사용 Idea	mothod	parameter	parameter 설명	사용 예시
Series --> DataFrame	.to_frame()			
관련 method 조회	dir			dir(pd.Serise) dir(pd.DataFrame)
결측값 비율	.isna().mean()			
인덱스, 컬럼명 중 해당되는 일부 건만 바꿀 수 있음	.rename()	index = { ___ }	딕셔너리로 인덱스명 변경	
		columns = {___}	딕셔너리로 컬럼명 변경	
모든 값이 True인지 확인	.all()			
하나라도 True인지 확인	.any()			
'gross' 컬럼이 몇 번째인지 확인	.get_loc()			movie.columns.get_loc('gross') + 1
원하는 위치에 컬럼 삽입	.insert()	loc	삽입 위치	movie.insert(loc=3, column = 'profit', value = movie['gross'] - movie['budget']
		column	삽입 시 컬럼명	movie의 3번째 열 위치에 profit이라는 이름으로 열을 삽입하고, 값은 value 값
		value	삽입 내용	
데이터 중 원하는 자료형의 컬럼만 선택	.select_dtypes()			movie.select_dtypes(include = ['number']) # 'int' 등 사용
컬럼명에 문자열 조건을 걸어 필터	.filter()	like	조건	movie.filter(like = 'facebook')
컬럼명에 정규 표현식 조건을 걸어 필터		regex	조건	movie(filter(regex = '₩d')
컬럼명에 정확한 문자열 조건을 걸어 필터 (매칭 되는 게 없어도 에러가 발생하지 않음)		item	조건	movie(filter(item = ['actor_name', 'asdfasdfafd')
항목 비교는 set을 사용하자(list는 멍청한 짓)				set(movie.columns) == set(new_col_order)
반올림 시 Numpy는 정확히 중간을 짝수로 반올림하기 때문에 아주 작은 수를 미리 더해주어야 함				np.round(4.5) --> np.round(4.5 + 0.00001) # 첫 번째 결과는 4, 두 번째는 5
두 개의 데이터프레임이 동일한지 검증	assert_frame_equal			from pandas.testing import assert_frame_equal assert_frame_equal(df1, df2)
열 방향 계산(결측값은 포함시키지 않음)	count, sum, ...	axis	0: index, 1: column	df.count(axis = 'columns')
누적 합계	.cumsum()	axis	0: index, 1: column	
누적 최대값, 최소값	.cummax() .cummin()			df['col1'].cummax()
전치	.T			df.describe().T # 굳이 컬럼 하나하나 잘라 붙일 필요 없음
숫자형과 범주형/카테고리형 변수 나눠서 describe	.describe()	include	np.number, np.int64, np.float64, np.object, pd.Categorical, 'int', 'float', 'int64', 'float64'	df.describe(include=[np.object, pd.Categorical])
describe로 percentile까지 확인 가능	.describe()	percentiles		df.describe(percentiles=[.01, .05, .10, .25, .5, .75, .9, .95, .99]).T
컬럼별 메모리 사용량 확인	.memory_usage()			df.memory_usage(deep=True) # deep=True로 정확한 값 측정
메모리가 많이 필요 없는 컬럼을 줄여주기	.astype()			df[col1].astype(np.int8) # 1, 0만 있는 컬럼은 1byte(8bit)로 변환
object형 컬럼만 선택..	.select_dtypes()	include		
고윳값 개수 확인..	.nunique()			df.select_dtypes(include=['object']).nunique()
고윳값이 많지 않은 경우 Categorical로 변경해 주면 메모리 절약 가능				
인덱스 설정 시 최소한의 메모리: RangeIndex, 모든 행 인덱스를 메모리에 저장: Int64Index	.Int64Index()			pd.Int64Index(df.index) # 이렇게 변환하는 게 메모리는 많이 잡아먹음
상위 100위, 하위 100위	.nlargest() .nsmallest()			df.nlargest(100, ['col1', 'col2'])
col1, col2 정렬 후 col1 클래스별 최대값만 남기기	.sort_values() .drop_duplicates()			df.sort_values(['col1', 'col2'], ascending=False).drop_duplicates(subset='col1')
컬럼별 오름차순, 내림차순 달리하기		ascending	리스트로 값을 주면 됨	df.sort_values(['col1', 'col2'], ascending=[True, False])
몇 번째 컬럼인지 찾기	.columns.get_loc()			df.columns.get_loc('col1')
스칼라값(단일값)을 찾을 때는 .loc, iloc보다 .at, .iat가 빠름	.at() .iat()			df.at['lemon', 'Tree'] df.iat[10, 10] # 둘 다 Series에서도 사용 가능
df에 조건 걸 시 대괄호 매번 치는 것보다 함수 쓰는게 좋음	.gt() / .ge() / .lt() / .le() .eq() / .ne()		초과, 이상, 미만, 이하 같다, 다르다	df['col1'].gt(120).mean() # 훨씬 간단
df에 조건 걸 시 대괄호 매번 치는 것보다 함수 쓰는게 좋음	.add(), .sub(), .mul(), .div(), .pow()		더하기, 빼기, 곱하기, 나누기, 제곱	
df에 조건 걸 시 대괄호 매번 치는 것보다 함수 쓰는게 좋음	.floordiv() .mod()		몫, 나머지	
nan을 특정값으로 대치하면서 연산	.add()	fill_value	특정값 지정	Series1.add(Series2, flil_value = 0)
데이터프레임간 정확한 비교는 .eq()가 아님	.equals()			df1.equals(df2)
고유한지 여부 확인	.is_unique			df['col1'].is_unique df.index.is_unique
인덱스 정렬 후, 인덱스를 통한 검색이 훨씬 빠름				df.sort_index().loc['Lemon'] crit1 = college['CITY'] == 'Miami' crit2 = college['STABBR'] == 'FL' college[crit1 & crit2] # 2.43ms college.index = college['CITY'] + ' , ' + college['STABBR'] college = college.sort_index() college.loc['Miami, FL'] #197ns
원소 포함 여부 확인	.isin()			a = ['a'] b = ['a', 'b', 'c'] a.isin[b]
Series는 Between method가 있음	.between()			df['col1'].between(1000, 2000)
변동률 계산 (윗 행 대비 변동률)	.pct_change()			df['col1'].pct_change()
query 사용	.query()			qs = "DEPARTMENT not in @top10_depts and GENDER == 'Female'" df.query(qs)
조건 외 값은 모두 대체	.where()	other	대체할 값, 디폴트는 na	crit = df['col1'] < 2000 df['col1'].where(crit, other = 2000) # col1이 2000 이상이면 2000으로 대체
상한, 하한 동시에 지정	.clip()	lower upper	하한 상한	df['col1'].clip(lower=1000, upper=2000)

