# 제주도 날씨 데이터를 활용한 관광객 수 예측 모델

TAVE 10기 데이터분석 2조 "데빌"

두 번째 프로젝트

# CONTENTS

# INTRODUCTION

1.1. Problem Definition

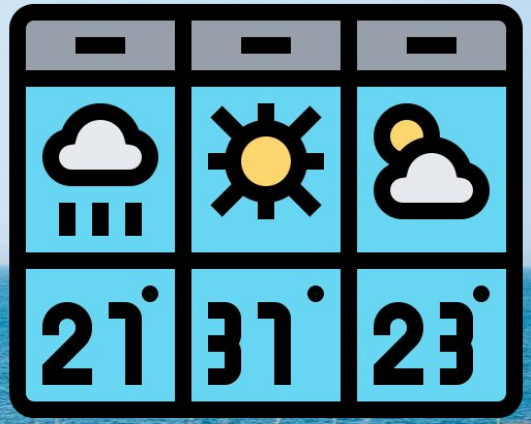
1.2. Project Necessity

# 1.1. Problem Definition

## Why Jeju?

- 날씨에 의해 크게 좌지우지되는 제주 여행

- 구체적으로 "어떤" 날씨 지표가 관광객 수에 영향을 줄까?



# 1.2. Project Necessity

## 기대 효과

- 기상 정보를 통한 보다 정확한 관광객 수 예측

- 예측 관광객 수를 토대로 효율적인 관광 산업 운영 가능

# METHODOLOGY

2.1. Data Source

2.2. Data Preprocessing

2.2. EDA : 히트맵

# 2.1. Data Source

1. 필요한 데이터 수집

(관광객수, 날씨정보)

-제주데이터허브 : 제주특별자치도_내국인관광객현황.xlsx

-국가통계포털(KOSIS) : 제주기상개황정보.xlsx

2. 한 해로는 데이터 수가 부족해 2015-2020 데이터 병합


2015 data

2020 data

2015-2020 data

# 2.2. Data Preprocessing

## 1. 관광객수 데이터 불러오기

```
import pandas as pd
import numpy as np

import warnings
warnings.filterwarnings(action='ignore')
```

```
# 제주 내국인 관광객 수 데이터(2015 ~2020)
df = pd.read_excel('jeju_tourist.xlsx')
df.head()
```

| | 구분연월 | 행태별 - 개별 여행 | 행태별 - 부분패키지 | 행태별 - 패키지 | 목적별 - 레저스포츠 | 목적별 - 회의및 업무 | 목적별 - 휴양및 관람 | 목적별 - 친지 방문 | 목적별 - 교육 여행 | 목적별 - 기타 방문 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2015-01-01 | 624526 | 96124 | 72018 | 127063 | 75791 | 521304 | 55457 | 3621 | 9432 |
| 1 | 2015-02-01 | 595357 | 103768 | 22462 | 134718 | 72552 | 459032 | 43058 | 202 | 12025 |
| 2 | 2015-03-01 | 635832 | 110797 | 45085 | 143528 | 78294 | 490004 | 44581 | 20466 | 14841 |
| 3 | 2015-04-01 | 600794 | 221013 | 117518 | 146259 | 83467 | 623087 | 27927 | 44893 | 13662 |
| 4 | 2015-05-01 | 760163 | 134508 | 120668 | 179944 | 87166 | 628479 | 29779 | 71746 | 18225 |

## 2. 행태별 관광객 수를 모두 더해 총 관광객수 'total' 구하기

```
# 월별 관광객 수: 형태별(개별여행), 형태별(부분 패키지), 형태별(패키지) 컬럼 값 병합
total = df.iloc[:,1] + df.iloc[:,2] + df.iloc[:,3]
total = pd.DataFrame(total, columns=['total'])
total.head()
```

→

| | total |
|---|---|
| 0 | 792668 |
| 1 | 721587 |
| 2 | 791714 |
| 3 | 939325 |
| 4 | 1015339 |

# 2.2. Data Preprocessing

## 3. 날씨 데이터 불러오기

```
# 제주도 기상개황정보 데이터(2015 ~ 2020)
df2 = pd.read_excel('jeju_weather.xlsx')
df2.head()
```

| | 내용 | 평균기온 (℃) | 평균강수량 (mm) | 평균상대습도 (%) | 평균해면기압 (hPa) | 이슬점온도 (℃) | 평균운량 (1/10) | 일조시간 (hr) | 최심신적설 (cm) | 평균풍속 (m/s) |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 201501 | 7.4 | 82.4 | 66.0 | 1024.6 | 1.2 | 6.5 | 94.2 | 1.8 | 3.9 |
| 1 | 201502 | 7.3 | 35.5 | 64.0 | 1022.7 | 0.7 | 6.5 | 93.5 | 0.5 | 3.7 |
| 2 | 201503 | 10.4 | 80.7 | 64.0 | 1021.9 | 3.3 | 5.3 | 183.8 | - | 2.9 |
| 3 | 201504 | 15.1 | 147.9 | 71.0 | 1015.9 | 9.0 | 6.2 | 163.4 | - | 2.9 |
| 4 | 201505 | 18.8 | 150.0 | 68.0 | 1010.7 | 12.1 | 5.3 | 239.7 | - | 2.6 |

