Report

Class 데이터과학

Title | Tashu 데이터 처리

학 과 명	컴퓨터공학과
교 수 명	이영석 교수님
학 번	201601989
이 름	김 진 섭
제 출 일	2019. 05.22

충남대학교
Chungnam National University

목 차

<과제 목표>

- RawData 처리

이번 과제에서는 타슈 대여 데이터, 날씨정보, 타슈 빌리는 위치에 대한 정보를 활용한 처리가 필요하다.

1. 타슈 대여 데이터
- 총 3년 상반기 하반기 포함하여 6개의 데이터
- 각각의 데이터가 header가 다를 수 도 있고, 포맷이 다를수도 있기 때문에 제어가 필요하다.

2. 날씨정보 데이터
- 3년의 데이터 정보가 있으며 시간 단위로 나누어진다.

3. 타슈를 빌리는 위치에 대한 정보
- Station 정보라고 하며 위치에 대한 정보가 있다.

- 데이터를 활용한 목표
> 데이터 분석 - 계절별, 요일별, 월별, 시간별 분석
> Top10 - 경로 분석
> 이용 경로(Chord diagram)
> 예측 영향도
> 하루 시간별 대여량 예측 모델 생성

사용하는 모델은 DT트리의 앙상블인 레인포레스트를 활용

- 데이터 시각화

바 차트 등을 활용하여 데이터를 시각화 하여 표시하여준다.

<Raw Data 처리>

- 2013년 상반기 데이터

libreOffice를 활용하여 회원인지에 대한 데이터를 삭제, csv로 변경

- 2013년 하반기 데이터

libreOffice를 활용하여 동일하게 처리

- 2014년 상반기 데이터

csv파일로 변경, 2013년도 헤더로 변경, 회원인지 구분과 총 대여시간 삭제

- 2014년 하반기 데이터

셀 서식을 활용하여 위 데이터와 동일하게 반납 일자와 대여 일자 변경
2013년도 헤더로 변경

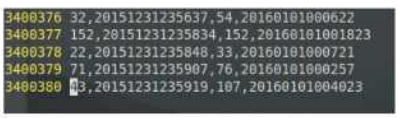
- 2015년 상반기 데이터

셀 서식 활용하여 위 데이터와 동일하게 반납 일자와 대여 일자 변경
2013년도 헤더로 변경

- 2015년 하반기 데이터

2013년도 헤더로 변경

해당 데이터들 모두 cat 명령어로 합쳐준다. (linux 가상머신 활용)
▶ tashu.csv 파일로 저장
▶ weather.csv 파일은 cat 명령어를 활용하여 3년치 데이터를 합해준다.
(추가로 weather의 날짜도 tashu와 동일한 포맷으로 세팅한다.)
▶ station.csv 파일은 기존에 있던 data에 있는 파일 사용



tashu.csv

station.csv

weather.csv

● 해당 데이터 완성 완료.

- total_rent.csv 파일 생성

Python을 활용하여 전처리를 수행하였다.

1. total_rent.csv 만들기

```
#전처리를 위해 필요한 패키지
import pandas as pd
import numpy as np
```

```
#날씨정보
weather = pd.read_csv('./Tashu/weather.csv')
weather.head()
```

	TIME	TEMPERATURE	RAINFALL	WINDSPEED	HUMIDITY	SNOWFALL
0	2013010100	-8.8	NaN	0.1	90.0	8.8
1	2013010101	-8.5	NaN	0.9	90.0	8.8
2	2013010102	-8.5	NaN	1.0	89.0	8.8
3	2013010103	-9.0	NaN	0.7	91.0	8.8
4	2013010104	-9.1	NaN	0.6	92.0	8.8

```
#station에 대한 정보
station = pd.read_csv('./Tashu/station.csv')
station = station.rename(columns = {'번호': 'STATION'})
station.head()
```

	STATION	키오스크번호	구별	명칭	위치	주소	거치대	좌표
0	1	1	유성구	무역전시관입구(택시승강장 앞)	엑스포다리 맞은편	유성구 도룡동 3-8	14	36.374325,127.387462
1	2	2	유성구	대전컨벤션 센터 앞	둔산대교 맞은편	유성구 도룡동 4-19	20	36.374472,127.392241
2	3	3	서구	한밭수목원(정문입구)	한밭수목원 내	서구 만년동 396	19	36.369855,127.388749
3	4	4	서구	초원아파트104동부근(버스정류장)	초원아파트 104동앞 목문 옥고 버스정류장 앞	서구 만년동 401	12	36.368192,127.379281
4	5	5	서구	둔산대공원 입구(버스정류장)	한밭수목원에서 평송수련원 가는길 버스정류장 앞	서구 둔산동 1521-10	13	36.365034,127.389361

- 날씨정보와 Station 정보를 가져온다.
- 전처리를 위해 필요한 패키지를 가져온다.

