

성별에 따른 데이터 시각화 및 나이브 베이즈이론을 활용한 예약 이행 여부 예측

데이터 출처 :

<https://www.kaggle.com/joniarroba/noshowappointments>

```
In [61]: import numpy as np #Numpy는 파이썬 언어를 위한 선형대수용 자료구조와 계산 함수를 제공
하는 라이브러리이다
import pandas as pd #구조화 된 데이터 조작 및 조작을 위한 데이터 munging 및 준비에 광범
위하게 사용
import matplotlib.pyplot as plt #히스토그램에서 라인 플롯, 히트 플롯까지 다양한 그래
프를 플로팅 할 수 있는 라이브러리
import seaborn as sns #Seaborn은 파이썬에서 매력적이고 유익한 통계 그래픽을 만들기 위한
라이브러리이다. matplotlib을 기반
```

```
In [62]: data = pd.read_csv('../LAB/data/No-show-Issue-Comma-300k.csv') #csv파일을
data라는 변수에 불러온다
```

```
In [63]: data.head(5) #불러온 csv파일이 들어있는 data 변수에서 맨 위에 5개의 데이터를 불러온다
```

Out[63]:

	Age	Gender	AppointmentRegistration	ApointmentData	DayOfTheWeek	Status	Diabet
0	19	M	2014-12-16T14:46:25Z	2015-01-14T00:00:00Z	Wednesday	Show-Up	0
1	24	F	2015-08-18T07:01:26Z	2015-08-19T00:00:00Z	Wednesday	Show-Up	0
2	4	F	2014-02-17T12:53:46Z	2014-02-18T00:00:00Z	Tuesday	Show-Up	0
3	5	M	2014-07-23T17:02:11Z	2014-08-07T00:00:00Z	Thursday	Show-Up	0
4	38	M	2015-10-21T15:20:09Z	2015-10-27T00:00:00Z	Tuesday	Show-Up	0

1. 데이터 점검

1.1 데이터 정리 및 오타 제거

일부 column의 오타를 알맞게 고치고 예약 등록날과 예약된 날의 간격의 수를 양수로 바꾸어준다.

```
In [64]: data.rename(columns = {'ApointmentData': 'AppointmentData', 'Alcoolism':
'Alcoholism', #철자가 틀린것을 고치기
'HiperTension': 'Hypertension', 'Handcap': 'Handicap'}, inplace = True)
```

##True일 경우 자기자신을 고치고 F

alse일 경우에는 바뀐 것을 다른 변수에 넣는 용도로 쓰인다
data.head(5)

Out[64]:

	Age	Gender	AppointmentRegistration	AppointmentData	DayOfTheWeek	Status	Diab
0	19	M	2014-12-16T14:46:25Z	2015-01-14T00:00:00Z	Wednesday	Show-Up	0
1	24	F	2015-08-18T07:01:26Z	2015-08-19T00:00:00Z	Wednesday	Show-Up	0
2	4	F	2014-02-17T12:53:46Z	2014-02-18T00:00:00Z	Tuesday	Show-Up	0
3	5	M	2014-07-23T17:02:11Z	2014-08-07T00:00:00Z	Thursday	Show-Up	0
4	38	M	2015-10-21T15:20:09Z	2015-10-27T00:00:00Z	Tuesday	Show-Up	0

```
In [65]: data.AwaitingTime = data.AwaitingTime.apply(abs) #등록날 과 예약된 날의 간격이  
음수로 되어있는데 이것을 절대값을 씌어주어 양수로 고쳐준다  
print(data.AwaitingTime.head())
```

```
0    29  
1     1  
2     1  
3    15  
4     6  
Name: AwaitingTime, dtype: int64
```

1.2 에러 체크 및 제거

데이터를 확인하여 NAN이나 맞지 않는 값을 찾아내 제거합니다

```
In [66]: print('Age:',sorted(data.Age.unique())) #각 데이터들의 잘못된 값이나 NAN값을 확  
인한다 SORTED는 값을 정렬해주고  
print('Gender:',data.Gender.unique()) #unique는 중복없이 값을 보여준다  
print('DayOfTheWeek:',data.DayOfTheWeek.unique())  
print('Status:',data.Status.unique())  
print('Diabetes:',data.Diabetes.unique())  
print('Alcoholism:',data.Alcoholism.unique())  
print('Hypertension:',data.Hypertension.unique())  
print('Handicap:',data.Handicap.unique())  
print('Smokes:',data.Smokes.unique())  
print('Scholarship:',data.Scholarship.unique())  
print('Tuberculosis:',data.Tuberculosis.unique())  
print('Sms_Reminder:',data.Sms_Reminder.unique())  
print('AwaitingTime:',sorted(data.AwaitingTime.unique()))
```

```
Age: [-2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,  
17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,  
35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52,  
53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70,  
71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88,  
89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 108
```

```
, 113]
Gender: ['M' 'F']
DayOfTheWeek: ['Wednesday' 'Tuesday' 'Thursday' 'Friday' 'Monday' 'Saturday' 'Sunday']
Status: ['Show-Up' 'No-Show']
Diabetes: [0 1]
Alcoholism: [0 1]
Hypertension: [0 1]
Handicap: [0 1 2 3 4]
Smokes: [0 1]
Scholarship: [0 1]
Tuberculosis: [0 1]
Sms_Reminder: [0 1 2]
AwaitingTime: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 133, 134, 135, 139, 140, 141, 142, 146, 147, 149, 153, 154, 156, 161, 162, 163, 168, 169, 170, 173, 174, 175, 176, 181, 182, 183, 184, 187, 189, 191, 195, 196, 203, 205, 206, 211, 217, 218, 224, 225, 226, 230, 231, 232, 238, 239, 245, 250, 253, 258, 259, 261, 265, 266, 268, 273, 275, 278, 279, 280, 281, 282, 287, 288, 294, 295, 296, 300, 301, 302, 303, 309, 314, 315, 321, 322, 329, 332, 334, 343, 349, 350, 398]
```

-1과 -2같이 잘못된 나이와 데이터 정리를 위해 100세 이상의 나이를 모두 제거 한다

```
In [67]: data = data[(data.Age >= 0) & (data.Age <= 100)]
```

1.3 전체 객체에서 남자 여자 비율 및 예약 이행 비율 확인

```
In [68]: df = pd.crosstab(index = data['Gender'], columns = data.Status).reset_index()
##crosstab은 그룹화할 행을 인덱스에 그룹화할 열을 columns에 넣고 각 행에 따른 Column의 개수를 보여준다
df['Sum'] = pd.DataFrame([(data['Gender']=='F').sum()], [(data['Gender']=='M').sum()])
df
```

Out[68]:

Status	Gender	No-Show	Show-Up	Sum
0	F	59879	140593	200472
1	M	30840	68655	99495

```
In [69]: all_oj = (data['Gender']=='M').sum() + (data['Gender']=='F').sum() # 전체 객체수
print("Man : ", round((data['Gender']=='M').sum() /all_oj , 5)*100 , '%') #round는 퍼센트를 구할때 유용한 라이브러리로, 5는 5개의 자리를
print("Woman : ",round((data['Gender']=='F').sum() /all_oj , 5)*100 , '%') #나타낸다는 것을 의미한다
```

```
Man : 33.169 %
Woman : 66.831 %
```

```
In [70]: FM = data[(data['Gender']== 'F')] #여자만 있는 데이터프레임 생성
f_oj = (data['Gender']== 'F').sum() # 여자 객체수
print("WoMan No-Show : ", round((FM['Status']== 'No-Show').sum() /f_oj ,
5)*100 , '%') #여자가 예약 이행 안한 비율
print("WoMan Show-up : ", 100-round((FM['Status']== 'No-Show').sum() /f_
oj , 5)*100 , '%') #여자가 예약 이행 한 비율
```

```
WoMan No-Show : 29.869 %
WoMan Show-up : 70.131 %
```

```
In [71]: MM = data[(data['Gender']== 'M')] #남자만 있는 데이터프레임 생성
m_oj = (data['Gender']== 'M').sum() # 남자 객체수
print("Man No-Show : ", round((MM['Status']== 'No-Show').sum() /m_oj , 5
)*100 , '%') #남자가 예약 이행 안한 비율
print("Man Show-up : ", 100-round((MM['Status']== 'No-Show').sum() /m_oj
, 5)*100 , '%') #남자가 예약 이행 한 비율
```

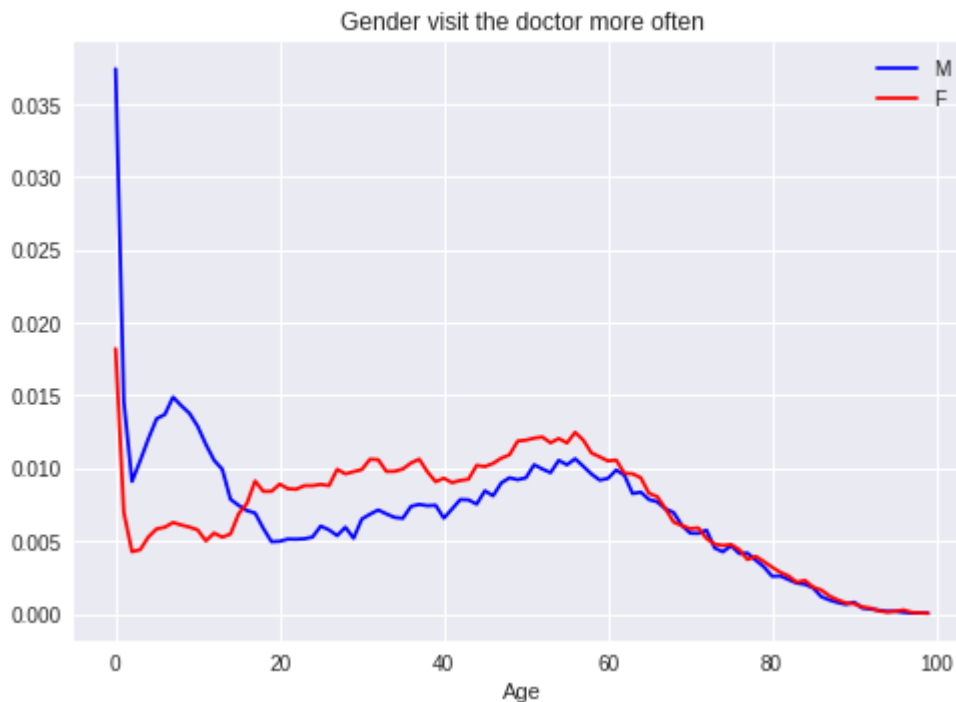
```
Man No-Show : 30.997 %
Man Show-up : 69.003 %
```

2. 데이터 탐구

2.1 여러가지 데이터를 이용하여 남자와 여자 예약 이행 여부 시각화

나이에 따른 예약 이행 여부 시각화

```
In [126]: df = data[data.Status == 'Show-Up'] ## 예약 이행한 것만 추출후 df에 저장
range_df = pd.DataFrame()
range_df['Age'] = range(100)
men = range_df.Age.apply(lambda x: len(df[(df.Age == x) & (df.Gender ==
'M')])) #이름없는함수이며 일회용으로 사용된다
women = range_df.Age.apply(lambda x: len(df[(df.Age == x) & (df.Gender =
= 'F')]))
plt.xlabel('Age') #X축의 이름
plt.title('Gender visit the doctor more often') #그래프의 이름 설정
plt.plot(range(100),men/(data['Gender']== 'M').sum(), 'b') #남자의 연령별 예
약 이행 비율
plt.plot(range(100),women/(data['Gender']== 'F').sum(), color = 'r') #여
자의 연령별 예약 이행 비율
plt.legend(['M', 'F'])
plt.show()
```



각종 질병 발생 시에 남녀 예약 이행 비율

```
In [19]: men_smoke = range_df.Age.apply(lambda x: len(df[(df.Age == x) & (df.Gender == 'M') & (df.Smokes == 1)])) #남자이며 흡연자
women_smoke = range_df.Age.apply(lambda x: len(df[(df.Age == x) & (df.Gender == 'F') & (df.Smokes == 1)])) #여자이며 흡연자

men_tension = range_df.Age.apply(lambda x: len(df[(df.Age == x) & (df.Gender == 'M') & (df.Hypertension == 1)])) #남자이며 고혈압
women_tension = range_df.Age.apply(lambda x: len(df[(df.Age == x) & (df.Gender == 'F') & (df.Hypertension == 1)])) #여자이며 고혈압

men_Diabetes = range_df.Age.apply(lambda x: len(df[(df.Age == x) & (df.Gender == 'M') & (df.Diabetes == 1)])) #남자이며 당뇨병
women_Diabetes = range_df.Age.apply(lambda x: len(df[(df.Age == x) & (df.Gender == 'F') & (df.Diabetes == 1)])) #여자이며 당뇨병

men_Tuber = range_df.Age.apply(lambda x: len(df[(df.Age == x) & (df.Gender == 'M') & (df.Tuberculosis == 1)])) #남자이며 결핵
women_Tuber = range_df.Age.apply(lambda x: len(df[(df.Age == x) & (df.Gender == 'F') & (df.Tuberculosis == 1)])) #여자이며 결핵

plt.figure(figsize = (15,15)) ## 그래프 사이즈
plt.subplot(221) ## 행 열 그리고 그 행열에 위치
plt.plot(range(100),men_smoke/men) # 세로축을 비율로 만들기위해 전체 남자에서 남자흡연자를 나눈다.
plt.plot(range(100),women_smoke/women, color = 'r')
plt.title('Smoking') #제목
plt.legend(['M','F'], loc = 2)
plt.xlabel('Age')

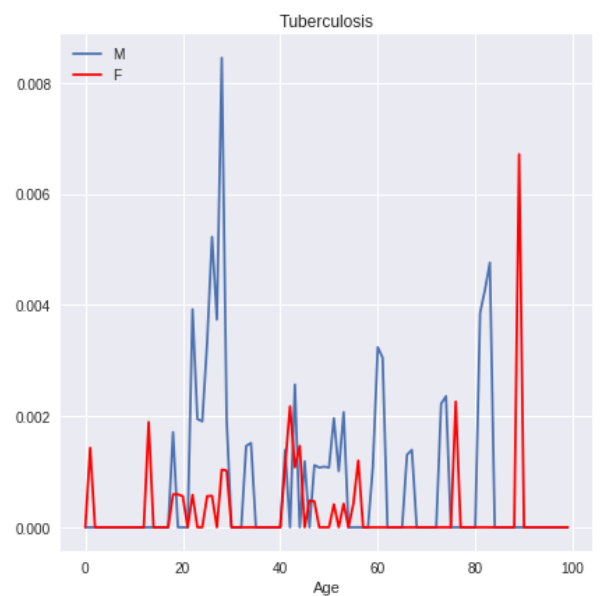
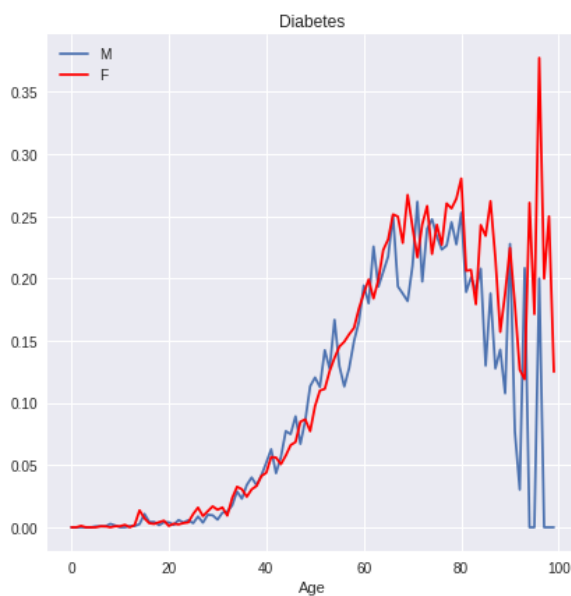
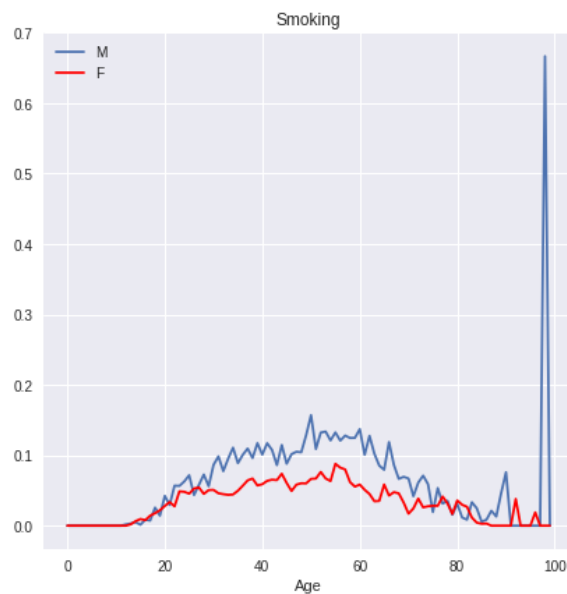
plt.subplot(222)
plt.plot(range(100),men_tension/men)
plt.plot(range(100),women_tension/women, color = 'r')
plt.title('Hyper Tension')
plt.legend(['M','F'], loc = 2)
```

```
plt.xlabel('Age')

plt.subplot(223)
plt.plot(range(100),men_Diabetes/men)
plt.plot(range(100),women_Diabetes/women, color = 'r')
plt.title('Diabetes')
plt.legend(['M','F'], loc = 2)
plt.xlabel('Age')

plt.subplot(224)
plt.plot(range(100),men_Tuber/men)
plt.plot(range(100),women_Tuber/women, color = 'r')
plt.legend(['M','F'], loc = 2)
plt.xlabel('Age')

plt.title('Tuberculosis')
plt.show()
```



