빅데이터 융합 개론 – 프로젝트 코드 설명

작업 환경: Google Colab

Step1. 파일 전처리

1. 파일 업로드 및 불러오기

```
# ✅ Step 1. 파일 전처리 단계
import pandas as pd
import numpy as np
import warnings

warnings.filterwarnings("ignore", category=FutureWarning)

# 1. 파일 로드
base_rate_df = pd.read_csv("/base_rate.csv", encoding='utf-8')
house_price_df = pd.read_csv("/house_price_avg.csv", encoding='euc-kr')
income_df = pd.read_csv("/income.csv", encoding='utf-8', header=1)
interest_rate_raw = pd.read_csv("/interest_rate.csv", encoding="euc-kr")
marriage_df = pd.read_csv("/mariage_per_year.csv", encoding='utf-8-sig')
pir_df = pd.read_csv("/pir.csv", encoding='utf-8')
```

: 세션 저장소에 업로드된 csv 파일들을 불러오는 코드. (warnings 구문은 월보간 시, resample('M') 대신 resample('ME')를 쓰라고 경고하는 메시지 방지 위함)

2. 기준금리 리샘플링

```
# 2. 기준금리 처리
base_rate_df['날짜'] = pd.to_datetime(base_rate_df['날짜'])
base_rate_monthly = base_rate_df.set_index('날짜').resample('D').interpolate().resample('M').last().reset_index()
```

: 불러온 base_rate.csv (기준금리 데이터)는 연간 데이터이므로, 분석 향상을 위해 .resample('M') 메서드로 월 보간 진행 (매년 -> 매월) → 월 보간 시, 선형 보간을 통해 결측치 처리(월말 기준값으로 인덱스 리셋)

3. 주택담보대출 리샘플링

```
# 3. 주담대금리 처리
interest_rate_transposed = interest_rate_raw.set_index('계정항목별').T
interest_rate_transposed.index.name = '기간'
interest_rate_numeric = interest_rate_transposed.reset_index()
interest_rate_numeric.iloc[:, 1:] = interest_rate_numeric.iloc[:, 1:].apply(pd.to_numeric, errors='coerce')
interest_rate_numeric['연도'] = interest_rate_numeric['기간'].str.extract(r'(\d{4})')[0].astype(int)

numeric_cols = interest_rate_numeric.select_dtypes(include=[np.number]).columns.difference(['연도'])
loan_avg = interest_rate_numeric.groupby('연도')[numeric_cols].mean().mean(axis=1).to_frame(name='주담대금리')
loan_avg.index = pd.to_datetime(loan_avg.index.astype(str), format='%Y')
loan_avg.index.name = '기준일'
loan_daily = loan_avg.resample('D').interpolate()
monthly_loan = loan_daily.resample('M').last().reset_index()
```

: 마찬가지로 주담대 또한 연데이터이므로, 월보간 진행해줌 (전치 -> 인덱스 리셋 -> 연도 추출 및 연도별 평균 계산 -> 일별 보간 후 월별 리샘플링)

4. 혼인건수 및 PIR지수 리샘플링

```
# 4. 혼인건수 처리
marriage_df = marriage_df[marriage_df['지역'] == '전국'][['연도', '혼인건수']]
marriage_df['연도'] = pd.to_datetime(marriage_df['연도'], format='%Y')
marriage_monthly = marriage_df.set_index('연도').resample('D').interpolate().resample('M').last().reset_index()

# 5. PIR 처리
pir_row = pir_df[pir_df['지역'].str.contains('전국|전체')].iloc[0] if pir_df['지역'].str.contains('전국|전체').any() else pir_df.iloc[0]
pir_data = pd.DataFrame({
    '기준일': pd.date_range(start='2019-01-01', end='2023-12-31', freq='Y'),
    'PIR': [pir_row[str(y)] for y in range(2019, 2024)]
})
pir_month = pir_data.set_index('기준일').resample('D').interpolate().resample('M').last().reset_index()
```

: 혼인건수는 동일. PIR지수는 전국 또는 전체 행을 선택(없으면 첫 행 선택) 후, 연도별 PIR 지수 구성 후 월별로 변환

5. 평균소득 리샘플링

```
# 6. 평균소득 처리 (컬럼명 직접 매핑)
filtered = income_df[
    (income_df['성별(1)'] == '총 계') &
    (income_df['연령대별(1)'] == '총 계') &
    (income_df['연령대별(2)'] == '소계')
]

col_map = ['평균소득', '평균소득.1', '평균소득.2', '평균소득.3', '평균소득.4']
평균소득 = []
for col in col_map:
    try:
        평균소득.append(float(filtered[col].values[0]) * 10000)
    except:
        평균소득.append(np.nan)

income_data = pd.DataFrame({
    '기준일': pd.date_range(start='2019-01-01', periods=5, freq='Y'),
    '평균소득': 평균소득
})
income_data = income_data.set_index('기준일').resample('D').interpolate().resample('M').last().reset_index()
```

: 총계/소계에 해당하는 행을 필터링 후 평균소득의 단위인 만원을 곱해주고, 예외 시 NaN 처리 후, 연도별 평균소득 df 생성, 이 데이터로 월 보간 진행

6. 평균주택 매매가 리샘플링

```
# 7. 평균주택가격 처리 (텍스트 제거 + 숫자화)
house_price_long = house_price_df.loc[:, '2019년 1월':]
house_price_long = house_price_long.replace(r'[^0-9.]', '', regex=True).apply(pd.to_numeric, errors='coerce')
house_price_long.columns = pd.date_range(start='2019-01-01', periods=house_price_long.shape[1], freq='MS')
house_price_nation = house_price_long.mean().reset_index()
house_price_nation.columns = ['기준일', '평균주택가격']
```

: 첫 줄에서 연속된 월별 데이터만 추출(데이터는 월별로 이미 전처리됨). 숫자 외의 문자는 제거하고 숫자형으로 전부 변환해줌(text -> numeric). 컬럼 이름을 날짜로 지정(start="", periods="", freq='')한 후에 인덱스를 리셋(reset_index())하고 사용할 평균주택가격 데이터 구성(마지막 줄)

7. 기준월 변환 함수 정의(to_month_str)

```
# 8. 기준월 변환 함수
def to_month_str(df, date_col='기준일'):
    df = df.copy()
    df[date_col] = pd.to_datetime(df[date_col])
    df['기준월'] = df[date_col].dt.to_period('M').astype(str)
    return df.drop(columns=[date_col])
```