## 4. 날씨 데이터와 전체관광객수를 병합해 'jeju' 데이터프레임 만들기

```
# 데이터프레임 병합
jeju = pd.concat([df2, total], axis=1)
jeju.head(3)
```

| | 내용 | 평균기온 (℃) | 평균강수량 (mm) | 평균상대습도 (%) | 평균해면기압 (hPa) | 이슬점온도 (℃) | 평균운량 (1/10) | 일조시간 (hr) | 최심신적설 (cm) | 평균풍속 (m/s) | total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 201501 | 7.4 | 82.4 | 66.0 | 1024.6 | 1.2 | 6.5 | 94.2 | 1.8 | 3.9 | 792668 |
| 1 | 201502 | 7.3 | 35.5 | 64.0 | 1022.7 | 0.7 | 6.5 | 93.5 | 0.5 | 3.7 | 721587 |
| 2 | 201503 | 10.4 | 80.7 | 64.0 | 1021.9 | 3.3 | 5.3 | 183.8 | - | 2.9 | 791714 |

# 2.2. Data Preprocessing

## 5. 결측값이 너무 많은 칼럼 제거

```
# '최심신적설'은 결측값이 너무 많기에 존재하지 않으니 drop
jeju.drop(columns=['최심신적설 (cm)'], inplace=True)
```

## 6. heatmap에서 깨지지 않도록 칼럼 이름을 영어로 변경

```
# heatmap시 깨지지 않도록 칼럼 이름 변경
jeju.columns = ['date', 'temperature(℃)', 'rain(mm)', 'humidity(%)', 'pressure(hPa)', 'dew_point(℃)', 'cloud(1/10)', 'daylig
jeju.head(3)
```

| | date | temperature(℃) | rain(mm) | humidity(%) | pressure(hPa) | dew_point(℃) | cloud(1/10) | daylight(hr) | windspeed(m/s) | total |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 201501 | 7.4 | 82.4 | 66.0 | 1024.6 | 1.2 | 6.5 | 94.2 | 3.9 | 792668 |
| 1 | 201502 | 7.3 | 35.5 | 64.0 | 1022.7 | 0.7 | 6.5 | 93.5 | 3.7 | 721587 |
| 2 | 201503 | 10.4 | 80.7 | 64.0 | 1021.9 | 3.3 | 5.3 | 183.8 | 2.9 | 791714 |

# 2.2. Data Preprocessing

## 7. 칼럼 확인

```
# 칼럼 확인
jeju.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 72 entries, 0 to 71
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   date             72 non-null     int64
 1   temperature(℃)   72 non-null     float64
 2   rain(mm)         72 non-null     float64
 3   humidity(%)      72 non-null     float64
 4   pressure(hPa)    72 non-null     float64
 5   dew point(℃)     72 non-null     float64
 6   cloud(1/10)      72 non-null     object
 7   daylight(hr)     72 non-null     float64
 8   windspeed(m/s)   72 non-null     float64
 9   total            72 non-null     int64
dtypes: float64(7), int64(2), object(1)
memory usage: 5.8+ KB
```

| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 내용 | 평균기온 ( | 평균강수량 | 평균상대습 | 평균해면기 | 이슬점온도 | 평균운량 ( | 일조시간 ( | 최심신적설 | 평균풍속 (r |
| 2 | 201501 | 7.4 | 82.4 | 66 | 1024.6 | 1.2 | 6.5 | 94.2 | 1.8 | 3.9 |
| 3 | 201502 | 7.3 | 35.5 | 64 | 1022.7 | 0.7 | 6.5 | 93.5 | 0.5 | 3.7 |
| 4 | 201503 | 10.4 | 80.7 | 64 | 1021.9 | 3.3 | 5.3 | 183.8 | - | 2.9 |
| 5 | 201504 | 15.1 | 147.9 | 71 | 1015.9 | 9 | 6.2 | 163.4 | - | 2.9 |
| 6 | 201505 | 18.8 | 150 | 68 | 1010.7 | 12.1 | 5.3 | 239.7 | - | 2.6 |
| 7 | 201506 | 22 | 186.4 | 80 | 1007.3 | 18.1 | 7.6 | 129.7 | - | 2.3 |
| 8 | 201507 | 25.6 | 329.7 | 81 | 1007 | 21.8 | 6.5 | 196.1 | - | 3.1 |
| 9 | 201508 | 26.4 | 248.6 | 78 | 1008.4 | 22.1 | 5.9 | 193.6 | - | 2.5 |
| 10 | 201509 | 23.2 | 172.9 | 73 | 1013.3 | 17.9 | 5.4 | 199.2 | - | 2.7 |
| 11 | 201510 | 19.2 | 31 | 64 | 1019.2 | 11.9 | 3.7 | 223 | - | 3.1 |
| 12 | 201511 | 15.2 | 173.2 | 76 | 1022.3 | 10.7 | 7.7 | 58.5 | - | 3 |
| 13 | 201512 | 10 | 90 | 66 | 1025.1 | 3.8 | 7.5 | 51.9 | - | 3.6 |
| 14 | 201601 | 6.2 | 100.3 | 66 | 1024.1 | | - | 73.3 | - | 3 |
| 15 | 201602 | 7.4 | 91.7 | 61.5 | 1024.6 | -0. | - | 142.1 | - | 3.1 |
| 16 | 201603 | 11 | 117.5 | 65 | 1021.1 | 4. | - | 198 | - | 2.4 |
| 17 | 201604 | 15.7 | 176.8 | 75 | 1013.7 | 10. | - | 175.7 | - | 2.4 |
| 18 | 201605 | 18.9 | 164.8 | 73 | 1012.3 | 13. | - | 198.5 | - | 2.1 |
| 19 | 201606 | 22 | 222 | 85.5 | 1008.2 | 19. | - | 110.5 | - | 2 |
| 20 | 201607 | 26.6 | 150.9 | 86 | 1008.4 | 23. | - | 201 | - | 2.2 |
| 21 | 201608 | 28.1 | 123.3 | 75.5 | 1005.8 | 23. | - | 270.4 | - | 2.4 |
| 22 | 201609 | 23.8 | 183.3 | 81.5 | 1012.1 | 20. | - | 120.5 | - | 2.2 |
| 23 | 201610 | 20.3 | 358 | 78.5 | 1017.2 | 16. | - | 89 | - | 2.5 |
| 24 | 201611 | 14.2 | 58 | 70.5 | 1021.7 | 8. | - | 134.7 | - | 2.4 |
| 25 | 201612 | 10 | 64.1 | 67.5 | 1023.9 | 4. | - | 133.6 | - | 2.6 |
| 26 | 201701 | 7.1 | 49.9 | 66.5 | 1024.5 | | - | 133 | - | 2.8 |
| 27 | 201702 | 7.4 | 60.5 | 60 | 1023.2 | -0. | - | 170.3 | - | 2.9 |
| 28 | 201703 | 10.1 | 42.3 | 59.5 | 1019.6 | | - | 191 | - | 2.6 |
| 29 | 201704 | 15.7 | 137 | 67.5 | 1014 | 8. | - | 194.9 | - | 2.4 |
| 30 | 201705 | 19.4 | 48.9 | 69 | 1013.4 | 12. | - | 250.6 | - | 2.1 |
| 31 | 201706 | 22.1 | 154.1 | 78.5 | 1009 | 17. | - | 177.3 | - | 2 |
| 32 | 201707 | 28.4 | 43.5 | 85.5 | 1008.6 | 25. | - | 196.1 | - | 2.3 |
| 33 | 201708 | 28.5 | 195.7 | 79 | 1007.2 | 24. | - | 241 | - | 2.3 |
| 34 | 201709 | 24.1 | 93.1 | 73.5 | 1011.9 | 18. | - | 148.1 | - | 2.5 |
| 35 | 201710 | 19.6 | 196.7 | 74 | 1018.6 | 14. | - | 153.3 | - | 2.8 |
| 36 | 201711 | 13.2 | 13 | 65 | 1022.3 | 6. | - | 166.7 | - | 2.2 |
| 37 | 201712 | 7.3 | 19.4 | 61 | 1025.4 | 0. | - | 130.6 | - | 2.8 |
| 38 | 201801 | 5.4 | 60.9 | 68 | 1023.2 | -0. | 0 | 118.6 | - | 2.7 |
| 39 | 201802 | 5.9 | 48.6 | 62 | 1023.1 | -1. | 0 | 135.2 | - | 2.8 |
| 40 | 201803 | 11.6 | 134.8 | 75.5 | 1019.1 | 7. | 0 | 191.8 | - | 2.6 |
| 41 | 201804 | 15.9 | 212.3 | 72.5 | 1015.9 | 10. | 0 | 197.5 | - | 2.3 |
| 42 | 201805 | 18.9 | 227.5 | 80 | 1012 | 14. | 0 | 153.1 | - | 2.5 |
| 43 | 201806 | 22.2 | 257.7 | 84 | 1008.1 | 19. | 0 | 130.7 | - | 2.2 |
| 44 | 201807 | 27 | 36 | 87 | 1008.1 | 24. | 0 | 229.8 | - | 2.3 |
| 45 | 201808 | 28.3 | 268.1 | 85 | 1006.5 | 25. | 0 | 237.1 | - | 2.8 |
| 46 | 201809 | 23.7 | 330.8 | 80 | 1012.2 | 19. | 0 | 136.7 | - | 2.4 |
| 47 | 201810 | 17.9 | 278.6 | 66.5 | 1018.2 | 11. | 0 | 199.6 | - | 2.5 |
| 48 | 201811 | 14.2 | 42.1 | 68 | 1022.5 | 8. | 0 | 163.4 | - | 2.1 |
| 49 | 201812 | 9.3 | 70.8 | 70 | 1025.6 | 3. | 0 | 88.2 | - | 2.6 |
| 50 | 201901 | 7.1 | 23 | 65 | 1025.4 | 0.9 | 5.9 | 136.6 | - | 2.4 |
| 51 | 201902 | 8.4 | 37.8 | 65 | 1023 | 1.9 | 6.8 | 127.8 | - | 2.3 |

