인 『분개장_2.csv』 파일을 받았다. 이는 아래의 링크를 누르면 다운로드 받을 수 있다.

『분개장_2.csv』

『분개장_2.csv』 데이터 파일의 구조는 다음과 같다.

칼럼명	데이터 타입	설명
전표일자	String	전표일자
전표번호	String	전표번호
전표라인번호	String	전표라인번호 : 하나의 전표세트에서 라인의 번호
계정코드	String	계정코드
계정과목	String	계정과목
차변금액	String	차변금액
대변금액	String	대변금액
거래처코드	String	거래처코드
입력사원	String	입력사원코드
입력일자	String	입력일자와 시간

『분개장_2.csv』 의 레코드는 10000개를 초과하기 때문에 Fraudit 교육용 버전의 File >> Import >> Text File or Excel Spreadsheet 메뉴를 사용해서는 안되고 **pandas를 이용하여 데이터프레임으로 불러와야 한다.**

동일한 전표일자와 동일한 전표번호를 가지는 세트를 하나의 전표세트라고 한다. 예를 들어, 아래의 네모 선 안에 있는 것이 하나의 전표세트이며 이 전표세트의 레코드수는 3개이다.

전표일자	전표번호	전표라인번호	계정코드	계정과목	차변금액	대변금액	거래처코드	입력사원	입력일자
20170101	1	1	12002	매출채권미수금(카드)	132000	0	80413	260	2017-01-06 오후 3:22:38
20170101	1	2	10800	외상매출금	0	66000	12282	260	2017-01-06 오후 3:22:38
20170101	1	3	10800	외상매출금	0	66000	9660	260	2017-01-06 오후 3:22:38
20170101	46	1	10800	외상매출금	332090	0	11777	517	2017-02-28 오전 11:30:04
20170101	46	2	40401	제품매출	0	301900	11777	517	2017-02-28 오전 11:30:04
20170101	46	3	25500	부가세예수금	0	30190	11777	517	2017-02-28 오전 11:30:04

다음은 계정과목과 간주계정과목의 관계를 나타낸 표이다.

계정코드	계정과목	간주계정과목	계정코드	계정과목	간주계정과목
10100	현금	현금	10200	당좌예금	보통예금
10301	보통예금	보통예금	10800	외상매출금	매출채권
11000	받을어음	매출채권	11300	진행률미수금	미수금
12001	매출채권미수금(일반)	매출채권	12002	매출채권미수금(카드)	매출채권
12300	미수금	미수금	13100	선급금	선급금
13500	부가세대급금	부가세대급금	14600	상품	재고자산
14700	제품	재고자산	14900	원자재	재고자산
15700	재공품	재고자산	15800	미완성공사(도급)	재고자산
25100	외상매입금	매입채무	25301	미지급금(일반)	미지급금
25303	미지급금(4대보험)	미지급금	25500	부가세예수금	부가세예수금
25900	선수금	선수금	31400	장기미지급금	미지급금
40100	상품매출	매출	40401	제품매출	매출
40700	교육수입	매출	41100	임대수입	매출
41200	기타수입	매출	45100	상품매출원가	매출원가
45500	제품매출원가	매출원가	46100	교육수입원가	매출원가
46200	임대수입원가	매출원가			

15 전표세트에서 **간주계정과목이 '현금'인 계정과목이 차변이면서 해당 차변금액이 0을 초과하고, 간주계정과목이 '매출'인 계정과목이 대변이면서 해당 대변금액이 0을 초과하는 경우를** 포함하는 전표세트를 현금매출 전표세트라고 가정한다. 현금매출 전표세트의 개수를 숫자로만 나타내라. 단, 천단위 콤마는 나타내지 않는다. (4점)

해설

	상세분석	전표_인덱스	현금_차변빈도	매출액_대변빈도	차변금액	대변금액
0	1	823	2	2	80	80
1	1	6616	1	1	394440	394440
2	1	6863	2	2	80	80
3	1	27152	2	1	10000	10000
4	1	47305	2	1	10000	10000

```
import pandas as pd
from pandas import *
#01. 현금_계정코드
현금_계정코드 = [10100]
#02. 매출_계정코드
매출_계정코드 = [40100, 40401, 40700, 41100, 41200]
#03. 현금_차변빈도, 매출액_대변빈도 칼럼 삽입
df_merged['현금_차변빈도'] = list(map(lambda x : 1 if x[0] in 현금_계정코드 and x[1] > else 0,zip(df_merged['계정코드'], df_merged['차변금액'])))
df_merged['매출액_대변빈도'] = list(map(lambda x : 1 if x[0] in 매출_계정코드 and x[1] > else 0, zip(df_merged['계정코드'], df_merged['대변금액'])))
#04. groupby : 전표번호, 전표일자
gb_전표_인덱스 = df_merged.groupby(['전표일자', '전표번호'])
# 거래처_전표_요약
df_현금_매출_총괄 =  DataFrame(
                    {'현금_차변빈도' : gb_전표_인덱스['현금_차변빈도'].sum(),
                    '매출액_대변빈도' : gb_전표_인덱스['매출액_대변빈도'].sum(),
                    '차변금액' : gb_전표_인덱스['차변금액'].sum(),
                    '대변금액' : gb_전표_인덱스['대변금액'].sum(),
                    }).reset_index()
df_현금_매출_총괄 = df_현금_매출_총괄[(df_현금_매출_총괄['현금_차변빈도'] >0) & (df_현금_
매출_총괄['매출액_대변빈도'] >0)]
# 상세분석 칼럼 삽입
len(df_현금_매출_총괄)
```

정답 : 5

16 정상적인 매출거래 전표세트의 정의는 다음과 같다.