In [5]:
```
#필요없는 변수 삭제 (Station 변수)
del station['키오스크번호']
del station['구별']
del station['명칭']
del station['위치']
del station['주소']
del station['거치대']
del station['좌표']
station.head()
```

Out[5]:

	STATION
0	1
1	2
2	3
3	4
4	5

- station의 필요없는 변수는 제거한다.

```
#데이터 틀 만들어주기, for문 두개 중첩해서 시간별 station 생성
Dindex = -1
df1 = pd.DataFrame()
df2 = pd.DataFrame()
for i in station['STATION']:
    if(Dindex != -1) :
        df2['STATION'] = stationD
        df2['TIME'] = timeD
        df2['TEMPERATURE'] = tempD
        df2['RAINFALL'] = rainD
        df2['WINDSPEED'] = windD
        df2['HUMIDITY'] = humiD
        df2['SNOWFALL'] = snowD
        df1 = pd.concat([df1, df2],ignore_index=True)
    Dindex = Dindex + 1
    Windex = 0
    stationD = []
    timeD = []
    tempD = []
    rainD = []
    windD = []
    humiD = []
    snowD = []
    for j in weather['TIME']:
        stationD.append(station['STATION'][Dindex])
        timeD.append(weather['TIME'][Windex])
        tempD.append(weather['TEMPERATURE'][Windex])
        rainD.append(weather['RAINFALL'][Windex])
        windD.append(weather['WINDSPEED'][Windex])
        humiD.append(weather['HUMIDITY'][Windex])
        snowD.append(weather['SNOWFALL'][Windex])
        Windex = Windex+1
df = df1
df.head()
```

- 위 Weather 데이터와 station 데이터를 통하여 만들고 싶은 데이터를 생성한다.
- 우선 df1과 df2를 선언하여 둔다.
- for문을 돌며 Station의 위치정보 하나 당 날씨정보를 붙여주는 작업을 수행한다.
- 이중 for문을 활용하는데 내부 for문에서는 Station 하나 당 그 시간대에 맞는 weather 날씨를 붙여주고 외부 for문에서는 그 것을 Station의 개수만큼 해주는 용도로 사용한다.
- 즉 카디날리티 곱과 같은 행위를 수행하는 과정이다.
- 해당 데이터를 df에 넣어준다.

```
# 년 월 일 시간 분리하기.
rent_year = []
rent_month =[]
rent_day = []
rent_time = []

for D in df['TIME']:
    Year = list(map(str, list(str(D))[0]+list(str(D))[1]+list(str(D))[2]+list(str(D))[3]))
    rent_year.append(Year[0] + Year[1]+Year[2]+ Year[3])
    Month = list(map(str, list(str(D))[4]+list(str(D))[5]))
    rent_month.append(Month[0] + Month[1])
    Day = list(map(str, list(str(D))[6]+list(str(D))[7]))
    rent_day.append(Day[0] + Day[1])
    Time = list(map(str, list(str(D))[8]+list(str(D))[9]))
    rent_time.append(Time[0] + Time[1])

df['YEAR'] = rent_year
df['MONTH'] = rent_month
df['DAY'] = rent_day
df['HOUR'] = rent_time
df.head()
```

	STATION	TIME	TEMPERATURE	RAINFALL	WINDSPEED	HUMIDITY	SNOWFALL	YEAR	MONTH	DAY	HOUR
0	1	2013010100	-8.8	NaN	0.1	90.0	8.8	2013	01	01	00
1	1	2013010101	-8.5	NaN	0.9	90.0	8.8	2013	01	01	01
2	1	2013010102	-8.5	NaN	1.0	89.0	8.8	2013	01	01	02
3	1	2013010103	-9.0	NaN	0.7	91.0	8.8	2013	01	01	03
4	1	2013010104	-9.1	NaN	0.6	92.0	8.8	2013	01	01	04

- 년, 월, 일, 시간 데이터를 생성한다. for문과 list map을 사용하며 돌아준다.
- 기존에 가지고 있던 데이터 TIME에는 yyyymmddhh의 타입을 가지고 있으므로 해당 데이터를 쪼개어 구분 짓는 역할을 수행한다.

```
# 요일 정보 삽입
import datetime
weekday = []
i = 0
for go in df['TIME'] :
    weekday.append(datetime.datetime(int(df['YEAR'][i]),int(df['MONTH'][i]),int(df['DAY'][i])).weekday())
    i = i + 1
df['WEEKDAY'] = weekday
df.head()
```

	RENT_STATION	RENT_DATE	RETURN_STATION	RETURN_DATE
0	43.0	20130101055603	34.0	2.01301e+13
1	97.0	20130101060400	NaN	2.01301e+13
2	2.0	20130101060406	10.0	2.01301e+13
3	106.0	20130101105305	105.0	2.01301e+13
4	4.0	20130101112223	4.0	2.01301e+13

- 요일 정보를 삽입하여준다.
- for문을 df의 데이터 개수만큼 돌려주며 요일에 대한 정보를 datetime을 사용해 넣어준다.
- 요일은 0,1,2,3,4,5,6로 매칭되어 들어가게 된다.

```
#season에 대한 함수 정의
def season(month):
    if month >=3 and month <=5 : return 0 #봄
    if month >=6 and month <=8 : return 1 #여름
    if month >=9 and month <=11 : return 2 #가을
    else : return 3 #Winter

#Season 추가하기
df['SEASON'] = df.apply(lambda x:season(float(x['MONTH'])),axis = 1)
df.head(5)
```

	STATION	TIME	TEMPERATURE	RAINFALL	WINDSPEED	HUMIDITY	SNOWFALL	YEAR	MONTH	DAY	HOUR	WEEKDAY
0	1	2013010100	-8.8	NaN	0.1	90.0	8.8	2013	01	01	00	1
1	1	2013010101	-8.5	NaN	0.9	90.0	8.8	2013	01	01	01	1
2	1	2013010102	-8.5	NaN	1.0	89.0	8.8	2013	01	01	02	1
3	1	2013010103	-9.0	NaN	0.7	91.0	8.8	2013	01	01	03	1
4	1	2013010104	-9.1	NaN	0.6	92.0	8.8	2013	01	01	04	1