### 문제점

'cloud(1/10)' 칼럼에 결측값은 없으나,

'-', '0' 으로 채워진 행이 많아

이러한 실질적인 결측값들을 채워주는 함수 필요

-> 2015, 2019, 2020년 데이터는 존재하므로

3년 각 월의 평균값으로 채워줄 예정

# 2.2. Data Preprocessing

## 8. 결측값 채우기 (방법1)

**1**

```
# date 컬럼의 "월"만 month 컬럼에 저장

jeju["month"] = 0
for i in range(len(jeju)):
    jeju["month"][i] = jeju["date"][i] % 100
jeju
```

→

| | date | month |
|---|---|---|
| 0 | 201501 | 1 |
| 1 | 201502 | 2 |
| 2 | 201503 | 3 |
| 3 | 201504 | 4 |
| 4 | 201505 | 5 |
| ... | ... | ... |
| 67 | 202008 | 8 |
| 68 | 202009 | 9 |
| 69 | 202010 | 10 |
| 70 | 202011 | 11 |
| 71 | 202012 | 12 |

**2**

```
# 최종 결측값 채우기 함수
def fill(column):
  # 평균 구하기
  meanList = [0 for i in range(12)]

  for i in range(len(jeju[column])):
    if jeju[column][i] == "-":
      jeju[column][i] = 0
    for j in range(12):
      if jeju["month"][i] == j+1:
        meanList[j] = meanList[j] + jeju[column][i]
  meanList = [round(x/3, 1) for x in meanList]

  # 평균으로 채우기
  for i in range(len(jeju[column])):
    if jeju[column][i] == 0:
      for j in range(12):
        if jeju["month"][i] == j+1:
          jeju[column][i] = meanList[j]
```

**3**

```
fill("cloud(1/10)")
jeju["cloud(1/10)"][13:60]
```

→

```
13    6.4
14    5.3
15    5.3
16    5.6
17    6.9
18    7.3
19    6.0
20    6.3
21    4.6
22    6.0
23    6.8
24    6.6
25    6.4
26    5.3
27    5.3
28    5.6
29    6.9
30    7.3
31    6.0
32    6.3
33    4.6
34    6.0
35    6.8
36    6.6
37    6.4
38    5.3
39    5.3
40    5.6
41    6.9
42    7.3
43    6.0
44    6.3
45    4.6
46    6.0
47    6.8
48    5.9
49    6.8
50    5.4
51    5.5
52    5.5
53    6.2
54    7.5
55    6.2
56    6.8
57    4.8
58      5
59    6.8
Name: cloud(1/10), dtype: object
```

결측값들이 월별 평균값으로 잘 대체됨을 확인!

# 2.2. Data Preprocessing

## 8. 결측값 채우기 (방법2)

```
# 평균운량 결측치 채우기
for i in range(len(jeju['cloud(1/10)'])):
    if jeju['cloud(1/10)'][i] == '-':
        jeju['cloud(1/10)'][i] = 0
jeju = jeju.astype({'cloud(1/10)':'float'})

for j in range(12):
    sum=jeju['cloud(1/10)'][j]+jeju['cloud(1/10)'][j+48]+jeju['cloud(1/10)'][j+60]
    mean = round(sum/3,1)
    for k in range(1,4):
        jeju['cloud(1/10)'][j+(12*k)]=mean
```

```
jeju['cloud(1/10)'][13:60]
```

```
13    6.4
14    5.3
15    5.3
16    5.6
17    6.9
18    7.3
19    6.0
20    6.3
21    4.6
22    6.0
23    6.8
24    6.6
25    6.4
26    5.3
27    5.3
28    5.6
29    6.9
30    7.3
31    6.0
32    6.3
33    4.6
34    6.0
35    6.8
36    6.6
37    6.4
38    5.3
39    5.3
40    5.6
41    6.9
42    7.3
43    6.0
44    6.3
45    4.6
46    6.0
47    6.8
48    5.9
49    6.8
50    5.4
51    5.5
52    5.5
53    6.2
54    7.5
55    6.2
56    6.8
57    4.8
58    5.0
59    6.8
Name: cloud(1/10), dtype: float64
```

결측값들이 월별 평균값으로 잘 대체됨을 확인!