① 계정과목이 ['상품매출', '제품매출','교육수입', '임대수입', '기타수입'] 중 하나 이상이며, 대변금액이 0을 초과하거나 차변금액이 0 미만인 경우가 포함된다.
② 계정과목이 ['현금', '당좌예금','보통예금','외상매출금', '받을어음', '진행률미수금', '매출채권미수금(일반)', '매출채권미수금(카드)', '미수금', '선수금'] 중 하나 이상이며, 차변금액이 0을 초과하거나 대변금액이 0미만 경우가 포함된다.

따라서, 정상적인 매출거래가 아닌 전표세트의 정의는 다음과 같다.

① 간주계정과목이 매출이면서, 대변금액이 0을 초과하거나 차변금액이 0 미만인 경우가 포함된다.
② 나머지 계정과목이 ['현금', '당좌예금','보통예금','외상매출금 '받을어음', '진행률미수금', '매출채권미수금(일반)', '매출채권미수금(카드)', '미수금', '선수금'] 이 아닌 항목들로만 나타나는 경우이다.

다음은 정상적인 매출거래와 관련된 **항목**만을 발췌한 표이다.

계정코드	계정과목	간주계정과목	계정코드	계정과목	간주계정과목
10100	현금	**현금**	10200	당좌예금	**보통예금**
10301	보통예금	**보통예금**	10800	외상매출금	**매출채권**
11000	받을어음	**매출채권**	11300	진행률미수금	**미수금**
12001	매출채권미수금(일반)	**매출채권**	12002	매출채권미수금(카드)	**매출채권**
12300	미수금	**미수금**	25900	선수금	**선수금**
40100	상품매출	**매출**	40401	제품매출	**매출**
40700	교육수입	**매출**	41100	임대수입	**매출**
41200	기타수입	**매출**			

정상적인 매출거래가 아닌 전표세트의 개수를 숫자로만 나타내라. 단, 천단위 콤마는 나타내지 않는다. (4점)

해설

```
import pandas as pd
from pandas import *
매출_관련계정코드 = [40100, 40401, 40700, 41100, 41200]
매출상대_관련계정코드 = [10100, 10200, 10301, 10800, 11000, 11300, 12001,
                        12002, 12300, 25900]
# 매출_대변빈도, 매출상대_차변빈도 칼럼 삽입
df_merged['매출상대_차변빈도'] = list(map(lambda x : 1 if x[0] in 매출상대_관련계정코드
and (x[1] >0 or x[2] <0) else 0, zip(df_merged['계정코드'], df_merged['차변금액'],
df_merged['대변금액'])))
df_merged['매출_대변빈도'] = list(map(lambda x : 1 if x[0] in 매출_관련계정코드 and (x[1]
<0 or x[2] >0) else 0, zip(df_merged['계정코드'], df_merged['차변금액'], df_merged['대변
금액'])))
#04. groupby : 전표번호, 전표일자
gb_전표_인덱스 = df_merged.groupby(['전표일자', '전표번호'])
#05. summarize 테이블
df_매출_관련계정_총괄 = DataFrame({
          '매출상대_차변빈도' : gb_전표_인덱스['매출상대_차변빈도'].sum(),
          '매출_대변빈도' : gb_전표_인덱스['매출_대변빈도'].sum(),
          '차변합계' : gb_전표_인덱스['차변금액'].sum(),
          '대변합계' : gb_전표_인덱스['대변금액'].sum(),
                    }
                    ).reset_index()
```

```
df_매출_관련계정_총괄 = df_매출_관련계정_총괄[ (df_매출_관련계정_총괄['매출상대_차변빈도']
==0) & (df_매출_관련계정_총괄['매출_대변빈도'] >0) ]
```

	전표일자	전표번호	매출상대_차변빈도	매출_대변빈도	차변합계	대변합계
95	20170102	3000	0	1	3000000	3000000
6620	20170202	3000	0	1	40000000	40000000
13012	20170302	3000	0	1	50000000	50000000
19728	20170402	3000	0	1	60000000	60000000

정답 4

17 입력일자 칼럼의 시간이 오후 7시 이상부터 오전 6시 미만까지인 레코드의 개수를 숫자로만 입력하라. 단, 입력일자 칼럼의 레코드가 없거나, None, NaN, nan인 경우는 레코드 개수에 포함하지 않는다. (4점)

해설

```
import re
# 입력시간 추출 함수
def extract_hour(date_string):
    pattern = r'(오전|오후) ₩d+'
    match = re.search(pattern, date_string)
    if match:
        return match.group()
    else:
        return None
# '입력시간' 칼럼 생성
df_merged['입력시간'] = df_merged['입력일자'].apply(extract_hour)
# 오후8시 이상에서 ~ 오전6시 미만 까지 입력된 항목을 1로 처리한다. 단, nan은 0으로 처리
한다.
입력시간 = ['오후 7', '오후 8', '오후 9','오후 10','오후 11',
        '오전 12', '오전 1','오전 2','오전 3','오전 4','오전 5']
df_merged['야간입력'] = [1 if i in 입력시간 else 0 for i in df_merged['입력시간']]
df_야간입력 = df_merged[df_merged['야간입력'] ==1]
len(df_야간입력)
```

정답 : 6868

18 전표세트를 가장 많이 입력한 입력사원이 입력한 전표세트의 개수를 숫자만 입력하라. 단, 천단위 콤마는 나타내지 않는다. (4점)