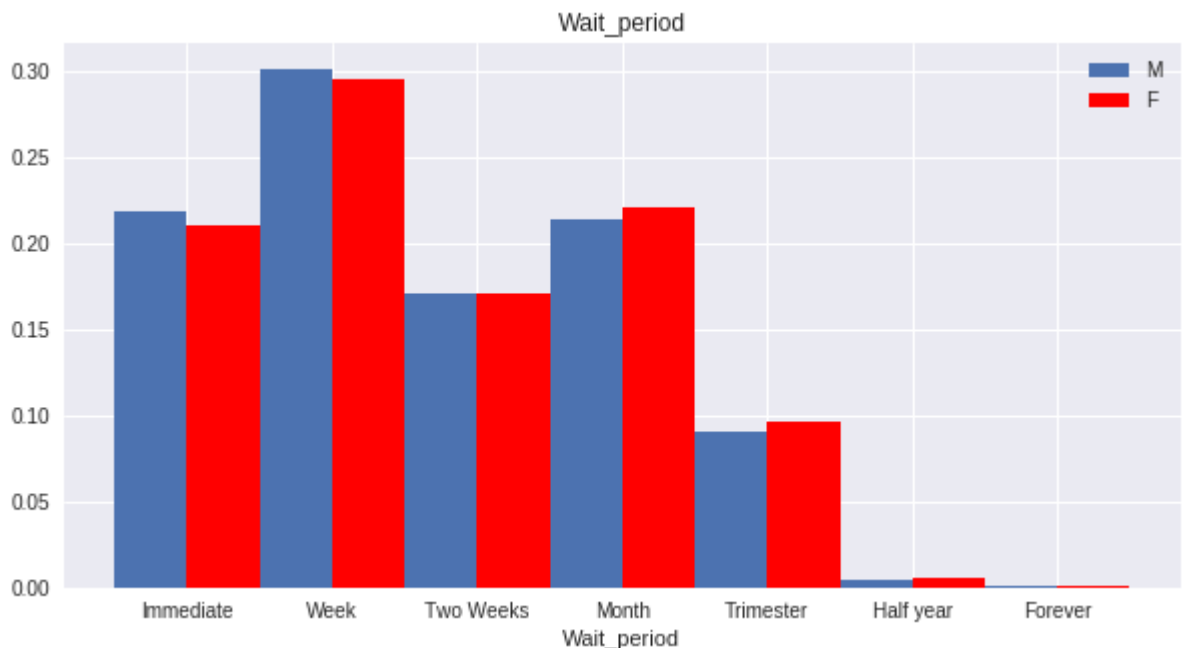
대부분의 질병은 남녀 비율이 비슷하지만 결핵에 한해서는 남자의 비율이 높다

예약 등록날과 예약 당일날과의 간격

```
In [86]: bins = [0, 2, 7, 14, 30, 90, 180, 9999] # 예약 시간과의 공백이 막연히 숫자로만 되어 있어서 문자로 만들어주기 위한 기준배열
labels = ["Immediate", "Week", "Two Weeks", "Month", "Trimester", "Half year", "Forever"]
wait_period = pd.cut(data.AwaitingTime, bins, labels=labels) # cut는 한 column에서 리스트를 사용하여 기준을 나눈다.
data['Wait_period'] = wait_period # 새로운 column을 만들어 이것을 넣고 예약 이행자들만 추출한다
df = data[data.Status == 'Show-Up']
wait_df = pd.DataFrame(labels)
men_wait = wait_df[wait_df.columns[0]].apply(lambda x: len(df[(df.Wait_period == x) & (df.Gender == 'M')]))
women_wait = wait_df[wait_df.columns[0]].apply(lambda x: len(df[(df.Wait_period == x) & (df.Gender == 'F')]))

plt.figure(figsize = (10,5)) # 그래프 사이즈
plt.bar(range(7), men_wait/len(df[df.Gender == 'M']), width = 0.5) # 막대를 7개 만들고 막대 크기를 0.5로 한다
plt.bar(range(7)+0.5*np.ones(len(range(7))), women_wait/len(df[df.Gender == 'F']), width = 0.5, color = 'r')
plt.xticks(range(7) + 0.25*np.ones(len(range(7))), labels) # x축의 라벨 삽입

plt.title('Wait_period')
plt.xlabel('Wait_period')
plt.legend(['M', 'F'])
plt.show()
```



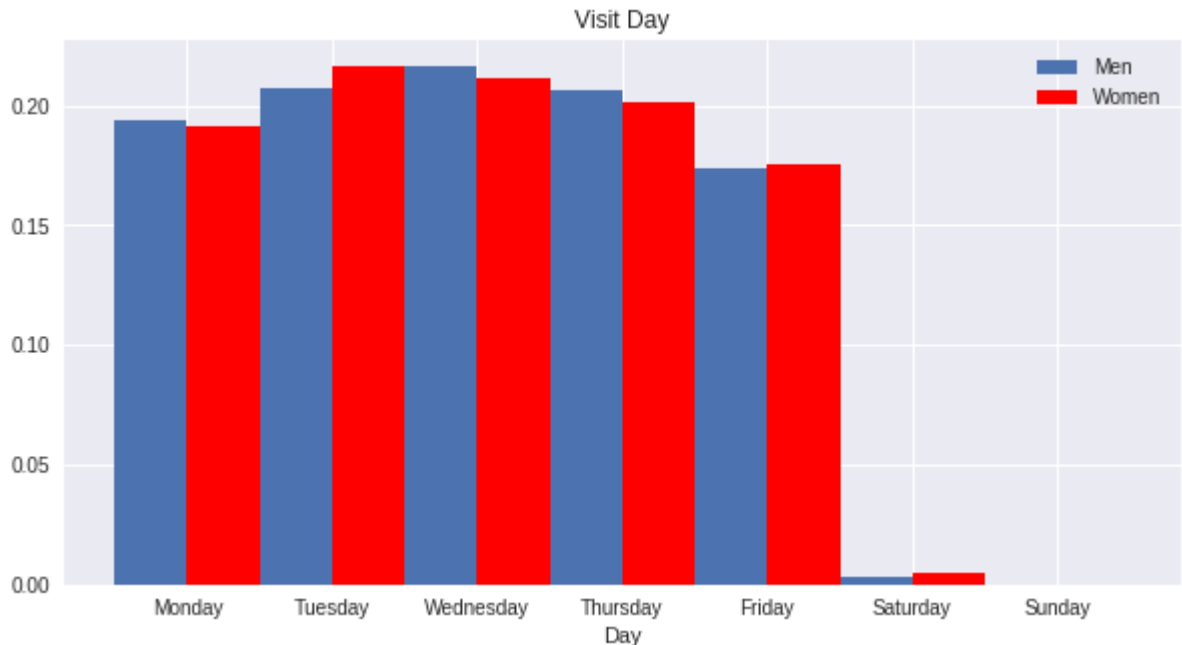
어떤 요일을 가장 많이 선호하는가에 대한 시각화

```
In [87]: Days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'] # 요일을 리스트로 만든다
Days_df = pd.DataFrame(Days) # 그것을 데이터 프레임으로 만든다

men_days = Days_df[Days_df.columns[0]].apply(lambda x: len(df[(df.DayOfTheWeek == x) & (df.Gender == 'M')]))
women_days = Days_df[Days_df.columns[0]].apply(lambda x: len(df[(df.DayOfTheWeek == x) & (df.Gender == 'F')]))
```

```
plt.figure(figsize = (10,5))
plt.bar(range(7), men_days/len(df[df.Gender == 'M']), width = 0.5)
plt.bar(range(7)+0.5*np.ones(len(range(7))), women_days/len(df[df.Gender == 'F']), width = 0.5, color = 'r')
plt.xticks(range(7) + 0.25*np.ones(len(range(7))),Days)

plt.title('Visit Day')
plt.xlabel('Day')
plt.legend(['Men', 'Women'])
plt.show()
```



예약 공백기간에서 남자들은 최근 예약을 잘 이행하는 반면에 여자들은 오래 된 예약을 더 잘 지키는 경향이 있다 요일 같은 경우에는 주말에 멀수록 예약자들이 많았고 남녀 비율은 요일마다 들쭉날쭉하므로 큰 영향이 없어보였다.

월 별 예약 이행자에 대한 시각화

```
In [219]: dts = data.copy() #데이터복사를 한 이유는 원본 데이터가 들어있는 변수에 영향을 주지 않기
           #위함이다
           dts['Month'] = dts['AppointmentData'].apply(lambda x: x[5:7]) #5:7은 날짜
           #에서 월만 추출한다는것이다
           dts = dts[dts.Status == 'Show-Up']
           dts= dts[["Month", "Gender"]]

           dts = pd.crosstab(index = dts['Month'], columns = dts.Gender) #월을 인덱스
           #에 컬럼에 성별을 넣는 새로운 데이터프레임 생성

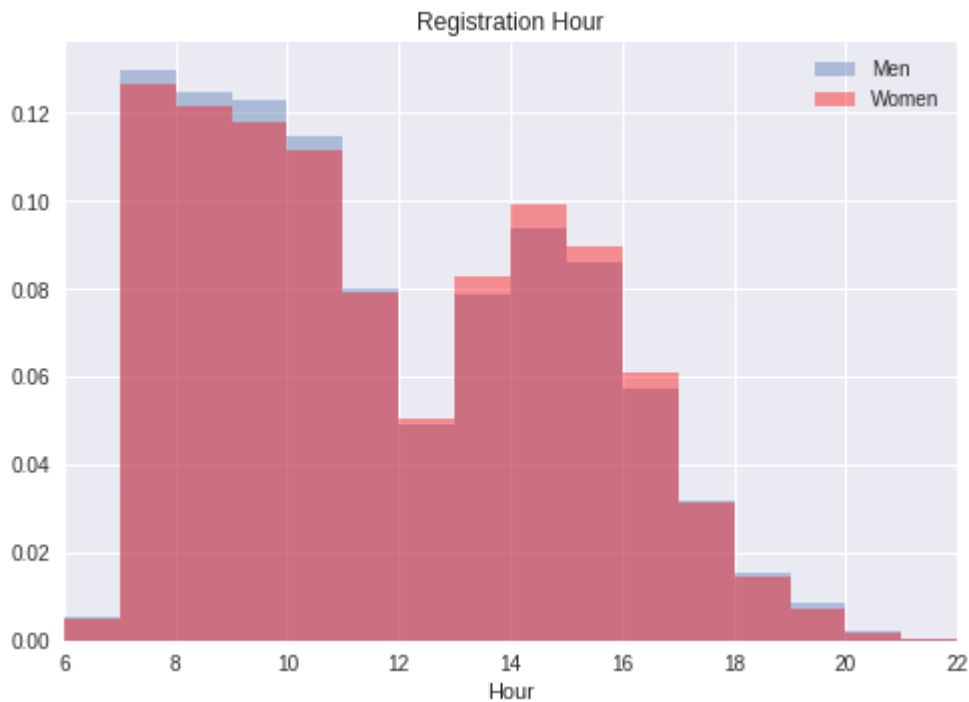
           dts['F'] = round((dts['F']/f_oj),3) *100
           dts['M'] = round((dts['M']/m_oj),3) *100
           plt.figure(figsize=(11,8))
           sns.heatmap(dts, annot=True, fmt='g') ##annot는 각 셀에 값을 표시할지 여부 fmt는 포
           #현 방식 f는 실수형
           plt.show()
```




병원 이용객은 여름 가을에 가장 많았으며 특히 휴가철인 7월에 절정을 찍었다. 하지만 남녀의 차이에는 큰 영향이 없었다.