- season 칼럼을 생성하는 작업이다.
- 함수를 정의하여 해당 월이 어떤 계절인지 확인하여준다.
- 데이터프레임의 apply함수와 lambda를 활용하여 데이터를 넣어준다.

```
#Rent Count를 위해서 가져온다.
data = pd.read_csv('./Tashu/tashu.csv')
data.head(5)
```

	RENT_STATION	RENT_DATE	RETURN_STATION	RETURN_DATE
0	43.0	20130101055603	34.0	2.01301e+13
1	97.0	20130101060400	NaN	2.01301e+13
2	2.0	20130101060406	10.0	2.01301e+13
3	106.0	20130101105305	105.0	2.01301e+13
4	4.0	20130101112223	4.0	2.01301e+13

- 얼마나 Rent하였는지 RentCount를 하기 위하여 가져온다.

```
#Rent Count를 만들기 위한 비교변수를 생성하여준다.
for_Weather = []

for D in data['RENT_DATE']:
    Year = list(map(str, list(str(D))[0]+list(str(D))[1]+list(str(D))[2]+list(str(D))[3]))
    if Year[0] =="'" :
        Year = list(map(str, list(str(D))[1]+list(str(D))[2]+list(str(D))[3]+list(str(D))[4]))
        Month = list(map(str, list(str(D))[5]+list(str(D))[6]))
        Day = list(map(str, list(str(D))[7]+list(str(D))[8]))
        Time = list(map(str, list(str(D))[9]+list(str(D))[10]))
    else :
        Month = list(map(str, list(str(D))[4]+list(str(D))[5]))
        Day = list(map(str, list(str(D))[6]+list(str(D))[7]))
        Time = list(map(str, list(str(D))[8]+list(str(D))[9]))
    for_Weather.append(Year[0] + Year[1]+Year[2]+ Year[3]+Month[0] + Month[1]+Day[0] + Day[1]+Time[0] + Time[1])
data['Time'] = for_Weather
data['Count'] = 1
data.head()
```

- Rent Count를 하기 위해서 가지고 있는 RentTime 데이터를 위처럼 년도 월 일 시간만 표현하는 데이터로 만들어준다.
- 이는 groupby를 수행하여주기 위하여 Time과 Count를 생성하는 것이다.

```
#data 값을 그룹지어준다.
data = data.dropna()
data['Time'] = data['Time'].astype(int)
data['RENT_STATION'] = data['RENT_STATION'].astype(int)
CountGroup = data.groupby(['Time','RENT_STATION'])['Count'].sum().reset_index()

#CountGroup을 merge 시킬 준비
CountGroup['Time'] = CountGroup['Time'].astype(str)
CountGroup['RENT_STATION'] = CountGroup['RENT_STATION'].astype(str)
CountGroup['ForMerge'] = CountGroup['Time'] + CountGroup['RENT_STATION']
del CountGroup['Time']
del CountGroup['RENT_STATION']

#df에 merge하기 위한 작업
df['TIME'] = df['TIME'].astype(str)
df['STATION'] = df['STATION'].astype(str)
df['ForMerge'] =  df['TIME'] + df['STATION']

df = pd.merge(df, CountGroup, on = ['ForMerge'], how ='left')
df = df.fillna(0)
```

- Count값을 계산하여주는 코드이다.
- data들을 groupby로 묶어주로 sum()을 수행하여준다.
- CountGroup과 df의 merge하기 위하여 유도 변수를 생성하여주고 left로 merge를 수행하여준다.

```
# total_rent.csv로 생성한 데이터를  저장한다.
del df['TIME']
del df['ForMerge']
df.to_csv("./Tashu/total_rent.csv", index = False)
df.info()
```

- total_rent.csv로 저장하여준다.

2. total_return.csv 만들기

- 위와 동일하되 반납하는 일자로만 변경하여 수행하므로 설명은 생략

```
# total_return.csv로 생성한 데이터를  저장한다.
del df['TIME']
del df['ForMerge']
df.to_csv("./Tashu/total_return.csv", index = False)
df.info()
```

- total_return.csv로 저장하여준다.

< HW 1_a >

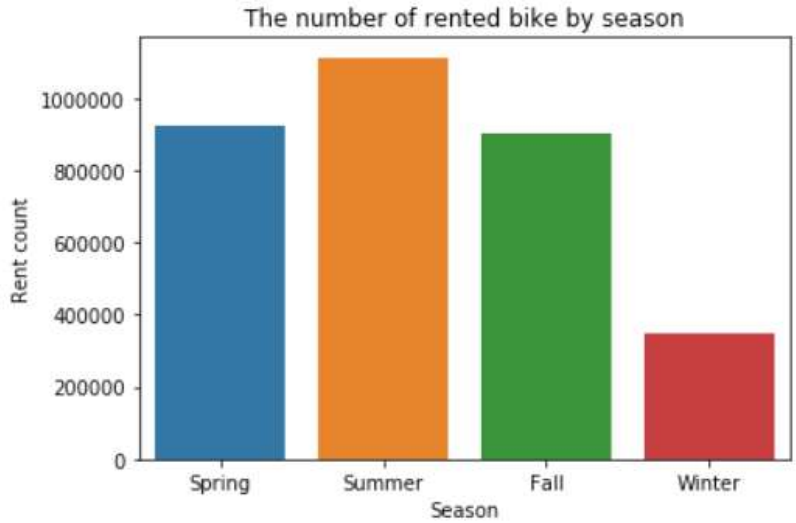
```
#필요한 패키지를 가져온다.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gmplot
import statistics
```

```
# 사용할 데이터를 가져온다.
train = pd.read_csv('./Tashu/total_rent.csv')
train.head(5)
```

- 사용할 데이터와 pakage를 가져옵니다.