해설

```
# groupby : 전표번호, 전표일자
gb_일자_번호_입력사원 = df_merged.groupby(['전표일자', '전표번호', '입력사원'])
# summarize 테이블
df_요약_1 = DataFrame({'빈도' : gb_일자_번호_입력사원['입력사원'].nunique(),
                       }
                       ).reset_index()
# groupby : 전표번호, 전표일자
gb_입력사원 = df_요약_1.groupby(['입력사원'])
df_요약_2 = DataFrame({'전표세트빈도' : gb_입력사원['빈도'].sum(),
                       }
                       ).reset_index()
```

	입력사원	전표세트빈도
0	0	238

```
1     260   16013
2     361    4392
3     498     519
4     517   59492
5    1181       2
6    1389       2
7    1446      10
8    1456       1
9    1483       8
10  10450       2
```

정답 : 59492

19 당신은 ㈜한공의 분개패턴을 분석하고자 한다. 이를 위해 분개장_2.csv를 pandas 라이브러리를 이용하여 데이터프레임으로 불러온 후 **계정코드_1** 칼럼을 다음과 같이 삽입한다. 여기서는 데이터프레임명을 'df_전체'라고 가정한다(여러분들은 자신이 설정한 데이터프레임명으로 적절하게 수정할 수 있다).

```
>>> df_전체['계정코드_1'] = [ i[2] if i[0] > 0 and i[1] == 0
...                     else(-i[2] if i[0] < 0 and i[1] == 0
...                     else(-i[2] if i[0] == 0 and i[1] > 0
...                     else( i[2] if i[0] == 0 and i[1] < 0
...                     else( i[2] if i[0] == 0 and i[1] == 0
...                     else 0))))
...                     for i in zip(df_전체['차변금액'],
...                     df_전체['대변금액'], df_전체['계정코드'])
...                                                          ]
```

분개패턴은 **하나의 전표세트**에서 **중복을 제외한 계정코드_1의 합계와 중복을 제외한 계정코드_1의 표준편차의 조합**으로 구분한다고 가정한다. **중복을 제외하는 이유**는 하나의 전표세트에서 **여러 거래처에 대하여 중복되는 계정코드_1이 나오는 경우가 많기 때문에 이러한 영향을 제외하기 위한 것**이다. 분개패턴을 기준으로 분개장에서 발생빈도가 1(즉, 1년 동안 발생한 빈도가 1)인 전표세트의 갯수를 숫자만 나타내라. 단, 천단위 콤마는 나타내지 않는다. (4점)

해설

```
import pandas as pd
from pandas import *
#계정코드_1 칼럼 삽입
df_merged['계정코드_1'] = [ i[2] if i[0]>0 and i[1]==0
          else( -i[2] if i[0]<0 and i[1] ==0
          else( -i[2] if i[0] ==0 and i[1] >0
          else( i[2] if i[0] ==0 and i[1] <0
          else( i[2] if i[0] ==0 and i[1] ==0
          else 0))))
          for i in zip(df_merged['차변금액'], df_merged['대변금액'], df_merged['계정코드'])
            ]
#groupby : 전표번호, 전표일자
gb_전표번호_일자 = df_merged.sort_values(['전표일자','전표번호']).groupby(['전표일자','전표번호'])
```

```
#칼럼 삽입
df_merged['계정코드_1_합계_중복제외'] = gb_전표번호_일자['계정코드_1'].transform(lambda x: x.unique().sum())
df_merged['계정코드_1_표준편차_중복제외'] = gb_전표번호_일자['계정코드_1'].transform(lambda x: x.unique().std())

df_merged['계정코드_1_개수_중복제외'] = gb_전표번호_일자['계정코드_1'].transform('nunique')

df_merged['차변합계'] = gb_전표번호_일자['차변금액'].transform("sum")
df_merged['차변Max'] = gb_전표번호_일자['차변금액'].transform("max")
# groupby : 계정코드1합계_중복제외,계정코드1표준편차_중복제외
gb_계정코드_1_중복제외 = df_merged.groupby(['계정코드_1_합계_중복제외','계정코드_1_표준편차_중복제외'])
#인덱스 칼럼 삽입
df_merged['계정코드_1_중복제외_index'] = gb_계정코드_1_중복제외.ngroup()
#09. df_분개패턴 summarize 생성
gb_전표_인덱스 = df_merged.groupby(['전표일자','전표번호'])
df_merged['전표_인덱스'] = gb_전표_인덱스.ngroup()
df_분개패턴1 = DataFrame(
 {'계정코드_1_중복제외_레코드' : gb_계정코드_1_중복제외['계정코드_1_개수_중복제외'].max(),
  '계정코드_1_전표세트_중복제외_빈도' : gb_계정코드_1_중복제외['전표_인덱스'].nunique(),
  '입력사원_중복제외_빈도' : gb_계정코드_1_중복제외['입력사원'].nunique(),
  '입력사원_중복제외' : gb_계정코드_1_중복제외['입력사원'].unique(),
  '차변합계' : gb_계정코드_1_중복제외['차변금액'].sum(),
  '차변Max' : gb_계정코드_1_중복제외['차변금액'].max()
   }).reset_index()
len(df_분개패턴1[df_분개패턴1['계정코드_1_전표세트_중복제외_빈도'] ==1])
```