: 데이터프레임과 데이터프레임의 '기준일' 컬럼을 인자값으로 리시브. 'df'라는 변수에다가 받은 데이터프레임을 복사한 후, df의 '기준일'에 해당하는 컬럼에 기준월이라는 컬럼으로 전부 형식화(기준일 삭제 및 '2020-01' 형식으로 변환, 기준월만 반환-> return)

8. 보간된 데이터들 전부 기준월 변환

```
# 9. 기준월 생성
df_base_rate = to_month_str(base_rate_monthly, date_col='날짜')
df_loan = to_month_str(monthly_loan, date_col='기준일')
df_income = to_month_str(income_data, date_col='기준일')
df_house = to_month_str(house_price_nation, date_col='기준일')
df_marriage = to_month_str(marriage_monthly, date_col='연도')
df_pir = to_month_str(pir_month, date_col='기준일')
```

: to_month_str() 함수를 통해 '기준월' 컬럼으로 형식화 된 데이터 얻음

9. 데이터 병합 및 정렬 후 전체 선형 보간(결측치 처리)

```
# 10. 병합 및 결측치 보간
final_df = df_base_rate ₩
    .merge(df_loan, on='기준월', how='outer') ₩
    .merge(df_income, on='기준월', how='outer') ₩
    .merge(df_house, on='기준월', how='outer') ₩
    .merge(df_marriage, on='기준월', how='outer') ₩
    .merge(df_pir, on='기준월', how='outer')

final_df = final_df.sort_values('기준월')
final_df = final_df.set_index('기준월').interpolate().ffill().bfill().reset_index()
```

: 'final_df'라는 데이터 프레임을 모든 보간된 데이터를 병합시킨 데이터 프레임으로 구성.

기준월을 기준으로 월별 정렬을 한번 한 후에, 전체에 대해 선형 보간을 진행(앞으로 채우고(.ffill()) 뒤로 채우고(.bfill()) 인덱스 리셋(.reset_index())

10. 최종 통합 데이터 프레임 출력해서 확인해보기

```
final_df
```

	기준월	기준금리	주담대금리	평균소득	평균주택가격	혼인건수	PIR
0	2019-01	1.463889	3.384580	3090000.0	330419.365854	237050.205479	5.4
1	2019-02	1.463889	3.341328	3090000.0	329831.097561	235081.997260	5.4
2	2019-03	1.463889	3.293443	3090000.0	329211.902439	232902.909589	5.4
3	2019-04	1.463889	3.247102	3090000.0	328506.926829	230794.115068	5.4
4	2019-05	1.463889	3.199216	3090000.0	328100.658537	228615.027397	5.4
...	...	...	...	...	...	...	...
67	2024-08	3.266091	4.761316	3630000.0	425456.097561	222412.000000	6.3
68	2024-09	3.254317	4.761316	3630000.0	427611.902439	222412.000000	6.3
69	2024-10	3.145833	4.761316	3630000.0	428793.487805	222412.000000	6.3
70	2024-11	3.000000	4.761316	3630000.0	429425.487805	222412.000000	6.3
71	2024-12	3.000000	4.761316	3630000.0	429609.463415	222412.000000	6.3

72 rows × 7 columns

: 6개 행(2019 – 2024) 에서 72개 행(6 years * 12 months)으로 잘 보간됨.

Step2. 데이터 시각화(데이터 추이 확인용) – matplotlib과 seaborn 활용

1. 연도별 추이를 위한 연도 추출

```
import matplotlib.pyplot as plt
import seaborn as sns

# 1. 연도 컬럼 생성 (기준월에서 연도 추출)
final_df['year'] = final_df['기준월'].str[:4]

# 2. 연도별 평균 집계
annual_df = final_df.groupby('year').mean(numeric_only=True).reset_index()
```

: final_df 에는 기준월 데이터로 정렬되어있음 -> year이라는 컬럼을 생성하여 연도별 평균을 칼럼값으로 (numeric data로) 집계 -> annual_df 로 생성

2. 코랩환경에서의 한글 깨짐 문제 현상 -> 영어 약어로 정의

```
# 3. 컬럼명 변경 (영어 약어로)
annual_df = annual_df.rename(columns={
    '기준금리': 'base_rate',
    '주담대금리': 'mortgage_rate',
    '평균소득': 'avg_income',
    '평균주택가격': 'avg_house_price',
    'PIR': 'pir_index',
    '혼인건수': 'num_marriages'
})

# 4. 영어 약어 ↔ 전체 이름 딕셔너리
abbr_map = {
    'BIR': 'Base Interest Rate',
    'MLR': 'Mortgage Loan Rate',
    'INC': 'Income',
    'HPI': 'House Price Index',
    'PIR': 'Price-to-Income Ratio',
    'MAR': 'Marriage Count'
}
```

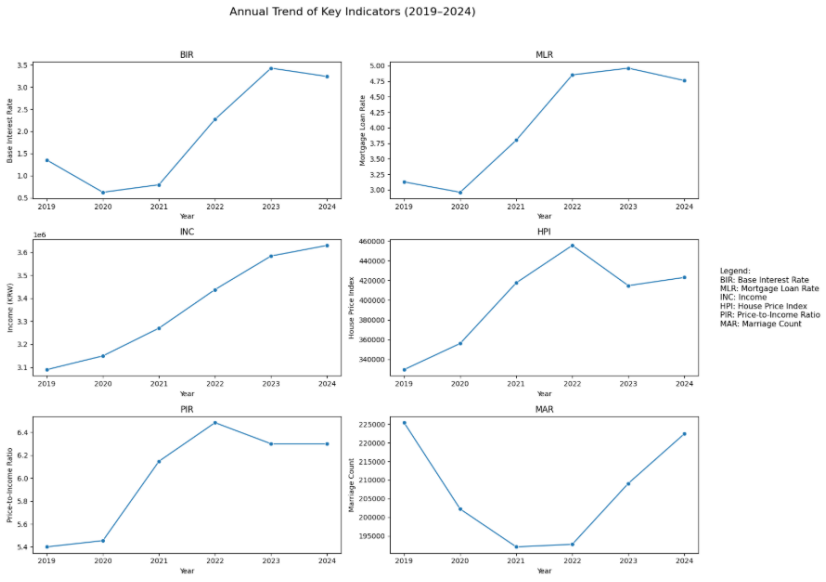
: annual_df의 컬럼명을 영어로 변환(한글 오류 해소) 및 약어 딕셔너리 생성(플롯 및 범례에 사용)