- Season

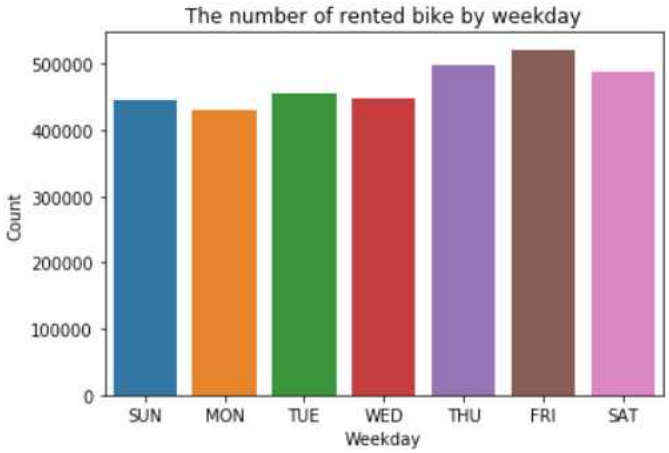
```
#계절별 데이터
group_season = train.groupby(['SEASON'])['Count'].sum().reset_index()
ax = sns.barplot(x = group_season['SEASON'], y=group_season['Count'])
ax.set(xlabel='Season', ylabel='Rent count')
season = [ 'Spring', 'Summer', 'Fall', 'Winter' ]
plt.xticks(np.arange(4), season)
plt.title('The number of rented bike by season')
plt.show()
```



- 계절별 데이터를 groupby를 활용하여 Count하여 바 그래프로 만들어 주었습니다.
- 여름에 가장 많이 대여를 하는 것을 알 수 있습니다.
- 겨울에 가장 적게 대여를 하는 것을 알 수 있습니다.

- WeekDay

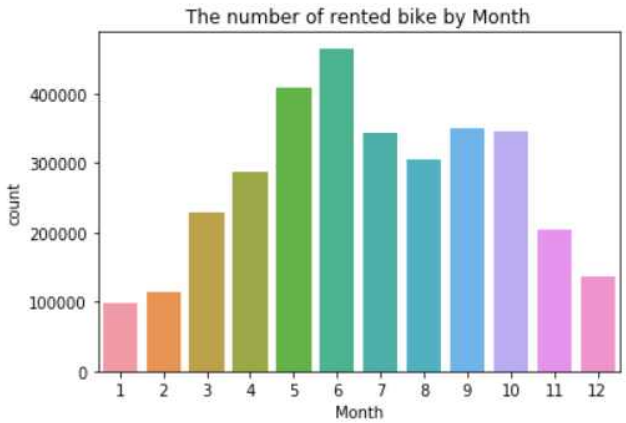
```
# 요일별 데이터
group_weekday = train.groupby(['WEEKDAY'])['Count'].sum().reset_index()
ax = sns.barplot(x=group_weekday['WEEKDAY'], y=group_weekday['Count'])
ax.set(xlabel='Weekday', ylabel='Count')
weekday = ['SUN', 'MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT']
plt.xticks(np.arange(7), weekday)
plt.title('The number of rented bike by weekday')
plt.show()
```



- 요일별 데이터를 Count하여 보여줍니다. 금요일에 가장 많이 대여를 하는 것을 확인할 수 있습니다.

- Month

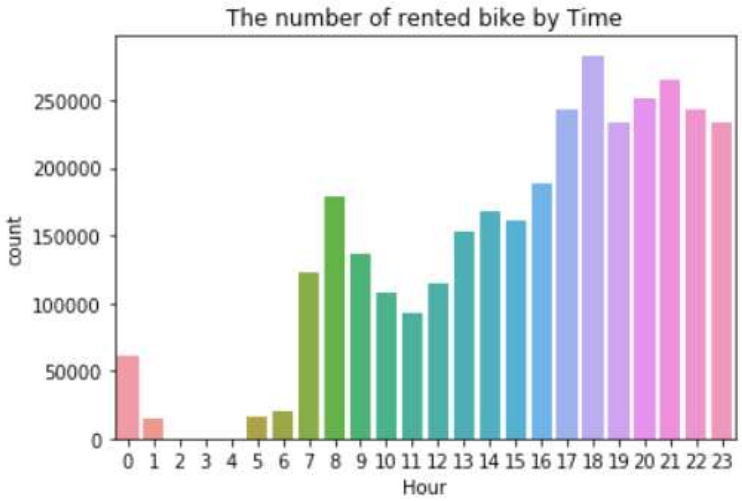
```
#월 별 데이터
group_mn = train.groupby(['MONTH'])['Count'].sum().reset_index()
ax = sns.barplot(x=group_mn['MONTH'], y=group_mn['Count'])
ax.set(xlabel='Month', ylabel='count')
plt.xticks(np.arange(12))
plt.title('The number of rented bike by Month')
plt.show()
```



- 월 별 대여수를 분석하기 위하여 만들 그래프이다.
- 6월에 가장 많이 대여를 하는 것을 알 수 있습니다.