정답 326

20 동일한 거래처에 대한 거래금액 중 가장 큰 금액과 두 번째로 큰 금액과의 비율을 검토하게 되면 가장 큰 금액이 어느 정도 예외적으로 큰 금액인지를 직관적으로 가늠할 수 있다. 예를 들어, 어떤 거래처에 대한 매출채권의 두 번째 큰 금액이 1,000,000인데 가장 큰 금액이 100,000,000이라고 가정하자. 통상적으로 매출 거래처에 대한 거래금액의 급격한 변화는 매우 특이한 상황(경우에 따라서는 부정 등의 가능성이 있을 수 있음)이라고 볼 수 있다. 이러한 테스트를 **Relative Size Factor(RSF) 테스트**라고 한다. **거래처별**로 **외상매출금계정과목 차변의 「가장 큰 금액/두번째로 큰 금액」 인 RSF가 100을 초과하는 거래처의 개수**를 숫자로 나타내라. 단, 천단위 콤마는 나타내지 않는다. **컴퓨터의 성능에 따라 코드를 처리하는데 1~3분 정도 걸릴 수 있다.** (4점)

해설

```
import pandas as pd
from pandas import *
# 인덱스 칼럼 삽입
df_merged['S_index'] = gb_거래처_계정과목.ngroup()
# Max, Second Max 칼럼 삽입
df_merged['차변_Max'] = gb_거래처_계정과목['차변금액'].transform(lambda x : x.nlargest(1).max())
df_merged['차변_2nd_Max'] = gb_거래처_계정과목['차변금액'].transform(lambda x : x.nlargest(2).min())
df_merged['대변_Max'] = gb_거래처_계정과목['대변금액'].transform(lambda x : x.nlargest(1).max())
df_merged['대변_2nd_Max'] = gb_거래처_계정과목['대변금액'].transform(lambda x : x.nlargest(2).min())
# 현금_차변빈도, 매출액_대변빈도 칼럼 삽입
df_merged['차변_RSF'] = list(map(lambda x : x[0]/x[1] if x[1] !=0 else -1,
                        zip(df_merged['차변_Max'], df_merged['차변_2nd_Max']))
```

```
                                                )
                                                )


df_merged['대변_RSF'] = list(map(lambda x : x[0]/x[1] if x[1] !=0 else -1,
                                 zip(df_merged['대변_Max'], df_merged['대변_2nd_Max'])
                                                )
                                                )

# df_RSF
df_RSF = DataFrame(
{'차변_Max' : gb_거래처_계정과목['차변_Max'].max(),
'차변_2nd_Max' : gb_거래처_계정과목['차변_2nd_Max'].max(),
'차변_RSF' : gb_거래처_계정과목['차변_RSF'].max(),
'대변_Max' : gb_거래처_계정과목['대변_Max'].max(),
'대변_2nd_Max' : gb_거래처_계정과목['대변_2nd_Max'].max(),
'대변_RSF' : gb_거래처_계정과목['대변_RSF'].max()
}).reset_index()
df_RSF.insert(0, 'S_index', range(0, len(df_RSF)))
cols = ['S_index','거래처코드','계정과목','차변_RSF','차변_Max', '차변_2nd_Max','대변_RSF','대변_Max','대변_2nd_Max']
df_RSF = df_RSF[cols]
# 외상매출금만 발췌
df_외상매출금 = df_RSF[(df_RSF['계정과목'] =='외상매출금') & (df_RSF['차변_RSF'] >100)]
print(len(df_외상매출금))
```

정답 : 14

21 ㈜한공은 이제까지 향후 매출액을 결정론적 접근방식에 의하여 단일 평균값으로 추정하였다. 그러나 이 방법으로는 순이익 예측 모형에서의 불확실성을 제대로 반영하지 못한다는 컨설팅 의견을 받았다. ㈜한공의 재무이사는 다음 회계연도의 당기순이익(Net Incom)에 대한 몬테카를로 시뮬레이션을 실행하려고 한다. 이를 위해서 추정손익계산서를 먼저 작성하고자 한다. 양식은 [추정손익계산서.tbl]에 있다. 추정손익계산서의 금액 단위는 원이다.

추정손익계산서.tbl

[추정손익계산서.tbl]을 열면 다음과 같이 되어 있다. 아래 월별 순매출에 들어갈 금액은 [분개장_1.csv]을 pandas 라이브러리를 이용하여 불러온 데이터프레임에서 간주계정과목이 매출인 계정과목의 대변금액 합계에서 차변금액 합계를 차감한 월별 순매출이다.

	계정	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
0	순매출	2,213,609,724	2,294,452,146	3,099,637,278	2,295,492,192	2,324,502,601	2,972,654,128	2,111,991,252	2,287,331,800	2,374,479,748	2,679,100,074	3,046,800,918	5,027,566,905
1	매출원가	0	0	0	0	0	0	0	0	0	0	0	0
2	매출총이익	0	0	0	0	0	0	0	0	0	0	0	0
3	판관비	0	0	0	0	0	0	0	0	0	0	0	0
4	감가상각	0	0	0	0	0	0	0	0	0	0	0	0
5	EBIT	0	0	0	0	0	0	0	0	0	0	0	0
6	이자비용	0	0	0	0	0	0	0	0	0	0	0	0
7	EBT	0	0	0	0	0	0	0	0	0	0	0	0
8	법인세	0	0	0	0	0	0	0	0	0	0	0	0
9	순이익	0	0	0	0	0	0	0	0	0	0	0	0

한편, 과거 100개월 동안의 매출원가율은 다음의 [과거매출원가율.tbl]에 있다.

과거매출원가율.tbl

몬테카를로 시뮬레이션을 수행하기 위한 분포는 다음과 같다.