# 2.2. Data Preprocessing

## 8. 결측값 채우기 (방법3)

```python
def fill(column):
    for i in range(len(jeju[column])):
        add = 0
        count = 0

        if (jeju[column][i] == '0') or (jeju[column][i] == '-'):
            month = jeju['month'][i]

            for j in range(len(jeju[column])):
                if (jeju['month'][j] == month) and (jeju[column][j] != 0) and (jeju[column][j] != '-'):
                        add += float(jeju[column][j])
                        count += 1

            jeju[column][i] = add / count
```

```python
fill('cloud(1/10)')
```

→

```python
jeju['cloud(1/10)'][13:60] #
```

```
13    4.775
14      4.0
15     3.95
16    4.175
17      5.2
18      5.5
19      4.5
20    4.725
21    3.425
22    4.525
23    5.075
24     4.95
25    4.775
26      4.0
27     3.95
28    4.175
29      5.2
30      5.5
31      4.5
32    4.725
33    3.425
34    4.525
35    5.075
36     4.95
37    4.775
38      4.0
39     3.95
40    4.175
41      5.2
42      5.5
43      4.5
44    4.725
45    3.425
46    4.525
47    5.075
48      5.9
49      6.8
50      5.4
51      5.5
52      5.5
53      6.2
54      7.5
55      6.2
56      6.8
57      4.8
58        5
59      6.8
Name: cloud(1/10), dtype: object
```

결측값들이 채워지긴 했으나, 값이 조금 틀림

원인: 이중 조건문 안에 잘못 구현된 부분이 있을 것으로 예상

# 2.3. EDA: Heatmap

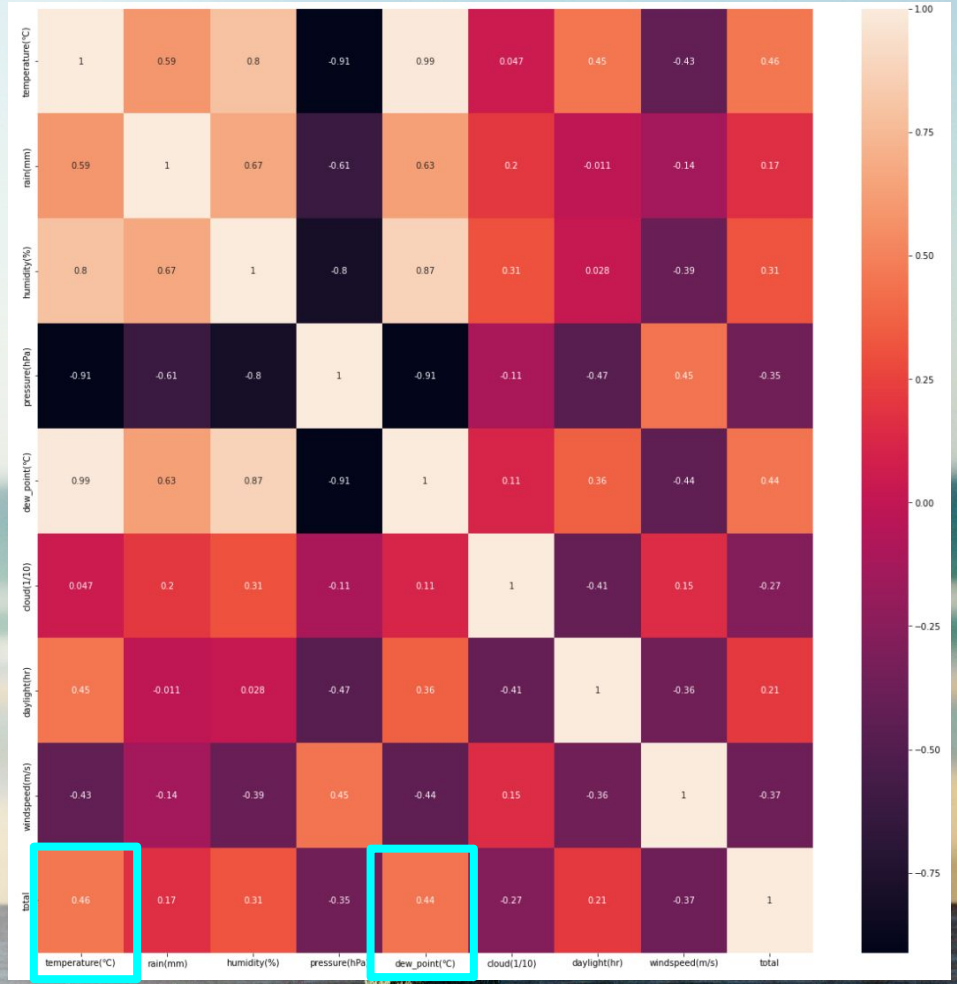
## 1. 히트맵을 통한 상관계수 확인

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

```
# 상관계수 plot 그리기
plt.figure(figsize=(20,20))
sns.heatmap(jeju.corr(), annot=True)
```