- Hour

```
#시간 별 데이터
group_hr = train.groupby(['HOUR'])['Count'].sum().reset_index()
ax = sns.barplot(x=group_hr['HOUR'], y=group_hr['Count'])
ax.set(xlabel='Hour', ylabel='count')
plt.title('The number of rented bike by Time')
plt.show()
```



- 시간 별로 groupby를 활용하여 만들어낸 그래프입니다.
- 18시에 가장 많이 대여하는 것을 확인할 수 있습니다.

< HW 1_b >

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gmplot
import statistics
```

```
Tashu = pd.read_csv('./Tashu/tashu.csv')
Tashu = Tashu.dropna()
Tashu['Count'] = 1
Tashu = Tashu[Tashu['RENT_STATION'] != Tashu['RETURN_STATION']]
Tashu.head(5)
```

```
C:₩Users₩KIMJINSEOP₩Anaconda3₩lib₩site-packages₩IPython₩core₩interactiveshell.py:3049: DtypeWarning: Columns (1,3) have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

	RENT_STATION	RENT_DATE	RETURN_STATION	RETURN_DATE	Count
0	43.0	20130101055603	34.0	2.01301e+13	1
2	2.0	20130101060406	10.0	2.01301e+13	1
3	106.0	20130101105305	105.0	2.01301e+13	1
5	21.0	20130101113953	105.0	2.01301e+13	1
6	90.0	20130101120833	91.0	2.01301e+13	1

- 필요한 패키지와 Tashu 데이터를 가져온다. Tashu 데이터는 dropna를 활용하여 데이터를 지워주고 Count 유도 변수를 설정한다.
- RENT_STATION과 RETURN_STATION이 다른 것에 대하여만 처리를 하여주기 위하여 수행하여준다.(추가점수)

```
RentData = Tashu.groupby(['RENT_STATION', 'RETURN_STATION'])['Count'].sum().nlargest(10).reset_index()
RentData['RENT_STATION'] = RentData['RENT_STATION'].astype(int)
RentData['RETURN_STATION'] = RentData['RETURN_STATION'].astype(int)
RentData.head(10)
```

	RENT_STATION	RETURN_STATION	Count
0	21	105	17220
1	105	21	12154
2	56	32	11868
3	32	56	11118
4	105	22	8074
5	107	105	7912
6	22	105	7463
7	21	22	6556
8	1	3	6135
9	105	107	6065

- 가장 많이 사용하는 위치의 Top10을 계산하여준다.

```
station = pd.read_csv('./Tashu/Station.csv')
del station['키오스크번호']
del station['구별']
del station['명칭']
del station['위치']
del station['주소']
del station['거치대']
station = station.rename(columns = {'번호': 'RENT_STATION'})
station.head()
```

- 필요없는 Station 정보를 모두 지워준다.
- merge를 사용하기 위하여 del연산을 수행한다.

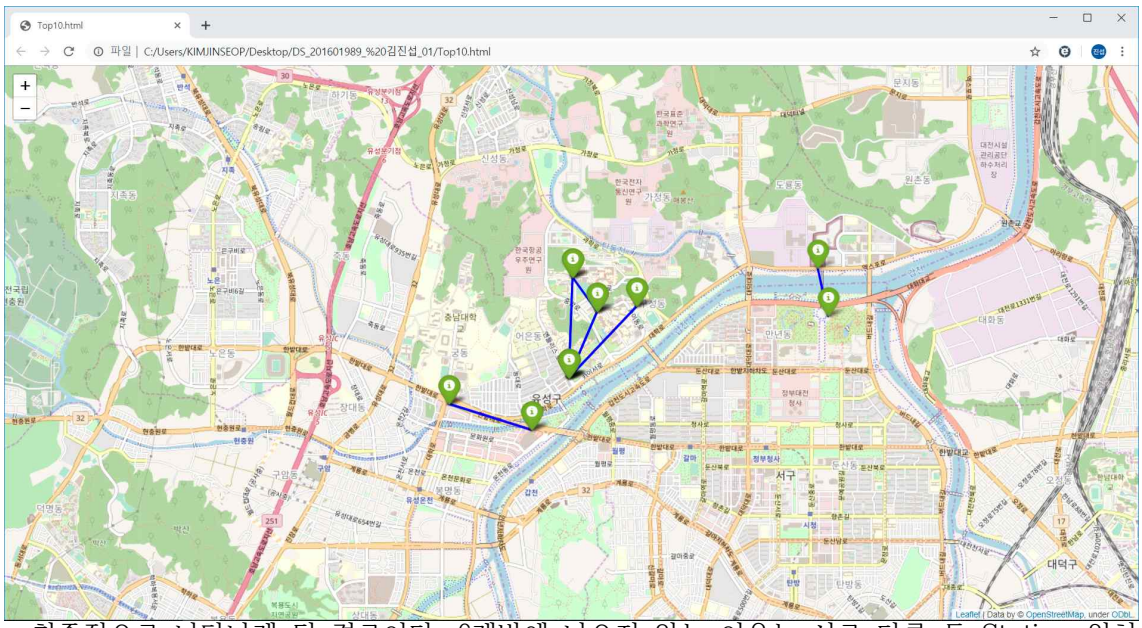
```
RentData = pd.merge(RentData,station, on = ['RENT_STATION'], how = 'left')
station = station.rename(columns = {'RENT_STATION': 'RETURN_STATION'})
RentData = RentData.rename(columns = {'좌표': 'RENT_LOC'})
RentData = pd.merge(RentData,station, on = ['RETURN_STATION'], how = 'left')
RentData = RentData.rename(columns = {'좌표': 'RETURN_LOC'})
RentData.head()
```