구분	분포	파라미터
월별 순매출	N(mu, sigma)	mu : 추정손익계산서의 월별 순매출 sigma : 추정손익계산서의 월별 순매출 * 0.1
월별 매출원가율	[과거매출원가율.tbl]의 자료를 distift() 함수를 이용하여 최적 분포를 찾는다.	
월별 판관비율	Tri(min, mode, max)	min : 0.06 mode : 0.07 max : 0.08

구분	수치	비고
월별 감가상각	100,000,000원	고정금액
월별 이자비용	80,000,000원	고정금액
월별 법인세	월별 EBT × 24%	변동금액
발행주식수	100,000주	발행주식수

결과 변수의 산식은 다음과 같다.

구분	산식
매출총이익	순매출 - 매출원가
EBIT	매출총이익 - 판관비 - 감가상각비
EBT	EBIT - 이자비용
순이익	EBT - 법인세

㈜한공이 주당순이익(EPS) 목표인 57,000원 이상 달성할 확률을 소수점 다섯째 자리에서 반올림하여 구하라(예: 0.12345→0.1235). 단, 시드는 np.random.seed(0)으로 고정시키며, 시뮬레이션 횟수는 10000으로 하고, 다음과 같은 라이브러리를 활용하라. (6점)

```
>>> import numpy as np
>>> import pandas as pd
>>> from pandas import *
>>> import mcerp
>>> from mcerp import *
>>> import matplotlib.pyplot as plt
>>> from scipy.stats import *
>>> from distfit import distfit
```

해설

```
# 매출원가율에 대해서 어레이 만들기
매출원가율 = np.array(과거매출원가율['매출원가율'])
# distfit() 함수를 실행시킨다.
dist = distfit()
# 기초자료_임대율의 자료에 이론적인 최적 분포 찾기
dist.fit_transform(매출원가율)
dist.plot()
plt.show()
print(dist.summary)
# 시뮬레이션 횟수 정하기
nR =10000
mcerp.npts = nR
# 시드 고정하기
np.random.seed(0)
# 입력변수의 파라미터 정하기
```

```
순매출_평균 = [ i for i in 추정손익계산서[0][1:]]
순매출_표준편차 = [ i*0.1 for i in 순매출_평균]
# 입력변수의 분포 및 고정금액 설정하기
순매출_입력분포_리스트=[N(i[0],i[1]) for i in zip(순매출_평균, 순매출_표준편차)]
매출원가율_입력분포 = uv(ss.lognorm(0.0257374,loc=0.219736,scale=0.398737))
판관비율_입력분포 = Tri(0.06, 0.07, 0.08)
월별_감가상각비 = 100_000_000
세율 = 0.24
월별_이자비용 = 80_000_000
발행주식수 = 100_000
# 결과변수의 산식 설정하기
순이익 = 0
for i in 순매출_입력분포_리스트:
    매출총이익 = i - i*매출원가율_입력분포
    EBIT = 매출총이익 - i*판관비율_입력분포 - 월별_감가상각비
    EBT = EBIT - 월별_이자비용
    순이익 = 순이익 + EBT - EBT*세율
# EPS 구하기
EPS = 순이익/발행주식수
# 가로축과 세로축의 포맷 지정
# 그래프 그리기
plt.figure("EPS")
plt.ticklabel_format(style='plain', axis='x', scilimits=(0,0))
plt.ticklabel_format(style='plain', axis='y', scilimits=(0,0))
EPS.plot()
plt.axvline(x =57000, color ='red')
plt.show()
# 확률 구하기
print(EPS >= 57000)
```

정답 : 0.8703

22~23

시내에 위치한 한 A호텔은 대대적인 리모델링을 고려하고 있으며, 수익의 평균을 극대화하기 위해 최적의 요금과 객실 크기 조합을 결정해야 한다. 현재 A호텔은 세 가지 객실 형태를 운영하고 있으며 각 객실의 총합은 450개이다. 각 객실의 가격, 일일 점유율, 가격/수요 탄력성은 다르다. 탄력성에 대한 불확실성을 고려할 때 A호텔은 각 객실 유형에 대해 최적의 가격을 설정하는 동시에 예상 수익을 극대화하기를 원한다. 현재 호텔은 다음과 같은 450개의 객실을 보유하고 있으며 기초 자료의 내용은 다음과 같다.

객실 형태	일 요금 ①	일일 평균 판매 (예약 객실 수) ②	수익 ① × ②	가격/수요 탄력성	새로운 일 요금 ③	판매될 것으로 추정되는 객실수 ④	예상수익
스탠다드	$85	250	$21,250	U(-4.5, -1.0)	$70 ~ $90	아래 공식 참조	③ × ④
골드	$98	100	$9,800	U(-1.5, -0.5)	$90 ~ $110	아래 공식 참조	③ × ④
플래티늄	$139	50	$6,950	U(-3.0, -1.0)	$120 ~ $149	아래 공식 참조	③ × ④

가격/수요 탄력성은 예를 들어, 스탠다드 객실의 가격을 1% 인하하면 판매되는 객실 수가 3% 증가하고, 가격을 1% 인상하면 판매되는 객실 수가 3% 감소하는 것을 의미한다.