3. 시각화 – subplot을 통해 6개의 그래프(3행 2열) 생성

```
# 5. 플롯 그리기
fig, axes = plt.subplots(3, 2, figsize=(14, 12))
fig.suptitle('Annual Trend of Key Indicators (2019-2024)', fontsize=16)

# 각 subplot
sns.lineplot(data=annual_df, x='year', y='base_rate', marker='o', ax=axes[0, 0])
axes[0, 0].set_title('BIR')
axes[0, 0].set_ylabel('Base Interest Rate')
axes[0, 0].set_xlabel('Year')

sns.lineplot(data=annual_df, x='year', y='mortgage_rate', marker='o', ax=axes[0, 1])
axes[0, 1].set_title('MLR')
axes[0, 1].set_ylabel('Mortgage Loan Rate')
axes[0, 1].set_xlabel('Year')

sns.lineplot(data=annual_df, x='year', y='avg_income', marker='o', ax=axes[1, 0])
axes[1, 0].set_title('INC')
axes[1, 0].set_ylabel('Income (KRW)')
axes[1, 0].set_xlabel('Year')

sns.lineplot(data=annual_df, x='year', y='avg_house_price', marker='o', ax=axes[1, 1])
axes[1, 1].set_title('HPI')
axes[1, 1].set_ylabel('House Price Index')
axes[1, 1].set_xlabel('Year')

sns.lineplot(data=annual_df, x='year', y='pir_index', marker='o', ax=axes[2, 0])
axes[2, 0].set_title('PIR')
axes[2, 0].set_ylabel('Price-to-Income Ratio')
axes[2, 0].set_xlabel('Year')

sns.lineplot(data=annual_df, x='year', y='num_marriages', marker='o', ax=axes[2, 1])
axes[2, 1].set_title('MAR')
axes[2, 1].set_ylabel('Marriage Count')
axes[2, 1].set_xlabel('Year')
```

4. 범례 생성 및 레이아웃 조정 -> 출력

```
# 범례
legend_text_eng = '₩n'.join([f'{k}: {v}' for k, v in abbr_map.items()])
fig.text(1.02, 0.5, f'Legend:₩n{legend_text_eng}', fontsize=11, va='center')

# 레이아웃 조정 및 출력
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



Step3. 상관관계 히트맵 – 변수 관계 파악하기

1. 약어 추출 및 간략화

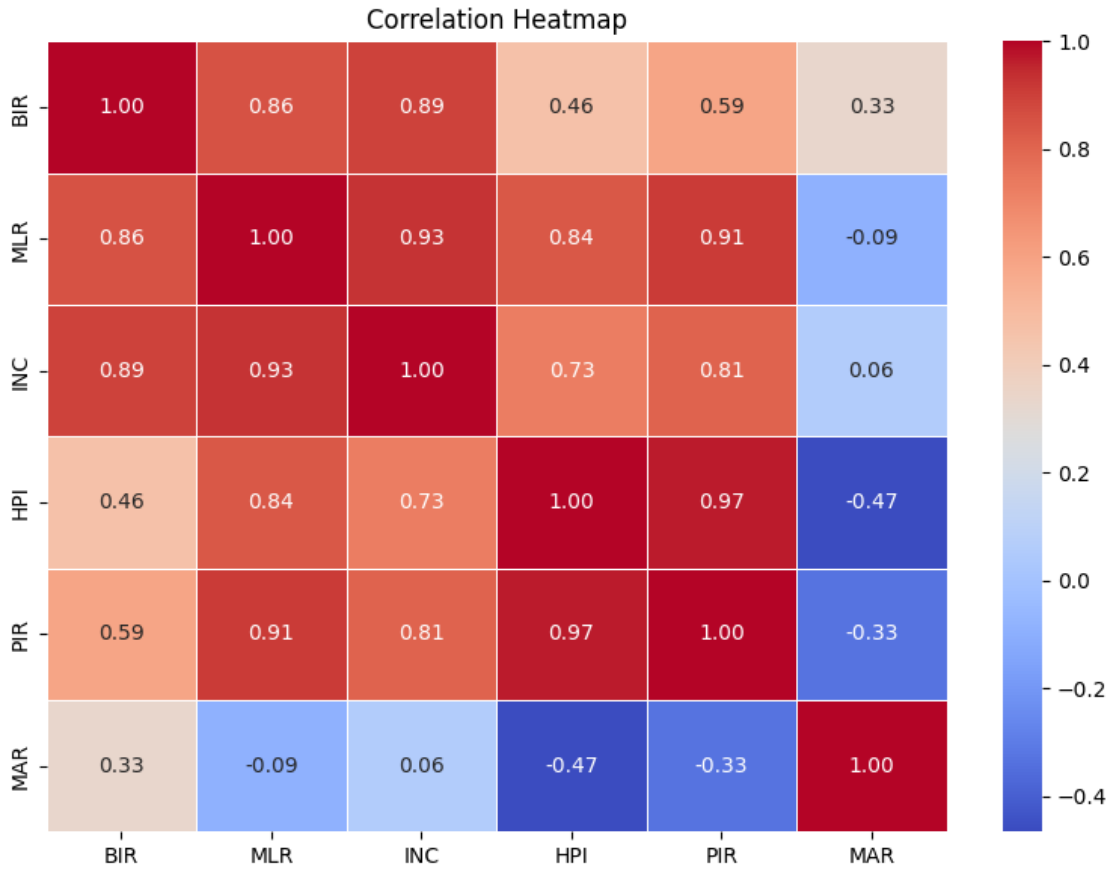
```
import matplotlib.pyplot as plt
import seaborn as sns

# 약어 기반 컬럼만 추출
corr_df = final_df[['기준금리', '주담대금리', '평균소득', '평균주택가격', 'PIR', '혼인건수']]

# 약어 대체
corr_df.columns = ['BIR', 'MLR', 'INC', 'HPI', 'PIR', 'MAR']
```

: final_df 으로부터 각 컬럼을 추출 -> 약어컬럼의 기반이 됨 -> corr_df 생성 후 각 컬럼명을 더 간략화된 약어로 대체

2. 상관행렬 계산 및 히트맵 출력