하루 중에 어떤 시간에 가장 많이 예약을 등록하러 오는지에 대한 시각화

```
In [105]: data['Time'] = data['AppointmentRegistration'].apply(lambda x: int(x[11:13])) ##커널밀도 추정 히스토그램을 스무디하게 평활화
sns.distplot(data.Time[data.Gender == 'M'], bins = range(24), norm_hist = True, kde = False)
sns.distplot(data.Time[data.Gender == 'F'], bins = range(24), norm_hist = True, kde = False, color = 'r')
##norm_hist y의 값을 퍼센트 값
plt.xlim([6,22]) #x축 범위 설정 병원이 6시부터 22시까지 운영하기 때문이다
plt.title('Registration Hour')
plt.xlabel('Hour')
plt.legend(['Men', 'Women'])
plt.show()
```



남자들은 주로 아침에 많이 예약을 등록하고 여자는 낮에 많이 예약을 등록한다

2.2 두 가지 질병을 가진 사람의 예약 이행 여부 비교

지금까지는 한 가지의 조건만으로 비교했는데 이번엔 알코올 의존증과 폐암을 동시에 비교한다

```
In [119]: data2 = data.copy()
data2.eval("Show = Status == 'Show-Up'", inplace=True) # inplace는 새 dataframe
을 반환할지 기존 것을 변경할지말이다
# 기본적으로 false로

설정된다
data2.eval("No = Status == 'No-Show'", inplace=True) # eval은 파이썬 표현식
만 포함 가능하다
groups = data2[['Gender', 'Smokes', 'Alcoholism', "Show", "No"]].groupby
(['Gender', 'Smokes', 'Alcoholism'])
# 컬럼을

가져오고 groupby는 그중에서 색인으로 할것을 뽑는다
gps = pd.DataFrame(groups.mean()) # 데이터 프레임 생성
gps["No-count"] = groups.sum()["No"] # 예약 불 이행자의 객체수를 더한다
gps["Sh-count"] = groups.sum()["Show"] # 예약 이행자의 객체수를 더한다
gps
```

Out[119]:

			Show	No	No-count	Sh-count
Gender	Smokes	Alcoholism				
F	0	0	0.704274	0.295726	56211.0	133867.0
		1	0.649186	0.350814	388.0	718.0
	1	0	0.664078	0.335922	2509.0	4960.0
		1	0.576141	0.423859	771.0	1048.0
M	0	0	0.693024	0.306976	27964.0	63131.0
		1	0.663296	0.336704	666.0	1312.0

	1	0	0.681319	0.318681	1218.0	2604.0
		1	0.618462	0.381538	992.0	1608.0

신기하게도 두 가지 병을 앓고 있는 사람들의 예약 이행 비율이 건강한사람들보다 더 작았고 이것은 남녀 모두에게 나타났다.

밀접한 관계의 합병증

당뇨병을 앓고 있는 사람들은 고혈압을, 고혈압을 가지고 있는 사람들은 당뇨병을 가지고 있는 경우가 많은데 그 만큼 두 가지의 병의 관계는 깊다. 그래서 이번엔 밀접한 관계에 있는 두 가지의 병을 묶어서 수치를 계산해 보았다.

```
In [25]: groups = data2[['Gender', 'Hypertension', 'Diabetes', "Show", "No"]].group
by(['Gender', 'Hypertension', 'Diabetes'])
gps = pd.DataFrame(groups.mean())
gps["No-count"] = groups.sum()["No"]
gps["Sh-count"] = groups.sum()["Show"]
gps
```

Out[25]:

			Show	No	No-count	Sh-count
Gender	Hypertension	Diabetes				
F	0	0	0.686039	0.313961	47631.0	104079.0
		1	0.735242	0.264758	740.0	2055.0
	1	0	0.749374	0.250626	8113.0	24258.0
		1	0.750294	0.249706	3395.0	10201.0
M	0	0	0.675870	0.324130	25675.0	53537.0
		1	0.707723	0.292277	439.0	1063.0
	1	0	0.745917	0.254083	3376.0	9911.0
		1	0.754277	0.245723	1350.0	4144.0

밀접한 관계에 있는 당뇨와 고혈압의 경우에는 위와는 정반대로 약속 이행 비율이 더 높게 나타났다 이것 또한 남녀 모두에게 나타났다

3. 데이터 예측

코드 출처 <https://www.kaggle.com/somrikbanerjee/d/joniarroba/noshowappointments/predicting-show-up-no-show>

먼저 반의 데이터로 기계학습을 한 뒤에 이것을 기반으로 나머지 반의 데이터의 예약 이행 여부 확률을 구한다

남자의 경우는 홀수개의 데이터이기 때문에 하나를 제외한다

```
In [121]: FM_train = FM[['Age', 'Diabetes', 'Hypertension', 'Tuberculosis', 'Smokes',
                        '# 나이, 질병, 고혈압, 당뇨병, 흡연등의 컬럼을 가져와
                        'Alcoholism', 'Scholarship']].iloc[:100236] #학습을 시킨다
F_train = FM.Status[:100236]

FM_test = FM[['Age', 'Diabetes', 'Hypertension', 'Tuberculosis', 'Smokes',
               'Alcoholism', 'Scholarship']].iloc[100236:]
F_test = FM.Status[100236:]

MM_train = MM[['Age', 'Diabetes', 'Hypertension', 'Tuberculosis', 'Smokes',
               'Alcoholism', 'Scholarship']].iloc[:49747]
M_train = MM.Status[:49747]
MM_test = FM[['Age', 'Diabetes', 'Hypertension', 'Tuberculosis', 'Smokes',
               'Alcoholism', 'Scholarship']].iloc[49747:]
M_test = FM.Status[49747:]
```

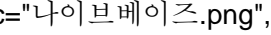
naive_bayes --- <http://laonple.blog.me/220867768192>

naive_bayes

나이브 베이즈는 분류기를 만들 수 있는 간단한 기술로써 단일 알고리즘을 통한 훈련이 아닌 일반적인 원칙에 근거한 여러 알고리즘들을 이용하여 훈련된다. 모든 나이브 베이즈 분류기는 공통적으로 모든 특성 값은 서로 독립임을 가정한다.

나이브 베이즈의 장점.

- 첫째, 일부의 확률 모델에서 나이브 베이즈 분류는 지도 학습 (Supervised Learning) 환경에서 매우 효율적으로 훈련 될 수 있다.
- 둘째, 분류에 필요한 파라미터를 추정하기 위한 트레이닝 데이터의 양이 매우 적다는 것이다.
- 셋째, 간단한 디자인과 단순한 가정에도 불구하고, 나이브 베이즈 분류는 많은 복잡한 실제 상황에서 잘 작동한다.

MultinomialNB는 다중 분산 데이터에 대한 나이브 베이즈인 알고리즘을 구현하고 텍스트 분류에 사용되는 2 개의 고전적인 나이브 베이즈인의 변형 중 하나이다.  클래스와 특성의 값이 훈련 데이터에서 발생 하지 않는 경우에는, 빈도수 기반의 확률 추정치는 0이 된다. 이것은 이 추정치가 곱해질 때 다른 확률의 모든 정보를 없앨 수 있기 때문에 문제가 될 수 있다. 따라서, 대부분의 확률 추정에서는 그 값이 0이 되지 않도록 가짜 수(pseudocount)라는 작은 샘플 보정값을 통합하여 사용하는 경우가 대부분이다. 나이브 베이즈의 이러한 정규화 방법은 가짜 수(pseudocount) 1일 경우 라플라스 정규화(Laplace smoothing)라고 불리고, 일반적으로 리드스톤 정규화(Lidstone smoothing)라고 불린다.

```
In [139]: from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import MultinomialNB

clf = MultinomialNB().fit(FM_train, F_train)
print('women Accuracy:', round(accuracy_score(F_test,
```

```
                                clf.predict(FM_test)), 5) * 100,  
                                '%')  
  
clf2 = MultinomialNB().fit(MM_train, M_train)  
print('man Accuracy:', round(accuracy_score(M_test,  
                                clf2.predict(MM_test)), 5) * 100  
      , '%')
```

women Accuracy: 70.029 %

man Accuracy: 70.143 %



유방암 진단 데이터

로지스틱 회귀분석(Logistic Regression)을 이용한 유방암 환자 예측

[데이터 다운로드](#)

[소스코드 출처](#)

데이터 설명

1. ID 번호
2. 진단 (M = 악성, B = 양성) 3-32
 - 10 개의 실수 값 피처가 각 셀 행에 대해 계산됩니다.
3. 반지름 (중심에서 주변까지의 거리의 평균)
4. 질감 (그레이 스케일 값의 표준 편차)
5. 둘레
6. 면적
7. 매끄러움 (반지름 길이의 국부적 인 변화)
8. 치밀도 (i) 대칭성 (j) 프랙탈 차원 ("해안선 근사법"-1)
9. 오목 점 (윤곽의 오목한 부분의 수)
 - 이 피처의 평균, 표준 오차 및 "최악"또는 가장 큰 (세 가지 가장 큰 값의 평균)이 각 이미지에 대해 계산되어 30 개의 피처가 생성됩니다. 예를 들어 필드 3은 평균 반경, 필드 13은 반경 SE, 필드 23은 최악 반경입니다.
 - 모든 기능 값은 4 자리 유효 숫자로 레코딩됩니다.
 - 누락 된 속성 값: 없음
 - 분포: 양성 357, 악성 212

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import KFold # K-fold cross validation
from sklearn import metrics
```

```
/home/ccnlml/anaconda3/lib/python3.6/site-packages/sklearn/cross_validation.py:44: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
"This module will be removed in 0.20.", DeprecationWarning)
```

```
In [2]: # 데이터 받기
df = pd.read_csv('data/BreastCancerWisconsin.csv')
```

```
In [3]: df.head(5)
```

```
Out[3]:
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smooth
0	842302	M	17.99	10.38	122.80	1001.0	0.11840
1	842517	M	20.57	17.77	132.90	1326.0	0.08474
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960
3	84348301	M	11.42	20.38	77.58	386.1	0.14250
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030

5 rows x 33 columns

```
In [4]: # 데이터 전처리
df.drop('id',axis=1,inplace=True)
df.drop('Unnamed: 32',axis=1,inplace=True)
# 데이터 프레임 크기 출력
len(df)
```

```
Out[4]: 569
```

```
In [5]: # 데이터 이상값 검정을 위한 unique값 출력
df.diagnosis.unique()
```

```
Out[5]: array(['M', 'B'], dtype=object)
```

```
In [6]: # 더미코딩
df['diagnosis'] = df['diagnosis'].map({'M':1,'B':0})
df.head()
```

```
Out[6]:
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	1	17.99	10.38	122.80	1001.0	0.11840
1	1	20.57	17.77	132.90	1326.0	0.08474
2	1	19.69	21.25	130.00	1203.0	0.10960
3	1	11.42	20.38	77.58	386.1	0.14250
4	1	20.29	14.34	135.10	1297.0	0.10030

5 rows x 31 columns

```
In [7]: # 데이터 기초 통계량
df.describe()
```

```
Out[7]:
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000

mean	0.372583	14.127292	19.289649	91.969033	654.889104	0.096360
std	0.483918	3.524049	4.301036	24.298981	351.914129	0.014064
min	0.000000	6.981000	9.710000	43.790000	143.500000	0.052630
25%	0.000000	11.700000	16.170000	75.170000	420.300000	0.086370
50%	0.000000	13.370000	18.840000	86.240000	551.100000	0.095870
75%	1.000000	15.780000	21.800000	104.100000	782.700000	0.105300
max	1.000000	28.110000	39.280000	188.500000	2501.000000	0.163400

8 rows x 31 columns

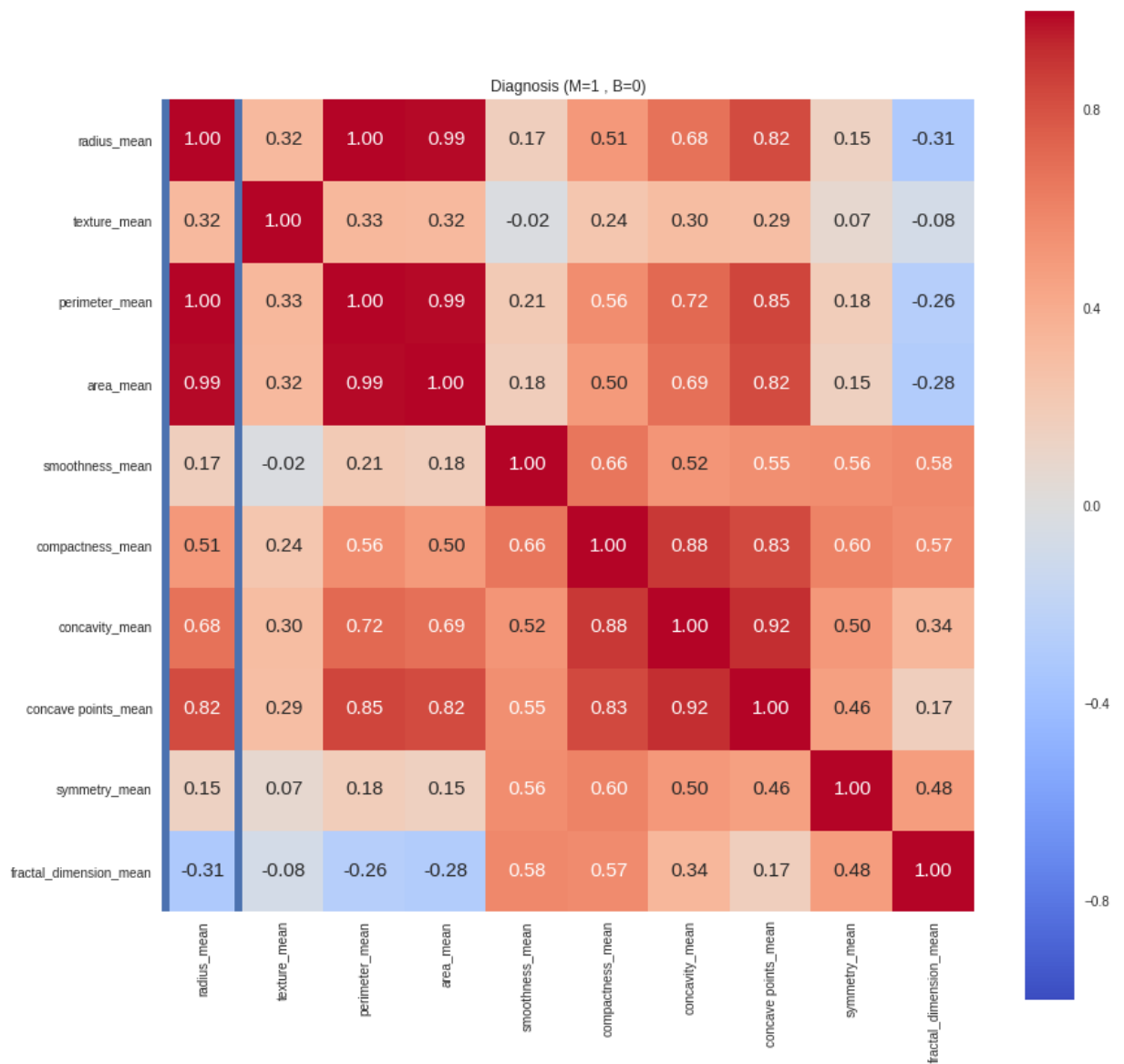
```
In [8]: features_mean=list(df.columns[1:11]) #속성값 가져오기
features_mean
```

```
Out[8]: ['radius_mean',
'texture_mean',
'perimeter_mean',
'area_mean',
'smoothness_mean',
'compactness_mean',
'concavity_mean',
'concave points_mean',
'symmetry_mean',
'fractal_dimension_mean']
```

```
In [9]: #상관관계 파악을 위한 히트맵(hit map)작성
corr = df[features_mean].corr()
plt.figure(figsize=(14,14))
sns.heatmap(corr, cbar = True, square = True, annot=True, fmt= '.2f',an
not_kws={'size': 15},
xticklabels= features_mean, yticklabels= features_mean,
cmap= 'coolwarm')
```