- merge를 활용하여 좌표정보를 넣어준다.

```
import folium
top10_lat= RentData['RENT_LOC'].apply(lambda e: e.split(",")[0]).astype(float)
top10_lon= RentData['RENT_LOC'].apply(lambda e: e.split(",")[1]).astype(float)
rtop10_lat= RentData['RETURN_LOC'].apply(lambda e: e.split(",")[0]).astype(float)
rtop10_lon= RentData['RETURN_LOC'].apply(lambda e: e.split(",")[1]).astype(float)
map_Top10 = folium.Map(location = [statistics.median(top10_lat),statistics.median(top10_lon)], zoom_start = 14)

index = 0
for i in top10_lat :
    folium.Marker(location=[top10_lat[index],top10_lon[index]],icon=folium.Icon(color='green')).add_to(map_Top10)
    folium.Marker(location=[rtop10_lat[index],rtop10_lon[index]],icon=folium.Icon(color='green')).add_to(map_Top10)
    folium.PolyLine(locations=[[top10_lat[index],top10_lon[index]],[rtop10_lat[index],rtop10_lon[index]]], color = 'blue').add_to(map_Top10
    index = index + 1
map_Top10.save('Top10.html')
```

- folium을 활용하여 지도를 그려주는데, 지도의 중심은 top10의 rentStation의 중앙으로 위치시켜주고, 각각의 점을 그리고 PolyLine을 활용해 이어준다.
- 그 정보를 Top10.html에 넣어준다.



- 최종적으로 나타나게 된 경로이다. 6개밖에 나오지 않는 이유는 서로 다른 두 Station 위치에서 Rent와 Return을 하는 위치가 반대되게 빌리는 경우가 있기 때문이다.

< HW 1_c >

```
import pandas as pd
import holoviews as hv

from holoviews import opts,dim
from bokeh.sampledata.les_mis import data

hv.extension('matplotlib')
hv.output(fig='svg', size=200)
```

- Chord를 그리기 위해서 저는 holoviews를 사용하였습니다.

```
Tashu = pd.read_csv('./Tashu/tashu.csv')
Tashu = Tashu.dropna()
Tashu['Count'] = 1
Tashu.head(5)
```

```
C:₩Users₩KIMJINSEOP₩Anaconda3₩lib₩site-packages₩IPython₩core₩interactiveshell.py:3049: DtypeWarning: Columns (1,3) have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

- 타슈 데이터를 가져옵니다.

```
RentData = Tashu.head(144)
del RentData['RENT_DATE']
del RentData['RETURN_DATE']
RentData['RENT_STATION'] = RentData['RENT_STATION'].astype(int)
RentData['RETURN_STATION'] = RentData['RETURN_STATION'].astype(int)
RentData.head(10)
```

- Tashu의 데이터를 가져오는데, 필요 없는 DATE정보를 지워줍니다. 그리고, Chord 그래프는 144개까지 표현하여줄 수 있으므로, 144개의 데이터만 가져왔습니다.

```
station = pd.read_csv('./Tashu/Station.csv')
del station['키오스크번호']
del station['구별']
del station['좌표']
del station['위치']
del station['주소']
del station['거치대']
station = station.rename(columns = {'번호': 'RENT_STATION'})
station.head()
```

- 이름 정보를 사용하기 위해서 station을 가져왔습니다. 그런데, 이름정보가 한글이라 깨지는 문제가 발생하여 여기서는 head()를 사용하였습니다.

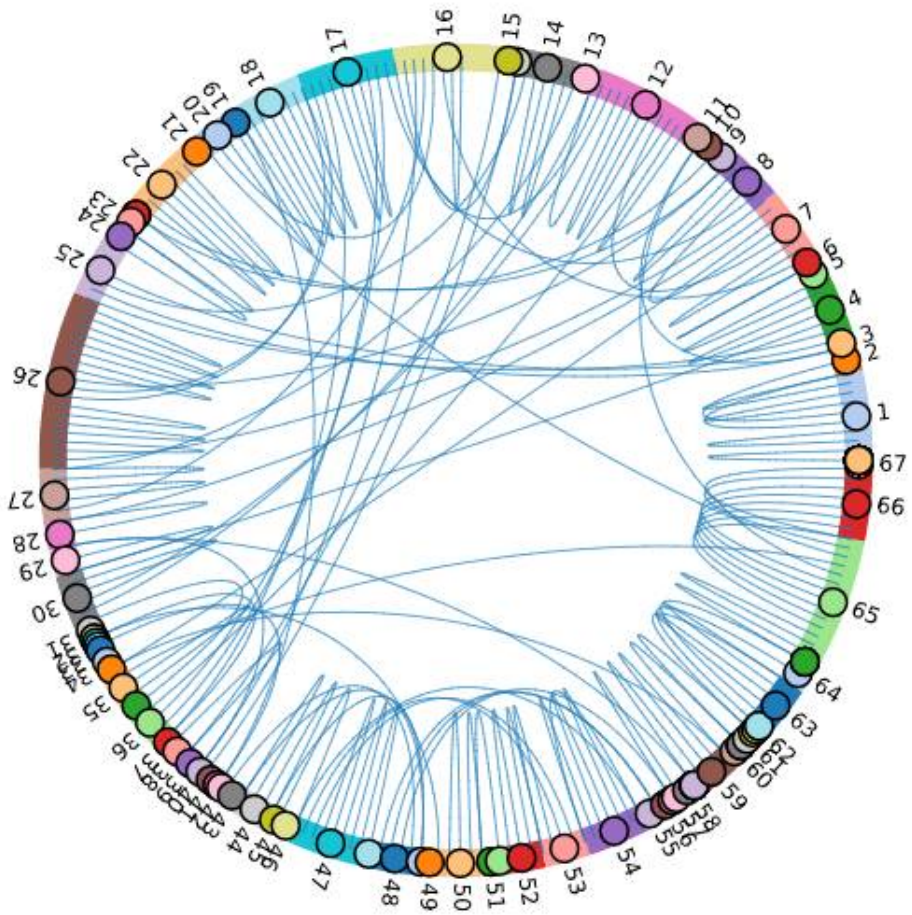
```
nodes = hv.Dataset(station, 'index')
nodes.data.head()
```

	index	RENT_STATION	명칭
0	0	1	무역전시관입구(택시승강장 앞)
1	1	2	대전컨벤션 센터 앞
2	2	3	한밭수목원(정문입구)
3	3	4	초원아파트104동부근(버스정류장)
4	4	5	둔산대공원 입구(버스정류장)

- nodes로 위 정보를 넣어줍니다.