```
# 히트맵 시각화
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.tight_layout()
plt.show()
```

: 히트맵의 경우, seaborn의 .heatmap() 함수 활용



3. 분석결과 분석

변수 쌍	상관계수	관계 해석	원인 분석
BIR–MLR	0.86	매우 강한 양의 상관	기준금리(BIR)가 오르면, 시중은행의 주담대금리(MLR)도 거의 동시에 연상됨. 이는 통화정책 전이 효과 때문.
BIR–INC	0.89	매우 강한 양의 상관	기준금리가 경기 과열 시기 상승하므로, 소득(INC)이 증가하는 경기확장기와 동반 상승하는 경향. 그러나 인과는 복잡.
BIR–HPI	0.46	중간 양의 상관	예상 외로 양의 상관인데, 이는 기준금리 인상 시점이 주택 가격이 이미 많이 오른 이후이기 때문일 수 있음 (정책 지연 효과).
BIR–PIR	0.59	중간 양의 상관	PIR = 집값 / 소득, 기준금리 인상은 소득 증가 시기와 겹치고, 집값이 단기적으로 덜 떨어지기 때문에 긍정적 상관 가능.
BIR–MAR	0.33	약한 양의 상관	기준금리와 혼인건수는 직접적 관계는 적고, 소득 향상기와 일치한 상승으로 인해 약한 양의 상관 가능성. 실질적 인과는 희박.

변수 쌍	상관계수	관계 해석	원인 분석
MLR–INC	0.93	매우 강한 양의 상관	주담대금리는 기준금리 영향을 받으며, 소득이 증가하는 시기에 대출수요가 늘어나며 동반 상승하기도 함.
MLR–HPI	0.84	매우 강한 양의 상관	집값이 오르면 대출 수요도 증가 → 주담대금리 상승 압력. 은행의 위험 프리미엄 반영.
MLR–PIR	0.91	매우 강한 양의 상관	PIR 상승은 집값 급등 혹은 소득 감소 의미. 둘 다 MLR 증가와 동반될 수 있음. 집값 폭등 시기와 강하게 연결.
MLR–MAR	-0.09	거의 무관 (약한 음의 상관)	대출금리 높을수록 결혼 비용 부담이 크지만, 직접적 영향보다는 간접 효과. 사실상 상관 없음.

변수 쌍	상관계수	관계 해석	원인 분석
INC–HPI	0.73	강한 양의 상관	소득 증가 시점에 자산 투자 수요가 늘고, 주택 수요 증가 → 집값 상승. 주택 구매 여력과 연결됨.
INC–PIR	0.81	강한 양의 상관	이론적으로는 소득 증가 → PIR 하락이지만, 실제로는 집값이 더 가파르게 오르기 때문에 동시에 상승할 수 있음.
INC–MAR	0.06	상관 없음	소득과 결혼은 직접 관계보다 주거·일자리 안정성과 연결됨. 즉, 소득이 높다고 무조건 결혼 증가 아님.

변수 쌍	상관계수	관계 해석	원인 분석
HPI–PIR	0.97	거의 완전한 양의 상관	PIR = HPI / INC이므로, 구조상 매우 높은 상관. 특히 주택가격이 소득보다 급격히 오를 경우.
HPI–MAR	-0.47	중간 음의 상관	주택가격 상승은 결혼에 필요한 주거 마련 비용을 증가시킴 → 결혼 포기 또는 연기 유도. 강한 부정적 영향.

변수 쌍	상관계수	관계 해석	원인 분석
PIR–MAR	-0.33	약한 음의 상관	PIR 상승은 주거비 부담 증가를 의미 → 혼인율에 부정적 영향. 하지만 관계는 선형보다는 임계값이 있을 수 있음.

주요 영향 변수	혼인건수와의 상관	의미
`HPI` (집값)	-0.47	주거 부담 → 혼인 저하
`PIR` (주거부담지수)	-0.33	집값 대비 소득 부담 상승 → 결혼 지연
`MLR`, `BIR`, `INC`	거의 무관	직접적 영향보다는 간접·구조적 영향

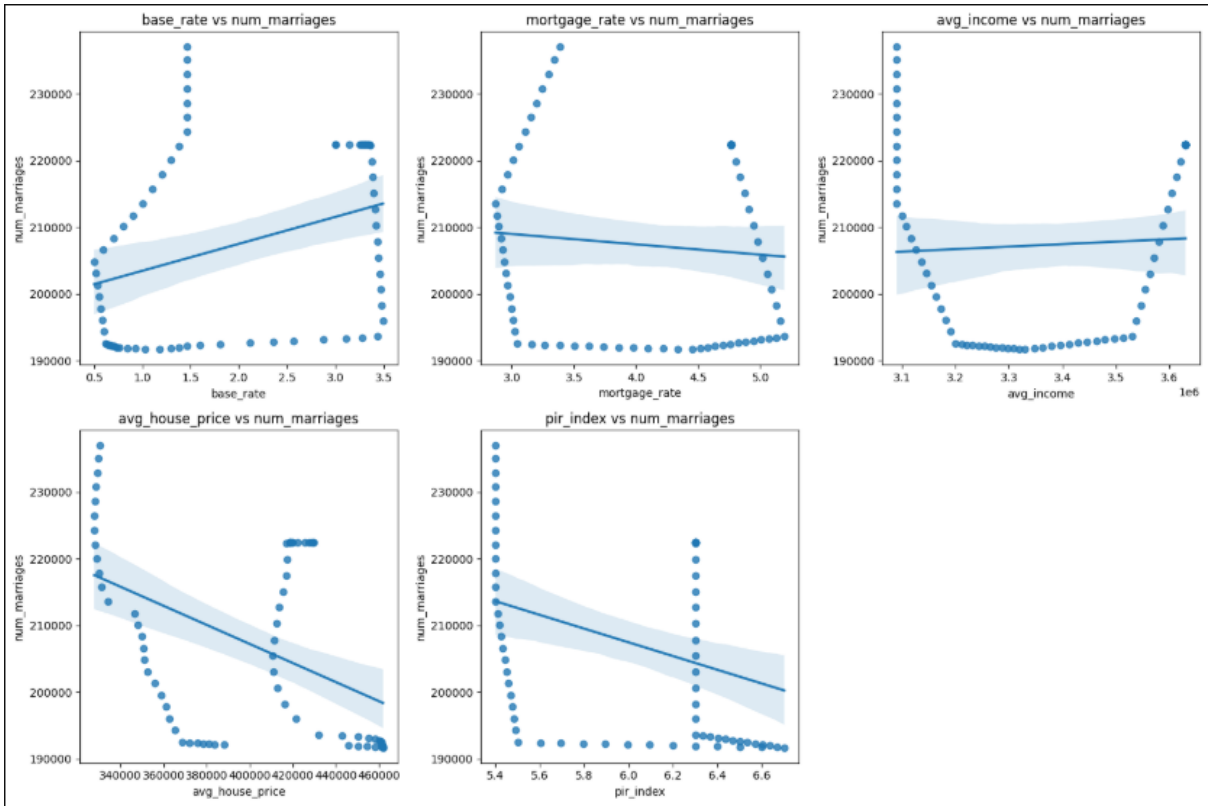
Step3. 상관관계 분석 시각화

1. 컬럼 영어 약어 작업 및 특징 컬럼 행벡터 취하기