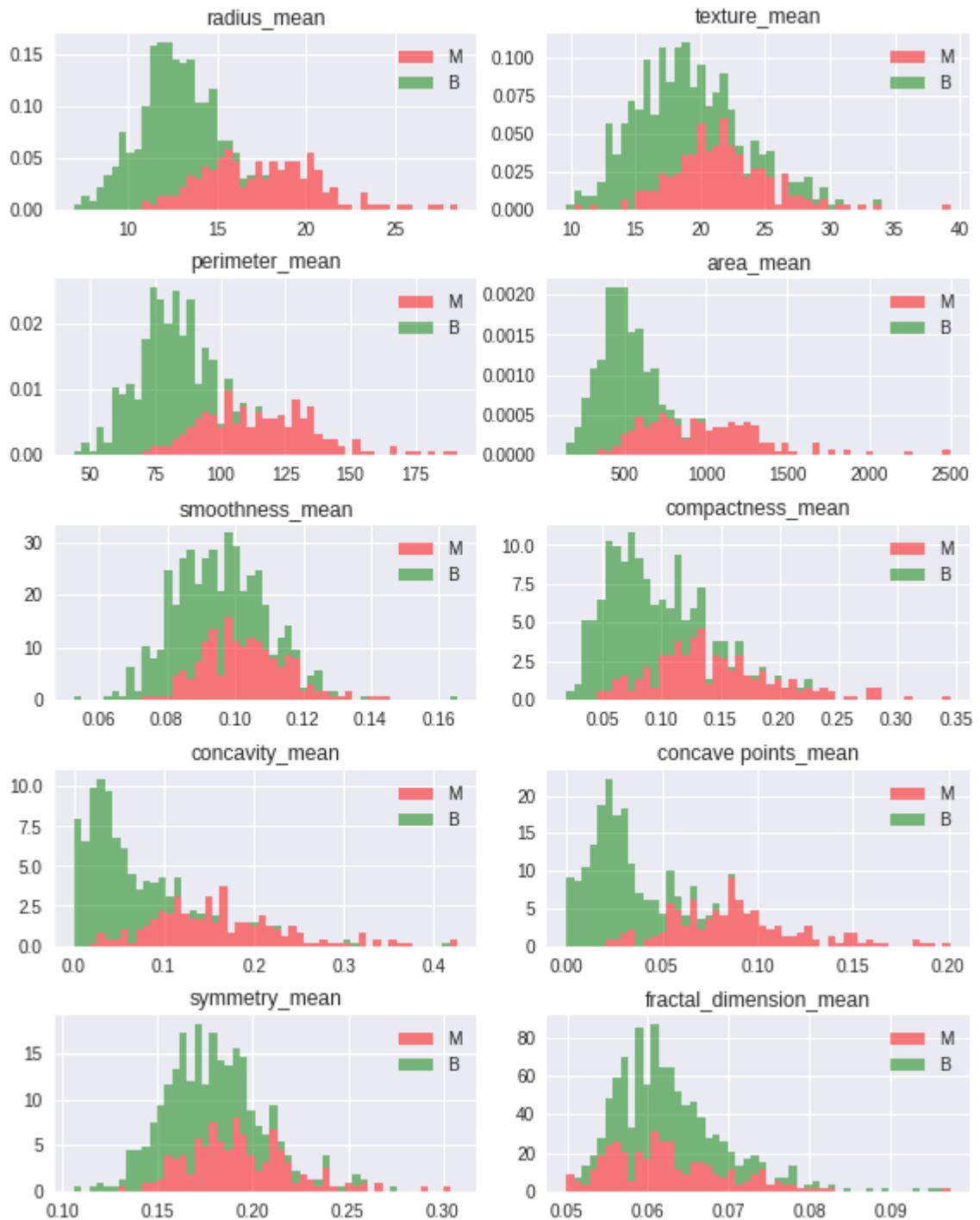
```
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6e519c02b0>
```

```
In [10]: #diagnosis에 대한 히스토그램 작성
plt.hist(df['diagnosis'])
plt.title('Diagnosis (M=1 , B=0)')
plt.show()
```

```
In [11]: # 진단 속성을 두개 속성으로 쪼갬다
dfM=df[df['diagnosis'] ==1]
dfB=df[df['diagnosis'] ==0]
```

```
In [12]: plt.rcParams.update({'font.size': 8})
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(8,10))
axes = axes.ravel()
for idx,ax in enumerate(axes):
    ax.figure
    binwidth= (max(df[features_mean[idx]]) - min(df[features_mean[idx]]))
    )/50
    ax.hist([dfM[features_mean[idx]],dfB[features_mean[idx]]], bins=np.a
range(min(df[features_mean[idx]]), max(df[features_mean[idx]]) + binwidth
h, binwidth) , alpha=0.5,stacked=True, normed = True, label=['M','B'],co
lor=['r','g'])
    ax.legend(loc='upper right')
    ax.set_title(features_mean[idx])
plt.tight_layout()
plt.show()
```



```
In [13]: # 트레이닝데이터, 테스트 데이터 분할
traindf, testdf = train_test_split(df, test_size = 0.3)
```

```
In [14]: def classification_model(model, data, predictors, outcome):
    model.fit(data[predictors],data[outcome])
    predictions = model.predict(data[predictors])

    # 정확도 출력
    accuracy = metrics.accuracy_score(predictions,data[outcome])
    print("Accuracy : %s" % "{0:.3%}".format(accuracy))

    # 5개의 fold값으로 k-fold cross-validation
    kf = KFold(data.shape[0], n_folds=5)
    error = []
    for train, test in kf:
        train_predictors = (data[predictors].iloc[train,:])
        train_target = data[outcome].iloc[train]
```

```

        model.fit(train_predictors, train_target)

        error.append(model.score(data[predictors].iloc[test,:], data[outcome].iloc[test]))
        print("Cross-Validation Score : %s" % "{0:.3%}".format(np.mean(error)))

    model.fit(data[predictors],data[outcome])

```

```

In [15]: # Logistic Regression model
predictor_var = ['radius_mean', 'perimeter_mean', 'area_mean', 'compactness_mean', 'concave points_mean']
outcome_var='diagnosis'
model=LogisticRegression()
# 모델 평가
classification_model(model,traindf,predictor_var,outcome_var)

```

```

Accuracy : 89.196%
Cross-Validation Score : 88.750%
Cross-Validation Score : 90.000%
Cross-Validation Score : 90.000%
Cross-Validation Score : 90.601%
Cross-Validation Score : 89.443%

```



국제 항공사 승객 데이터

선형회귀분석(**Linear Regression**)을 이용한 국제 항공사 승객 예측 문제

[데이터 다운로드](#)

[소스 출처](#)

데이터 설명

1 년 1 개월이 주어지면 국제선 승객 수를 1,000 명 단위로 예측하는 것이 문제.

자료는 1949 년 1 월부터 1960 년 12 월까지, 또는 12 년 동안 144 회의 관찰을 통해 수집.

```
In [187]: # 데이터 읽어오기
dataset = pandas.read_csv('data/international-airline-passengers.csv', u
secols=[1], engine='python', skipfooter=3)
```

월별 태양 흑점 데이터

선형회귀분석(**Linear Regression**)을 이용한 태양 흑점 예측 문제

[데이터 다운로드](#)

데이터 설명

230 년 (1749-1983 년) 동안 관찰 된 월별 태양 흑점 수

단위는 카운트이며 2,820 회의 관측값이 있음.

```
In [169]: # 데이터 읽어오기
dataset = pandas.read_csv('data/zueroich-monthly-sunspot-numbers-.csv', u
secols=[1], engine='python', skipfooter=3)
```

```
In [188]: import pandas
import matplotlib.pyplot as plt
import numpy
import math
from sklearn.metrics import mean_squared_error
```

```
In [189]: dataset.head(5)
```

Out[189]:

International airline passengers: monthly totals in thousands. Jan 49 ? Dec 60	
0	112
1	118

2	132
3	129
4	121

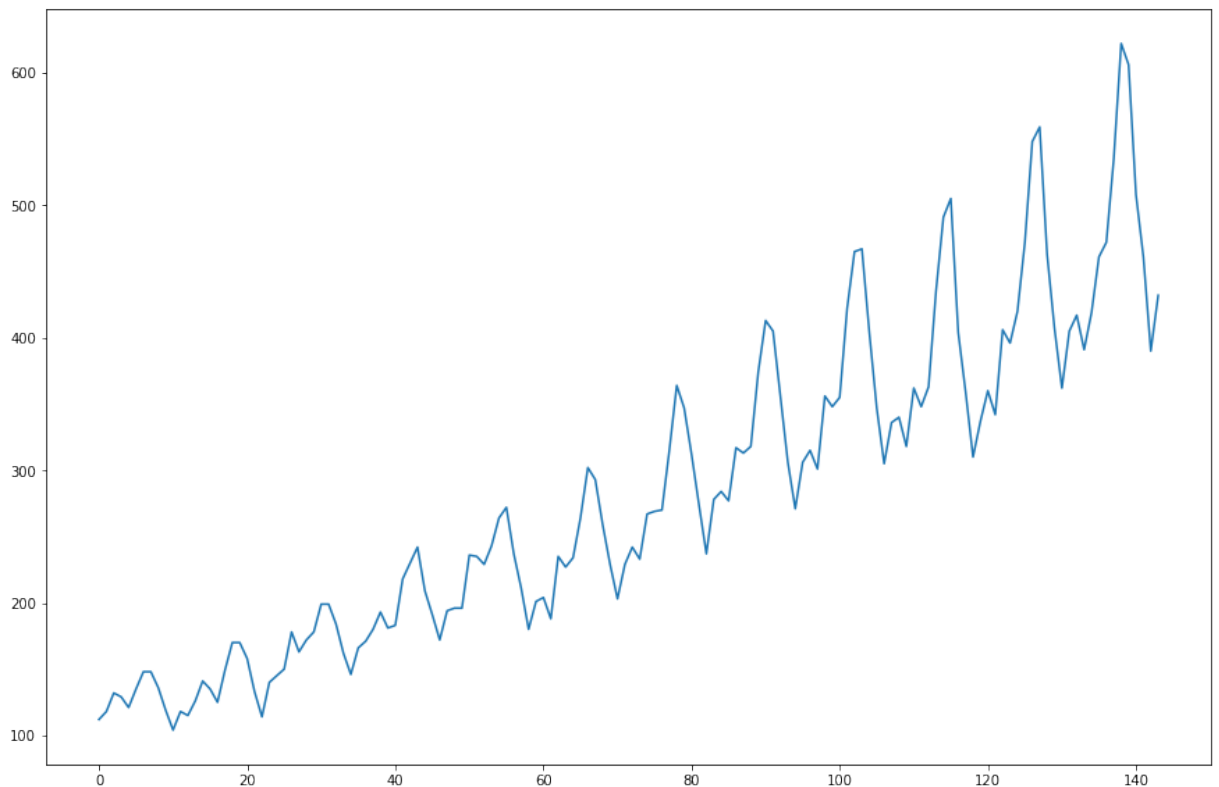
```
In [190]: dataset.tail(5)
```

Out[190]:

	International airline passengers: monthly totals in thousands. Jan 49 ? Dec 60
139	606
140	508
141	461
142	390
143	432

```
In [191]: plt.clf()
plt.figure(figsize=(15, 10))
plt.plot(dataset)
plt.show()
```

<matplotlib.figure.Figure at 0x7fd616e53320>



```
In [192]: # seed값을 고정한다.
numpy.random.seed(7)
```

```
In [193]: dataset.head(5)
```

Out[193]:

	International airline passengers: monthly totals in thousands. Jan 49 ? Dec 60
0	112

1	118
2	132
3	129
4	121

```
In [194]: # 최대 최소 정규화
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
```

```
In [195]: # 훈련데이터와 테스트 데이터 분리
# 관측치의 0.67%를 훈련데이터로, 나머지를 테스트 데이터로 활용
# 주의할점 : 시계열 데이터는 값의 순서가 중요하므로 임의 샘플링(random sampling)을 통한
# 데이터 분리는 피해야 한다

train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
print(len(train), len(test))

96 48
```

```
In [196]: # 배열을 dataset matrix 형태로 바꾼다.
# X는 주어진 시간 (t)에 승객의 수이고 Y가 다음 시간 (t+1)에 승객의 수인 데이터 세트를 생성
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)
```

```
In [197]: # reshape into X=t and Y=t+1
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
```

```
In [198]: # 회귀분석
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(trainX, trainY)
```

```
Out[198]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
In [199]: # make predictions
trainPredict_lm = lm.predict(trainX)
testPredict_lm = lm.predict(testX)
```

```
In [200]: # invert predictions
trainPredict_lm = scaler.inverse_transform(trainPredict_lm.reshape(-1, 1))
trainY = scaler.inverse_transform(trainY.reshape(-1, 1))
testPredict_lm = scaler.inverse_transform(testPredict_lm.reshape(-1, 1))
```

```
testY = scaler.inverse_transform(testY.reshape(-1, 1))
```

평균 제곱근 편차

평균 제곱근 편차(Root Mean Square Deviation; RMSD) 또는 평균 제곱근 오차(Root Mean Square Error; RMSE)는 추정 값 또는 모델이 예측한 값과 실제 환경에서 관찰되는 값의 차이를 다룰 때 흔히 사용하는 척도이다. 정밀도(precision)를 표현하는데 적합하다. 각각의 차이값은 잔차(residual)라고도 하며, 평균 제곱근 편차는 잔차들을 하나의 척도로 종합할 때 사용된다.

```
In [201]: # RME(root mean squared error) 계산
trainScore = math.sqrt(mean_squared_error(trainY, trainPredict_lm))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY, testPredict_lm))
print('Test Score: %.2f RMSE' % (testScore))
```