각 객실형태에 대해서 판매될 것으로 추정되는 객실수는 아래의 공식을 사용하여 구할 수 있다.

$$\text{판매될 것으로 추정되는 객실 수} = H + \frac{E \times H \times (N - C)}{C}$$

H : 일일 평균 판매
E : 가격/수요 탄력성
N : 새로운 일 요금
C : 현행 일 요금

판매될 것으로 추정되는 객실수의 합이 430이하가 될 확률이 80%를 초과하여야 한다. A호텔의 모든 객실의 가격은 1달러 단위로 표시된다. A호텔은 현재 450개 객실을 초과하여 확장할 계획은 없다. A호텔은 예상수익의 합의 평균을 최대화하기 위한 각 객실 형태의 새로운 요금을 구하고자 한다.

아래의 코드를 앞 부분에 포함하기 바란다.

```
>>> import numpy as np
>>> import pandas as pd
>>> from pandas import *
>>> import mcerp
>>> from mcerp import *
>>> import scipy.stats as ss
>>> from scipy.stats import *
>>> from geneticalgorithm import geneticalgorithm as ga
>>> import matplotlib.pyplot as plt
>>> np.random.seed(0)    # seed를 0으로 고정시킨다.
>>> mcerp.npts = 10000
```

22 A호텔의 예상수익의 합의 평균을 최대화하는 각 객실형태의 새로운 요금을 구한 경우, 예상수익의 합의 평균을 소수점 첫째자리에서 반올림하여 숫자만 입력하라. (5점)

해설

```
import numpy as np
import pandas as pd
from pandas import *
import mcerp
from mcerp import *
import scipy.stats as ss
from scipy.stats import *
from geneticalgorithm import geneticalgorithm as ga
import matplotlib.pyplot as plt
np.random.seed(0)    # seed를 0으로 고정시킨다.
mcerp.npts = 10000

# 입력분포 설정
입력변수_1 = U(-4.5, -1.0)  # 균등분포
입력변수_2 = U(-1.5, -0.5)      # 균등분포
입력변수_3 = U(-3.0, -1.0)   # 균등분포
# 입력분포를 어레이화
입력변수_1 = 입력변수_1._mcpts
입력변수_2 = 입력변수_2._mcpts
입력변수_3 = 입력변수_3._mcpts
# 공식에 따른 과거 일일평균 판매수
H_1 =250
H_2 =100
H_3 =50
# 공식에 따른 현행 요금
C_1 =85
C_2 =98
C_3 =139
def f(X):
    # 전역변수 반영
    global 입력변수_1
    global 입력변수_2
    global 입력변수_3

    global H_1
    global H_2
    global H_3

    global C_1
    global C_2
    global C_3
   # 결정변수_1,2,3,4(혼합형, 인덱스혼합형, 주식형, 채권형의 투자비율) 설정
    결정변수_1 = X[0]
    결정변수_2 = X[1]
    결정변수_3 = X[2]
    판매추정객실수_1=np.round(H_1+입력변수_1*H_1*(결정변수_1-C_1)/C_1)
    판매추정객실수_2=np.round(H_2+입력변수_2*H_1*(결정변수_2 - C_2)/C_2)
    판매추정객실수_3=np.round(H_3+입력변수_3*H_1*(결정변수_3 - C_3)/C_3)
    판매추정객실수_합 = 판매추정객실수_1+ 판매추정객실수_2+판매추정객실수_3
    #제약조건 페널티
    페널티 =0
```

```
    if np.count_nonzero(판매추정객실수_합 <=430) / 판매추정객실수_합.size
<=0.8 :
        페널티 =1000000
    # 0.01을 곱하는 것은 ga() 함수의 파라미터 중 variable_type이 정수이므
로 1% 변동 반영
    결과변수 = 결정변수_1 * 판매추정객실수_1 + 결정변수_2 * 판매추정객실수
_2 + 결정변수_3 * 판매추정객실수_3 - 페널티
    return -np.mean(결과변수)    # 최대화이므로 -를 반영한다.
결정변수_범위 = np.array([[70,90],
                    [90,110],
                    [120,149]])
model = ga(function = f,
           dimension =3,
           variable_type ='int',  # 변동단위가 1%이므로 정수처리를 한다.
           variable_boundaries = 결정변수_범위)
model.run()
```

The best solution found:
[90. 101. 123.]

Objective function:
−41417.6337

정답 : 41418

23 예상수익의 합의 평균이 $39,000 이상일 확률을 소수점 넷째 자리에서 반올림하여 구하라(예: 0.1234→0.123). (5점)

해설

```
model = ga(function = f,
           dimension =3,
           variable_type ='int',  # 변동단위가 1%이므로 정수처리를 한다.
           variable_boundaries = 결정변수_범위)
결정변수_1, 결정변수_2, 결정변수_3 =90, 101, 123
결과변수 = 결정변수_1 * (H_1 + 입력변수_1*H_1*(결정변수_1 - C_1)/C_1) \
         + 결정변수_2 * (H_2 + 입력변수_2*H_1*(결정변수_2 - C_2)/C_2) \
         + 결정변수_3 * (H_3 + 입력변수_3*H_1*(결정변수_3 - C_3)/C_3) \
plt.figure('결과변수')
plt.hist(결과변수, bins =100)
특정값 =39000
plt.axvline(x = 특정값, color ='red')
plt.show()
# 기대수익률이 0보다 클 확률을 구해야 하므로 0을 특정값으로 지정한다.
# count와 bins를 구한다.
count, bins_count = np.histogram(결과변수,
                  bins = [np.min(결과변수), 특정값 , np.max(결과변수)])
# pdf(Probability Density Function : 확률밀도함수)를 구한다.
pdf = count / sum(count)
print(pdf)
```

[0.1765 0.8235]

정답 : 0.824

24~27

B은행은 최근 대규모의 전세자금 대출사기 사건이 발생하였다. 대출사기에 동원된 수법은 여러 사람이 여러

사람과 가공의 전세계약을 체결하게 한 후 전세자금대출을 받고 잠적하는 것이었다. B은행의 내부감사인은 전세자금대출에 필요한 전세계약서를 데이터베이스화하여 가공의 전세계약을 파악하려고 한다. 아래는 전세계약을 데이터베이스화한 파일이다.