```
hv.Chord((RentData, nodes)).select(value=(5, None)).opts(
    opts.Chord(cmap='Category20', edge_color=dim('source').astype(str), labels='RENT_STATION', node_color=dim('index').astype(str)))
```

- Chord의 label과 데이터들을 위처럼 설정하여줍니다.



최종적으로 나온 Chord 그래프입니다. 한글 표기를 하면 깨지는 문제가 발생하여 Station의 번호로 표기하였습니다.

< HW 2_a >

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
```

- 패키지를 가져온다.

```
train_data = pd.read_csv('./Tashu/total_rent.csv')
train_data['MONTH'] = train_data['MONTH'].astype('object')
train_data['HOUR'] = train_data['HOUR'].astype('object')
train_data['WEEKDAY'] = train_data['WEEKDAY'].astype('object')
train_data['SEASON'] = train_data['SEASON'].astype('object')
train_data.head(5)
```

	RENT_STATION	RETURN_STATION	RENT_YEAR	RENT_MONTH	RENT_DAY	RENT_TIME	SEA
0	43.0	34.0	2013	1	1	5	
1	97.0	NaN	2013	1	1	6	
2	2.0	10.0	2013	1	1	6	

- 사용할 데이터를 가져온다.

```
train_data = train_data.fillna(0)
del train_data['STATION']
del train_data['YEAR']
del train_data['DAY']
train_data.head()
```

	RENT_MONTH	RENT_TIME	SEASON	TEMPERATURE	RAINFALL	WINDSPEED	HUMIDITY	SI
0	1	5	3	-9.4	0.0	0.5	92.0	
1	1	6	3	-9.0	0.0	1.4	93.0	
2	1	6	3	-9.0	0.0	1.4	93.0	
3	1	10	3	-5.3	0.0	0.4	98.0	
4	1	11	3	-3.4	0.0	0.9	93.0	

- 학습에 필요 없는 칼럼 데이터를 삭제하여 줍니다.

```
X_train = train_data.iloc[:, :-1].values
y_train = train_data.iloc[:, -1].values
```

- 학습을 위한 데이터를 생성한다.

```
# Randomforest를 통한 학습 진행 tree 개수는 50개
rf = RandomForestRegressor(n_estimators= 50)
rf.fit(X_train, y_train)
```

- 학습을 할 Tree를 만들어주고 학습을 수행한다.

```
Importance = np.round(rf.feature_importances_,2)
print(Importance)
```

```
[0.26 0.04 0.04 0.09 0.   0.03 0.49 0.04 0.01]
```

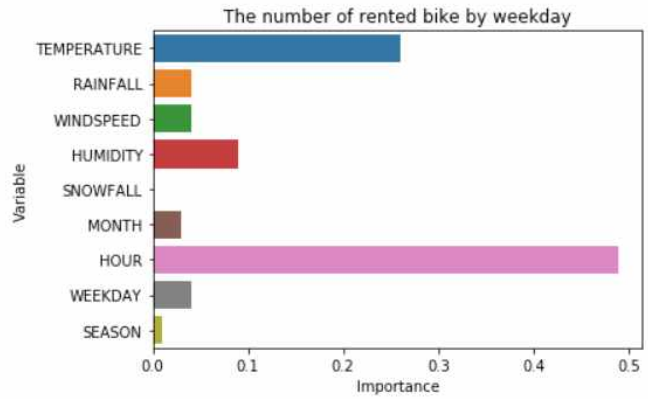
- 각 변수별 Importance를 계산하여 나타내어주는데 소수점 2째 자리수 까지 표현하여줍니다.

```
TrainColume = ('TEMPERATURE','RAINFALL','WINDSPEED','HUMIDITY','SNOWFALL','M
df ={'Imt' : Importance, 'Colume' : TrainColume}
df = pd.DataFrame(df)
print(df)
```

```
    Imt       Colume
0  0.26  TEMPERATURE
1  0.04     RAINFALL
2  0.04    WINDSPEED
3  0.09     HUMIDITY
4  0.00     SNOWFALL
5  0.03        MONTH
6  0.49         HOUR
7  0.04      WEEKDAY
8  0.01       SEASON
```

- 각각의 예측 영향도를 보여준다.

```
ax = sns.barplot(x=df['Imt'], y=df['Colume'])
ax.set(xlabel='Importance', ylabel='Variable')
plt.title('The number of rented bike by weekday')
plt.show()
```



- 최종 결과를 시각화하여 보여준다. 시간과 온도가 가장 큰 영향을 끼치는 것을 확인할 수 있다.