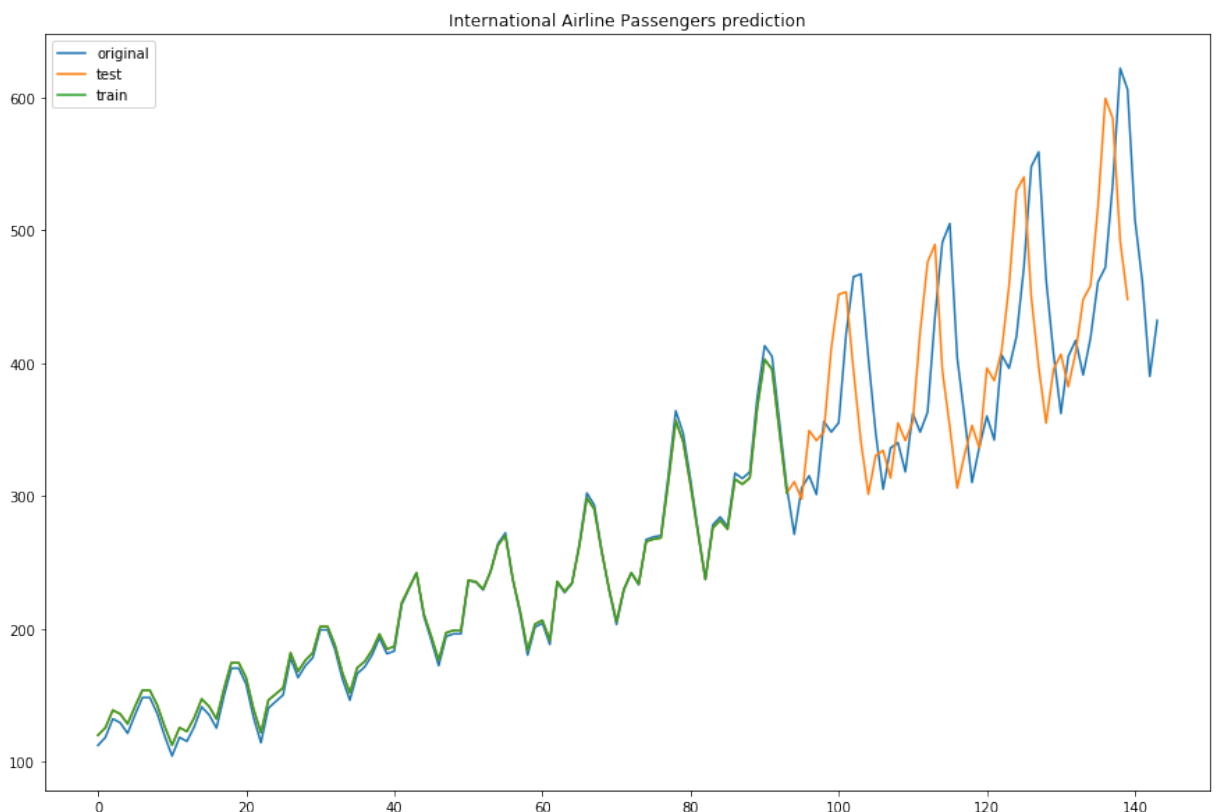
Train Score: 22.72 RMSE

Test Score: 48.76 RMSE

```
In [202]: # 그래프를 그리기 위해 train data 예측값과 test data 예측값을 합친다
totalPredict_lm = numpy.append(trainPredict_lm, testPredict_lm)
```

```
In [203]: plt.clf()
plt.figure(figsize=(15, 10))
# plot baseline and predictions
plt.title('International Airline Passengers prediction')
plt.plot(scaler.inverse_transform(dataset), label='original')
plt.plot(totalPredict_lm, label='test')
plt.plot(trainPredict_lm, label='train')
plt.legend()
plt.show()
```

<matplotlib.figure.Figure at 0x7fd616dfcbe0>



In []:



전력 생산량 데이터

선형회귀분석(**Linear Regression**)을 이용한 전력 데이터 분석

[데이터 다운로드](#)

[소스 출처](#)

연도별 전력 생산량 데이터

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math
import seaborn as sns
from sklearn.metrics import mean_squared_error
```

```
In [2]: # 데이터 읽어오기
data = pd.read_excel('data/GeneratedEnergy.xlsx')
data.head(5)
```

Out[2]:

	year	water power	steam power	fire power	internal-combustion power	nuclear power	group	substitution	
0	2000	5609822	119947533	26863140	293861	108963740	0	0	47
1	2001	4150753	135436741	29032971	324939	112133033	0	0	41
2	2002	5311047	138929484	38336951	353023	119102905	0	0	44
3	2003	6886983	140269475	40374646	370125	129671763	0	0	48
4	2004	5861434	145364710	55451941	406895	130714816	3552538	350183	44

```
In [3]: # 시계열 분석에서 year column은 불필요함으로 제거한다
dataset = pd.DataFrame(data.drop('year', axis=1))
```

```
In [4]: dataset.describe()
```

Out[4]:

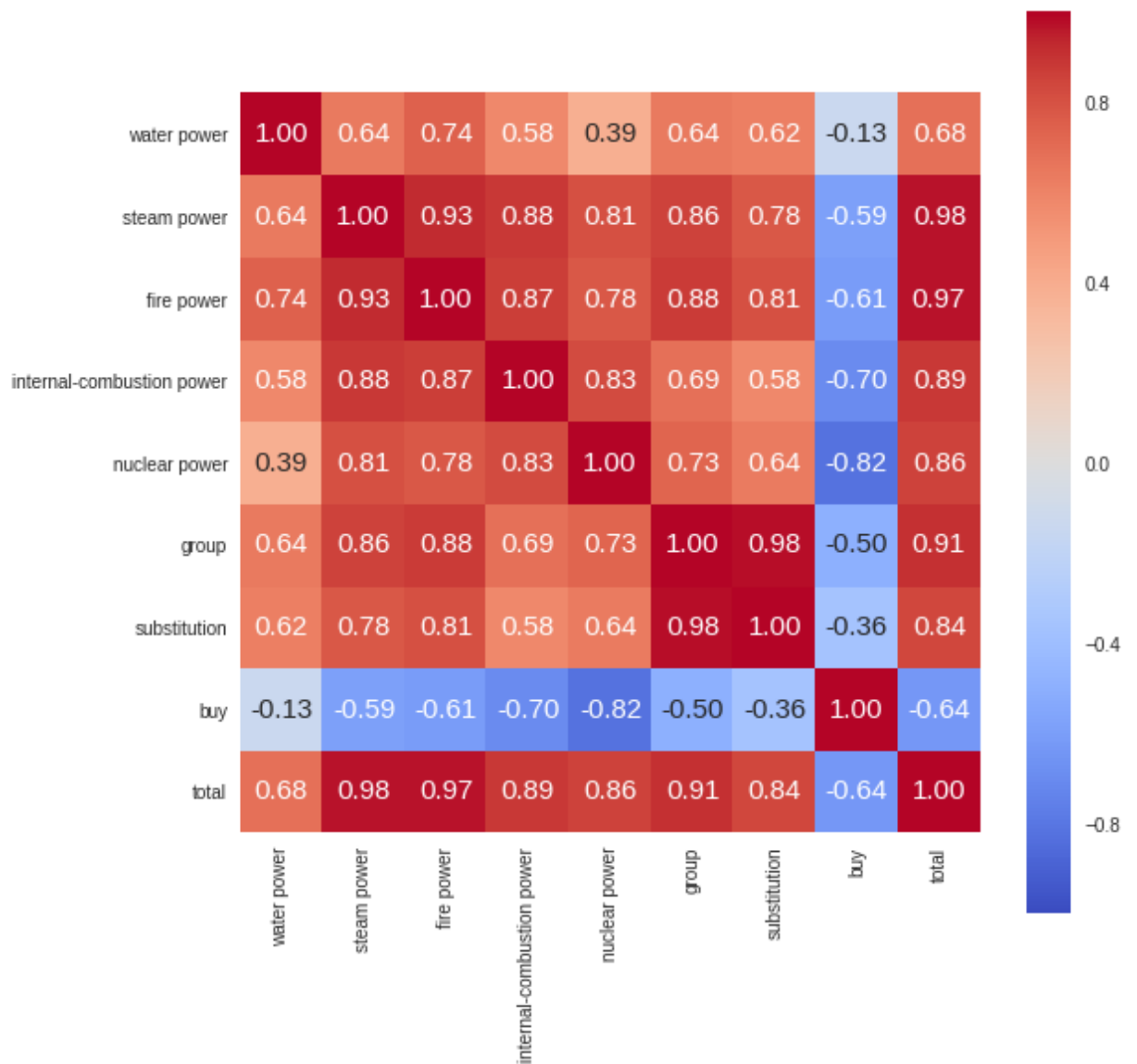
	water power	steam power	fire power	internal-combustion power	nuclear power	group
count	1.600000e+01	1.600000e+01	1.600000e+01	16.000000	1.600000e+01	1.600000e+01
mean	6.155056e+06	1.772373e+08	7.332181e+07	570168.875000	1.407112e+08	6.979149e+07
std	1.224840e+06	3.575108e+07	3.118344e+07	172408.381448	1.623582e+07	7.092117e+07

min	4.150753e+06	1.199475e+08	2.686314e+07	293861.000000	1.089637e+08	0.000000e+00
25%	5.287940e+06	1.440909e+08	5.168262e+07	397702.500000	1.304541e+08	1.947548e+00
50%	5.718601e+06	1.785352e+08	7.082885e+07	610760.000000	1.472749e+08	4.303480e+00
75%	7.088987e+06	2.112659e+08	1.008186e+08	705388.500000	1.504850e+08	1.254998e+01
max	8.393928e+06	2.185853e+08	1.244000e+08	820533.000000	1.647624e+08	2.201871e+01

```
In [5]: features=list(dataset.columns[:]) #속성값 가져오기
features
```

```
Out[5]: ['water power',
         'steam power',
         'fire power',
         'internal-combustion power',
         'nuclear power',
         'group',
         'substitution',
         'buy',
         'total']
```

```
In [6]: #상관관계 파악을 위한 히트맵(hit map)작성
corr = dataset[features].corr()
plt.figure(figsize=(9,9))
sns.heatmap(corr, cbar = True, square = True, annot=True, fmt= '.2f',an
not_kws={'size': 15},
           xticklabels= features, yticklabels= features,
           cmap= 'coolwarm')
plt.show()
```



```
In [7]: # 배열을 dataset matrix 형태로 바꾼다.
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back)]
        dataX.append(a)
        dataY.append(dataset[i + look_back])
    return np.array(dataX), np.array(dataY)
```

```
In [8]: from sklearn.preprocessing import MinMaxScaler # 최대 최소 정규화
from sklearn.linear_model import LinearRegression
```

```
dataset = dataset['buy'] # 생산량 column 읽기
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
```

```
/home/ccnlml/anaconda3/lib/python3.6/site-packages/sklearn/utils/validat
ion.py:429: DataConversionWarning: Data with input dtype int64 was converte
d to float64 by MinMaxScaler.
```

```
warnings.warn(msg, _DataConversionWarning)
```

```
/home/ccnlml/anaconda3/lib/python3.6/site-packages/sklearn/preprocessing
/data.py:321: DeprecationWarning: Passing 1d arrays as data is deprecate
d in 0.17 and will raise ValueError in 0.19. Reshape your data either us
ing X.reshape(-1, 1) if your data has a single feature or X.reshape(1, -
1) if it contains a single sample.
```

```
warnings.warn(DEPRECATION_MSG_1D, DeprecationWarning)
/home/ccnlml/anaconda3/lib/python3.6/site-packages/sklearn/preprocessing
/data.py:356: DeprecationWarning: Passing 1d arrays as data is deprecate
d in 0.17 and will raise ValueError in 0.19. Reshape your data either us
ing X.reshape(-1, 1) if your data has a single feature or X.reshape(1, -
1) if it contains a single sample.
warnings.warn(DEPRECATION_MSG_1D, DeprecationWarning)
```

```
In [9]: # 7:3 비율로 훈련데이터와 테스트 데이터 분할
train_size = int(len(dataset) * 0.7)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size], dataset[train_size:len(dataset)]
train = np.array(train)
test = np.array(test)
```

```
In [10]: # reshape into X=t and Y=t+1
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
```

```
In [11]: #회귀분석
lm = LinearRegression()
lm.fit(trainX, trainY)
```

```
Out[11]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
In [12]: # make predictions
trainPredict_lm = lm.predict(trainX)
testPredict_lm = lm.predict(testX)
```

```
In [13]: # invert predictions
trainPredict_lm = scaler.inverse_transform(trainPredict_lm.reshape(-1, 1))
trainY = scaler.inverse_transform(trainY.reshape(-1, 1))
testPredict_lm = scaler.inverse_transform(testPredict_lm.reshape(-1, 1))
testY = scaler.inverse_transform(testY.reshape(-1, 1))
```