**r>0.8** : 강한 상관이 있다
**0.6<r<0.8** : 상관이 있다
**0.4<r<0.6** : 약한 상관이 있다

*total과 (어느 정도) 상관이 있어 보이는 변수 :*
temperature, dew_point



# 2.3. EDA: Pair plot

## 2. 다중공선성 확인

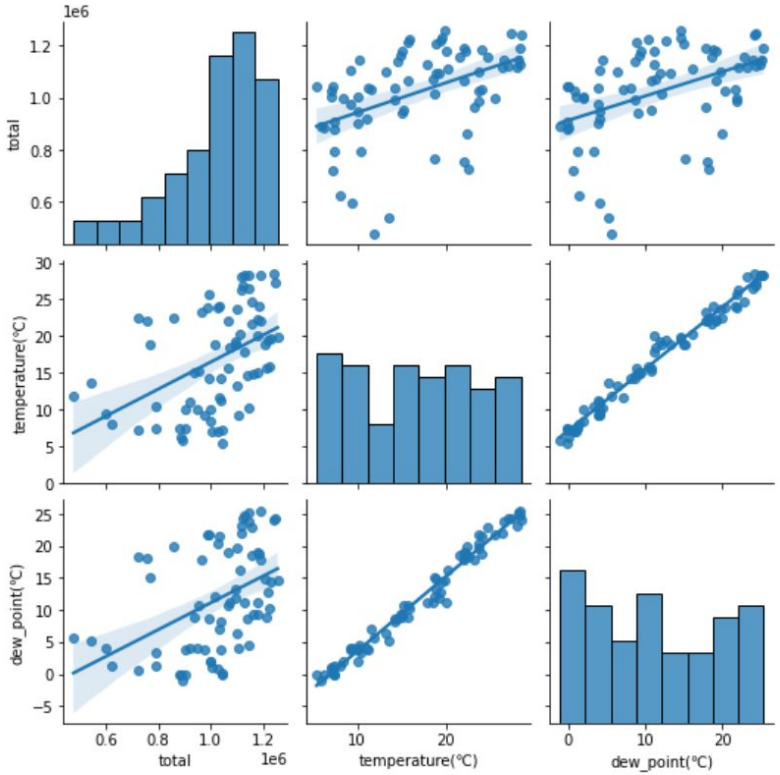
결과

- temperature과 dew_point 두 변수의 다중공선성 문제

  -> temperature 하나만 변수로 채택

- 모델링시 다중선형회귀가 아닌 단순선형회귀로 작업

```
# pairplot으로 더 살펴보기
sns.pairplot(jeju[['total','temperature(℃)', 'dew_point(℃)']], kind='reg')
plt.show()
```




하지만, *temperature*과 **dew_point*로 놓고 봤을 때 ***다중공선성*** *문제가 의심됨 때문에 독립변수를 **temperature* 하나만 갖고 **단순선형회귀**로 작업하는 게 나을 듯!

# Modeling

3.1. Linear Regression

3.2. Ridge Regression

3.3. Lasso Regression

3.4. Random Forest

# 3.1. Linear Regression

변수선택

```
from sklearn.model_selection import train_test_split
```

```
features = ['temperature(℃)']
X = jeju[features]
Y = jeju['total']

# train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,  random_state=97 )
```

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(57, 1)
(15, 1)
(57,)
(15,)
```

변수 : 다중공선성을 고려해

'temperature'만 사용

# 3.1. Linear Regression

## 모델링

```
1 from sklearn.linear_model import LinearRegression
2 from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error
```

```
1 linear = LinearRegression()
2 linear.fit(x_train, y_train)
```

4]: LinearRegression()

```
1 y_pred_linear = linear.predict(x_test)
2 y_pred_linear
```

5]:
```
array([1039506.6852639 ,  923589.99209105, 1117624.45674994,
       1110064.67241258,  937449.59670954, 1110064.67241258,
        895870.78285407, 1048326.43365748, 1089905.24751296,
       1141563.77381825,  952569.16538426, 1062186.03827597,
        972728.59028389, 1006747.619802  , 1128964.13325598])
```

```
1 print(f"RMSE: {mean_squared_error(y_test, y_pred_linear, squared=False)}")
2 print(f"R2 Score: {r2_score(y_test, y_pred_linear)}")
3 print(f"MAPE: {mean_absolute_percentage_error(y_test, y_pred_linear)}")
```

```
RMSE: 136488.25271320625
R2 Score: -0.08057096648892714
MAPE: 0.11081960648237686
```

RMSE: 136488

R2 Score: -0.08

MAPE: 0.11

# 3.2. Lasso Regression

변수선택

```
from sklearn.model_selection import train_test_split
```

```
features = ['temperature(℃)','dew_point(℃)']
X = jeju[features]
Y = jeju['total']

# train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,  random_state=97 )
```

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(57, 2)
(15, 2)
(57,)
(15,)
```