전세계약.tbl

전세계약.tbl의 구조는 다음과 같다.

칼럼명	데이터 타입	설명
임대차_계약번호	Integer	임대차 계약번호 시퀀스 번호
대출종류	String	대출종류
건전성	String	건전성
담보여부	String	담보여부
대출유형	String	대출유형
임대인	String	임대인 명
임대인_주민번호	String	임대인 주민등록번호
임차인	String	임차인 명
임차인_주민번호	String	임차인 주민등록번호
전세금	Integer	전세계약금

24 임대인과 임차인이 다음의 관계에 있으면 임대인과 임차인 모두 가공의 전세계약 공모 가능성이 높다고 본다.

(예 1) : A, B는 공모가능성이 높음

임대인	임대인_주민번호	임차인	임차인_주민번호
A	1	B	2
B	2	A	1

(예 2) : A, B, C는 공모가능성이 높음

임대인	임대인_주민번호	임차인	임차인_주민번호
A	1	B	2
B	2	C	3
C	3	A	1

(예 3) : A, B, C, D는 공모가능성이 높음

임대인	임대인_주민번호	임차인	임차인_주민번호
A	1	B	2
B	2	C	3
C	3	D	4
D	4	A	1

이와 같은 관계를 가지는 임대인과 임차인의 집단(상기 예 3에서 A,B,C,D)을 하나의 사이클 집단이라고 한다. 이러한 사이클 집단이 몇개가 있는지 그 숫자만 입력하라. 단, 이러한 거래를 찾기 위해서는 networkx 라이브러리의 simple_cycles() 함수를 이용할 수 있다. (3점)

해설

```
import pandas as pd
import networkx as nx
df = 전세계약.toDF()
DG = nx.from_pandas_edgelist(df, '임대인_주민번호', '임차인_주민번호',
create_using=nx.DiGraph())
def 사이클_찾기(graph):
    cycles = nx.simple_cycles(graph)
    return cycles
for cycle in 사이클_찾기(DG):
    print(cycle)
```

```
['020706-2769537', '260509-1759390']
['880600-2210932', '600620-2630320']
['700215-2781570', '900022-1239764']
['680508-2017443', '480610-1544490', '360115-2185327']
['680508-2017443', '480610-1544490', '680224-1835367', '290629-1057647']
['740329-2501538', '360902-1791124']
['020111-1584587', '360902-1791124']
['920611-1170276', '360902-1791124']
```

정답 : 8

25 하나의 임대인이 여러명의 임차인과 관련이 있는 경우 그러한 임대인이 몇 명인지 그 숫자만 입력하라. 단, 이러한 거래를 찾기 위해서 networkx 라이브러리의 graph.out_degree() 함수와 graph.neighbors() 함수를 이용할 수 있다. (3점)

해설

```
def 하나의_임대인_여러명_임차인(graph):
    임대인= [node for node in graph.nodes if node in df['임대인_주민번호'].values]
    return [i for i in 임대인 if graph.out_degree(i) >1]
print(하나의_임대인_여러명_임차인(DG))
```

```
['020706-2769537', '260509-1759390']
['880600-2210932', '600620-2630320']
['700215-2781570', '900022-1239764']
['680508-2017443', '480610-1544490', '360115-2185327']
['680508-2017443', '480610-1544490', '680224-1835367', '290629-1057647']
['740329-2501538', '360902-1791124']
['020111-1584587', '360902-1791124']
['920611-1170276', '360902-1791124']
```

정답 : 23

26 여러명의 임대인이 하나의 임차인과 관련이 있는 경우 그러한 임차인이 몇 명인지 그 숫자만 입력하라. 단, 이러한 거래를 찾기 위해서 networkx 라이브러리의 graph.in_degree() 함수와 graph.predecessors() 함수를 이용할 수 있다. (3점)

해설

```
def 여러명_임대인_하나의_임차인(graph):
    임차인 =[node for node in graph.nodes if node in df['임차인_주민번호'].values]
    return [i for i in 임차인 if graph.in_degree(i) >1]
print(여러명_임대인_하나의_임차인(DG))
```

```
['360902-1791124', '680508-2017443', '500022-2931534', '680224-1835367',
'290629-1057647', '360115-2185327']
```

정답 : 6

27 하나의 임대인이 다른 계약의 임차인이 되는 경우 그러한 임대인이 몇 명인지 그 숫자만 입력하라. (3점)

해설

```
def 하나의_임대인_다른계약_임차인(graph):
    임대인 =[node for node in graph.nodes if node in df['임대인_주민번호'].values]
    return [i for i in 임대인 if i in df['임차인_주민번호'].values]
print(하나의_임대인_다른계약_임차인(DG))
```

```
['360902-1791124',    '680508-2017443',    '480610-1544490',    '700215-2781570',
'240515-2332071',    '240411-1473558',    '880600-2210932',    '990103-1169215',
'020111-1584587',    '900022-1239764',    '740329-2501538',    '920611-1170276',
'670113-2918486',    '020706-2769537',    '260509-1759390',    '600620-2630320',
'940616-2261613',    '770919-1430484',    '810207-1097459',    '050406-2738073',
'760704-1298033',    '680224-1835367',    '290629-1057647',    '360115-2185327']
```

정답 : 24

28~29

당신은 적요란에 포함된 데이터를 분석하여 지역별, 연령별 연봉금액을 비교하려고 한다. 적요란에 포함된 데이터만 추출하여 csv 파일로 변환한 것이 『적요.csv』 파일이며 이는 아래의 링크를 누르면 다운로드 받을 수 있다.

『적요.csv』

『적요.csv』 데이터의 첫 번째와 두 번째 레코드는 다음과 같다.

첫 번째 레코드	yejun83@naver.com, 920524-8131112, 052-038-7998, 세종특별자치시 노원구 삼성922로, 연봉 111736000원, 강성민
두 번째 레코드	연봉 136778000원, juweon50@naver.com, 세종특별자치시 용산구 가락3로, 이현주, 740921-6402472, 031-680-3538

레코드에 입력된 내용은 이름, 주소, 주민등록번호, 전화번호, 이메일, 연봉이며 이는 랜덤으로 섞여 들어가 있다. 각 항목에 대한 설명은 다음과 같다.

연봉	연봉 XXXXX원 또는 연봉 ₩XXXXX 형식으로 입력
주소	광역시도의 명칭은 다음과 같다. '서울특별시', '부산광역시', '대구광역시', '인천광역시', '광주광역시', '대전광역시', '울산광역시', '세종특별자치시', '경기도', '강원도', '충청북도', '충청남도', '전라북도', '전라남도', '경상북도', '경상남도', '제주특별자치도'
주민등록번호	2000년 이후 출생자는 없음. 즉 00XXXX-XXXXXXX, 01XXXX-XXXXXXX... 형태의 주민번호는 없다.

28 『적요.csv』 를 데이터프레임으로 불러온 후 광역시도별 연봉의 합계를 구한 결과 연봉의 합계가 가장 높은 광역시도의 이름을 입력하라. (3점)

해설

```
import pandas as pd
df = pd.read_csv('적요.csv')
# 시도 이름 발췌
def extract_sido(text):
    pattern = r'(경기도|강원도|충청북도|충청남도|전라북도|전라남도|경상북도|경상남도
|제주특별자치도|\w*특별시|\w*광역시|\w*자치시)'
    match = re.search(pattern, text)
    if match:
        return match.group(1)
    return None
# 연봉 발췌
def extract_salary(text):
    pattern = r'연봉\s*(\d+[,\d]*원|₩\d+[,\d]*)'
    match = re.search(pattern, text)
    if match:
        return match.group(1).replace('₩','').replace('원','').replace(',','')
    return None
# 나이 계산
기준일자 = datetime(2023, 5, 27)
def calculate_age(ssn):
    birth_year =int(ssn[:2])
    birth_month =int(ssn[2:4])
    birth_day =int(ssn[4:6])
    birth_year = birth_year +1900
    birth_date = datetime(birth_year, birth_month, birth_day)
    age = 기준일자.year - birth_date.year - ((기준일자.month,기준일자.day) <
(birth_date.month,birth_date.day))
    return age
#
df['시도명'] = df['적요'].apply(extract_sido)
df['연봉'] = df['적요'].apply(extract_salary).astype('int64')
df['주민등록번호_앞자리'] = df['적요'].apply(extract_ssn_front)
df['만나이'] = df['주민등록번호_앞자리'].apply(calculate_age)
df['나이대'] = df['만나이'].astype('str')
df['나이대'] = [i[0] for i in df['나이대']]
#
gb_시도명 = df.groupby(['시도명'])
df_요약_시도명=pd.DataFrame({'연봉합계' : gb_시도명['연봉'].sum()}).reset_index()
print(df_요약_시도명.sort_values('연봉합계'))
```

```
       시도명          연봉합계
0      강원도  685107434000
3     경상북도  690868608000
15    충청남도  692795639000
16    충청북도  693682537000
1      경기도  697448157000
13    전라북도  697876556000
14  제주특별자치도  699714762000
12    전라남도  713500738000
2     경상남도  713548688000
7    부산광역시  766729030000
8    서울특별시  775067522000
11   인천광역시  775666394000
5    대구광역시  777259854000
10   울산광역시  779778736000
9   세종특별자치시  780056307000
6    대전광역시  786589901000
4    광주광역시  786857182000
```

정답 : 광주광역시

29 『적요.csv』를 데이터프레임으로 불러온 후 주민등록번호의 앞 6자리와 기준일자인 2023년 5월 27일과의 차이를 통하여 만나이를 구하고 만나이를 기준으로 30대, 40대, 50대, 60대, 70대, 80대, 90대별 연봉의 합계를 구한 결과 연봉의 합계가 가장 높은 나이대를 입력하라. 예를 들어 만나이 30살 이상에서 40살 미만이면 30대이며 이 나이대의 연봉의 합계가 가장 높다면 30을 입력한다. (3점)

해설

```
import pandas as pd
gb_나이대 = df.groupby(['나이대'])
df_요약_나이대=pd.DataFrame({'연봉합계' : gb_나이대['연봉'].sum()}).reset_index()
print(df_요약_나이대.sort_values('연봉합계'))
```

```
  나이대          연봉합계
7   9   597313213000
0   2  1182027860000
3   5  1772502355000
4   6  1772975984000
2   4  1785474136000
6   8  1786512669000
1   3  1787979745000
5   7  1827762083000
```

정답 : 70