사용 Idea	mothod	parameter	parameter 설명	사용 예시
조건에 맞는 값은 모두 na로 대체(.where와 정반대)	.mask()			crit = df['col1'] >= 1000 df.mask(crit)

set간 합집합, 차집합, 대칭 차집합	.union(), .difference()
	.symmetric_difference()
				set1.union(set2) # set1 | set2 와 같음 // set1.difference(set2)
				set1.symmetric_difference(set2) # set1 ^set2 와 같음

인덱스 순서가 완벽히 동일하지 않으면 카디션 곱 발생 --> 연산 전 인덱스를 정렬하는 것이 중요				
tmp1 = df['col1'] tmp2 = df['col1']				# tmp1과 tmp2는 같은 객체를 참조하므로 동일한 것. # tmp1의 원소를 수정하면 df와 tmp2도 영향을 받는다
결측값 여부 확인	.hasnans			Series1.hasnans # True, False
데이터프레임 중 결측값 색깔 표시	.style.highlight_null('yellow')			df.style.highlight_null('yellow')
열별 최대값에 색깔 표시	.style.highlight_max()	axis='columns'		df.style.highlight_max() df.style.highlight_max(axis = 'columns') # 이렇게 하면 행별 최대값
에러 반영한 자료형 변환 : to_numeric	pd.to_numeric()	errors = 'coerce' errors = 'ignore'		pd.to_numeric(df['col1'], errors = 'coerce') # 에러를 nan으로 반환 --> float로 바뀜 # 'ignore'시 원래의 값 반환
열별 최대값의 인덱스 번호 반환	.idxmax()	axis='rows'		df.idxmax() # axis='rows' --> 행 별
aggregation, groupby, apply				
groupby의 네 가지 메서드와 받아들일 수 있는 함수 : agg(함수 결과가 스칼라), filter(함수 결과가 불리언), transformation(함수 결과가 전달된 데이터와 동일한 길이의 Series), apply(스칼라, Series, DataFrame)				
groupby에 사용할 수 있는 종합함수들				min, max, mean, median, sum, count, std, var size, describe, nunique, idxmin, idxmax. value_counts
groupby + agg() 사용 예시 1 : 각 컬럼에 적용할 함수를 딕셔너리로 표현				df.groupby(['col1','col2']).agg({'col3':['sum','min'], 'col4':['mean','var']})
groupby + agg() 사용 예시 2 : 각 함수를 각각의 열에 적용(사용자 함수는 따옴표 생략)				df.groupby(['col1','col2'])['col3','col4'].agg([def1, def2, 'mean', 'max'])
groupby 후 MultiIndex 처리하는 방법 - 컬럼명 # 행 인덱스는 reset_index() 활용	.get_level_values()			level0 = df_grouped.columns.get_level_values(0) level1 = df_grouped.columns.get_level_values(1) df_grouped.columns = level0 + '_' + level1
	.reset_index()	**col_level**	기존의 컬럼 레벨 중 어디로 맞출 것인지 위부터 0, 가장 아래는 -1	df.reset_index(col_level=-1)
groupby 기준 열을 인덱스로 만들지 않음	.groupby()	**as_index=False**	reset_index() 필요 없음	df.groupby(['col1'], as_index=False)['col2'].agg('mean')
함수 속성 이름 변경 --> groupby 연산 후 생성되는 컬럼명 preset	.__name__			def1.__name__ = 'Lemon Def' df.groupby(['col1'])['col2'].agg([def1]) --> 생성 컬럼명이 'Lemon Def'
*args : 정해지지 않은 수의 인수 --> 0개도 가능 **kwargs : 정해지지 않은 수의 키워드 인수(key : item)				
메서드에 대한 정의(시그니처) 확인	import inspect inspect.signature(메서드)			inspect.signature(df_grouped.agg) # groupby된 df 객체의 agg 메서드에 대한 시그니처 <Signature (arg, *args, **kwargs) >
groupby에 사용자 함수 사용시 인수 input				df.groupby(['col1'])['col2'].agg(def1, arg1, arg2 ...) # 인수가 들어갈 경우 함수는 한 가지밖에 사용하지 못함
클로저, 지역변수, 전역변수, 함수 반환	클로저는 개념이 어려워 추가 공부가 필요함			def calc():
사용 가능한 함수 확인				print([attr for attr in dir(대상) if not attr.startswith('_')])

	from IPython.display import display
	display()
				for name, group in df.groupby(['col1']):
				  print(name)
				  display(group.head(3)) # 그룹별 name과 데이터프레임을 출력

group 개수 출력	.groupby() .ngroups			df_grouped = df.groupby('col1') df_grouped.ngroups # 그룹 수 출력됨 df['col1'].nunique() # 동일한 결과
한 함수에 대한 그룹별 필터링 ※ 그룹핑된 결과가 아니라 원래 df에서 필터된 결과가 나옴 즉, 그룹단위로 필터는 하되, 결과는 그룹핑 안 된 형식	**.groupby()** **.filter()**			df_grouped = df.groupby('col1') df_grouped_filtered = df_grouped.filter(def1, arg1, arg2 ...)
전 행(row) 비 변화율 --- ① 그룹 없이				def func1(s): # s는 계산할 시리즈 데이터 return (s / s.iloc[0) / s.iloc[0] func1(df['col1'])
전 행(row) 비 변화율 --- ② 그룹별	**.groupby()** **.transform()**			df.groupby(['col1'])['col2'].transform(func1) # col1별 그룹 각각에 대해 col2의 전 행 비 변화율을 보여줌
pivot	.pivot()			df.pivot(index = 'col1', columns='col2', values ='col3') # col1의 값 별(행), col2의 값 별(열), col3의 값
if-then-else	np.where()			np.where(df['col1'] > df['col2'], 'A', 'B') # col1의 값이 더 크면 A, 아니면 B를 반환하는 시리즈
카테고리 순서 정렬되어 출력되게 하기	.Categorical()			order = ['Jan', 'Feb', 'Mar', 'Apr'] df['month'] = pd.Categorical(df['month'], categories=order, ordered=True) # 이 후 pivot을 하면 month에 대해서는 위에서 지정한 order 순으로 출력됨 # pd.Categorical(df['col1'])을 하면 col1의 전체 길이와 카테고리 클래스가 정리됨. pd.Categorical(df['col1']).codes는 원래 col1의 값을 카테고리 클래스 중 몇 번째인지 숫자로 변환한 값.

groupby + apply()	.groupby()
	.apply())
				grouped.apply(func1)
				
				여기서 함수 a의 return 값이 스칼라면 apply 결과도 그룹별 스칼라로,
				return 값이 Series면 apply 결과도 그룹별 Series로 출력됨
				예시)
				def func1(df):
				  cal1 = df['B']*df['C'] # df['B'][0]*df['C'][0]부터 차례차례 계산
				  cal2 = df['B']*df['C'].mean() # df['B'][0] * df['C'].mean() 고정값
				  return cal1 # apply 적용시 그룹의 모든 행의 cal1값이 출력
				
				def a(df):
				  cal1 = df['B']*df['C']
				  return cal1.sum() # apply 적용 시 그룹별 ca1의 sum() 값이 출력됨

pd.cut()	.cut()			bins = [-np.inf, 200, 500, 1000, 2000, np.inf] # 나눌 구간 기준
				labels = ['L1', 'L2', 'L3', 'L4', 'L5', 'L6] # 구간별 이름
				cuts = pd.cut(df['col1'], bins=bins) # cut 된 col1 값을 출력
				df.groupby(cut2)['col1'].value_counts() # df에 cuts를 붙이지 않아도 바로 사용 가능
				.unstack() # pivot처럼 cut별 그룹별 정리

그룹별 항목의 개수	.size()			>> df.groupby(['col1', 'col2']).size() col1 col2 A B 10 C 15 D 12 B A 5 **>> df.loc[[('A', 'B'), ('B', 'A')]] # 인덱스가 중복일 때는 튜플로 지정)** A B 10 B A 5
df.apply(sorted, axis=1) 과 np.sort(df)의 속도 비교 # np.sort가 압승	.apply + axis=1 은 속도저하의 주원인			

사용 Idea	mothod	parameter	parameter 설명	사용 예시
pd.DataFrame()		columns	columns명 설정	pd.DataFrame(array, columns = col_name) # col_name은 리스트 형식
stack, melt, unstack, pivot				
열을 인덱스로 만들기(↔ reset_index())	.set_index()			df.set_index('col1')
stack()은 컬럼명을 "인덱스"로 만듦	기존 인덱스 + 새로운 인덱스 = 멀티 인덱스			df.stack() # 기존의 인덱스에 새로운 인덱스(컬럼명)가 합쳐진 멀티인덱스 생성

melt()는 컬럼명을 "열"로 만듦	기존 열 + 새로운 열 = 열 두 줄			df.melt(id_vars = ['col1'], # 첫 번째 기준열
				value_vars = ['col2', 'col3', 'col4'] # 두 번째 기준열이 될 컬럼명
				var_name = 'New_col' # 두 번째 기준열의 이름
				value_name = 'value_nm') # 데이터 열의 새로운 이름

여러 종류의 변수명 그룹이 있고, stack을 하고 싶을 때
wide_to_long : stack할 변수명들이 연속된 숫자로 끝나면 좋음
	.wide_to_long()			stubs = ['A', 'B'] # 컬럼명이 A_1, A_2, A_3, B_1, B_2, B_3의 형태로 되었을 때
				pd.wide_to_long(df, stubnames = stubs, # stack할 그룹 명
				i=['col1'], # 살려둘 기준 컬럼
				j='New_cnt', # stubs의 끝자리 숫자가 출력됨. 그 컬럼의 이름
				sep='_') # stubname과 끝자리 숫자를 구분하는 기호

unstack()은 "인덱스"를 컬럼명으로 만듦	.unstack()			df_stacked = df.stack() df_stacked.unstack() # 인덱스가 다중일 때 왼쪽부터 0임. # 따라서 .unstack(0)을 실행하면 바깥쪽 인덱스가 unstack이 됨

pivot()은 "열"을 컬럼명으로 만듦	.pivot()			df_melted.pivot(index = 'col1', # 남기고 싶은 열. 인덱스로 형태가 변하긴 한다
				columns = 'col2', # stack/melt되었던 컬럼. 컬럼명이 된다
				values = 'col3') # 값