```
In [14]: totalPredict_lm = np.append(trainPredict_lm, testPredict_lm)
```

```
In [15]: len(dataset)
```

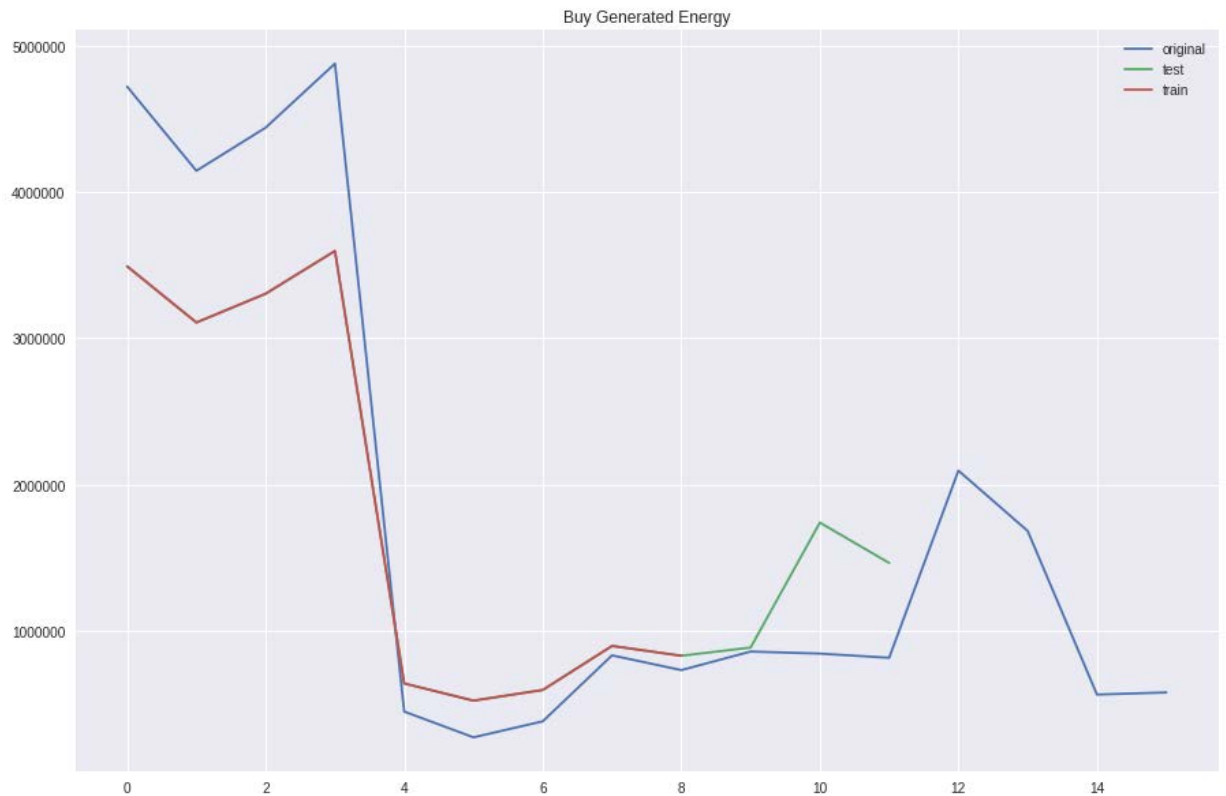
```
Out[15]: 16
```

```
In [16]: plt.clf()
plt.figure(figsize=(15, 10))
# plot baseline and predictions
plt.title('Buy Generated Energy')
plt.plot(scaler.inverse_transform(dataset), label='original')
plt.plot(totalPredict_lm, label='test')
plt.plot(trainPredict_lm, label='train')
plt.legend()
plt.show()
```

```
/home/ccnlml/anaconda3/lib/python3.6/site-packages/sklearn/preprocessing
/data.py:374: DeprecationWarning: Passing 1d arrays as data is deprecate
d in 0.17 and will raise ValueError in 0.19. Reshape your data either us
ing X.reshape(-1, 1) if your data has a single feature or X.reshape(1, -
```

```
1) if it contains a single sample.
warnings.warn(DEPRECATION_MSG_1D, DeprecationWarning)
```

```
<matplotlib.figure.Figure at 0x7fcb2b0d4358>
```



주별 최대 전력 생산량 데이터

[데이터 다운로드](#)

```
In [17]: # 데이터 읽어오기
mydata2 = pd.read_excel('data/weekly_max.xlsx')
```

```
In [18]: # 시계열 분석을 위해 요일을 없애고 한개의 column으로 통일한다
mydata2 = mydata2.drop('week', axis=1)
mydata2.head(5)
```

Out[18]:

	sun	mon	tue	wed	thu	fri	sat
0	65000	78800	78000	77000	65500	75500	67500
1	62000	77200	77800	78500	78200	76800	67400
2	62000	75000	75500	75000	75100	75200	65000
3	56000	68000	68300	68300	68900	68700	60500
4	56400	68500	68800	69000	68200	67800	60500

```
In [19]: mydata2 = np.array(mydata2).reshape(-1, 1)
len(mydata2)
```

Out[19]: 350

```
In [20]: dataset = pd.DataFrame(mydata2)
```

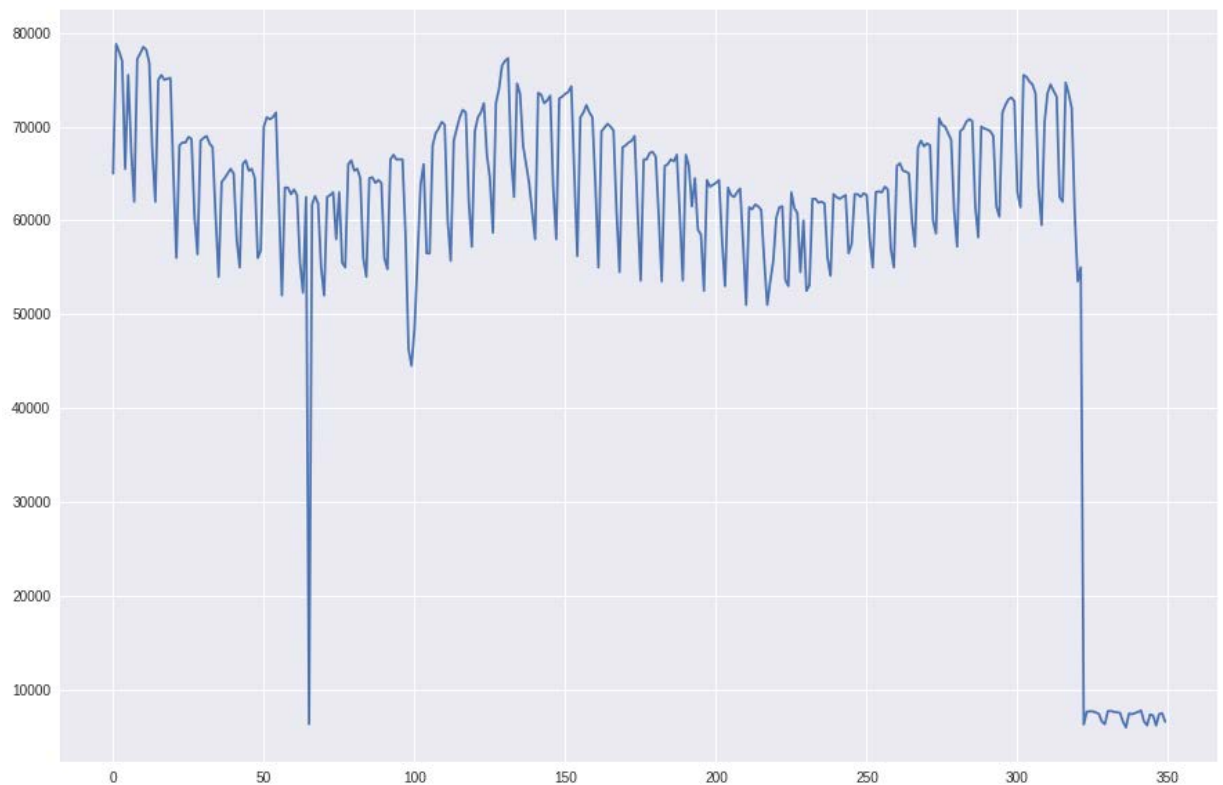
```
dataset.head(5)
```

Out[20]:

	0
0	65000
1	78800
2	78000
3	77000
4	65500

```
In [21]: plt.clf()  
plt.figure(figsize=(15, 10))  
plt.plot(dataset)  
plt.show()
```

<matplotlib.figure.Figure at 0x7fcb240b1f98>



```
In [22]: # 최대 최소 정규화  
from sklearn.preprocessing import MinMaxScaler  
  
scaler = MinMaxScaler(feature_range=(0, 1))  
dataset = scaler.fit_transform(dataset)
```

```
In [23]: # 훈련데이터와 테스트 데이터 분리  
# 관측치의 0.67%를 훈련데이터로, 나머지를 테스트 데이터로 활용  
# 주의할점 : 시계열 데이터는 값의 순서가 중요하므로 임의의 샘플링(random sampling)을 통한  
# 데이터 분리는 피해야 한다  
  
train_size = int(len(dataset) * 0.67)  
test_size = len(dataset) - train_size  
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]  
]
```

```
print(len(train), len(test))
```

234 116

```
In [24]: # 배열을 dataset matrix 형태로 바꾼다.
# X는 주어진 시간 (t)에 승객의 수이고 Y가 다음 시간 (t+1)에 승객의 수인 데이터 세트를 생성
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return np.array(dataX), np.array(dataY)
```

```
In [25]: # reshape into X=t and Y=t+1
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
```

```
In [26]: #회귀분석
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(trainX, trainY)
```

```
Out[26]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
In [27]: # make predictions
trainPredict_lm = lm.predict(trainX)
testPredict_lm = lm.predict(testX)
```

```
In [28]: # invert predictions
trainPredict_lm = scaler.inverse_transform(trainPredict_lm.reshape(-1, 1))
trainY = scaler.inverse_transform(trainY.reshape(-1, 1))
testPredict_lm = scaler.inverse_transform(testPredict_lm.reshape(-1, 1))
testY = scaler.inverse_transform(testY.reshape(-1, 1))
```

```
In [29]: # RME(root mean squared error) 계산
trainScore = math.sqrt(mean_squared_error(trainY, trainPredict_lm))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY, testPredict_lm))
print('Test Score: %.2f RMSE' % (testScore))
```

Train Score: 6903.73 RMSE

Test Score: 15594.11 RMSE

```
In [30]: # 그래프를 그리기 위해 train data 예측값과 test data 예측값을 합친다
totalPredict_lm = np.append(trainPredict_lm, testPredict_lm)
```

```
In [31]: plt.clf()
plt.figure(figsize=(15, 10))
# plot baseline and predictions
plt.title('International Airline Passengers prediction')
plt.plot(scaler.inverse_transform(dataset), label='original')
plt.plot(totalPredict_lm, label='test')
plt.plot(trainPredict_lm, label='train')
plt.legend()
```

```
plt.show()
```

<matplotlib.figure.Figure at 0x7fcbldccd128>



In []:

택시운행 데이터 불러오기

데이터 소스

뉴욕의 2013년 택시 운행 데이터를 읽어보겠다. 용량은 50GB이고, 날짜, 탑승지, 하차장소, 요금 등의 정보를 포함하여 170만개의 운행정보를 가지고 있다. 데이터 용량을 줄이기 위해 전체 데이터 중 0.5%에 해당하는 85만건의 운행정보만 읽어보겠다.

이 데이터에 대한 소개는 hubcab.org 를 참고하기 바란다.

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
In [2]: # 데이터 path
data_filename = 'data/nyc_data.csv'
fare_filename = 'data/nyc_fare.csv'
```

```
In [3]: # 데이터 읽기
data = pd.read_csv(data_filename, parse_dates=['pickup_datetime', 'dropoff_datetime']) # 승차 데이터
fare = pd.read_csv(fare_filename, parse_dates=['pickup_datetime']) # 요금 데이터
```

```
In [4]: data.head(3)
```

Out[4]:

	medallion	hack_license	vendor
0	76942C3205E17D7E7FE5A9F709D16434	25BA06A87905667AA1FE5990E33F0E2E	VTS
1	517C6B330DBB3F055D007B07512628B3	2C19FBEE1A6E05612EFE4C958C14BC7F	VTS
2	ED15611F168E41B33619C83D900FE266	754AEBD7C80DA17BA1D81D89FB6F4D1D	CMT

```
In [5]: fare.head(3)
```

Out[5]:

	medallion	hack_license	vendor
0	76942C3205E17D7E7FE5A9F709D16434	25BA06A87905667AA1FE5990E33F0E2E	VTS
1	517C6B330DBB3F055D007B07512628B3	2C19FBEE1A6E05612EFE4C958C14BC7F	VTS
2	ED15611F168E41B33619C83D900FE266	754AEBD7C80DA17BA1D81D89FB6F4D1D	CMT

matplotlib을 활용한 그래프 그리기

```
In [6]: # 승차 데이터 속성값 출력
data.columns
```

```
Out[6]: Index(['medallion', 'hack_license', 'vendor_id', 'rate_code',
              'store_and_fwd_flag', 'pickup_datetime', 'dropoff_datetime',
              'passenger_count', 'trip_time_in_secs', 'trip_distance',
              'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
              'dropoff_latitude'],
              dtype='object')
```

```
In [7]: # 요금 데이터 속성값 출력
fare.columns
```

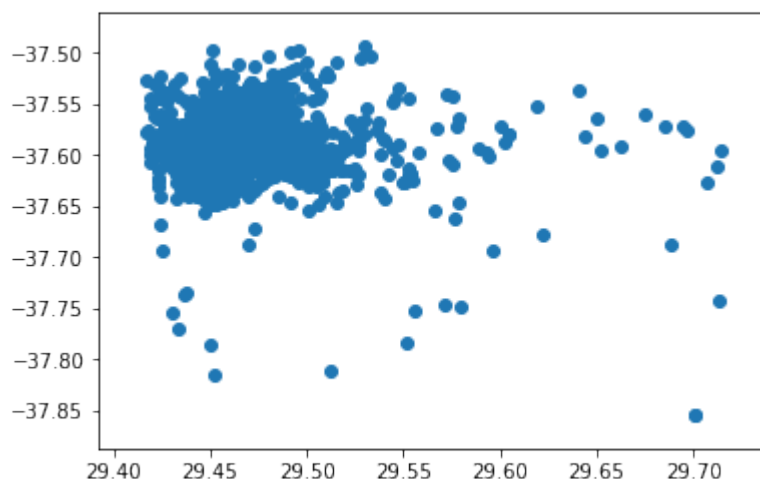
```
Out[7]: Index(['medallion', 'hack_license', 'vendor_id', 'pickup_datetime',
              'payment_type', 'fare_amount', 'surcharge', 'mta_tax', 'tip_amooun
              t',
              'tolls_amount', 'total_amount'],
              dtype='object')
```

```
In [8]: p_lng = data.pickup_longitude # 승차 경도
p_lat = data.pickup_latitude # 승차 위도
d_lng = data.dropoff_longitude # 하차 위도
d_lat = data.dropoff_latitude # 하차 경도
```

```
In [9]: def lat_lng_to_pixels(lat, lng):
        lat_rad = lat * np.pi / 180.0
        lat_rad = np.log(np.tan((lat_rad + np.pi / 2.0) / 2.0))
        x = 100 * (lng + 180.0) / 360.0
        y = 100 * (lat_rad - np.pi) / (2.0 * np.pi)
        return (x, y)
```

```
In [10]: px, py = lat_lng_to_pixels(p_lat, p_lng)
```

```
In [11]: # 산점도 출력
plt.clf()
plt.scatter(px, py)
plt.show()
```



```
In [12]: # 지도처럼 출력
```

```
plt.clf()
plt.figure(figsize=(8, 6))
plt.scatter(px, py, s=.1, alpha=.03) #점 크기 설정(s) 및 투명도 설정(alpha)
plt.axis('equal') #x,y축 비율 맞춤
plt.xlim(29.40, 29.55)
plt.ylim(-37.63, -37.54)
plt.axis('off') #축 없애기
plt.show()
```