-

< HW 2_b >

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
```

- 패키지를 가져온다.

```
train_data = pd.read_csv('./Tashu/total_rent.csv')
train_data['MONTH'] = train_data['MONTH'].astype('object')
train_data['HOUR'] = train_data['HOUR'].astype('object')
train_data['WEEKDAY'] = train_data['WEEKDAY'].astype('object')
train_data['SEASON'] = train_data['SEASON'].astype('object')
train_data.head(5)
```

	STATION	TEMPERATURE	RAINFALL	WINDSPEED	HUMIDITY	SNOWFALL	YEAR	MONTH	DAY	HOUR	WEEKDAY	SEASON	Count
0	1	-8.8	0.0	0.1	90.0	8.8	2013	1	1	0	1	3	0.0
1	1	-8.5	0.0	0.9	90.0	8.8	2013	1	1	1	1	3	0.0
2	1	-8.5	0.0	1.0	89.0	8.8	2013	1	1	2	1	3	0.0
3	1	-9.0	0.0	0.7	91.0	8.8	2013	1	1	3	1	3	0.0
4	1	-9.1	0.0	0.6	92.0	8.8	2013	1	1	4	1	3	0.0

- train_data들을 모두 object 모드로 설정하여준다.

```
train_data = train_data[train_data.YEAR == 2015]
train_data = train_data[train_data.MONTH == 1]
train_data = train_data[train_data.DAY == 1]
train_data = train_data[train_data.STATION == 3]
train_data.tail()
train_data = train_data.fillna(0)
train_data.head()
```

	STATION	TEMPERATURE	RAINFALL	WINDSPEED	HUMIDITY	SNOWFALL	YEAR	MONTH	DAY	HOUR	WEEKDAY	SEASON	Count
70080	3	-4.1	0.3	4.2	82.0	0.5	2015	1	1	0	3	3	0.0
70081	3	-5.7	0.0	2.8	74.0	0.5	2015	1	1	1	3	3	0.0
70082	3	-6.4	0.0	2.6	60.0	0.5	2015	1	1	2	3	3	0.0
70083	3	-6.5	0.0	2.7	53.0	0.5	2015	1	1	3	3	3	0.0
70084	3	-6.6	0.0	2.5	61.0	0.5	2015	1	1	4	3	3	0.0

- 데이터를 2015년 1월 1일 3번 station으로 줄여서 확인하고, na값을 0으로 채워준다.

```
X = train_data['HOUR']
del train_data['STATION']
del train_data['YEAR']
del train_data['MONTH']
del train_data['WEEKDAY']
del train_data['DAY']
del train_data['HOUR']
del train_data['SEASON']
train_data = train_data.fillna(0)
train_data.head()
```

	TEMPERATURE	RAINFALL	WINDSPEED	HUMIDITY	SNOWFALL	Count
70080	-4.1	0.3	4.2	82.0	0.5	0.0
70081	-5.7	0.0	2.8	74.0	0.5	0.0
70082	-6.4	0.0	2.6	60.0	0.5	0.0
70083	-6.5	0.0	2.7	53.0	0.5	0.0
70084	-6.6	0.0	2.5	61.0	0.5	0.0

- 사용하지 않는 변수들을 삭제하여 준다.

```
X_train = train_data.iloc[:, :-1].values
y_train = train_data.iloc[:, -1].values
```
- X_train 데이터와 y_train 데이터를 가져온다.

```
predictions = rf.predict(X_train)
```

```
# RSME 계산
errors = np.sqrt(np.mean((predictions - y_train)**2))
print('Error:',round(errors,2), 'degrees.')
```

```
Error: 0.61 degrees.
```

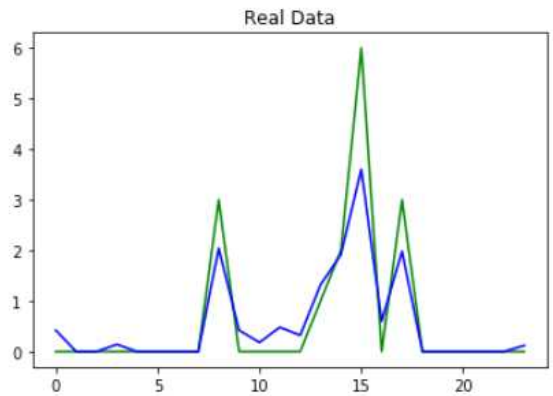
- Randomforest로 예측을 수행하고 오차를 계산하여준다.

오차의 결과는 RSME = 0.61 degrees정도임을 확인할 수 있다.

```
predictions = rf.predict(X_test)
```
- 예측을 수행한다.

```
plt.title('Real Data')
plt.plot(X,y_train,color='green',linestyle='solid')
plt.plot(X,predictions,color='blue',linestyle='solid')

plt.show()
```



- 최종 데이터에 대한 예측 결과를 실제 값과 예측값을 비교하여 보여주는 그래프이다.
- 초록색 선이 실제 데이터이다.
- 파란색 선이 예측 데이터이다.