pivot_table	.pivot_table()	index columns values aggfunc fill_value : null값 처리		df.pivot_table(index = ['col1', 'col2'], columns = ['col3', 'col4'], values = ['col5', 'col6'], aggfunc = [np.sum, np.mean], fill_value=0)
인덱스나 열 레벨에 이름 붙이기	.rename_axis()	axis='columns'		df.rename_axis(['Cat1', 'Cat2'], axis='columns') # 열 레벨(변수명)이 2중일 일 때 이름을 붙여줌 # [None, None]으로 하면 삭제가 됨
멀티인덱스 정렬순서 바꾸기	.swaplevel()	axis='index'		df.swaplevel('col1', 'col3', axis='index') # 인덱스에서 이름이 col1인 것과 col3인 것의 위치를 바꿈
컬럼명도 인덱스임 = sort_index 적용 가능	.sort_index()	axis='columns'		df.sort_index(axis='columns')
데이터프레임의 문자열 나누기	.str.split()	pat='/' expand=True	구분자. 공백이 디폴트 True일 경우 컬럼을 쪼갬	df['col1'].str.split(pat='/', expand=True) # /를 기준으로 컬럼을 나눔
컬럼의 첫 글자 따기	.str			df['col1'].str[0] # 컬럼값의 첫 글자만 딴 컬럼
MultiIndex 컬럼명에서 한 레벨 없애기	.droplevel()			df.columns.droplevel(0) # 컬럼명이 여러줄일 때 0(첫번째=제일위) 라인을 지움
melt와 pivot_table 예시				sensors.melt(id_vars=['Group', 'Property'], var_name='Year') ₩ .pivot_table(index=['Group', 'Year'], columns='Property', values='value') ₩ .reset_index() ₩ .rename_axis(None, axis='columns')
Pandas 객체 합치기				>> df Name Amount 0 Apple 5 1 Banana 4
직접 행 만들어서 추가 - 리스트 사용	.loc[] +[]			new_row = ['Lemon', 3] df.loc[2] = new_row ※ 기존 인덱스 번호를 사용하게 되면 데이터를 덮어씌운게 된다.
직접 행 만들어서 추가 - 딕셔너리, Series 사용	.loc[] + {} .loc[] + pd.Series			df.loc[len(df)] = {'Name':'Peach', 'Amount':7} df.loc[len(df)] = pd.Series({'Name':'Melon', 'Amount':2})
직접 행 만들어서 추가 - 딕셔너리 + append	.append()	이 방법은 인덱스를 재설정 함		df.append({ 'Name' : 'Apple', 'Amount' : 2 } , ignore_index=True)
직접 행 만들어서 추가 - Series + append	.append()	인덱스 이름을 미리 설정해서 기존 인덱스 유지 가능		s1 = pd.Series({'Name' : 'Pineapple', 'Amount' : 3 }, name = len(df)) s2 = pd.Series({'Name' : 'Orange', 'Amount' : 2 }, name = '하핫') # 새 인덱스를 df의 길이(len(df))로 설정하기 위함 df.append([s1, s2]) # 여러 개의 시리즈 추가
pd.concat()	pd.concat()	keys=[]	합치는 df 각각의 이름이 MultiIndex로 붙음	pd.concat([df1, df2], keys=['2016', '2017'], names=['Year', 'Team_name']) # None으로 하나는 생략 가능
		names=[]	MultiIndex의 이름	
		join = 'inner'	인덱스 기준으로 합침 디폴트는 'outer'	pd.concat([df1, df2], join = 'inner', axis='columns')
(2010, 2020, 2030) 과 (1, 2, 3)을 (2010, 1), (2020, 2), (2030, 3)으로 묶기	**zip()**			a = (2010, 2020, 2030) b = (1, 2, 3) zip(a, b) # 튜플 형태로 결과물
위를 딕셔너리 하나로 출력	dict(zip ())			dict(zip(a, b)) # 딕셔너리 형태로 결과물 {2010:1, 2020:2, 2030:3}
	pd.concat(dict(zip ())			pd.concat(dict(zip (years, dfs)) # 각 year가 key로 MultiIndex의 이름으로 붙으며 합침
join으로 합치기	.join	lsuffix	왼쪽에서 오는 데이터 변수에 붙는 문자열	df_2020.join(df_2021, lsuffix = '_2020', rsuffix='_2021', how='outer')
		rsuffix	오른쪽에서 오는 데이터 변수에 붙는 문자열	
		how	join 방식 'outer', 'inner'	
		on	join 기준 변수 설정 merge와 같은 효과	
변수명 전체에 문자열 붙이기	.add_suffix()			df.add_suffix('_2020') # df 모든 변수명에 _2020이 붙음
merge로 합치기	.merge()	suffixes	변수명 꼬리 문자열 설정	df1.merge(df2, left_index=True, right_index=True, how='outer', suffixes=('_2020', '_2021'), on=['col1', col2'])
시계열 분석				
파이썬의 날짜 시간 형식 : date, time, datetime	.date() .time() .datetime()			import datetime date = datetime.date(year = 2020, month = 9, day = 1) # 2020-09-01 time = datetime.time(hour = 12, minute = 30, second = 10, microsecond = 123456) # 12:30:10.123456 dt = datetime.datetime(year = 2020, month = 9, day = 7, hour = 12, minute = 30, second = 10, microsecond = 123456) # 2020-09-07 12:30:10.123456)
	.timedelta()	date와 datetime에는 timedelta를 더할 수 있으나, time에는 불가능		datetime.timedelta(weeks = 2, days = 5, hours = 10, minutes=20, seconds = 6.73, milliseconds=99, microseconds=8) # 19 days, 10:20:06.829008

사용 Idea	mothod	parameter	parameter 설명	사용 예시
Pandas의 timestamp	pd.Timestamp()	단일 스칼라값을 Timestamp로 변환 디폴트는 1970-01-01		pd.Timestamp(year=2012, month=12, day=21, hour=5, minute=10, second=8, microsecond=99) pd.Timestamp('2016/1/10') pd.Timestamp('2014-5/10') pd.Timestamp('Jan 3, 2019 20:45.56') pd.Timestamp('2016-01-05T05:34:43.123456789') pd.Timestamp(500)

	pd.to_Datetime()	리스트, Series를 Timestamp로 변환 가능
		디폴트는 1970-01-01
				pd.to_datetime('2015-5-13')
				pd.to_datetime('2015-5-13', format = '%Y-%m-%d') #이게 40배 빠름
				pd.to_datetime('2015-13-5', dayfirst=True)
				pd.Timestamp('Saturday September 30th, 2017')
				pd.to_datetime('Start Date: Sep 30, 2017 Start Time: 1:30 pm',
				format='Start Date: %b %d, %Y Start Time: %I:%M %p')
				pd.to_datetime(100, unit='D', origin='2013-1-1')
				
				s = pd.Series([10, 100, 1000, 10000])
				pd.to_datetime(s, unit='D')
				
				s = pd.Series(['12-5-2015', '14-1-2013', '20/12/2017', '40/23/2017'])
				pd.to_datetime(s, dayfirst=True, errors='coerce')

	pd.Timedelta()	Timestamp나 다른 timedelta와 가감 가능 # 주, 일, 시, 분, 초, milli/micro/nano		pd.Timedelta('12 days 5 hours 3 minutes 123456789 nanoseconds') pd.Timedelta(days=5, minutes=7.34) pd.to_timedelta('5 dayz', errors='ignore') pd.to_timedelta('67:15:45.454') pd.Timedelta('12 days 5 hours 3 minutes') * 2 pd.Timedelta('12 days') / pd.Timedelta('3 days')
	pd.DateOffset()	Timestamp와 가감 가능 # 연, 월, 일, 시, 분, 초 가능		pd.DateOffset(days=3, hours=8, seconds=10)
Timestamp의 속성과 메서드				ts = pd.Timestamp('2016-10-1 4:23:23.9') # Timestamp('2016-10-01 05:00:00')
올림	.ceil()			ts.ceil('h') # 시간으로 올림 # Timestamp('2016-10-01 05:00:00')
연, 월, 일, 시, 분, 초	.year .month .day .hour .minute .second			ts.year, ts.month, ts.day, ts.hour, ts.minute, ts.second
주, 월, 연 중 몇 번째 날인지	.dayofweek .dayofmonth .dayofyear			ts.dayofweek, ts.dayofyear, ts.daysinmonth
파이썬 datetime 형식으로 변환	.to_pydatetime()			ts.to_pydatetime()
Timedelta의 속성과 메서드				td = pd.Timedelta(125.8723, unit='h') # Timedelta('5 days 05:52:20.280000')
반올림	.round()			td.round('min') # Timedelta('5 days 05:52:00')

일, 시간, 분, 초, milli/micro/nano seconds 표시	.components			td.components # Components(days=5, hours=5, minutes=52, seconds=20,
				milliseconds=280, microseconds=0, nanoseconds=0)

초 단위로 변환	.total_seconds()			td.total_seconds() # 453140.28
dtype이 Timestamp인 변수를 인덱스로 만들어서 사용하기				df = df.set_index('col_ts') # Timestamp를 인덱스로 만들면 위와 같이 조회 가능 df.loc['2020-01-01 10:00:00':'2020-01-03 11:00:00'] # 슬라이스 가능 df.loc['2020-01-01':'2020-01-03 11'] # 시작 끝 형식이 다른 슬라이스도 가능 df.loc['2020-03-01 14] # 연월일시로 조회 df.loc['2020-04'] # 연월로 조회 df.loc['2020'] # 연으로 조회 crime.loc['Dec 2015'].sort_index() # 문자로 된 날짜도 사용 가능 + 인덱스 정렬
DatetimeIndex에 작동하는 메서드				
날짜 관계없이 시간 기준으로만 데이터 조회	.between_time()			crime.between_time('2:00', '5:00', include_end=False)
특정 시간의 데이터만 조회	.at_time()			crime.at_time('5:47')
첫 n일, 영업일, 주(월화수목금토일), 분기, 연말 / 뒤에서부터 세는 것도 있음	.first() .last()			crime_sort.first('5D') # 5일치 crime_sort.first('5B') # 5영업일치 crime_sort.first('7W') # 7주치 crime_sort.first('3QS') # 3분기의 마지막날까지(표시는 9/30이 찍힘) crime_sort.first('A') # 당해 연말까지
Timestamp 그룹핑				

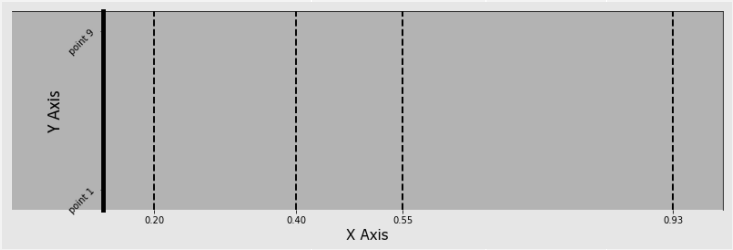
주단위로 그룹핑	.resample()	groupby같은 기능		df.resample('W').size() # 일요일을 마지막날로 일주일씩 카운트(월~일)
				df.resample('W--THU').size() # 금~목
				
				df.resample('Q').size() # 분기별(인덱스에는 분기 마지막일이 찍힘, 3/31, 6/30 ...)
				df.resample('QS-MAR').size() # 3월 시작 기준 분기별(분기 시작일이 찍힘, 3/1, 6/1...)

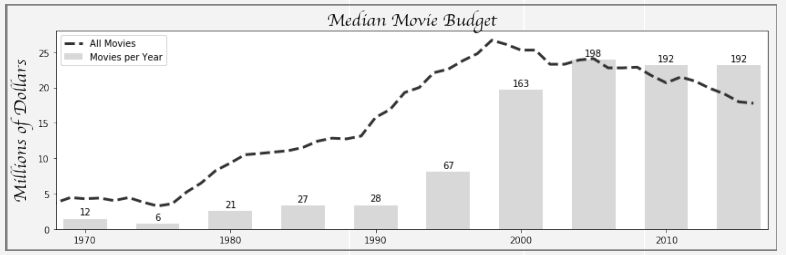
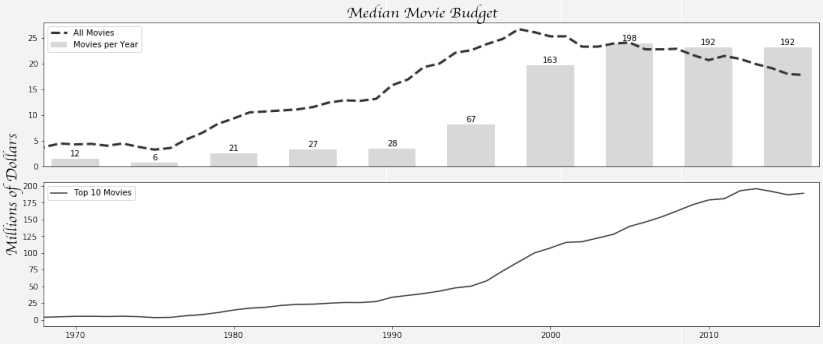
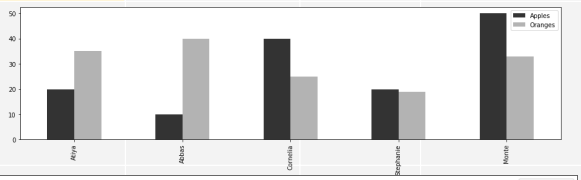
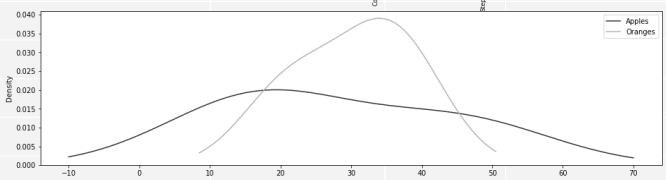
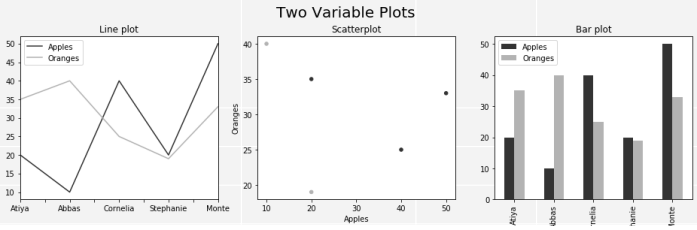
주단위로 그룹핑	.resample()	on	인덱스가 Timestamp가 아니어도 열로 선택가능	df.resample('W', on='REPORTED_DATE').size()
resample은 groupby처럼 사용 가능				crime_sort.resample('QS')['IS_CRIME', 'IS_TRAFFIC'].sum()

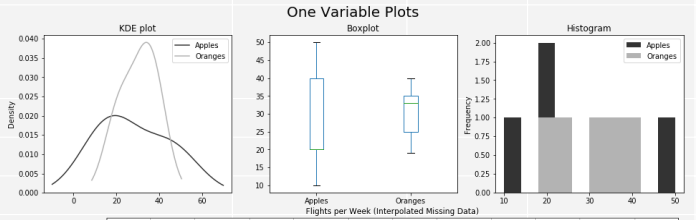
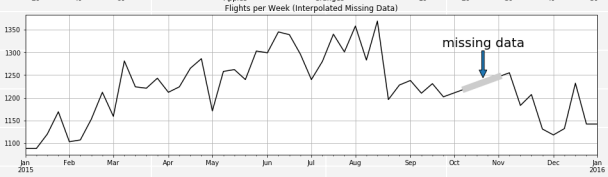
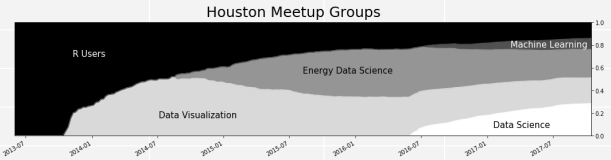
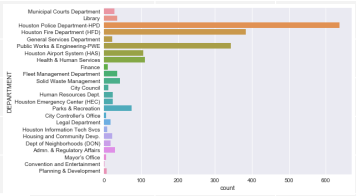
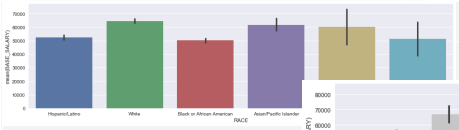
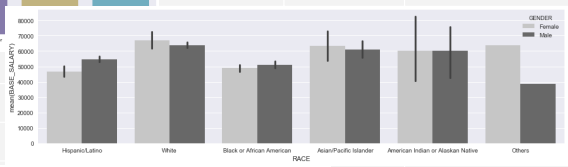
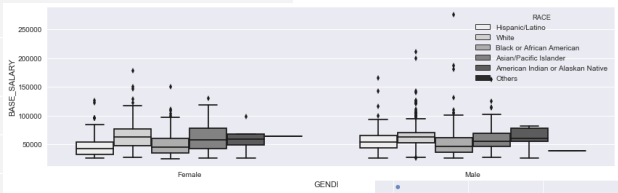
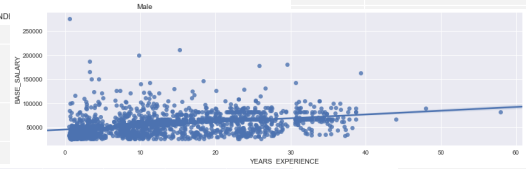
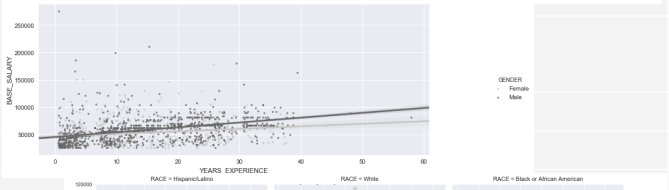
요일별, 연도별 그룹핑	.dt	.weekday_name
		.year
			요일별
			연도별
				crime['REPORTED_DATE'].dt.weekday_name.value_counts() # 요일별 count
				
				weekday = crime['REPORTED_DATE'].dt.weekday_name
				year = crime['REPORTED_DATE'].dt.year
				crime_wd_y = crime.groupby([year, weekday]).size() # 연, 요일별 count
				crime_table = crime_wd_y.rename_axis(['Year', 'Weekday']).unstack('Weekday')

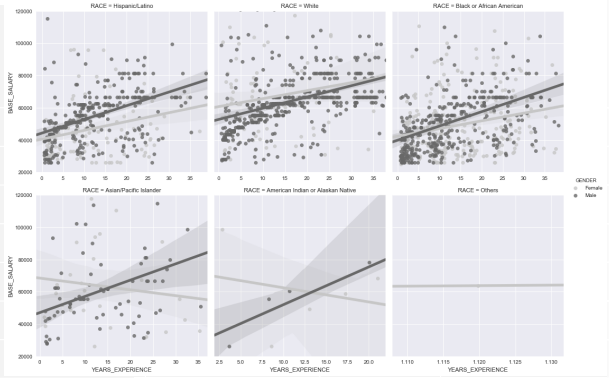
인덱스를 원하는 대로 정렬	.reindex()			days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'] wd_counts.reindex(days)
인덱스가 DatetimeIndex일 경우 인덱스를 그룹핑에 사용 groupby + lambda	.groupby() lambda	df의 인덱스를 인수로 받아 lambda에 사용		crime_sort.groupby(lambda x: x.weekday_name)['IS_CRIME', 'IS_TRAFFIC'].sum()
일반 변수와 시간 변수 함께 그룹핑 resample과 groupby 함께 사용 가능 ※ 인덱스는 DatetimeIndex여야함 --> 그룹별로 시간 텀이 다른 문제가 발생 --> pd.Grouper 아니면 pd.cut 사용해도 됨	.groupby() pd.Grouper()			employee.groupby('GENDER').resample('10AS')['BASE_SALARY'].mean() # 한 그룹은 1955, 1965.. 다른 그룹은 1958, 1968.. 과 같은 문제 발생 employee.groupby(['GENDER', pd.Grouper(freq='10AS')])['SALARY'].mean() # pd.Grouper(freq='10AS')로 10년 주기 시간을 함께 그룹핑 가능 cuts = pd.cut(employee.index.year, bins=5, precision=0) employee.groupby([cuts, 'GENDER'])['SALARY'].mean().unstack('GENDER')
시각화				
Matplotlib				

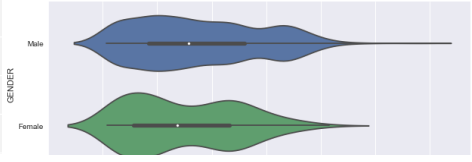
사용 Idea	mothod	parameter	parameter 설명	사용 예시
두 가지 방법이 있음: 1) 상태 기반, 2) 객체지향				x = [-3, 5, 7] y = [10, 2, 5] 1) 상태 기반 plt.figure(figsize=(15,3)) plt.plot(x, y) plt.xlim(0, 10) plt.ylim(-3, 8) plt.xlabel('X Axis') plt.ylabel('Y axis') plt.title('Line Plot') plt.suptitle('Figure Title', size=20, y=1.03) 2) 객체지향 fig, ax = plt.subplots(figsize=(15,3)) ax.plot(x, y) ax.set_xlim(0, 10) ax.set_ylim(-3, 8) ax.set_xlabel('X axis') ax.set_ylabel('Y axis') ax.set_title('Line Plot') fig.suptitle('Figure Title', size=20, y=1.03)

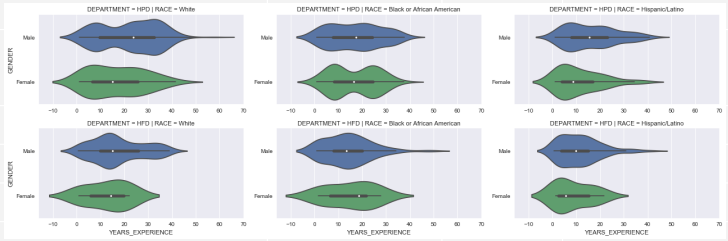
				**## 기본 fig, axes**
				fig, ax = plt.subplots(nrows=1, ncols=1, # fig와 ax 객체 생성, axes 개수도 지정
				, linewidth=5, edgecolor='.5') # fig 테두리 설정
				fig.get_size_inches() # 크기 확인
				fig.set_size_inches(14, 4) # Figure 크기 지정
				fig.set_facecolor('.9') # fig 색상 0~1사이 실수
				ax.set_facecolor('.7') # ax의 색상
				
				## Spine
				ax.spines['left'].set_positoin(('outward', -100)) # ax 4개 뼈대 중 left뼈대의 위치 변경
				ax.spines['right'].set_linewidth(5) # right 뼈대의 선 너비 변경
				ax.spines['top'].set_visible(False) # top 뼈대 보이지 않게 하기
				
				## Axis
				ax.xaxis.grid(True, which='major', linewidth=2, color='black', linestyle='--')
				ax.xaxis.set_ticks([.2, .4, .55, .93]) # x축 눈금 직접 지정
				ax.xaxis.set_label_text('X Axis', family='Verdana', fontsize=15)
				
				ax.set_ylabel('Y Axis', family='Calibri', fontsize=20)
				ax.set_yticks([.1, .9])
				ax.set_yticklabels(['point 1', 'point 9'], rotation=45) # y축 눈금에 숫자 대신 별칭
				
				text_kwargs=dict(fontsize=20, family='cursive') # font 설정을 미리 할 수도 있음
				ax.set_title('Median Movie Budget', **text_kwargs)
				ax.set_ylabel('Millions of Dollars', **text_kwargs)
				
				## Plot
				ax.plot(years, budget, linestyle='--', linewidth=3, color='.2', label='All Movies')
				#선 그래프
				ax.bar(years_5, ct_norm_5, 3, facecolor='.5', alpha=.3, label='Movies per Year')
				# 막대 그래프
				for x, y, v in zip(years_5, ct_norm_5, ct_5): # 레이블링
				ax.text(x, y + .5, str(v), ha='center')
				ax.legend() # 그래프별 범례
				fig # 그래프 프린트

				
				fig2, ax_array = plt.subplots(2, 1, figsize=(14,6), sharex=True) ax1 = ax_array[0] ax2 = ax_array[1] ax1.plot(years, budget, linestyle='--', linewidth=3, color='.2', label='All Movies') ax1.bar(years_5, ct_norm_5, 3, facecolor='.5', alpha=.3, label='Movies per Year') ax1.legend(loc='upper left') ax1.set_xlim(1968, 2017) plt.setp(ax1.get_xticklines(), visible=False) for x, y, v in zip(years_5, ct_norm_5, ct_5): ax1.text(x, y + .5, str(v), ha='center') ax2.plot(years, top10_roll.values, color='.2', label='Top 10 Movies') ax2.legend(loc='upper left') fig2.tight_layout() fig2.suptitle('Median Movie Budget', y=1.02, **text_kwargs) fig2.text(0, .6, 'Millions of Dollars', rotation='vertical', ha='center', **text_kwargs) import os path = os.path.expanduser('~/Desktop/movie_budget.png') fig2.savefig(path, bbox_inches='tight')
pandas의 시각화				
				Apples Oranges Atiya 20 35 Abbas 10 40 Cornelia 40 25 Stephanie 20 19 Monte 50 33 color = ['.2', '.7'] df.plot(kind='bar', color=color, figsize=(16,4))
				df.plot(kind='kde', color=color, figsize=(16,4))
다변량 (변수가 두 개 / x, y)				fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(16,4)) fig.suptitle('Two Variable Plots', size=20, y=1.02) df.plot(kind='line', color=color, ax=ax1, title='Line plot') df.plot(x='Apples', y='Oranges', kind='scatter', color=color, ax=ax2, title='Scatterplot') df.plot(kind='bar', color=color, ax=ax3, title='Bar plot')
				

사용 Idea	mothod	parameter	parameter 설명	사용 예시
단변량				fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(16,4)) fig.suptitle('One Variable Plots', size=20, y=1.02) df.plot(kind='kde', color=color, ax=ax1, title='KDE plot') df.plot(kind='box', ax=ax2, title='Boxplot') df.plot(kind='hist', color=color, ax=ax3, title='Histogram')
그래프에 특이사항 표시				ax.annotate(xy=(.8, .55), xytext=(.8, .77), xycoords='axes fraction', s='missing data',
				Pandas Cookbook p.587
seaborn의 시각화 **(pandas에 비해 groupby 등의 전처리 단계가 하나 빠짐)**				sns.countplot(y='DEPARTMENT', data=employee)
				ax = sns.barplot(x='RACE', y='BASE_SALARY', data=employee) ax.figure.set_size_inches(16, 4)
				ax = sns.barplot(x='RACE', y='BASE_SALARY', hue='GENDER', data=employee, palette='Greys')
				ax = sns.boxplot(x='GENDER', y='BASE_SALARY', data=employee, hue='RACE', palette='Greys') ax.figure.set_size_inches(14,4)
회귀선 그래프				ax = sns.regplot(x='YEARS_EXPERIENCE', y='BASE_SALARY', data=employee) ax.figure.set_size_inches(14,4)
회귀선이 각각				grid = sns.lmplot(x='YEARS_EXPERIENCE', y='BASE_SALARY', hue='GENDER', palette='Greys', scatter_kws={'s':10}, data=employee) grid.fig.set_size_inches(14, 4)

col 추가로 더 세분화				grid = sns.lmplot(x='YEARS_EXPERIENCE', y='BASE_SALARY',
				hue='GENDER', col='RACE', col_wrap=3, # col로 더 세분화
				palette='Greys', sharex=False,
				line_kws = {'linewidth':5},
				data=employee)
				grid.set(ylim=(20000, 120000))

				ax = sns.violinplot(x = 'YEARS_EXPERIENCE', y='GENDER', data=emp2) ax.figure.set_size_inches(10,4)

사용 Idea	mothod	parameter	parameter 설명	사용 예시
col, row로 세분화				sns.factorplot(x ='YEARS_EXPERIENCE', y='GENDER', col='RACE', row='DEPARTMENT', size=3, aspect=2, data=emp2, kind='violin')