<matplotlib.figure.Figure at 0x7f264843a2b0>

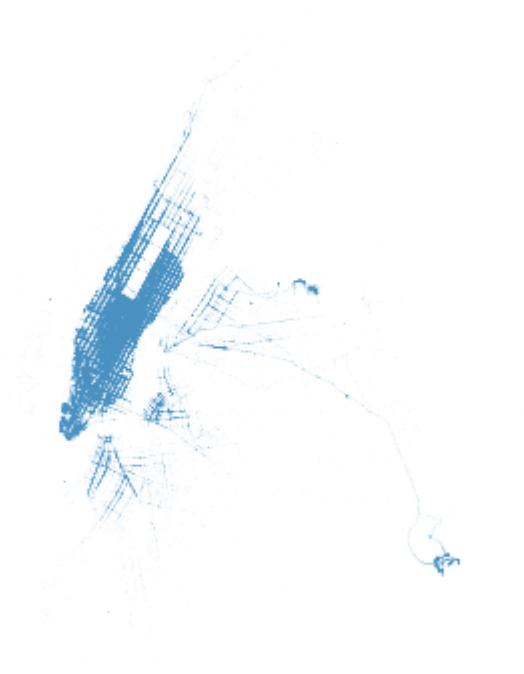


그림 라이브러리 **seaborn**

matplotlib로 그림을 그리려면 시행착오 과정을 많이 거쳐야 한다. 이를 편리하게 해결하는 라이브러리로 **seaborn**을 사용하겠다. **seaborn**을 설치하면 그림을 그릴 때 자동으로 깔끔하게 만들어준다.

```
In [13]: px.count(), px.min(), px.max()
```

```
Out[13]: (846945, 29.417137499999995, 29.714313055555561)
```

```
In [14]: px.mean(), px.median(), px.std()
```

```
Out[14]: (29.45134580776863, 29.449418333333337, 0.009761694274451149)
```

```
In [15]: import seaborn as sns
# 히스토그램 출력
data.trip_distance.hist(bins=np.linspace(0., 10., 100))
```

```
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2645998908>
```

데이터 전처리

```
In [16]: # 데이터 선택
```

```
data[['trip_distance', 'trip_time_in_secs']].head(5)
# 두개의 컬럼만 선택하고 상위 3개의 행에 해당하는 데이터만 읽음
```

Out[16]:

	trip_distance	trip_time_in_secs
0	0.61	300
1	3.28	960
2	1.50	386
3	0.00	0
4	1.31	360

```
In [17]: data.loc[840000]
# 첫번째 행(row)의 값을 읽음
```

```
Out[17]: medallion          B15C4DE03944CDA9D837DE778FA10FCD
hack_license      20A8952E97ED1F916C8BC5DFB5C7EE7A
vendor_id         CMT
rate_code         1
store_and_fwd_flag N
pickup_datetime   2013-12-28 18:06:56
dropoff_datetime  2013-12-28 18:36:40
passenger_count   1
trip_time_in_secs 1784
trip_distance     1.1
pickup_longitude  -73.9929
pickup_latitude   40.7684
dropoff_longitude -73.9943
dropoff_latitude  40.7542
Name: 840000, dtype: object
```

```
In [18]: # 0과 100000번 데이터 행을 읽는다
data.loc[[0, 100000]]
```

Out[18]:

	medallion	hack_license
0	76942C3205E17D7E7FE5A9F709D16434	25BA06A87905667AA1FE5990E33F0E2E
100000	7461F7106D33D3A5775F4245724606FD	BACEA353BB4106A005BB7836BDCAC0C3

```
In [19]: data.loc[1000:2000:10,
               ['trip_distance', 'trip_time_in_secs']]

# 행과 열을 동시에 선택할 수 있는데, 행을 먼저 정해주어야 한다.

data.loc[data.trip_distance>50]
```

Out[19]:

	medallion	hack_license
1853	DA2A0A9EBD9AFD25013CF83805B61D6D	A3BCE60F48302B63C4B7AAE443960704
6316	6BD1B641A1CD55803A21560299B985A7	A33DB5A909ADBFCE2AB057264141C8E9

8683	E95D27A43DC7A5F0C015409386BB49DB	8A6EB7FA7E6A3F4A50C717871C068D72
24381	A38882246FC6439948FA470459F94134	8A60E3668580091A4BB124E9BDAFDD89
54723	0F48C5AE255A294B4E88ECC3CED0CFC5	C92D893B061C29C152ADF42B15317C3E
90068	7B59F6023736C5217B613D17B4ED6A9C	F2D89276E3886C6A8D27AE463637FF36
115471	FB0FAE6EABFE14F45600AD382B52EFE5	0D39DD3359634605EAC8B451295E8186
119002	A51E7B7371667690E65D88307FBBA229	3A6EDD62A2933A11545527FDE55DAC52
134839	FA189EABBB4058AC0359AA1CF30F5313	C707B5A8C6AEF269564272AF66B37564
157050	C4580AE98B332B1E4B6B822B1C2D03BE	E2CB66B7D37105F7C3364DFF61E4D487
201876	DBCD64B365502F85569B4A264B798AAE	D404427B22B187FE0460668FDEC9937E
210667	BD4738379B8C2E8168F48CFFDE2F94E6	E6F2B8515FE5A32AACF884FBA21F0170
238505	4ADDFBA1E3FD055AB4854D003976F263	2B09130C92CF5199315044DFE89D6356
305690	A895384C359AABDA144AF574989DA1C3	CEEB6BCC5E4705A32043BCA2DBA78E3
450941	849397BEC809E0CBF9FF645FE7FD96A8	BC7E6894AF102A345E3D050121FAA1F2
504497	7237EC7ABD6114EDDC87A3AA846F8418	D52502537E2DF62C9BFFECF5A387E7E9
507107	50DA72F510E2F84A42712E13744FAC7B	EA9D03A766C1D32A6668FFF0C1EB4E4E
520570	68FADA2827FA3A1AB0FF32EC47F565EB	1E68C0830B73658C43635CAD45ECFDF
548988	A978A0AAE9B2CFEE310FACD97A09C319	CE56A27F53ABF411094B6CD708BFBA96
558665	5A5C516A820FE476E9D3E14101B669AC	C24585AA866FC76A4E09A05F55DC7E54
659208	E8E54922CFDFC480893C1E2021506CD4	BCCAFB9AB09496225A0FE39096FD72CC
672612	9B14258B0E3EF13923996FB8537B2EA0	42834D2C6C221FDA7CDAE3AC95A981D

벡터 연산

파이선은 벡터 연산을 지원한다. 즉, `for` 문을 쓸 필요없이 `Series` 나 `DataFrame` 구조에 대해서 모든 항목에 대해 연산을 한번에 처리할 수 있다. 아래는 초 단위로 된 차량 운행 시간을 분 단위로 한번에 바꾸는 것을 보인 예이다. 컬럼을 하나 추가하면서 이름을 `'trip_time_in_mins'`로 지정했다.

시리즈 (`Series`) 변수들을 연산할 때에는 인덱스가 같은 항목끼리 연산이 이루어진다.

```
In [20]: data['trip_time_in_mins'] = data.trip_time_in_secs / 60.0
data[['trip_time_in_secs', 'trip_time_in_mins']].head(3)
```

Out[20]:

	trip_time_in_secs	trip_time_in_mins
0	300	5.000000
1	960	16.000000
2	386	6.433333

```
In [21]: data.medallion.head(3)
data.medallion.str.slice(0, 4).head(3)

# 날짜와 시간
# 요일을 찾으려면 dt.dayofweek()를 이용하면 되는데 월요일은 1, 화요일은 2, ... 값이 리턴된다.

data.pickup_datetime.dt.dayofweek[::200000]

day_p = data.pickup_datetime.dt.day
day_d = data.dropoff_datetime.dt.day
selection = (day_p != day_d) # 타고 내린 날짜가 다른 경우를 찾음
print(len(data.loc[selection]))
data.loc[selection].head(3)
```

7716

Out[21]:

	medallion	hack_license	vehicle_type
2005	6385CA8C99985BFBAFB477A9BDFA28C9	08A78365909D2F09BF72B869C0B21FED	V
2008	D932DC772B89F69D30F03FB095424F97	F5AE2E36090433DFE4142AFC19AFD495	C
2010	33BB4B9DBFD87B7522909FEEB84896F4	412253C6258AF9DCE2D27DE714A84049	C

그룹핑

```
In [22]: #Group-by

weekly = data.groupby(data.pickup_datetime.dt.weekofyear)
len(weekly)
```

```

y = weekly.size()
y.head(10)

x = weekly.pickup_datetime.first()
x.head(3)

pd.Series(y.values, index=x).plot()
plt.ylim(0) # Set the lower y value to 0.
plt.xlabel('Week') # Label of the x axis.
plt.ylabel('Taxi rides') # Label of the y axis.

```

Out[22]: <matplotlib.text.Text at 0x7f26459b94a8>

```

In [23]: ### Joins
tip = fare[['medallion', 'tip_amount']].loc[fare.tip_amount>0].groupby('
medallion').mean()
print(len(tip))
tip.head(10)

```

13407

Out[23]:

	tip_amount
medallion	
00005007A9F30E289E760362F69E4EAD	1.815854
000318C2E3E6381580E5C99910A60668	2.857222
000351EDC735C079246435340A54C7C1	2.099111
0009986BDBAB2F9A125FEF49D0BFCCDD	2.220000
00115F46520039845A5F719C979BEA45	3.422222
00153E36140C5B2A84EA308F355A7925	2.704687
001C8EC421C9BE57D08576617465401A	2.895455
001D3B86C2ACDEE4D1B98AFE52969F3D	2.520455
001DFAC01BC0A32F48C3769DD1414778	2.164865
00244196AAA321571762E0CCC55EEAD9	2.009333

```

In [24]: # 히스토그램 출력
tip.hist(bins=np.linspace(0., 6., 50))
plt.xlabel('Average tip')
plt.ylabel('Number of taxis')

```

Out[24]: <matplotlib.text.Text at 0x7f2643aae630>

```

In [25]: # merge() 두개의 데이터프레임을 합하여 새로운 컬럼을 추가할 때 사용된다.
data_merged = pd.merge(data, tip, how='left', left_on='medallion', right
_index=True)
data_merged.head(3)

```

Out[25]:

	medallion	hack_license	vendor
0	76942C3205E17D7E7FE5A9F709D16434	25BA06A87905667AA1FE5990E33F0E2E	VTS

1	517C6B330DBB3F055D007B07512628B3	2C19FBEE1A6E05612EFE4C958C14BC7F	VTs
2	ED15611F168E41B33619C83D900FE266	754AEBD7C80DA17BA1D81D89FB6F4D1D	CMT

In []:

택시운행 데이터 분석

```
In [1]: import numpy as np
import pandas as pd
data = pd.read_csv('./data/nyc_data.csv')

# pandas로부터 NumPy로 바꾸는 것은 간단히 .value를 붙여주면 된다. 예를 들어
# Seires는 1차원 NumPy 어레이로 바뀐다.
# DataFrame은 2차원 NumPy로 바뀐다.
pickup = data[['pickup_longitude', 'pickup_latitude']].values
dropoff = data[['dropoff_longitude', 'dropoff_latitude']].values
```

```
In [2]: import numpy as np
x = np.arange(1, 11)
print(x)

[ 1  2  3  4  5  6  7  8  9 10]
```

```
In [3]: # 벡터를 1차원 어레이로 변경하는 방법
# 아래의 (1, -1)에서 -1의 의미는 차원의 수를 알아서 적절히 정하라는 뜻임
#아래의 결과를 보면 []와 [[]]의 차이가 벡터와 어레이임
x_row = x.reshape((1, -1))
x_row
```

```
Out[3]: array([[ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10]])
```

```
In [4]: x_col = x[:, np.newaxis]
x_col
```

```
Out[4]: array([[ 1],
               [ 2],
               [ 3],
               [ 4],
               [ 5],
               [ 6],
               [ 7],
               [ 8],
               [ 9],
               [10]])
```

```
In [5]: # 행렬 곱을 출력하는 함수
np.dot(x_col, x_row)
```

```
Out[5]: array([[ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10],
               [ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20],
               [ 3,  6,  9, 12, 15, 18, 21, 24, 27, 30],
               [ 4,  8, 12, 16, 20, 24, 28, 32, 36, 40],
               [ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50],
               [ 6, 12, 18, 24, 30, 36, 42, 48, 54, 60],
               [ 7, 14, 21, 28, 35, 42, 49, 56, 63, 70],
               [ 8, 16, 24, 32, 40, 48, 56, 64, 72, 80],
               [ 9, 18, 27, 36, 45, 54, 63, 72, 81, 90],
               [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]])
```

```
In [6]: np.dot(x_row, x_col)
```

```
Out[6]: array([[385]])
```

```
In [7]: #dot()함수와 달리 *는 항목별 곱셈을 수행한다
x_row * 3
```

```
Out[7]: array([[ 3,  6,  9, 12, 15, 18, 21, 24, 27, 30]])
```

```
In [8]: x_row * x_row
```

```
Out[8]: array([[ 1,  4,  9, 16, 25, 36, 49, 64, 81, 100]])
```

```
In [9]: #이 연산은 매트릭스 곱이 아니라 항목별 곱셈을 수행한다. 그런데 항목의 수가 서로 맞지 않기 때문
에
#broadcasting (자동확장)을 하여 아래와 같은 결과가 나오는 것임을 주의해야 한다
x_row * x_col
```

```
Out[9]: array([[ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10],
 [ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20],
 [ 3,  6,  9, 12, 15, 18, 21, 24, 27, 30],
 [ 4,  8, 12, 16, 20, 24, 28, 32, 36, 40],
 [ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50],
 [ 6, 12, 18, 24, 30, 36, 42, 48, 54, 60],
 [ 7, 14, 21, 28, 35, 42, 49, 56, 63, 70],
 [ 8, 16, 24, 32, 40, 48, 56, 64, 72, 80],
 [ 9, 18, 27, 36, 45, 54, 63, 72, 81, 90],
 [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]])
```

```
In [10]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib; matplotlib.rcParams['savefig.dpi'] = 144

# 데이터 읽기
data = pd.read_csv('./data/nyc_data.csv',
                    parse_dates=['pickup_datetime', 'dropoff_datetime'])

pickup = data[['pickup_longitude', 'pickup_latitude']].values # 승차 위경도
값을 pickup에 저장
dropoff = data[['dropoff_longitude', 'dropoff_latitude']].values # 하차 위
경도 값을 dropoff에 저장
pickup
```

```
Out[10]: array([[ -73.955925,  40.781887],
 [ -74.005501,  40.745735],
 [ -73.969955,  40.79977 ],
 ...,
 [ -73.993492,  40.729347],
 [ -73.978477,  40.772945],
 [ -73.987206,  40.750568]])
```

```
In [11]: # 하차 경도
lon = dropoff[:, 0]
lon
```

```
Out[11]: array([ -73.963181, -73.964943, -73.954567, ..., -74.013725, -73.963814,
 -73.970909])
```

```
In [12]: # 하차 위도
lat = dropoff[:, 1]
```

```
lat
```

```
Out[12]: array([ 40.777832,  40.755722,  40.787392, ...,  40.702332,  40.773922,
                40.795815])
```

```
In [35]: ## NumPy 어레이 연산
lon_min, lon_max = (-73.98470, -73.98165)
lat_min, lat_max = ( 40.76582,  40.76729)

in_lon = (lon_min <= lon) & (lon <= lon_max)
in_lon
```

```
Out[35]: array([False, False, False, ..., False, False, False], dtype=bool)
```

```
In [36]: in_lon.sum()
```

```
Out[36]: 61572
```

```
In [15]: in_lat = (lat_min <= lat) & (lat <= lat_max)

in_lonlat = in_lon & in_lat
in_lonlat.sum()
```

```
Out[15]: 1991
```

```
In [18]: # 요소들 중 0이 아닌 값들의 index를 반환하는 함수
np.nonzero(in_lonlat)[0]
```

```
Out[18]: array([    33,   1989,   2354, ..., 845273, 845827, 845989])
```

```
In [28]: # T는 전치(transpose)를 나타낸다
lon1, lat1 = dropoff.T
print(lon1)
print(lat1)
```

```
[-73.963181 -73.964943 -73.954567 ..., -74.013725 -73.963814 -73.970909]
[ 40.777832  40.755722  40.787392 ...,  40.702332  40.773922  40.795815]
```

