📏 예시: 리텐션율(Retention Rate)

- 1월에 100명의 고객이 가입
- 그 중 2월에도 계속 사용하는 사람 40명

$$1\text{개월 리텐션율} = \frac{40}{100} = 40\%$$

📊 리텐션 분석의 목적

- **서비스 품질 평가**: 고객이 계속 사용한다는 건 서비스가 유용하다는 뜻
- **이탈 예측 및 방지**: 고객이 언제 빠져나가는지를 파악하고 조치 가능
- **제품 개선 방향 도출**: 사용 흐름에서 어디서 문제가 발생하는지 분석
- **비즈니스 성장 예측**: 리텐션이 높을수록 장기 수익 예측이 유리함

분석 방식	설명
리텐션 커브	시간에 따른 리텐션율을 시각화한 곡선
코호트 분석(Cohort Analysis)	같은 시점에 가입한 사용자 그룹의 리텐션을 추적
생존 분석(Survival Analysis)	고객이 이탈하지 않고 남아 있을 확률 추정
이탈율(Churn Rate)	리텐션의 반대 개념 (얼마나 이탈했는지)

In [16]:
```python
# *리텐션 계산 및 커브 시삭화
initial_users = 100
retained_users = 35

retention_rate = (retained_users / initial_users) * 100
print(f"1개월 후 리텐션율: {retention_rate:.2f}%")  # 결과: 35.00%
```

```
churn_rate = 1 - retention_rate
print(f"이탈률: {churn_rate:.2%}")  # 결과: 65.00%
```

```
1개월 후 리텐션율: 35.00%
이탈률: -3400.00%
```

In [17]:
```
# *리텐션 계산 및 커브 시각화
import pandas as pd

# 예시 코호트 데이터
df = pd.DataFrame({
    'Month 0': [100, 120, 130],
    'Month 1': [60, 80, 90],
    'Month 2': [40, 65, 75]
}, index=['2024-01', '2024-02', '2024-03'])

# 리텐션율 계산
retention_rate = df.divide(df['Month 0'], axis=0).round(2)
print("리텐션율 테이블:")
print(retention_rate)
```

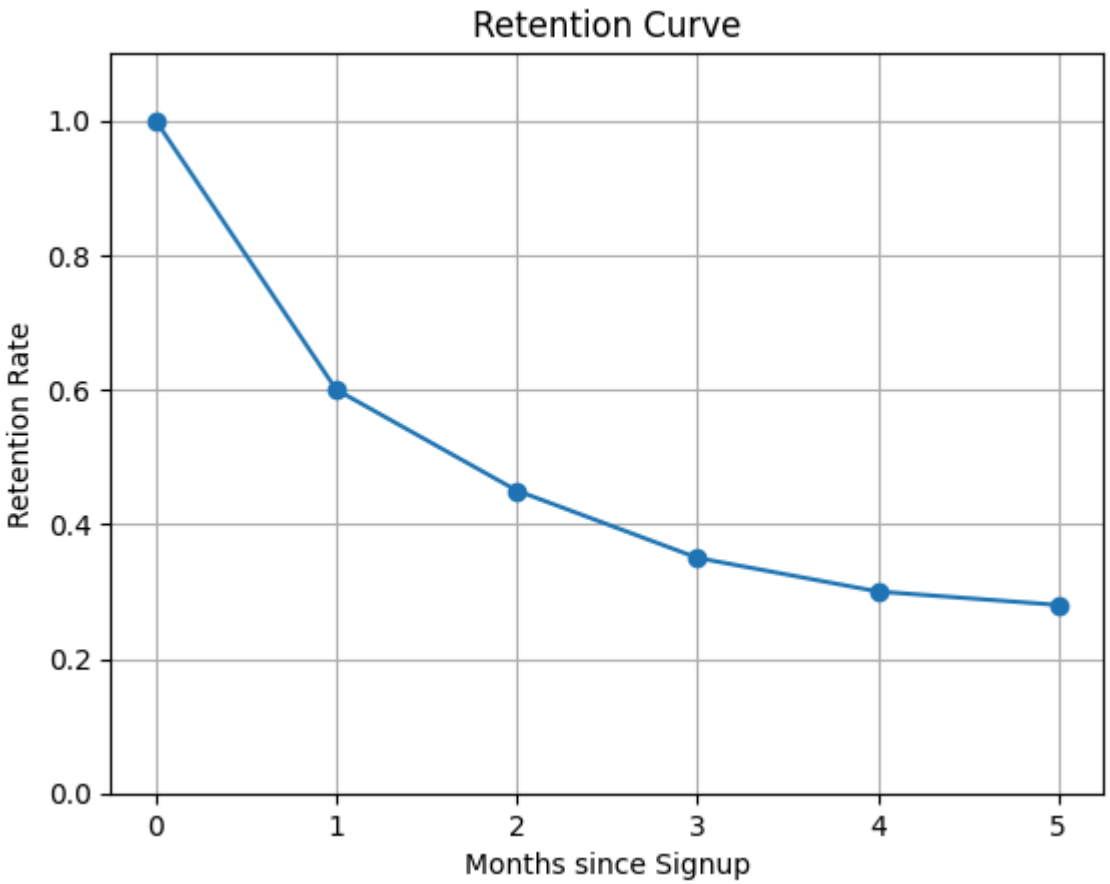
```
리텐션율 테이블:
         Month 0  Month 1  Month 2
2024-01      1.0     0.60     0.40
2024-02      1.0     0.67     0.54
2024-03      1.0     0.69     0.58
```

In [18]:
```
# *리텐션 계산 및 커브 시각화
# 각 고객의 잔존 기간 (월 단위)
retention_months = [1, 2, 3, 1, 2, 5, 4, 2]

average_retention = sum(retention_months) / len(retention_months)
print(f"평균 리텐션 기간: {average_retention:.2f}개월")
```

```
평균 리텐션 기간: 2.50개월
```

In [19]:
```
# *리텐션 계산 및 커브 시각화
import matplotlib.pyplot as plt

# 예시: 월별 리텐션율
retention_rates = [1.0, 0.6, 0.45, 0.35, 0.30, 0.28]
months = [0, 1, 2, 3, 4, 5]

plt.plot(months, retention_rates, marker='o')
plt.title("Retention Curve")
plt.xlabel("Months since Signup")
```

```
plt.ylabel("Retention Rate")
plt.ylim(0, 1.1)
plt.grid(True)
plt.show()
```



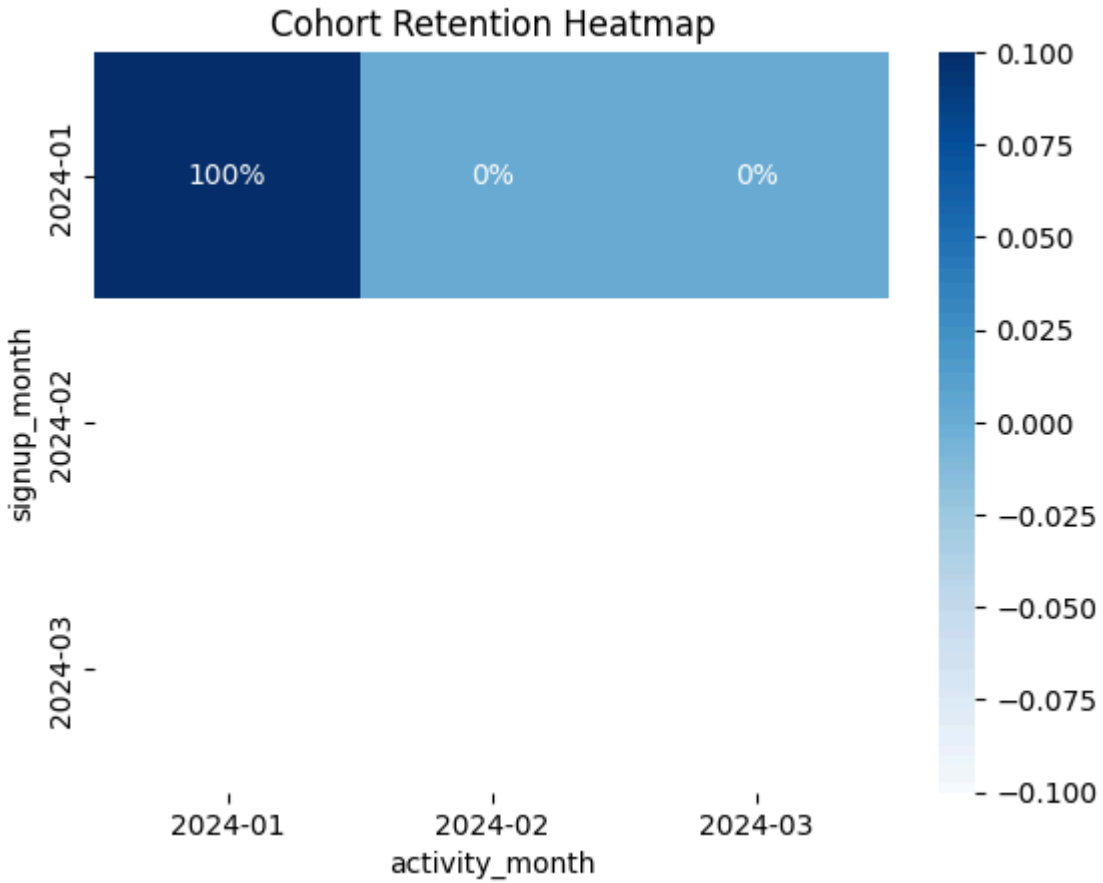
In [20]:
```
# *코호트 분석

import pandas as pd

# 코호트별 리텐션율 테이블 (데이터 주어졌다고 가정)
retention_data = pd.DataFrame({
    '1개월 후': [0.60, 0.67, 0.69],
    '2개월 후': [0.40, 0.54, 0.58]
}, index=['2024-01', '2024-02', '2024-03'])

# 평균 리텐션율로 비교
retention_data['평균 리텐션율'] = retention_data.mean(axis=1)
print(retention_data)
```

```
# 충성도 가장 높은 코호트 확인
best_cohort = retention_data['평균 리텐션율'].idxmax()
print(f"\n충성도 가장 높은 코호트: {best_cohort}")
```

```
         1개월 후  2개월 후  평균 리텐션율
2024-01   0.60   0.40    0.500
2024-02   0.67   0.54    0.605
2024-03   0.69   0.58    0.635

충성도 가장 높은 코호트: 2024-03
```

In [21]:

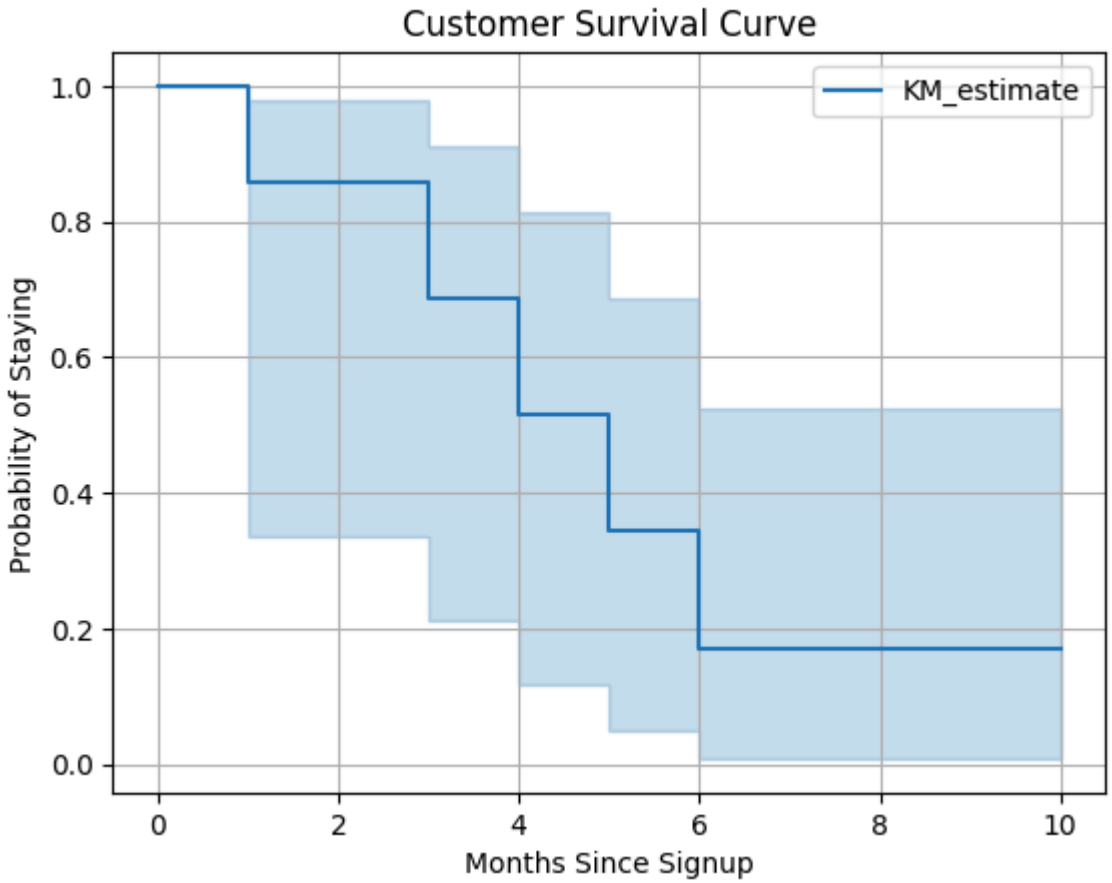
```
# *코호트 분석
import pandas as pd

# 샘플 로그인 데이터 (가입일, 행동일)
data = {
    'user_id': [1, 1, 2, 2, 3, 4, 5, 5],
    'signup_month': ['2024-01', '2024-02', '2024-01', '2024-03', '2024-02', '2024-02', '2024-01', '2024-02'],
    'activity_month': ['2024-01', '2024-02', '2024-01', '2024-03', '2024-02', '2024-03', '2024-01', '2024-03']
}

df = pd.DataFrame(data)

# 코호트 테이블 만들기
cohort = df.groupby(['signup_month', 'activity_month'])['user_id'].nunique().unstack(fill_value=0)

# 비율 계산 (0개월 기준 유지율)
cohort_percent = cohort.divide(cohort.iloc[:, 0], axis=0)

import seaborn as sns
sns.heatmap(cohort_percent, annot=True, fmt=".0%", cmap="Blues")
plt.title("Cohort Retention Heatmap")
plt.show()
```



In [22]:

```python
# *생존 분석 – 고객이 이탈하지 않을 확률
import pandas as pd
from lifelines import KaplanMeierFitter
import matplotlib.pyplot as plt

# 가상의 고객 생존 데이터
data = pd.DataFrame({
    'duration': [5, 3, 6, 2, 4, 10, 1],  # 사용 기간 (단위: 월)
    'event_observed': [1, 1, 1, 0, 1, 0, 1]  # 1: 이탈, 0: 아직 사용 중
})

# Kaplan-Meier 생존 곡선
kmf = KaplanMeierFitter()
kmf.fit(data['duration'], event_observed=data['event_observed'])

kmf.plot()
plt.title("Customer Survival Curve")
plt.xlabel("Months Since Signup")
```

```
plt.ylabel("Probability of Staying")
plt.grid(True)
plt.show()
```



In [23]:
```
# *코호트 간 리텐션율 차이가 유의미한가?
import math

# 코호트 A
n1, p1 = 50, 0.7
# 코호트 B
n2, p2 = 50, 0.55

# 합쳐진 성공률
p_pool = (p1 * n1 + p2 * n2) / (n1 + n2)

# z 계산
z = (p1 - p2) / math.sqrt(p_pool * (1 - p_pool) * (1/n1 + 1/n2))

# 유의수준 0.05에서 z_critical ≈ 1.96
```

```
print(f"z 통계량: {z:.4f}")
if abs(z) > 1.96:
    print("유의미한 차이 있음 (귀무가설 기각)")
else:
    print("유의미한 차이 없음 (귀무가설 채택)")
```

```
z 통계량: 1.5492
유의미한 차이 없음 (귀무가설 채택)
```

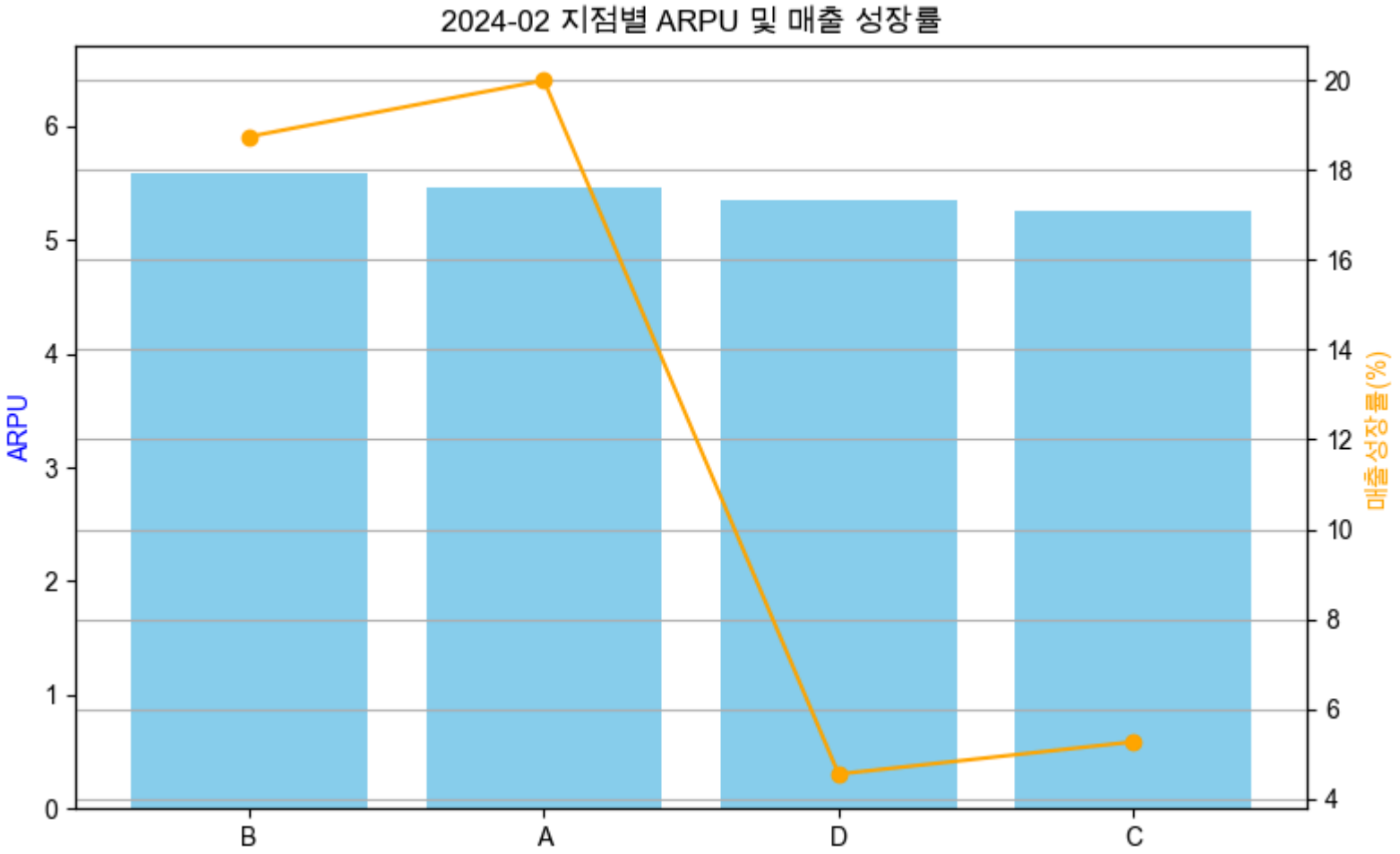
In [24]:

```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import font_manager, rc
font_path='/Library/Fonts/Arial Unicode.ttf'
font= font_manager.FontProperties(fname=font_path).get_name()
rc('font',family=font)
data = pd.DataFrame({
    '지점': ['A', 'A', 'B', 'B', 'C', 'C', 'D', 'D'],
    '월': ['2024-01', '2024-02'] * 4,
    '매출액': [1000, 1200, 800, 950, 950, 1000, 1100, 1150],
    '고객수': [200, 220, 160, 170, 180, 190, 210, 215]
})

# 2. 고객당 매출(ARPU) 계산
data['ARPU'] = data['매출액'] / data['고객수']

# 3. 전월 대비 매출 성장률 계산 (%)
data['매출성장률(%)'] = data.groupby('지점')['매출액'].pct_change() * 100

# 4. 2024-02 기준 성과 요약 정리
summary = data[data['월'] == '2024-02'][['지점', 'ARPU', '매출성장률(%)']]
summary = summary.sort_values(by='ARPU', ascending=False)

print("📊 2024년 2월 성과 기준 지점별 분석:")
print(summary)

# 5. 시각화: 지점별 ARPU & 매출 성장률
fig, ax1 = plt.subplots(figsize=(8, 5))

# ARPU bar chart
ax1.bar(summary['지점'], summary['ARPU'], color='skyblue', label='ARPU')
ax1.set_ylabel('ARPU', color='blue')
ax1.set_ylim(0, max(summary['ARPU'])*1.2)

# 성장률 선그래프
ax2 = ax1.twinx()
ax2.plot(summary['지점'], summary['매출성장률(%)'], color='orange', marker='o', label='매출성장률(%)')
ax2.set_ylabel('매출성장률(%)', color='orange')
```

```
plt.title("2024-02 지점별 ARPU 및 매출 성장률")
plt.grid(True)
plt.tight_layout()
plt.show()
# A 지점은 전월 대비 20% 성장하며 전체 지점 중 가장 높은 성장률을 기록했고,
# ARPU도 두 번째로 높아 전반적으로 우수한 성과를 보임
# 반면, C 지점은 ARPU와 성장률 모두 낮아 성과 개선이 필요함
```

📊 2024년 2월 성과 기준 지점별 분석:

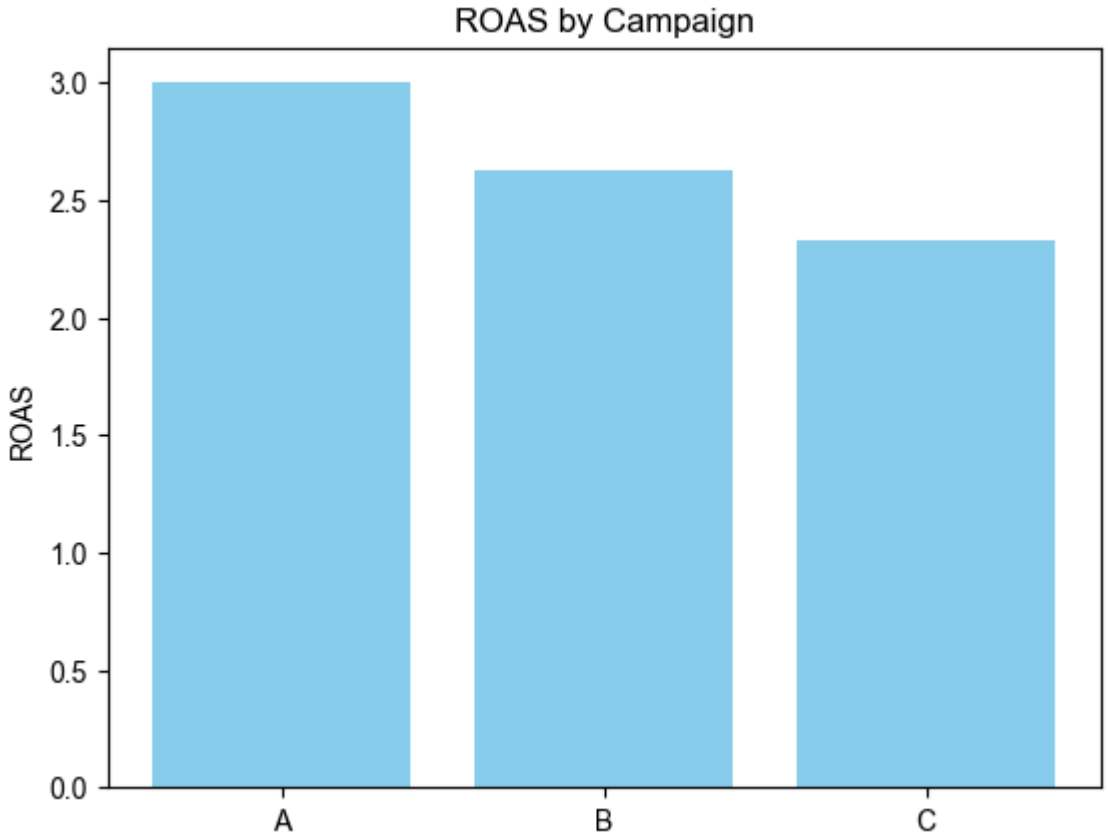
	지점	ARPU	매출성장률(%)
3	B	5.588235	18.750000
1	A	5.454545	20.000000
7	D	5.348837	4.545455
5	C	5.263158	5.263158



In [25]:
```
import pandas as pd
```

```python
data = pd.DataFrame({
    'Campaign': ['A', 'B', 'C'],
    'Impressions': [10000, 15000, 8000],
    'Clicks': [300, 400, 180],
    'Conversions': [50, 70, 20],
    'Cost': [500, 800, 300],
    'Revenue': [1500, 2100, 700]
})

# 클릭률(CTR), 전환율(CVR), 광고수익률(ROAS), 수익률(Profit Margin)
data['CTR (%)'] = (data['Clicks'] / data['Impressions']) * 100
data['CVR (%)'] = (data['Conversions'] / data['Clicks']) * 100
data['ROAS'] = data['Revenue'] / data['Cost']
data['Profit Margin (%)'] = ((data['Revenue'] - data['Cost']) / data['Revenue']) * 100

print("📊 캠페인별 성과분석 결과:")
print(data[['Campaign', 'CTR (%)', 'CVR (%)', 'ROAS', 'Profit Margin (%)']])

import matplotlib.pyplot as plt

plt.bar(data['Campaign'], data['ROAS'], color='skyblue')
plt.title("ROAS by Campaign")
plt.ylabel("ROAS")
plt.show()
```

```
📊 캠페인별 성과분석 결과:
  Campaign   CTR (%)    CVR (%)      ROAS  Profit Margin (%)
0        A  3.000000  16.666667  3.000000          66.666667
1        B  2.666667  17.500000  2.625000          61.904762
2        C  2.250000  11.111111  2.333333          57.142857
```


ROAS by Campaign
ROAS
3.0
2.5
2.0
1.5
1.0
0.5
0.0
A
B
C

***1개년 50억, 2개년 60억, 3개년 70억의 예산을 가지고 NPV가 가장 높아지는 안을 제시하시오**

가장 높은 값을 갖는 안을 찾기 위해서는 할인율(discount rate)이 필요
- 할인율이 주어지지 않았으므로, 일반적으로 사용되는 할인율을 가정
- 일반적으로 기업의 비용 기준으로 10%의 할인율을 사용할 수 있음

$$NPV = \sum_{t=1}^{n} \frac{CF_t}{(1+r)^t}$$

여기서 CF_t는 현재 시점을 기준으로 t년 후의 현금 흐름(현금 입출금)이며, r은 할인율

이를 바탕으로 안 별 NPV를 계산

1. 안 1:
$NPV_1 = \frac{10}{(1+0.10)^1} + \frac{20}{(1+0.10)^2} + \frac{15}{(1+0.10)^3}$
2. 안 2:
$NPV_2 = \frac{15}{(1+0.10)^1} + \frac{14}{(1+0.10)^2} + \frac{19}{(1+0.10)^3}$
3. 안 3:
$NPV_3 = \frac{12}{(1+0.10)^1} + \frac{11}{(1+0.10)^2} + \frac{30}{(1+0.10)^3}$
4. 안 4:
$NPV_4 = \frac{13}{(1+0.10)^1} + \frac{25}{(1+0.10)^2} + \frac{20}{(1+0.10)^3}$
5. 안 5:
$NPV_5 = \frac{16}{(1+0.10)^1} + \frac{30}{(1+0.10)^2} + \frac{24}{(1+0.10)^3}$

In [26]:
```python
import numpy as np

# 현금 흐름 설정
cash_flows = np.array([
    [10, 20, 15],
    [15, 14, 19],
    [12, 11, 30],
    [13, 25, 20],
    [16, 30, 24]
])

# 할인율
discount_rate = 0.1

# NPV 계산 함수 정의
def calculate_npv(cash_flows, discount_rate):
    npv = np.sum(cash_flows / (1 + discount_rate) ** np.arange(1, len(cash_flows) + 1))
    return npv

# 각 안의 NPV 계산
```

```
npvs = [calculate_npv(cash_flow, discount_rate) for cash_flow in cash_flows]

# 결과 출력
for i, npv in enumerate(npvs):
    print(f"안 {i+1}의 NPV: {npv}")

# 가장 높은 NPV를 가진 안 찾기
max_npv_index = np.argmax(npvs)
print(f"\n가장 높은 NPV를 가진 안: 안 {max_npv_index+1}")
```

```
안 1의 NPV: 36.889556724267464
안 2의 NPV: 39.481592787377906
안 3의 NPV: 42.539444027047324
안 4의 NPV: 47.505634861006754
안 5의 NPV: 57.37039819684447

가장 높은 NPV를 가진 안: 안 5
```

In [27]:

```
import numpy_financial as npf
import numpy as np

cash_flows = np.array([
    [10, 20, 15],  # 투자안 1의 현금 흐름
    [15, 14, 19],  # 투자안 2의 현금 흐름
    [12, 11, 30],  # 투자안 3의 현금 흐름
    [13, 25, 20],  # 투자안 4의 현금 흐름
    [16, 30, 24]   # 투자안 5의 현금 흐름
])
discount_rate = 0.1

npvs = [npf.npv(discount_rate, cash_flow) for cash_flow in cash_flows]

for i, npv in enumerate(npvs):
    print(f"투자안 {i+1}의 NPV: {npv:.2f}")

max_npv_index = np.argmax(npvs)
print(f"\n가장 높은 NPV를 가진 투자안: 투자안 {max_npv_index + 1}")
```

```
투자안 1의 NPV: 40.58
투자안 2의 NPV: 43.43
투자안 3의 NPV: 46.79
투자안 4의 NPV: 52.26
투자안 5의 NPV: 63.11

가장 높은 NPV를 가진 투자안: 투자안 5
```

In [28]:

```
# *IRR 구하기

# NPV가 0이 되는 할인율을 찾는 것
# 이는 투자 프로젝트의 수익성을 평가할 때 중요한 지표
import numpy_financial as npf

# 초기 비용은 음수 값으로, 그 후의 현금 흐름은 양수 또는 음수 값으로 표현
# 현금 흐름이 [-100, 50, 60, 70]인 경우, 이는 초기 비용이 100이고 그 후의 세 달의 현금 흐름이 각각 50, 60, 70임을 의미
cash_flows = [-100, 50, 60, 70]

# 내부수익률을 계산합니다.
irr = npf.irr(cash_flows)

print("내부수익률 (IRR): {:.2%}".format(irr))

import numpy as np
from scipy.optimize import fsolve

cash_flows = [-100, 50, 60, 70]

# NPV가 0이 되는 할인율을 찾는 함수 정의 (IRR을 계산하는 함수)
def irr_npv(irr, cash_flows):
    return np.sum([cf / (1 + irr) ** t for t, cf in enumerate(cash_flows)])

# 내부수익률(IRR)을 계산하는 함수
def calculate_irr(cash_flows):
    # IRR을 추정하기 위한 초기 값으로 0.1 (10%)을 설정
    irr_guess = 0.1
    irr = fsolve(irr_npv, irr_guess, args=(cash_flows,))
    return irr[0]

irr = calculate_irr(cash_flows)
print("내부수익률 (IRR): {:.2%}".format(irr))

import numpy as np
import numpy_financial as npf

cash_flows = np.array([
    [-50, 20, 30, 40],  # 투자안 1 (초기 투자 -50, 이후 현금 흐름)
    [-40, 15, 25, 35],  # 투자안 2
    [-60, 25, 35, 45],  # 투자안 3
    [-70, 30, 40, 50],  # 투자안 4
    [-80, 35, 45, 55]   # 투자안 5
])

irr_values = [npf.irr(cash_flow) for cash_flow in cash_flows]
```

```
for i, irr in enumerate(irr_values):
    print(f"투자안 {i + 1}의 IRR: {irr:.2%}")
```

```
내부수익률 (IRR): 33.87%
내부수익률 (IRR): 33.87%
투자안 1의 IRR: 31.69%
투자안 2의 IRR: 33.46%
투자안 3의 IRR: 30.45%
투자안 4의 IRR: 29.54%
투자안 5의 IRR: 28.83%
```

In [29]:

```
import numpy as np
from scipy.optimize import fsolve

cash_flows = np.array([
    [-50, 20, 30, 40],  # 투자안 1 (초기 투자 -50, 이후 현금 흐름)
    [-40, 15, 25, 35],  # 투자안 2
    [-60, 25, 35, 45],  # 투자안 3
    [-70, 30, 40, 50],  # 투자안 4
    [-80, 35, 45, 55]   # 투자안 5
])

def irr_npv(irr, cash_flows):
    return np.sum([cf / (1 + irr) ** t for t, cf in enumerate(cash_flows)])

def calculate_irr(cash_flow):
    # IRR이 0이 되는 해를 찾아야 하므로 초기 추정값을 0.1로 설정
    irr = fsolve(irr_npv, 0.1, args=(cash_flow,))
    return irr[0]

irr_values = [calculate_irr(cash_flow) for cash_flow in cash_flows]

for i, irr in enumerate(irr_values):
    print(f"투자안 {i + 1}의 IRR: {irr:.2%}")
```

```
투자안 1의 IRR: 31.69%
투자안 2의 IRR: 33.46%
투자안 3의 IRR: 30.45%
투자안 4의 IRR: 29.54%
투자안 5의 IRR: 28.83%
```