```
# 1. 컬럼명 영어 약어로 변경
final_df = final_df.rename(columns={
    '기준금리': 'base_rate',
    '주담대금리': 'mortgage_rate',
    '평균소득': 'avg_income',
    '평균주택가격': 'avg_house_price',
    '혼인건수': 'num_marriages',
    'PIR': 'pir_index'
})

# 2. 산점도 + 회귀선 시각화 (영어 컬럼 기준)
features = ['base_rate', 'mortgage_rate', 'avg_income', 'avg_house_price', 'pir_index']
```

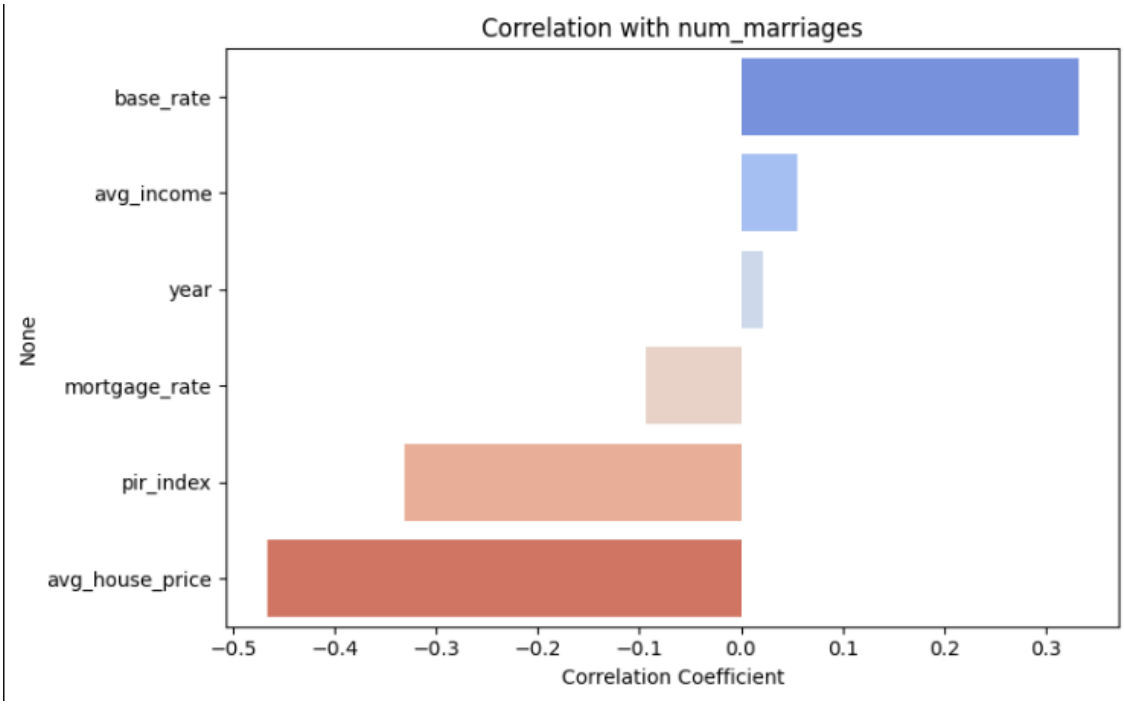
2. 산점도와 회귀선을 통해 각 변수간 상관관계 시각화

```
plt.figure(figsize=(15, 10))
for idx, col in enumerate(features, 1):
    plt.subplot(2, 3, idx)
    sns.regplot(x=final_df[col], y=final_df['num_marriages'])
    plt.title(f'{col} vs num_marriages')

plt.tight_layout()
plt.show()
```



3. 혼인건수와의 상관관계를 막대그래프로 시각화(hot vs cold)

```
# 3. 혼인건수 상관계수 추출 + 막대그래프
marriage_corr = final_df.drop(columns=['기준월']).corr()['num_marriages'].drop('num_marriages').sort_values(ascending=False)

plt.figure(figsize=(8, 5))
sns.barplot(x=marriage_corr.values, y=marriage_corr.index, palette='coolwarm')
plt.title('Correlation with num_marriages')
plt.xlabel('Correlation Coefficient')
plt.tight_layout()
plt.show()
```



Step4. 스케일링 및 회귀분석 -1. 릿지회귀

1. 스케일링 준비

```
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# 1. 데이터 준비
X_ridge = final_df[['base_rate', 'mortgage_rate', 'pir_index', 'avg_house_price', 'avg_income']].astype(float).to_numpy()
y_ridge_original = final_df['num_marriages'].astype(float).to_numpy().reshape(-1, 1)
```

: 릿지회귀 모델 및 정규화모델(StandardScaler)을 sklearn에서 import하고 데이터 정의

2. 스케일링

```
# 2. 스케일링
scaler_X = StandardScaler()
X_scaled = scaler_X.fit_transform(X_ridge)

scaler_y = StandardScaler()
y_scaled = scaler_y.fit_transform(y_ridge_original)
```

: x(입력값)과 y(출력값)에 따로 스케일링(정규화)을 적용

3. 릿지회귀 모델 학습

```
# 3. 모델 학습
model = Ridge(alpha=1.0)
model.fit(X_scaled, y_scaled)
```

4. 모델로 2025년 혼인건수 예측 및 결괏값 역정규화(보정)

```
# 4. 2025년 예측
X_input = np.array([[2.75, 4.17, 7.12, 408934, 2910000]])
X_input_scaled = scaler_X.transform(X_input)
y_scaled_pred = model.predict(X_input_scaled)
y_pred = scaler_y.inverse_transform(y_scaled_pred.reshape(-1, 1))[0][0]

# 5. 결과 보정 및 출력
rounded_prediction = int(max(0, round(y_pred)))  # 음수 방지
print(f"2025년 예측 혼인건수(릿지 회귀): {rounded_prediction}건")
```

: 2025년의 데이터(현재 기준)를 입력값으로 정의(정답 없음)하여 예측(model.predict())하고, y_pred에 inverse_transtorm()하여 보정한 값을 y_prediction에 저장

5. 결괏값 평가

```
# 6. 모델 평가
y_train_pred = model.predict(X_scaled)
y_train_pred_orig = scaler_y.inverse_transform(y_train_pred.reshape(-1, 1))
print("---" * 25)
print(f"MSE: {mean_squared_error(y_ridge_original, y_train_pred_orig):.2f}")
print(f"R² : {r2_score(y_ridge_original, y_train_pred_orig):.4f}")
```

```
2025년 예측 혼인건수(릿지 회귀): 235696건
---------------------------------------------------------------------------
MSE: 65961715.19
R² : 0.6671
```

✅ 예측 결과 해석

항목	값	해석
2025년 혼인건수 예측값	235,696건	최근 몇 년간의 평균(222,000~237,000건)과 유사한 합리적인 값
훈련 데이터 MSE	65,961,715	평균 제곱 오차로 보면 약 √66만 ≈ 8,100건 정도의 평균 오차가 있음
R² 점수	0.6671	타깃값의 66.7%를 설명하는 모델이라는 의미

Step4-2. 규제회귀 비교해보기

1. 데이터 및 모델 정의

```python
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np

# 1. 데이터 구성
X = final_df[['base_rate', 'mortgage_rate', 'pir_index', 'avg_house_price', 'avg_income']].astype(float)
y = final_df['num_marriages'].astype(float).to_numpy().reshape(-1, 1)

# 2. 정규화
scaler_X = StandardScaler()
X_scaled = scaler_X.fit_transform(X)

scaler_y = StandardScaler()
y_scaled = scaler_y.fit_transform(y)

# 3. 예측용 2025년 입력 데이터
X_input_raw = np.array([[2.75, 4.17, 7.12, 408934, 2910000]])
X_input_scaled = scaler_X.transform(X_input_raw)

# 4. 모델들 정의
models = {
    "Ridge": Ridge(alpha=1.0),
    "Lasso": Lasso(alpha=0.1),
    "ElasticNet": ElasticNet(alpha=0.1, l1_ratio=0.5)
}
```

: 릿지, 라쏘, 엘라스틱넷을 서로 비교해보기

2. 학습 및 평가

```python
results = []

for name, model in models.items():
    model.fit(X_scaled, y_scaled)

    # 학습 데이터에 대한 예측 및 평가
    y_pred_scaled = model.predict(X_scaled)
    y_pred_original = scaler_y.inverse_transform(y_pred_scaled.reshape(-1, 1))

    mse = mean_squared_error(y, y_pred_original)
    r2 = r2_score(y, y_pred_original)

    # 2025년 예측
    y_2025_scaled = model.predict(X_input_scaled)
    y_2025_original = scaler_y.inverse_transform(y_2025_scaled.reshape(-1, 1))[0][0]
    y_2025_final = round(max(0, y_2025_original))

    results.append({
        "모델": name,
        "R² ": round(r2, 4),
        "MSE": round(mse, 2),
        "2025년 예측 혼인건수": y_2025_final
    })
```

: predict(), scaled -> inverse_transform(), result.append() 하여 결괏값 테이블 생성

3. 출력

```python
# 결과 테이블 출력
results_df = pd.DataFrame(results)

results_df
```

	모델	R^2	MSE	2025년 예측 혼인건수
0	Ridge	0.6671	65961715.19	235696
1	Lasso	0.5628	86622324.24	210527
2	ElasticNet	0.5764	83937368.57	210802

: 릿지 회귀가 가장 적절함

Step4-3. 랜덤포레스트 적용

```
from sklearn.ensemble import RandomForestRegressor

# 1. 입력 데이터 및 타깃값 구성
X_rf = final_df[['base_rate', 'mortgage_rate', 'pir_index', 'avg_house_price', 'avg_income']].astype(float).to_numpy()
y_rf = final_df['num_marriages'].astype(float).to_numpy()

# 2. 랜덤 포레스트 모델 정의 및 훈련
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_rf, y_rf)

# 3. 2025년 입력값 구성 및 예측
X_2025_rf = np.array([[2.75, 4.17, 7.12, 408934, 2910000]])
y_2025_rf_pred = rf_model.predict(X_2025_rf)[0]

print(f"2025년 예측 혼인건수(랜덤 포레스트): {round(y_2025_rf_pred)}건")
```

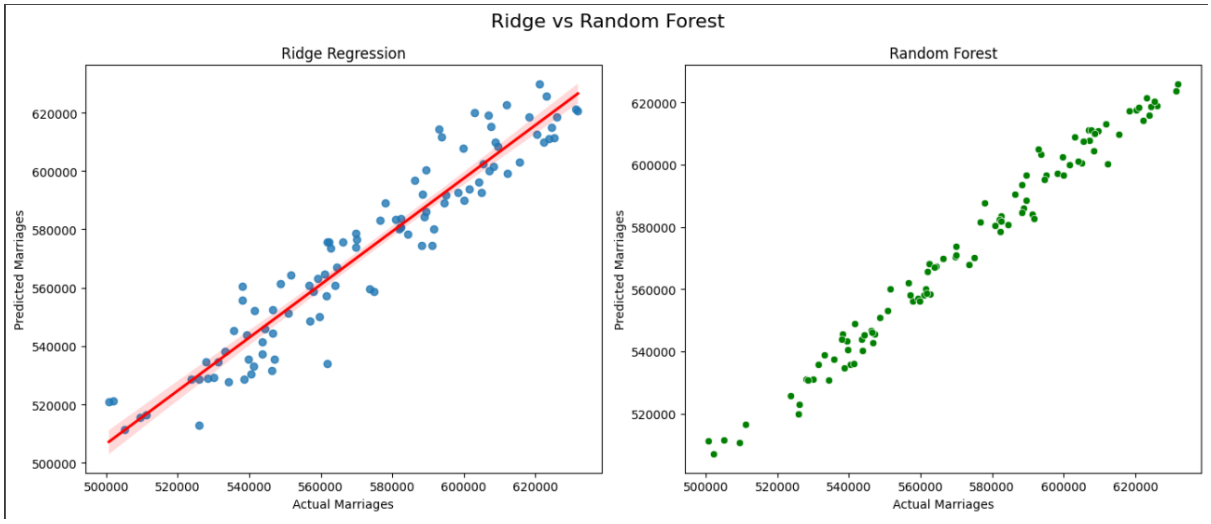
```
2025년 예측 혼인건수(랜덤 포레스트): 215638건
```

: 릿지와 랜덤포레스트의 값은 약 2만건 정도 차이나는 것을 확인

Step5. 두 분석단계를 시각화하여 비교

```
# 비교 데이터프레임 생성
comparison_df = pd.DataFrame({
    'Actual': y_true,
    'Ridge_Predicted': y_pred_ridge,
    'RF_Predicted': y_pred_rf
})

# 시각화
fig, axes = plt.subplots(1, 2, figsize=(14, 6))
fig.suptitle('Ridge vs Random Forest', fontsize=16)

# Ridge 회귀선 그래프
sns.regplot(data=comparison_df, x='Actual', y='Ridge_Predicted', ax=axes[0], line_kws={"color": "red"})
axes[0].set_title('Ridge Regression')
axes[0].set_xlabel('Actual Marriages')
axes[0].set_ylabel('Predicted Marriages')

# Random Forest 산점도 (회귀선 없음)
sns.scatterplot(data=comparison_df, x='Actual', y='RF_Predicted', ax=axes[1], color="green")
axes[1].set_title('Random Forest')
axes[1].set_xlabel('Actual Marriages')
axes[1].set_ylabel('Predicted Marriages')

plt.tight_layout()
plt.show()
```

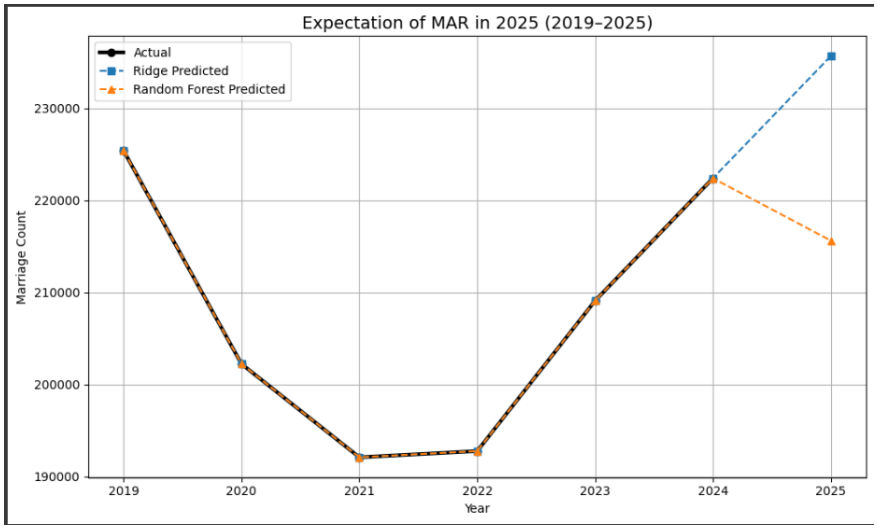
: comparison_df에 2019~2024 + 25예측값들을 전부 넣어 실측값, 릿지회귀값, 랜덤포레스트값으로 데이터프레임 구성. 이 데이터프레임을 시각화



: 이렇게 비교해서는 딱히 차이점을 못느낌을 알게됨. → 다르게 시각화 시도

Step6. 예측값을 추이기반 꺾은선 그래프로 비교

```python
# 연도 추출
final_df['연도'] = final_df['기준월'].str[:4]

# 연도 및 실제 평균 혼인건수
years = sorted(final_df['연도'].unique().astype(int).tolist())
actual = final_df.groupby('연도')['num_marriages'].mean().astype(int).tolist()

# 예측값 추가
years_with_2025 = years + [2025]
ridge_pred = actual + [round(rounded_prediction)]
rf_pred = actual + [round(y_2025_rf_pred)]

# 시각화
plt.figure(figsize=(10, 6))
plt.plot(years, actual, label='Actual', marker='o', linewidth=3, color='black')
plt.plot(years_with_2025, ridge_pred, label='Ridge Predicted', marker='s', linestyle='--')
plt.plot(years_with_2025, rf_pred, label='Random Forest Predicted', marker='^', linestyle='--')

plt.title('Expectation of MAR in 2025 (2019-2025)', fontsize=14)
plt.xlabel('Year')
plt.ylabel('Marriage Count')
plt.xticks(years_with_2025)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



: 릿지회귀는 추이를 따라가고, 랜덤포레스트는 그렇지 않음을 확인. 그러나, 현실에서의 상황을 고려한다면, 증가도, 감소도 큰폭이 이루어지지 않고, 감소한다면 했지, 증가는 안할거 같음(경험적 데이터). 그러므로 일단 두 방식의 검증이 필요함.

Step7. 검증단계(MAPE 사용)

1. 입력값 정의

```
from sklearn.metrics import mean_absolute_percentage_error
import numpy as np

# === 검증 입력값 원본
val_x_17_raw = np.array([[1.75, 3.27, 5.6, 255966, 27330000]])
val_x_18_raw = np.array([[1.5, 3.39, 5.7, 285680, 28100000]])

# === 정규화된 입력값 (Ridge 모델 학습 시 사용된 scaler_X 활용)
val_x_17_scaled = scaler_X.transform(val_x_17_raw)
val_x_18_scaled = scaler_X.transform(val_x_18_raw)
```

: 검증데이터는 정답이 있는 2017년과 2018년도 데이터를 활용(이 역시 정규화 진행)

2. 릿지회귀

```
# === Ridge 예측 (정규화된 X → 정규화된 y → 역정규화)
ridge_pred_17_scaled = ridge_model.predict(val_x_17_scaled)
ridge_pred_18_scaled = ridge_model.predict(val_x_18_scaled)

ridge_pred_17 = scaler_y.inverse_transform(ridge_pred_17_scaled.reshape(-1, 1))[0][0]
ridge_pred_18 = scaler_y.inverse_transform(ridge_pred_18_scaled.reshape(-1, 1))[0][0]
```

: 정규화된 x에 대해 예측 진행하고, 정규화된 예측 y를 다시 역정규화로 복원

3. 랜덤포레스트

```
# === Random Forest은 정규화 불필요
rf_pred_17 = rf_model.predict(val_x_17_raw)[0]
rf_pred_18 = rf_model.predict(val_x_18_raw)[0]
```

: 얘는 정규화가 필요없어서 바로 predict()

4. 정답 정의 및 MAPE 계산

```
# === 실제 혼인건수
true_17 = 264455
true_18 = 257622

# === MAPE 계산
ridge_mape_17 = mean_absolute_percentage_error([true_17], [ridge_pred_17])
ridge_mape_18 = mean_absolute_percentage_error([true_18], [ridge_pred_18])
rf_mape_17 = mean_absolute_percentage_error([true_17], [rf_pred_17])
rf_mape_18 = mean_absolute_percentage_error([true_18], [rf_pred_18])
```

: 2017년과 2018년의 정답 데이터를 기반으로 mean_absolute_percentage_error() 진행

➔ 사용이유: 비율로 한눈에 파악하기 쉬움.

5. 검증 출력

```
# === 결과 출력
print(f"📌 Ridge MAPE (2017): {ridge_mape_17:.2%}")
print(f"📌 Ridge MAPE (2018): {ridge_mape_18:.2%}")
print("--" * 30)
print(f"🌲 Random Forest MAPE (2017): {rf_mape_17:.2%}")
print(f"🌲 Random Forest MAPE (2018): {rf_mape_18:.2%}")
```

```
📌 Ridge MAPE (2017): 6.15%
📌 Ridge MAPE (2018): 4.53%
------------------------------------------------------------
🌲 Random Forest MAPE (2017): 14.31%
🌲 Random Forest MAPE (2018): 11.51%
```

릿지 회귀에 비해서는 랜덤포레스트가 정확도가 떨어짐.

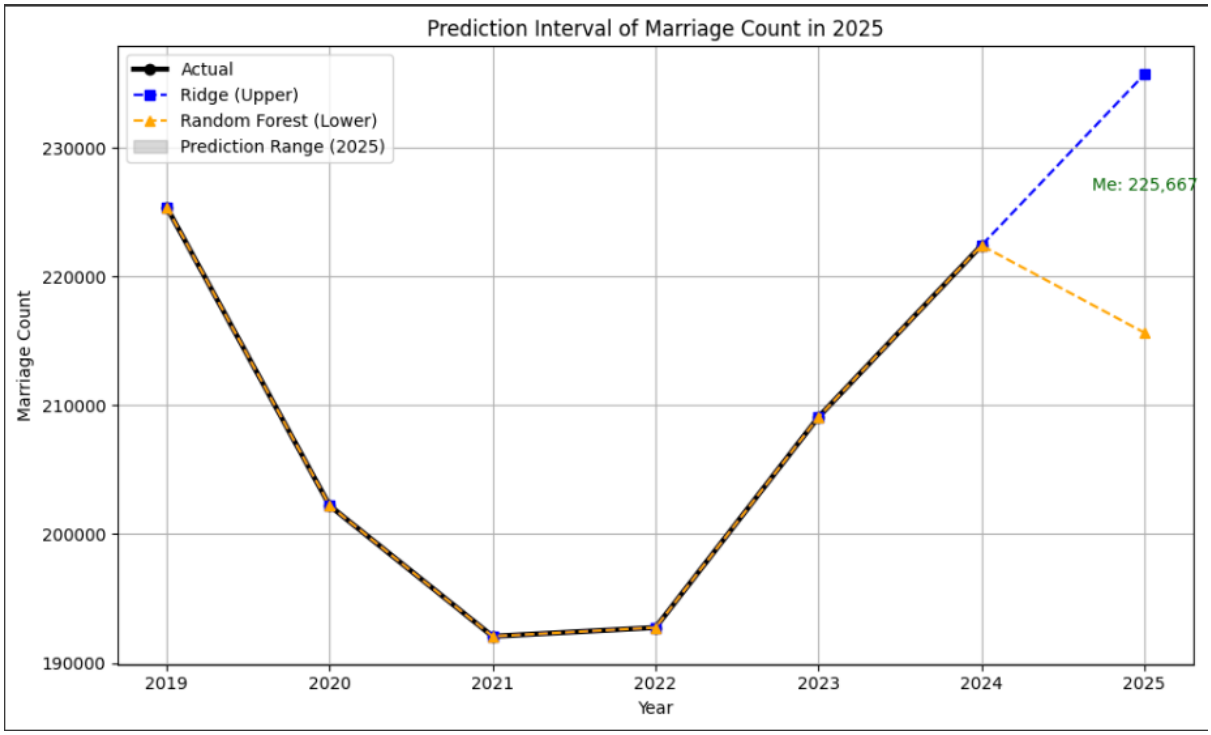
--> 그러나, 회귀라는 측면에서 릿지회귀는 추세를 따라가는 경향을 보이고, 랜덤포레스트는 외삽에 강함.
--> Ridge => 상한 / RandomForest => 하한으로 설정하여 분석

Step8. 최종 결괏값 출력 및 시각화

```
import matplotlib.pyplot as plt

upper_bound = round(rounded_prediction)  # Ridge 예측값
lower_bound = round(y_2025_rf_pred)          # Random Forest 예측값

# 연도 리스트
years_with_2025 = years + [2025]
actual = final_df.groupby('연도')['num_marriages'].mean().astype(int).tolist()

# 예측값들
ridge_pred = actual + [upper_bound]
rf_pred = actual + [lower_bound]

# 예측 구간
prediction_range = [ (ridge_pred[i] - rf_pred[i]) if i == len(ridge_pred)-1 else 0 for i in range(len(ridge_pred)) ]
prediction_mid = [ (ridge_pred[i] + rf_pred[i]) // 2 if i == len(ridge_pred)-1 else ridge_pred[i] for i in range(len(ridge_pred)) ]

# 시각화
plt.figure(figsize=(10, 6))
plt.plot(years, actual, label='Actual', marker='o', linewidth=3, color='black')
plt.plot(years_with_2025, ridge_pred, label='Ridge (Upper)', linestyle='--', marker='s', color='blue')
plt.plot(years_with_2025, rf_pred, label='Random Forest (Lower)', linestyle='--', marker='^', color='orange')

# 예측 구간 표시 (2025년만)
plt.fill_between([2025], [lower_bound], [upper_bound], color='gray', alpha=0.3, label='Prediction Range (2025)')

# 텍스트로 중앙값 표시
plt.text(2025, prediction_mid[-1] + 1000, f"Me: {prediction_mid[-1]:,}", ha='center', fontsize=10, color='darkgreen')

plt.title('Prediction Interval of Marriage Count in 2025')
plt.xlabel('Year')
plt.ylabel('Marriage Count')
plt.xticks(years_with_2025)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

: 코드에 관해서는 위의 내용들을 참조할 것. (비슷한 과정 위에 똑 같은 과정 존재)

→ 예측값 시각화단계인 step6 : ref.



: 상한/하한을 기점으로 중앙값(Me)를 책정하여 그래프위에 텍스트 기반으로 표시

➔ Plt.text() 구문 확인

⇨ 최종 예측값은 225,667(건)으로, 전년도 기준 약 1.5% 정도 증가할 것으로 예상됨.