변수 :

temperature, dew_point 사용

# 3.2. Lasso Regression

모델링

```
1 from sklearn.linear_model import Lasso
2 from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error
```

```
1 Lasso = Lasso(alpha = 0.01)
2 Lasso.fit(x_train,y_train)
```

51]: Lasso(alpha=0.01)

```
1 y_pred_lasso = Lasso.predict(x_test)
2 y_pred_lasso
```

52]:
```
array([1067054.02300157,  910460.28342096, 1089026.09677053,
       1124907.99883053,  947013.51427146, 1089164.61293164,
        888415.80157551, 1078594.1245305 , 1087406.40686729,
       1128373.25489779,  931417.88799257, 1053872.97955434,
        984238.07391247, 1008507.46636704, 1127753.08216872])
```

```
1 print(f"RMSE: {mean_squared_error(y_test, y_pred_lasso, squared=False)}")
2 print(f"R2 Score: {r2_score(y_test, y_pred_lasso)}")
3 print(f"MAPE: {mean_absolute_percentage_error(y_test, y_pred_lasso)}")
```

```
RMSE: 136721.00255182383
R2 Score: -0.08425944741760305
MAPE: 0.11044378989140367
```

RMSE: 136721

R2 Score: -0.08

MAPE: 0.11

# 3.3. Ridge Regression

변수선택

```
from sklearn.model_selection import train_test_split
```

```
features = ['temperature(℃)','dew_point(℃)']
X = jeju[features]
Y = jeju['total']

# train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,  random_state=97 )
```

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(57, 2)
(15, 2)
(57,)
(15,)
```

변수 :

temperature, dew_point 사용

# 3.3. Ridge Regression

## 모델링

```
1 from sklearn.linear_model import Ridge
2 from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error
```

```
1 ridge = Ridge()
2 ridge.fit(x_train, y_train)
```

58]: Ridge()

```
1 y_pred_ridge = ridge.predict(x_test)
2 y_pred_ridge
```

59]:
```
array([1065797.7731267 ,  911117.73332188, 1090233.79879148,
       1124169.84598863,  946641.18900956, 1090032.12646212,
        888838.99269869, 1077209.09664165, 1087462.32646915,
       1128870.96087884,  932414.08244263, 1054210.74742673,
        983752.05318775, 1008434.06191155, 1127722.03748404])
```

```
1 print(f"RMSE: {mean_squared_error(y_test, y_pred_ridge, squared=False)}")
2 print(f"R2 Score: {r2_score(y_test, y_pred_ridge)}")
3 print(f"MAPE: {mean_absolute_percentage_error(y_test, y_pred_ridge)}")
```

```
RMSE: 136646.44325052897
R2 Score: -0.08307719184215445
MAPE: 0.11045329530614735
```

RMSE: 136646

R2 Score: -0.08

MAPE: 0.11

# 3.4. Random Forest

변수선택

```
from sklearn.model_selection import train_test_split
```

```
features = ['temperature(℃)','dew_point(℃)']
X = jeju[features]
Y = jeju['total']

# train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,  random_state=97 )
```

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(57, 2)
(15, 2)
(57,)
(15,)
```

변수 :

temperature, dew_point 사용

# 3.4. Random Forest

```
from sklearn import ensemble
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

```
nTreeList = [1, 10, 50, 100, 200]
depths = [None, 5, 10]
RMSE = []
R2 = []
MAPE = []

def mapefunc(y, pred):
    return np.mean((y-pred)/y * 100)

for iTrees in nTreeList:
    for depth in depths:
        Forest = RandomForestRegressor(n_estimators=iTrees, max_depth=depth, oob_score=False, random_state=42)
        Forest.fit(x_train, y_train)
        y_pred_forest = Forest.predict(x_test)

        mse = mean_squared_error(y_test, y_pred_forest)
        rmse = np.sqrt(mse)
        RMSE.append(rmse)

        r2 = r2_score(y_test, y_pred_forest)
        R2.append(r2)

        mape = mapefunc(y_test, y_pred_forest)
        MAPE.append(mape)

        print("nTrees : ", iTrees, "depths : ", depth)
        print("RMSE : ", rmse)
        print("R2 : ", r2)
        print("MAPE : ", mape)
        print('-'*70)
```

```
nTrees :  1 depths :  None
RMSE :  228166.45149144955
R2 :  -2.0197175474690647
MAPE :  6.200651345138279
----------------------------------------------------------------------
nTrees :  1 depths :  5
RMSE :  225755.70185781983
R2 :  -1.9562434938450983
MAPE :  6.704527319265531
----------------------------------------------------------------------
nTrees :  1 depths :  10
RMSE :  228166.45149144955
R2 :  -2.0197175474690647
MAPE :  6.200651345138279
----------------------------------------------------------------------
nTrees :  10 depths :  None
RMSE :  195932.24168152962
R2 :  -1.2267658943002968
MAPE :  5.967376134827151
----------------------------------------------------------------------
nTrees :  10 depths :  5
RMSE :  192022.8274071953
R2 :  -1.1387915838563933
MAPE :  5.372553775190203
----------------------------------------------------------------------
nTrees :  10 depths :  10
RMSE :  195932.24168152962
R2 :  -1.2267658943002968
MAPE :  5.967376134827151
----------------------------------------------------------------------
nTrees :  50 depths :  None
RMSE :  177769.6869513036
R2 :  -0.8330662508480169
MAPE :  5.345410007744311
----------------------------------------------------------------------
nTrees :  50 depths :  5
RMSE :  172556.2048211217
R2 :  -0.7271255426578331
MAPE :  4.139293480656602
----------------------------------------------------------------------
nTrees :  50 depths :  10
RMSE :  177705.4504242448
R2 :  -0.8317417445435149
MAPE :  5.338412537256086
----------------------------------------------------------------------
nTrees :  100 depths :  None
RMSE :  180116.2112610568
R2 :  -0.8817778503439628
MAPE :  5.319497372483043
----------------------------------------------------------------------
nTrees :  100 depths :  5
RMSE :  178061.75626225234
R2 :  -0.8390945249950101
MAPE :  4.101124412993121
----------------------------------------------------------------------
nTrees :  100 depths :  10
RMSE :  180017.3541591351
R2 :  -0.8797127829878355
MAPE :  5.322125452094394
----------------------------------------------------------------------
nTrees :  200 depths :  None
RMSE :  184960.07661159607
R2 :  -0.984352120227596
MAPE :  5.144007408601468
----------------------------------------------------------------------
nTrees :  200 depths :  5
RMSE :  180541.48076772183
R2 :  -0.8906744119164882
MAPE :  4.269645692518886
----------------------------------------------------------------------
nTrees :  200 depths :  10
RMSE :  184809.95713506496
R2 :  -0.9811323009180324
MAPE :  5.172684028095197
----------------------------------------------------------------------
```

# 3.4. Random Forest

```
new_R2 = []
for i in R2:
    new_R2.append(abs(1-i))
new_R2
```

```
[3.0197175474690647,
 2.9562434938450983,
 3.0197175474690647,
 2.226765894300297,
 2.1387915838563933,
 2.226765894300297,
 1.8330662508480169,
 1.7271255426578331,
 1.831741744543515,
 1.8817778503439628,
 1.8390945249950101,
 1.8797127829878355,
 1.984352120227596,
 1.8906744119164882,
 1.9811323009180324]
```

```
print("lowest RMSE : ", min(RMSE), "index : ", RMSE.index(min(RMSE)))
print('-'*70)

print("best R2 : ", R2[new_R2.index(min(new_R2))], "index : ", new_R2.index(min(new_R2)))
print('-'*70)

print("lowest MAPE : ", min(MAPE), "index : ", MAPE.index(min(MAPE)))
```

```
lowest RMSE :  172556.2048211217 index :  7
----------------------------------------------------------------------
best R2 :  -0.7271255426578331 index :  7
----------------------------------------------------------------------
lowest MAPE :  4.101124412993121 index :  10
```

lowest RMSE: 172556

best R2 Score: -0.72

lowest MAPE: 4.10

# Result

| | Linear | Lasso | Ridge | Random Forest |
|---|---|---|---|---|
| RMSE | 136488 | 136721 | 136646 | 172556 |
| R2 Score | -0.08 | -0.08 | -0.08 | -0.72 |
| MAPE | 0.11 | 0.11 | 0.11 | 4.10 |

- R2 Score: 네 모델 모두 음수이기 때문에 설명력이 낮음
- MAPE: 해당 값이 작은 Linear Regression, Lasso Regression, Ridge Regression이 비교적 적합한 모델인 것으로 보임

# DISCUSSION

4.1. Implication

4.2. Limitation

# 4.1. Implication

## 프로젝트 의의 # 1

- 기상 정보 데이터에 포함된 항목 중 "기온"이 관광객 수와 가장 큰 상관관계가 있다는 사실을 Heatmap을 통한 시각화로 한 눈에 확인

## 프로젝트 의의 # 2

- 다양한 모델 학습 방법(Linear Regression, Lasso Regression, Ridge Regression, Random Forest)을 이용하여 날씨에 따른 관광객 수 예측 -> 날씨에 따른 관광객수를 예측하여 관광 정책 수립 시에 활용 가능

# 4.2. Limitation

## 프로젝트 한계 #1

- 데이터의 개수가 절대적으로 부족해 모델의 정확성이 기대에 미치지 못함
  -> 날씨뿐 아닌 관광객 수에 영향을 줄 수 있는 다양한 변수를 추가해서 데이터 수를 늘려 프로젝트 보완 가능

## 프로젝트 한계 #2

- 아직 학습 모델에 대한 지식이 부족해 원하는 결과를 얻지 못함
  -> 머신러닝 모델에 대한 보다 심도있는 학습 후 더욱 적합한 모델로 프로젝트 보완 가능


Thank you
데빌