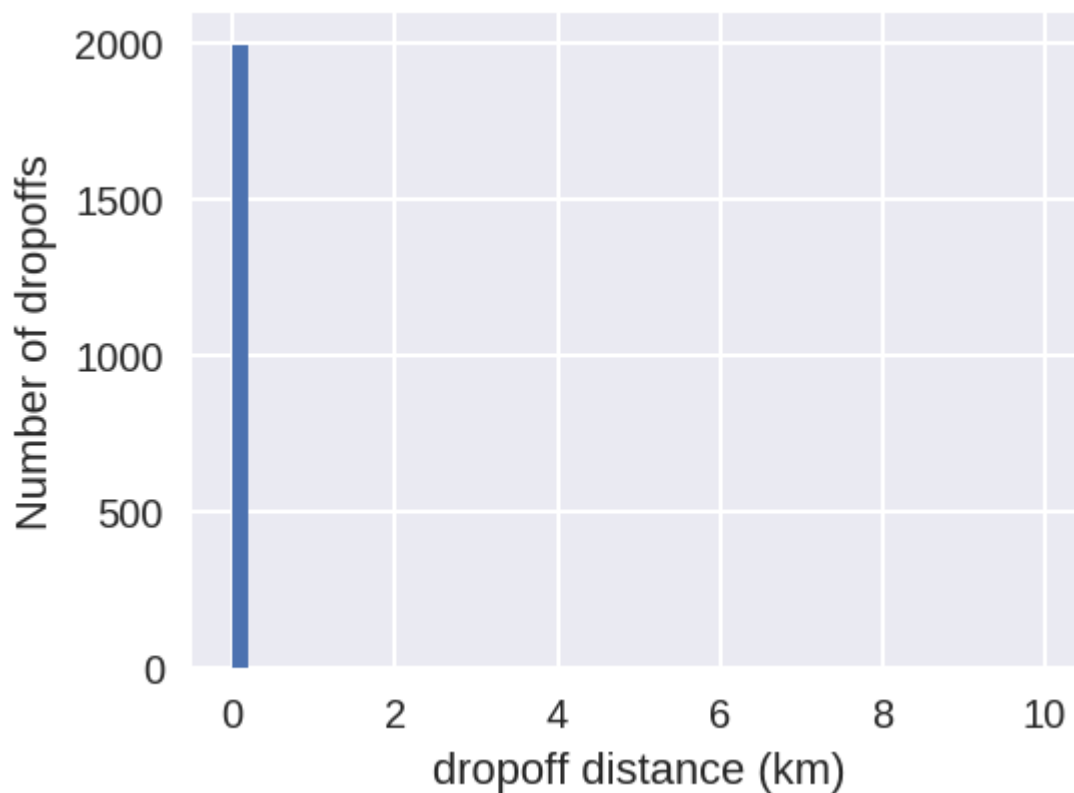
```
In [33]: # 직선거리 구하기
EARTH_R = 6372.8
def geo_distance(lon0, lat0, lon1, lat1):
    lat0 = np.radians(lat0)
    lon0 = np.radians(lon0)
    lat1 = np.radians(lat1)
    lon1 = np.radians(lon1)
    dlon = lon0 - lon1
    y = np.sqrt(
        (np.cos(lat1) * np.sin(dlon)) ** 2
        + (np.cos(lat0) * np.sin(lat1)
          - np.sin(lat0) * np.cos(lat1) * np.cos(dlon)) ** 2)
    x = np.sin(lat0) * np.sin(lat1) + \
        np.cos(lat0) * np.cos(lat1) * np.cos(dlon)
    c = np.arctan2(y, x)
    return EARTH_R * c
```

```
In [34]: distances = geo_distance(lon, lat, lon1, lat1)
distances
```

```
Out[34]: array([ 0.,  0.,  0., ...,  0.,  0.,  0.])
```

```
In [27]: plt.clf()
plt.figure(figsize=(4, 3))
plt.hist(distances[in_lonlat], np.linspace(0., 10., 50))
plt.xlabel('dropoff distance (km)')
plt.ylabel('Number of dropoffs')
plt.show()
```

<matplotlib.figure.Figure at 0x7f41d05bad68>



```
In [20]: evening = (data.pickup_datetime.dt.hour >= 19).values
n = np.sum(evening)
n
```

Out[20]: 242818

```
In [17]: # pandas는 필터링, 그룹핑 등 데이터를 다루는데 편리하고, NumPy는 매트릭스 연산을 빠르게 처리
# 해 주는 장점이 있다.
# 즉 처음에는 pandas로 데이터를 가공하고, 계산을 할 때 NumPy로 바꾸는 것이 편리하다.

weights = np.zeros(2 * n)

weights[:n] = -1
weights[n:] = +1

points = np.r_[pickup[evening],
               dropoff[evening]]
points.shape
```

Out[17]: (485636, 2)

```
In [36]: # 지리 좌표를 픽셀 좌표로 변환
def lat_lon_to_pixels(lat, lon):
    lat_rad = lat * np.pi / 180.0
    lat_rad = np.log(np.tan((lat_rad + np.pi / 2.0) / 2.0))
    x = 100 * (lon + 180.0) / 360.0
    y = 100 * (lat_rad - np.pi) / (2.0 * np.pi)
```

```

    return (x, y)

lon, lat = points.T
x, y = lat_lon_to_pixels(lat, lon)

lon_min, lat_min = -74.0214, 40.6978
lon_max, lat_max = -73.9524, 40.7982

x_min, y_min = lat_lon_to_pixels(lat_min, lon_min)
x_max, y_max = lat_lon_to_pixels(lat_max, lon_max)

bin = .00003
bins_x = np.arange(x_min, x_max, bin)
bins_y = np.arange(y_min, y_max, bin)

grid, _, _ = np.histogram2d(y, x, weights=weights,
                             bins=(bins_y, bins_x))

density = 1. / (1. + np.exp(-.5 * grid))

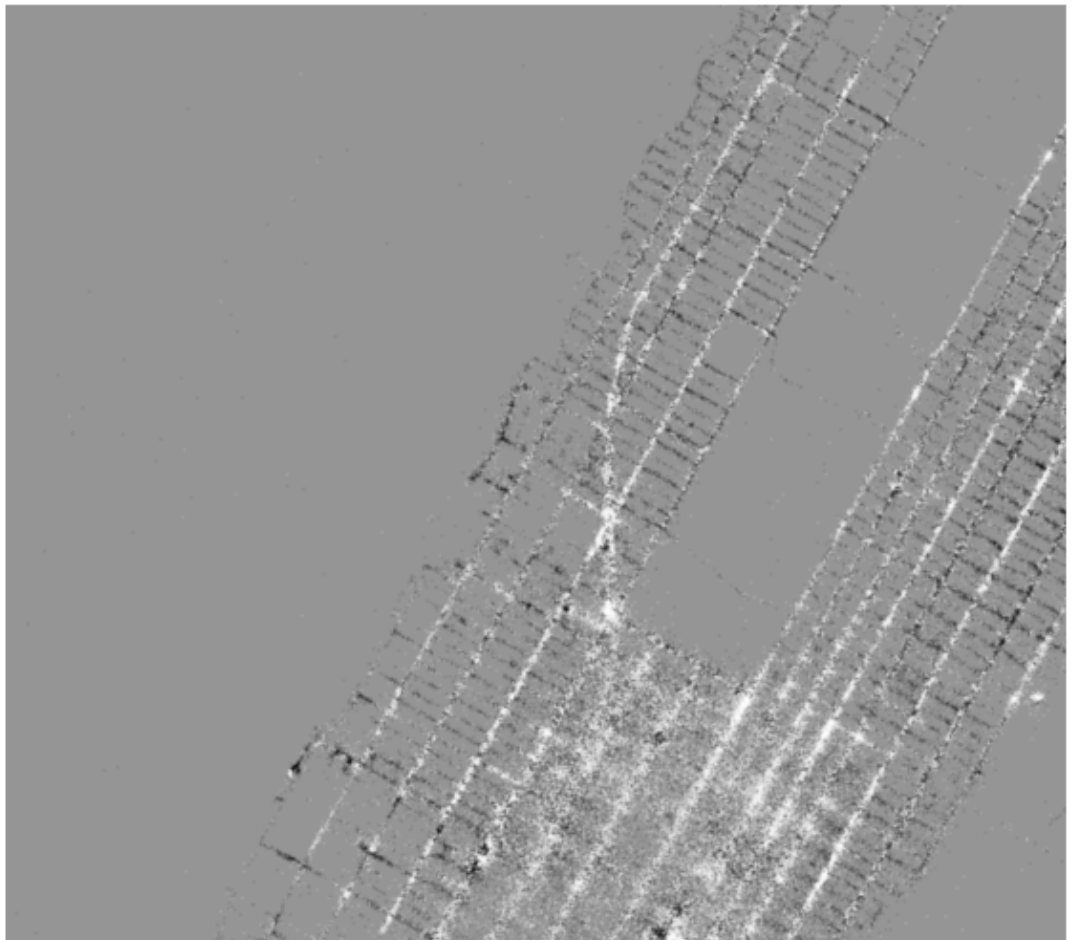
```

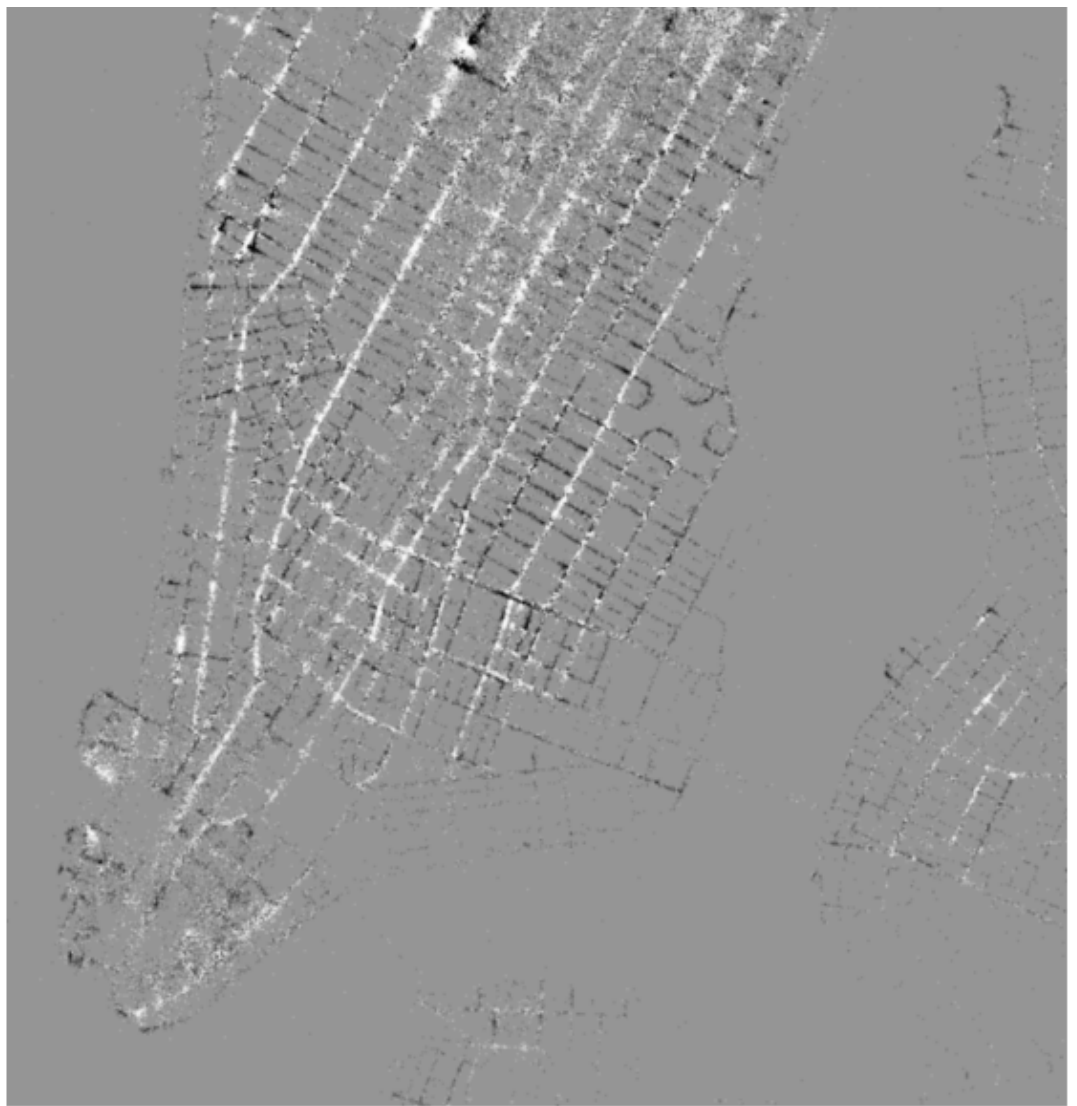
```

In [37]: # 밀도 그래프 출력
plt.clf()
plt.figure(figsize=(8, 8))
plt.imshow(density,
            origin='lower',
            interpolation='bicubic'
            )
plt.axis('off')
plt.tight_layout()
plt.show()

```

<matplotlib.figure.Figure at 0x7f524ae54cc0>





In []: