데이터문제해결및실습1

7주차-1
Chapter_06_[경진대회1] 자전거 대여 수요 예측_1

세종대학교
인공지능데이터사이언스학과
박동현 교수

- 본 강의는 골든래빗 출판사의 머신러닝/딥러닝 문제해결전략이 제공하는 강의 교안에 기반함.

Chapter

06

[경진대회]
자전거 대여 수요 예측 1

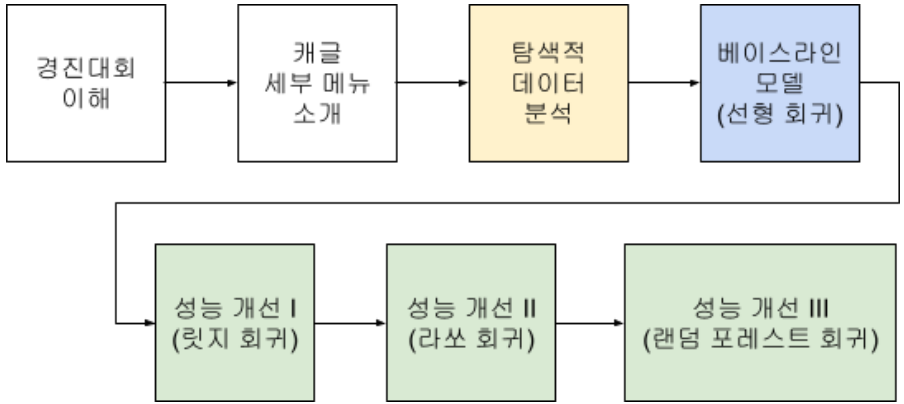
[경진대회] 자전거 대여 수요 예측

난이도	★☆☆		
경진대회명	자전거 대여 수요 예측 경진대회		
미션	날짜, 계절, 근무일 여부, 날씨, 온도, 체감 온도, 풍속 데이터를 활용하여 자전거 대여 수량 예측		
문제 유형	회귀	**평가지표**	RMSLE
데이터 크기	1.1MB	**참가팀 수**	3,242팀
제출 시 사용한 모델	랜덤 포레스트 회귀		
파이썬 버전	3.7.10		
사용 라이브러리 및 버전	• numpy (numpy==1.19.5) • pandas (pandas==1.3.2) • seaborn (seaborn==0.11.2) • matplotlib (matplotlib==3.4.3) • sklearn (scikit-learn==0.23.2) • datetime, calendar		

[경진대회] 자전거 대여 수요 예측

▪ 학습 목표

- 머신러닝 모델링 프로세스를 배워봅시다!
- 기본 회귀 모델을 배우게 됩니다.
- 경진대회 세부 메뉴도 알아봅니다.

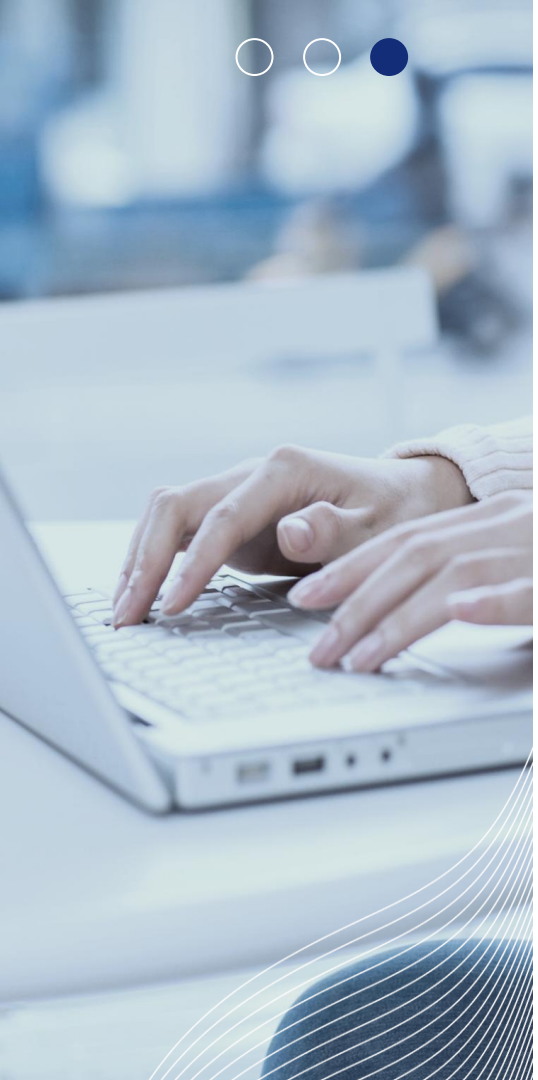


▪ 학습 키워드

- 유형 및 평가지표 : 회귀, RMSLE
- 탐색적 데이터 분석 : 분포도, 막대 그래프, 박스플롯, 포인트플롯, 산점도, 히트맵
- 머신러닝 모델 : 선형 회귀, 릿지 회귀, 라쏘 회귀, 랜덤 포레스트 회귀
- 피처 엔지니어링 : 파생 피처 추가, 피처 제거
- 하이퍼파라미터 최적화 : 그리드서치

▪ 핵심 요약

1. **머신러닝/딥러닝 문제해결 프로세스**: 크게 '경진대회 이해' → '탐색적 데이터 분석' → '베이스라인 모델' → '성능 개선' 순으로 진행
2. **타깃값 변환**: 타깃값이 정규분포에 가까울수록 회귀 모델의 성능이 좋음. 한쪽으로 치우친 타깃값은 로그변환하면 정규분포에 가까워지고, 결괏값을 지수변환하면 원래 타깃값 형태로 복원됨.
3. **이상치 제거**: 훈련 데이터에서 이상치를 제거하면 일반화 성능이 좋아질 수 있음
4. **파생 피처 추가**: 기존 피처를 분해/조합하여 모델링에 도움되는 새로운 피처를 만들 수 있음
5. **피처 제거**: 반대로 불필요한 피처를 제거하면 성능도 좋아지고, 훈련 속도도 빨라짐
6. **선형회귀, 릿지, 라쏘 모델**: 회귀 문제를 푸는 대표적인 모델, but 너무 기본적이라 실전에서 단독으로 최상의 성능을 기대하기는 어려움
7. **랜덤 포레스트 회귀 모델**: 여러 모델을 묶어 (대체로) 더 나은 성능을 이끌어내는 간단하고 유용한 기법입니다.
8. **그리드서치**: 교차 검증으로 최적의 하이퍼파라미터 값을 찾아주는 기법



6.1 경진대회 이해

6.2 경진대회 접속 방법 및 세부 메뉴

6.3 탐색적 데이터 분석

6.4 베이스라인 모델

6.5 성능 개선 I : 릿지 회귀 모델

6.6 성능 개선 II : 라쏘 회귀 모델

6.7 성능 개선 III : 랜덤 포레스트 회귀 모델

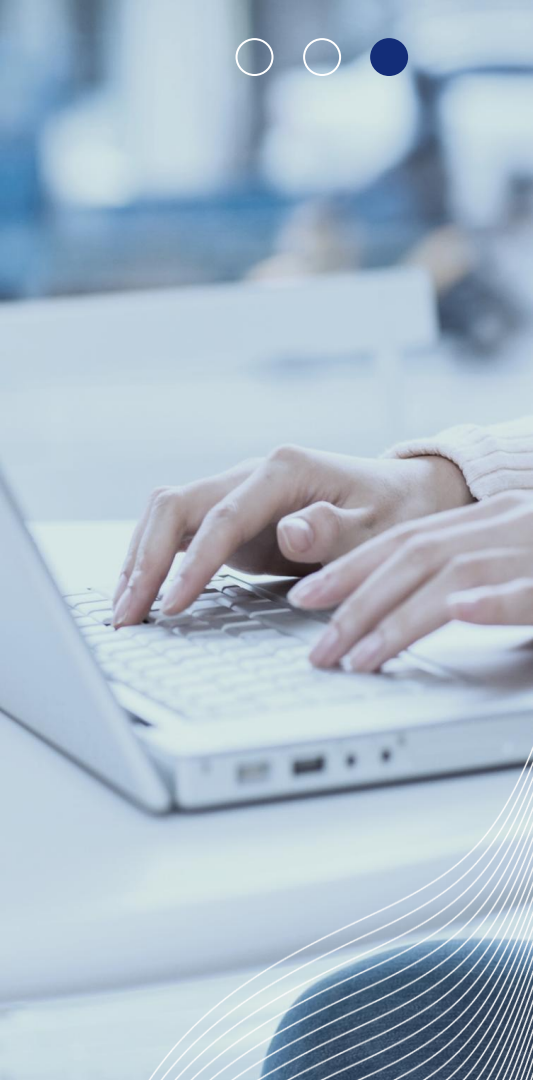
6.1 경진대회 이해

▪ 경진대회 Overview

- 워싱턴 D.C 자전거 무인 대여 시스템 과거 기록을 바탕으로 향후 자전거 대여 수요 예측
- 주어진 데이터는 2011녀부터 2012년까지 2년간 자전거 대여 데이터 + 날씨 데이터
  (대여 날짜, 시간, 요일, 계절, 날씨, 실제 온도, 체감 온도, 습도, 풍속, 회원 여부)
- 훈련 데이터는 매달 1일~19일까지 기록 / 테스트 데이터는 매달 20일~월말까지 기록
- 난이도가 낮은 연습용 대회 (몸 한 번 풀어볼까요?)

◉ 피처와 타깃값이란?

피처(feature, 특성, 특징) : 원하는 값을 예측하기 위해 활용하는 데이터
→ ex. 대여 날짜, 시간, 요일, 계절, 날씨, 실제 온도, 체감 온도, 습도, 풍속, 회원 여부
타깃값(target value, 목푯값, 목표변수, 타깃변수): 예측해야 할 값
→ ex. 대여 수량



6.1 경진대회 이해

6.2 경진대회 접속 방법 및 세부 메뉴

6.3 탐색적 데이터 분석

6.4 베이스라인 모델

6.5 성능 개선 I : 릿지 회귀 모델

6.6 성능 개선 II : 라쏘 회귀 모델

6.7 성능 개선 III : 랜덤 포레스트 회귀 모델

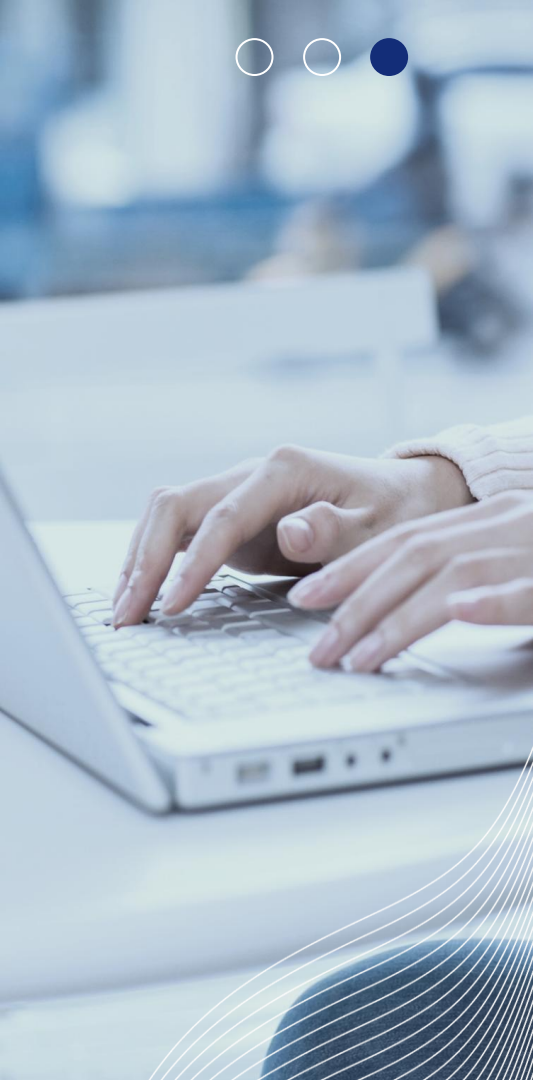
6.2 경진대회 접속 방법 및 세부 메뉴

▪ 경진대회 접속 방법

- https://www.kaggle.com/t/5a7cd9746791445184e06590aef6d68c
- Team Name – “615557-박동현-ch6-eda” –〉 “학번-이름-ch6-eda”
- Notebook 생성 – “615557-박동현-ch6-eda” –〉 “학번-이름-ch6-eda”
- 참고 노트북: https://www.kaggle.com/code/clyde1114/615557-ch6-eda1

▪ 경진대회 메뉴 설명

- Overview 메뉴 : 경진대회 전반을 소개
- Data 메뉴 : 주어진 데이터에 관해 설명
- Code 메뉴 : 다른 참가자가 공유한 코드(노트북)을 볼 수 있음
- Discussion 메뉴 : 인사이트, 주의사항, 질의응답 등 도움되는 내용이 올라옴
- Leaderboard 메뉴 : 참가자의 등수와 점수 확인 가능
- Rules 메뉴 : 대회 규정 메뉴
- Team 메뉴 : 팀을 꾸리는 메뉴



6.1 경진대회 이해

6.2 경진대회 접속 방법 및 세부 메뉴

6.3 탐색적 데이터 분석

6.4 베이스라인 모델

6.5 성능 개선 I : 릿지 회귀 모델

6.6 성능 개선 II : 라쏘 회귀 모델

6.7 성능 개선 III : 랜덤 포레스트 회귀 모델

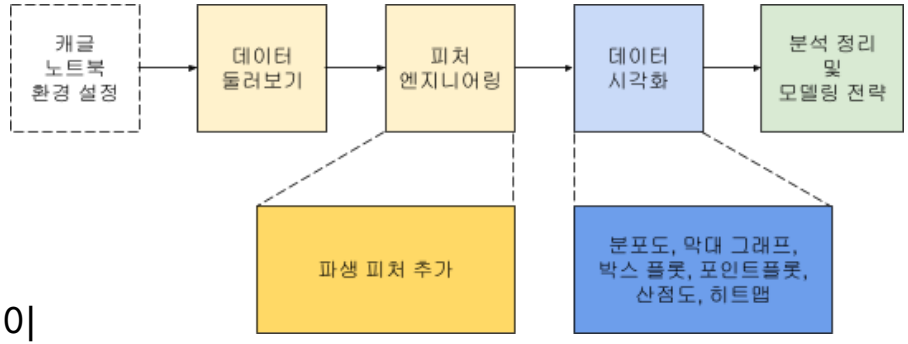
6.3 탐색적 데이터 분석

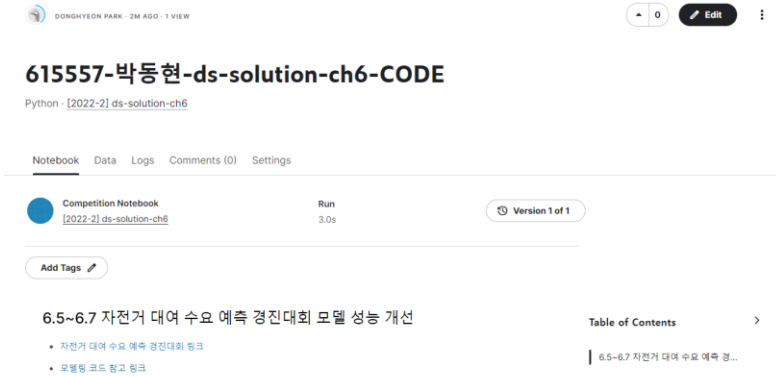
▪ 탐색적 데이터 분석이란?

- 주어진 데이터를 면밀히 살펴보는 단계

▪ 6.3.1 캐글 노트북 환경 설정

- 라이브러리는 시간이 지나면 조금씩 버전업 됨
- 라이브러리 버전을 일치시키기 위해 라이브러리 버전이 고정된 노트북 양식을 복사해서 작업하길 권장
  (장 시작 페이지, 표 마지막 줄의 노트북 양식 링크 참고)
- 복사 방법 : 공유된 노트북의 오른쪽 위 [Copy & Edit] 클릭하면 끝!





6.3 탐색적 데이터 분석

▪ 6.3.2 데이터 둘러보기

- 주어진 데이터가 어떻게 구성되어 있는지 살펴봅시다!
- 판다스로 훈련, 테스트, 제출 샘플 데이터를 DataFrame 형태로 불러오기

```
import numpy as np  # 넘파이 임포트
import pandas as pd # 판다스 임포트

data_path = '/kaggle/input/bike-sharing-demand/' # 데이터 경로

train = pd.read_csv(data_path + 'train.csv') # 훈련 데이터
test = pd.read_csv(data_path + 'test.csv')   # 테스트 데이터
submission = pd.read_csv(data_path + 'sampleSubmission.csv') # 제출 샘플 데이터
```

- 훈련 데이터와 테스트 데이터 크기 확인

```
train.shape, test.shape
```

```
((10886, 12), (6493, 9))
```

6.3 탐색적 데이터 분석

▪ 6.3.2 데이터 둘러보기

• 훈련 데이터 둘러보기

```
train.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

datetime	기록 일시 (1시간 간격)
season	계절 (1 : 봄, 2 : 여름, 3 : 가을, 4 : 겨울)
holiday	공휴일 여부 (0 : 공휴일 아님, 1 : 공휴일)
workingday	근무일 여부 (0 : 근무일 아님, 1 : 근무일)
weather	날씨 (1 : 맑음, 2 : 옅은 안개, 약간 흐림, 3 : 약간의 눈, 약간의 비와 천둥 번개, 흐림, 4 : 폭우와 천둥 번개, 눈과 짙은 안개)

temp	실제 온도
atemp	체감 온도
humidity	상대 습도
windspeed	풍속
casual	등록되지 않은 사용자(비회원) 수
registered	등록된 사용자(회원) 수
count	**자전거 대여 수량**

타깃값

6.3 탐색적 데이터 분석

▪ 6.3.2 데이터 둘러보기

- 테스트 데이터 둘러보기

```
test.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed
0	2011-01-20 00:00:00	1	0	1	1	10.66	11.365	56	26.0027
1	2011-01-20 01:00:00	1	0	1	1	10.66	13.635	56	0.0000
2	2011-01-20 02:00:00	1	0	1	1	10.66	13.635	56	0.0000
3	2011-01-20 03:00:00	1	0	1	1	10.66	12.880	56	11.0014
4	2011-01-20 04:00:00	1	0	1	1	10.66	12.880	56	11.0014

- 테스트 데이터에는 casual과 registered 피처가 빠짐
  - 예측 시 사용하는 데이터가 테스트 데이터인데, 테스트 데이터에 casual과 registered 피처가 없으므로 모델 훈련 시에도 훈련 데이터의 casual과 registered 피처를 빼야 함

분석 결과

casual, registered 피처 제거

6.3 탐색적 데이터 분석

▪ 6.3.2 데이터 둘러보기

- 제출 샘플 데이터 둘러보기

```
submission.head()
```

	datetime	count
0	2011-01-20 00:00:00	0
1	2011-01-20 01:00:00	0
2	2011-01-20 02:00:00	0
3	2011-01-20 03:00:00	0
4	2011-01-20 04:00:00	0

- 데이터를 구분하는 ID 값(여기서는 datetime)과 타깃값으로 구성
  - 현재 타깃값인 count는 모두 0으로 기록됨
  - 시간대별 대여 수량을 예측해 이 값을 바꾼 뒤 제출하면 됨
  - ID 값인 datetime은 데이터를 구분하는 역할만 함,
    타깃값 예측에는 아무런 도움을 주지 않아 추후 모델 훈련 시 제거할 계획

분석 결과

datetime 피처 제거

6.3 탐색적 데이터 분석

▪ 6.3.2 데이터 둘러보기

- info( ) 함수로 결측값 개수와 데이터 타입 파악 (훈련 데이터)

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
#   Column      Non-Null Count  Dtype
---  ------      --------------  -----
0   datetime    10886 non-null  object
1   season      10886 non-null  int64
2   holiday     10886 non-null  int64
3   workingday  10886 non-null  int64
4   weather     10886 non-null  int64
5   temp        10886 non-null  float64
6   atemp       10886 non-null  float64
7   humidity    10886 non-null  int64
8   windspeed   10886 non-null  float64
9   casual      10886 non-null  int64
10  registered  10886 non-null  int64
11  count       10886 non-null  int64
dtypes: float64(3), int64(8), object(1)
memory usage: 1020.7+ KB
```

6.3 탐색적 데이터 분석

▪ 6.3.2 데이터 둘러보기

- info( ) 함수로 결측값 개수와 데이터 타입 파악 (테스트 데이터)

```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6493 entries, 0 to 6492
Data columns (total 9 columns):
#   Column      Non-Null Count  Dtype
---  ------      --------------  -----
0   datetime    6493 non-null   object
1   season      6493 non-null   int64
2   holiday     6493 non-null   int64
3   workingday  6493 non-null   int64
4   weather     6493 non-null   int64
5   temp        6493 non-null   float64
6   atemp       6493 non-null   float64
7   humidity    6493 non-null   int64
8   windspeed   6493 non-null   float64
dtypes: float64(3), int64(5), object(1)
memory usage: 456.7+ KB
```

6.3 탐색적 데이터 분석

▪ 6.3.3 더 효과적인 분석을 위한 피처 엔지니어링

- 기본적인 분석을 마쳤다면 다음은 데이터 시각화를 할 차례
  - 다양한 관점에서 시각화 해보면 날 데이터raw data 상태에서는 찾기 어려운 경향, 공통점, 차이점 등이 드러날 수 있음
- 일부 데이터는 시각화하기에 적합하지 않은 형태일 수 있음(ex. datetime)
  - 시각화하기 전에 datetime 피처를 분석하기에 적합한 형태로 변환해야 함
- datetime 피처를 세부 구성요소(연도, 월, 일, 시간, 분, 초)로 나누기
  - datetime은 object 타입이라서 문자열처럼 다룰 수 있음

```
print(train['datetime'][100]) # datetime 100번째 원소
print(train['datetime'][100].split()) # 공백 기준으로 문자열 나누기
print(train['datetime'][100].split()[0]) # 날짜
print(train['datetime'][100].split()[1]) # 시간
```

```
2011-01-05 09:00:00
['2011-01-05', '09:00:00']
2011-01-05
09:00:00
```

분석 결과

연도, 월, 일, 시간, 분, 초 피처 추가

6.3 탐색적 데이터 분석

▪ 6.3.3 더 효과적인 분석을 위한 피처 엔지니어링

- 날짜 문자열을 다시 연도, 월, 일로 나누기

```
print(train['datetime'][100].split()[0]) # 날짜
print(train['datetime'][100].split()[0].split("-")) # "-" 기준으로 문자열 나누기
print(train['datetime'][100].split()[0].split("-")[0]) # 연도
print(train['datetime'][100].split()[0].split("-")[1]) # 월
print(train['datetime'][100].split()[0].split("-")[2]) # 일
```

```
2011-01-05
['2011', '01', '05']
2011
01
05
```

  - “-” 문자를 기준으로 연도, 월, 일을 구함

6.3 탐색적 데이터 분석

▪ 6.3.3 더 효과적인 분석을 위한 피처 엔지니어링

- 시간 문자열을 시, 분, 초로 나누기

```
print(train['datetime'][100].split()[1]) # 시간
print(train['datetime'][100].split()[1].split(":")) # ":" 기준으로 문자열 나누기
print(train['datetime'][100].split()[1].split(":")[0])# 시간
print(train['datetime'][100].split()[1].split(":")[1]) # 분
print(train['datetime'][100].split()[1].split(":")[2]) # 초
```

```
09:00:00
['09', '00', '00']
09
00
00
```

- “:” 문자를 기준으로 시, 분, 초를 구함

6.3 탐색적 데이터 분석

▪ 6.3.3 더 효과적인 분석을 위한 피처 엔지니어링

- 판다스 apply( ) 함수로 앞서 살펴본 로직을 datetime에 적용해 날짜(date), 연도(year), 월(month), 일(day), 시(hour), 분(minute), 초(second) 피처 생성

```
train['date'] = train['datetime'].apply(lambda x: x.split()[0]) # 날짜 피처 생성

# 연도, 월, 일, 시, 분, 초 피처를 차례로 생성
train['year'] = train['datetime'].apply(lambda x: x.split()[0].split('-')[0])
train['month'] = train['datetime'].apply(lambda x: x.split()[0].split('-')[1])
train['day'] = train['datetime'].apply(lambda x: x.split()[0].split('-')[2])
train['hour'] = train['datetime'].apply(lambda x: x.split()[1].split(':')[0])
train['minute'] = train['datetime'].apply(lambda x:
x.split()[1].split(':')[1])
train['second'] = train['datetime'].apply(lambda x:
x.split()[1].split(':')[2])
```

❶ train의 'datetime' 피처의 원소 각각에

❶ 공백으로 나눈 후 0번째 요소를

```
train['year'] = train['datetime'].apply(lambda x: x.split()[0].split('-')[0])
```

❸ 'year' 피처로 추가합니다.

❷ 람다 함수를 적용하여

❷ 다시 '-'로 나눈 후 0번째 요소를 취합니다.

〈Note〉
이처럼 기존 피처에서 파생된 피처를 '파생 피처'라고 함

6.3 탐색적 데이터 분석

▪ 6.3.3 더 효과적인 분석을 위한 피처 엔지니어링

- 요일 피처 생성
  - datetime과 calendar 라이브러리 활용

```
from datetime import datetime # datetime 라이브러리 임포트
import calendar

print(train['date'][100]) # 날짜
print(datetime.strptime(train['date'][100], '%Y-%m-%d')) # datetime 타입으로 변경
# 정수로 요일 반환
print(datetime.strptime(train['date'][100], '%Y-%m-%d').weekday())
# 문자열로 요일 반환
print(calendar.day_name[datetime.strptime(train['date'][100], '%Y-%m-%d').weekday()])
```

```
2011-01-05
2011-01-05 00:00:00
2
Wednesday
```

  - 0은 월요일, 1은 화요일, 2는 수요일순으로 매핑
  - 단, 모델 훈련 시에는 피처 값을 문자로 바꾸면 안 됨(머신러닝 모델은 숫자만 인식)

〈Warning〉
머신러닝 모델은 숫자만 인식하므로 모델을 훈련할 때 피처 값을 문자로 바꾸면 안 됨

6.3 탐색적 데이터 분석

▪ 6.3.3 더 효과적인 분석을 위한 피처 엔지니어링

- 앞 로직을 apply( ) 함수로 적용해 요일(weekday) 피처 추가

```
train['weekday'] = train['date'].apply(
   lambda dateString:
   calendar.day_name[datetime.strptime(dateString,"%Y-%m-%d").weekday()])
```

- season과 weather 피처 추가
  - 두 피처는 범주형 데이터인데 숫자로 표현되어 있어서 의미 파악이 어려움
  - 시각화 시 의미가 잘 드러나도록 map( ) 함수를 사용해 문자열로 변환

```
train['season'] = train['season'].map({1: 'Spring',
                                       2: 'Summer',
                                       3: 'Fall',
                                       4: 'Winter' })
train['weather'] = train['weather'].map({1: 'Clear',
                                         2: 'Mist, Few clouds',
                                         3: 'Light Snow, Rain, Thunderstorm',
                                         4: 'Heavy Rain, Thunderstorm, Snow,
Fog'})
```

분석 결과

요일 피처 추가

6.3 탐색적 데이터 분석

▪ 6.3.3 더 효과적인 분석을 위한 피처 엔지니어링

- 바뀐 훈련 데이터

```
train.head()
```

	datetim	② eason	holiday	workingda	② eather	temp	① date	year	month	day	hour	minute	second	weekday
0	2011-01-01 00:00:00	Spring	0	0	Clear	9.84	2011-01-01	2011	01	01	00	00	00	Saturday
1	2011-01-01 01:00:00	Spring	0	0	Clear	9.02	2011-01-01	2011	01	01	01	00	00	Saturday
2	2011-01-01 02:00:00	Spring	0	0	Clear	9.02	2011-01-01	2011	01	01	02	00	00	Saturday
3	2011-01-01 03:00:00	Spring	0	0	Clear	9.84	2011-01-01	2011	01	01	03	00	00	Saturday
4	2011-01-01 04:00:00	Spring	0	0	Clear	9.84	2011-01-01	2011	01	01	04	00	00	Saturday

- ① date ~ weekday 피처 추가됨
- ② season과 weather 피처는 숫자에서 문자로 바뀜
- date 피처 정보는 year, month, day 피처에 포함되므로 추후 date 피처 제거
- 세 달씩 '월'을 묶으면 '계절'이 되므로 추후 season 피처만 남기고 month 피처는 제거

분석 결과

date 피처 제거

분석 결과

month 피처 제거

6.3 탐색적 데이터 분석

6.3.4 데이터 시각화

- 시각화는 탐색적 데이터 분석에서 가장 중요한 부분
  - 데이터 분포나 데이터 간 관계를 한눈에 파악
  - 모델링에 도움될 정보를 얻을 수 있음
- 시각화에 앞서 두 라이브러리를 임포트

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

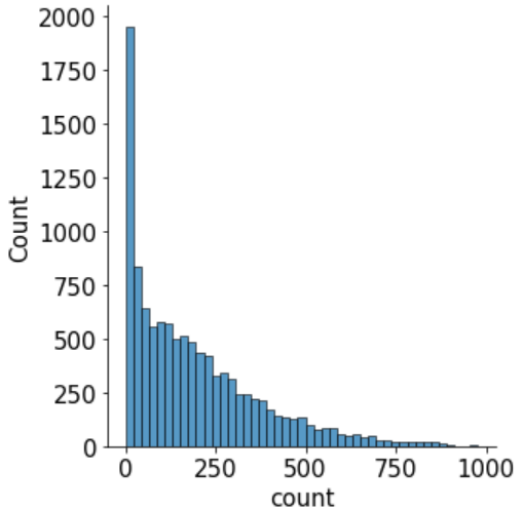
  - **matplotlib**은 파이썬으로 데이터를 시각화할 때 표준처럼 사용하는 라이브러리
  - **seaborn**은 matplotlib에 고수준 인터페이스를 덧씌운 라이브러리
- 그래프 자체에 관한 좀 더 자세한 설명은 〈머신러닝·딥러닝 문제해결 전략〉 4장 참고

6.3 탐색적 데이터 분석

▪ 6.3.4 데이터 시각화 | 분포도

- 분포도distribution plot는 수치형 데이터의 집계 값(총 개수나 비율 등)을 나타내는 그래프
- 타깃값(count)의 분포도 그려보기
  - 타깃값 분포도를 통해 타깃값을 그대로 사용할지 변환해 사용할지 결정할 수 있음

```
mpl.rc('font', size=15)       # 폰트 크기를 15로 설정
sns.displot(train['count']); # 분포도 출력
```



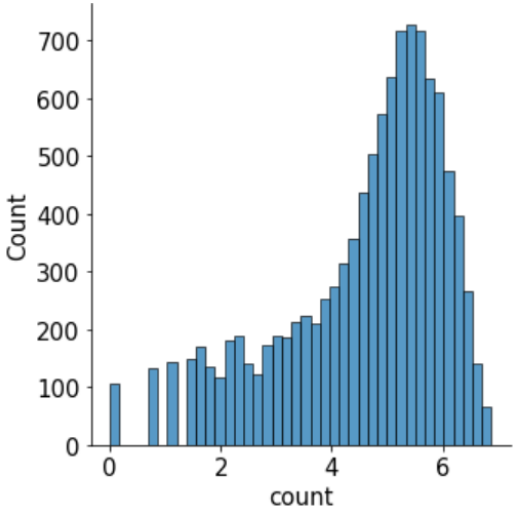
- x축은 타깃값 count, y축은 총 개수
- 분포가 왼쪽으로 편향됨
- 회귀 모델이 좋은 성능을 내려면 데이터가 정규분포를 따르는 게 좋음
  따라서, 타깃값을 정규분포에 가깝게 만들어주는 작업 필요
- 이때 사용하는 방법이 **로그변환**!

6.3 탐색적 데이터 분석

▪ 6.3.4 데이터 시각화 | 분포도

- 타깃값 로그변환

```
sns.displot(np.log(train['count']));
```



- 다만, 마지막에 지수변환해 실제 타깃값인 count로 복원해야 함

분석 결과

타깃값을 count가 아닌 log(count)로 변환해 사용

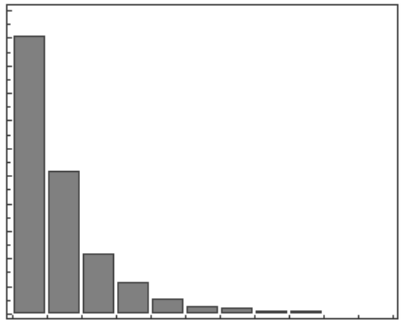
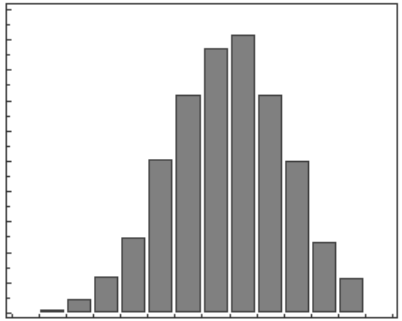
분석 결과

마지막에 지수변환해 count로 복원

6.3 탐색적 데이터 분석

▪ 6.3.4 데이터 시각화 | 분포도

- 기존 타깃값과 로그변환한 타깃값 비교

구분	기존 타깃값 활용 시	로그변환한 타깃값 활용 시
타깃값	y	log(y)
타깃값 분포	y값 분포(비대칭 분포) 	로그변환 후 y값 분포(정규분포) 
회귀 모델 예측 성능	나쁨	좋음
후처리	필요 없음	log(y)에서 실제 타깃값인 y를 복원하기 위해 지수변환을 해줘야 함

6.3 탐색적 데이터 분석

▪ 6.3.4 데이터 시각화 | 막대 그래프

- 연도, 월, 일, 시, 분, 초별 평균 대여 수량을 막대 그래프로 시각화

```
# 스텝 1 : m행 n열 Figure 준비
figure, axes = plt.subplots(nrows=2, ncols=2) # 2행 2열
plt.tight_layout()
figure.set_size_inches(10, 10)

# 스텝 2 : 서브플롯 할당
# 계절, 날씨, 공휴일, 근무일별 대여 수량 박스플롯
sns.boxplot(x='season', y='count', data=train, ax=axes[0, 0])
sns.boxplot(x='weather', y='count', data=train, ax=axes[0, 1])
sns.boxplot(x='holiday', y='count', data=train, ax=axes[1, 0])
sns.boxplot(x='workingday', y='count', data=train, ax=axes[1, 1])

# 스텝 3 : 세부 설정
# 3-1 서브플롯에 제목 달기
axes[0, 0].set(title='Box Plot On Count Across Season')
axes[0, 1].set(title='Box Plot On Count Across Weather')
axes[1, 0].set(title='Box Plot On Count Across Holiday')
axes[1, 1].set(title='Box Plot On Count Across Working Day')

# 3-2 x축 라벨 겹침 해결
axes[0, 1].tick_params(axis='x', labelrotation=10) # 10도 회전
```

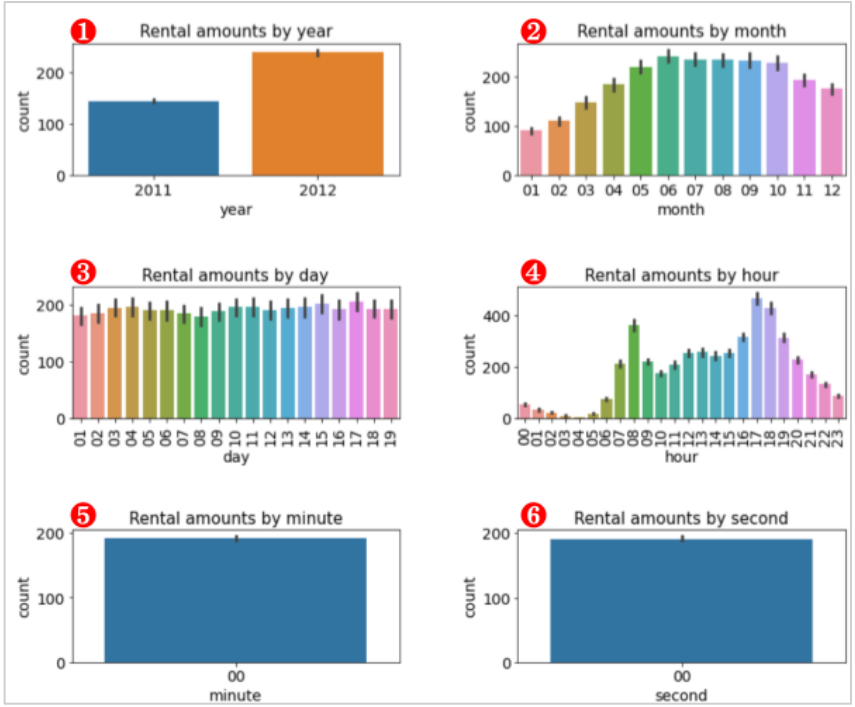
6.3 탐색적 데이터 분석

▪ 6.3.4 데이터 시각화 | 막대 그래프

- 연도, 월, 일, 시, 분, 초별 평균 대여 수량을 막대 그래프로 시각화
  - ① 연도별 평균 대여 수량
    - 2011년보다 2012년에 대여가 많음
  - ② 월별 평균 대여 수량
    - 날이 따뜻할수록 대여가 많다고 짐작해볼 수 있음
  - ③ 일별 평균 대여 수량
    - 훈련 데이터(1일~19일)와 테스트 데이터(20일~월말)의 day 피처가 서로 달라 day 피처는 모델링 시 사용 못함
  - ④ 시간별 평균 대여 수량
    - 쌍봉형(출퇴근, 등하교 시간에 이용량이 많음)
  - ⑤ 분별 평균 대여 수량 ⑥ 분별 평균 대여 수량
    - 아무런 정보가 없어 제거

분석 결과

day, minute, second 피처 제거



6.3 탐색적 데이터 분석

▪ 6.3.4 데이터 시각화 | 박스플롯

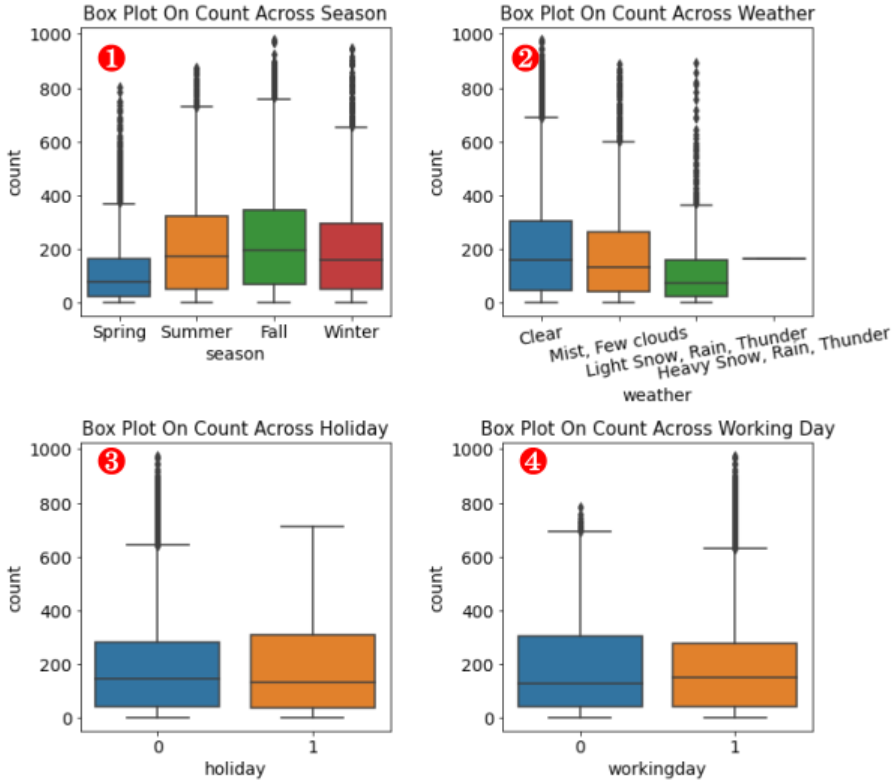
- 박스플롯box plot은 범주형 데이터에 따른 수치형 데이터 정보를 나타내는 그래프
- 계절, 날씨, 공휴일, 근무일(범주형 데이터)별 대여 수량(수치형 데이터)을 박스플롯으로 시각화

```
# 스텝 1 : m행 n열 Figure 준비
figure, axes = plt.subplots(nrows=2, ncols=2) # 2행 2열
plt.tight_layout()
figure.set_size_inches(10, 10)

# 스텝 2 : 서브플롯 할당
# 계절, 날씨, 공휴일, 근무일별 대여 수량 박스플롯
sns.boxplot(x='season', y='count', data=train, ax=axes[0, 0])
sns.boxplot(x='weather', y='count', data=train, ax=axes[0, 1])
sns.boxplot(x='holiday', y='count', data=train, ax=axes[1, 0])
sns.boxplot(x='workingday', y='count', data=train, ax=axes[1, 1])

# 스텝 3 : 세부 설정
# 3-1 서브플롯에 제목 달기
axes[0, 0].set(title='Box Plot On Count Across Season')
axes[0, 1].set(title='Box Plot On Count Across Weather')
axes[1, 0].set(title='Box Plot On Count Across Holiday')
axes[1, 1].set(title='Box Plot On Count Across Working Day')

# 3-2 x축 라벨 겹침 해결
axes[0, 1].tick_params(axis='x', labelrotation=10) # 10도 회전
```

6.3 탐색적 데이터 분석

▪ 6.3.4 데이터 시각화 | 박스플롯

- 박스플롯 결과 해석
  - ▪ ① 계절별 대여 수량 박스플롯
  - ▪ ② 날씨별 대여 수량 박스플롯
    - 날씨가 좋을수록 대여 수량이 많음
  - ▪ ③ 공휴일 여부에 따른 대여 수량 박스플롯
    - x축 라벨 0 = 공휴일 아님, 1 = 공휴일
    - 공휴일이 아닐 때 이상치outlier가 많음
  - ▪ ④ 근무일 여부에 따른 대여 수량 박스플롯
    - 근무일일 때 이상치가 많음
    - 참고로, 근무일은 공휴일과 주말을 뺀 나머지 날을 뜻함



6.3 탐색적 데이터 분석

▪ 6.3.4 데이터 시각화 | 포인트플롯

- 포인트플롯point plot은 범주형 데이터에 따른 수치형 데이터의 평균과 신뢰구간을 점과 선으로 표시
- 막대 그래프와 같은 정보를 제공하지만, 한 화면에 여러 그래프를 겹쳐 그리기에 적합

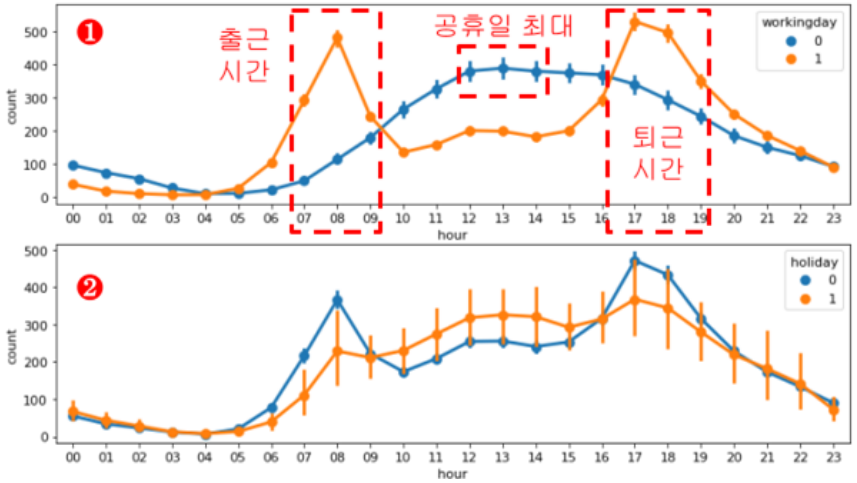
```
# 스텝 1 : m행 n열 Figure 준비
mpl.rc('font', size=11)
figure, axes = plt.subplots(nrows=5) # 5행 1열
figure.set_size_inches(12, 18)

# 스텝 2 : 서브플롯 할당
# 근무일, 공휴일, 요일, 계절, 날씨에 따른 시간대별 평균 대여 수량 포인트플롯
sns.pointplot(x='hour', y='count', data=train, hue='workingday', ax=axes[0])
sns.pointplot(x='hour', y='count', data=train, hue='holiday', ax=axes[1])
sns.pointplot(x='hour', y='count', data=train, hue='weekday', ax=axes[2])
sns.pointplot(x='hour', y='count', data=train, hue='season', ax=axes[3])
sns.pointplot(x='hour', y='count', data=train, hue='weather', ax=axes[4]);
```

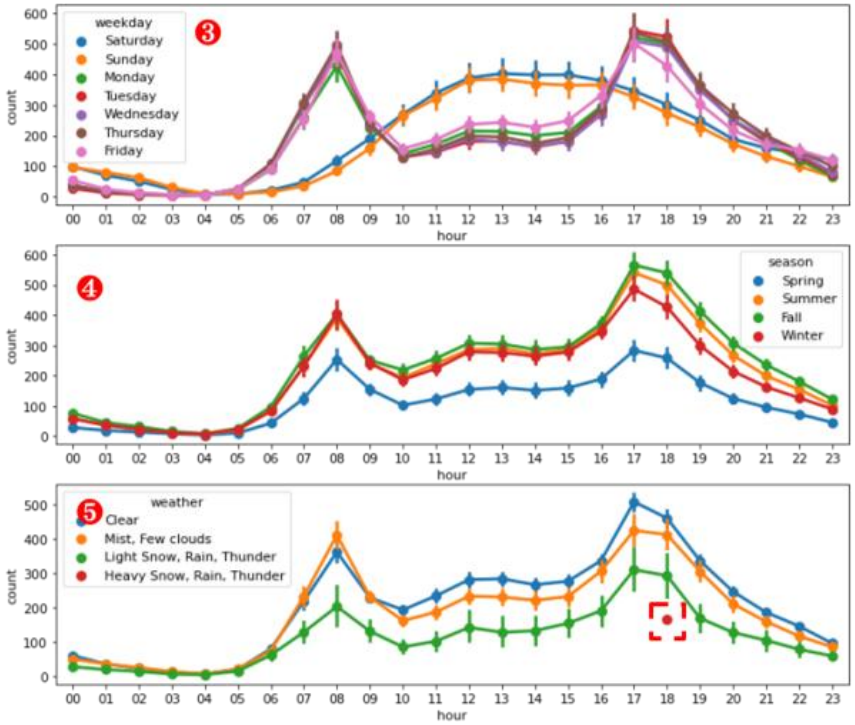
- hue 파라미터에 전달한 피처를 기준으로 그래프가 나뉨

6.3 탐색적 데이터 분석

▪ 6.3.4 데이터 시각화 | 포인트플롯

- 포인트플롯 결과





분석 결과

weather == 4인 데이터 제거

6.3 탐색적 데이터 분석

▪ 6.3.4 데이터 시각화 | 회귀선을 포함한 산점도 그래프

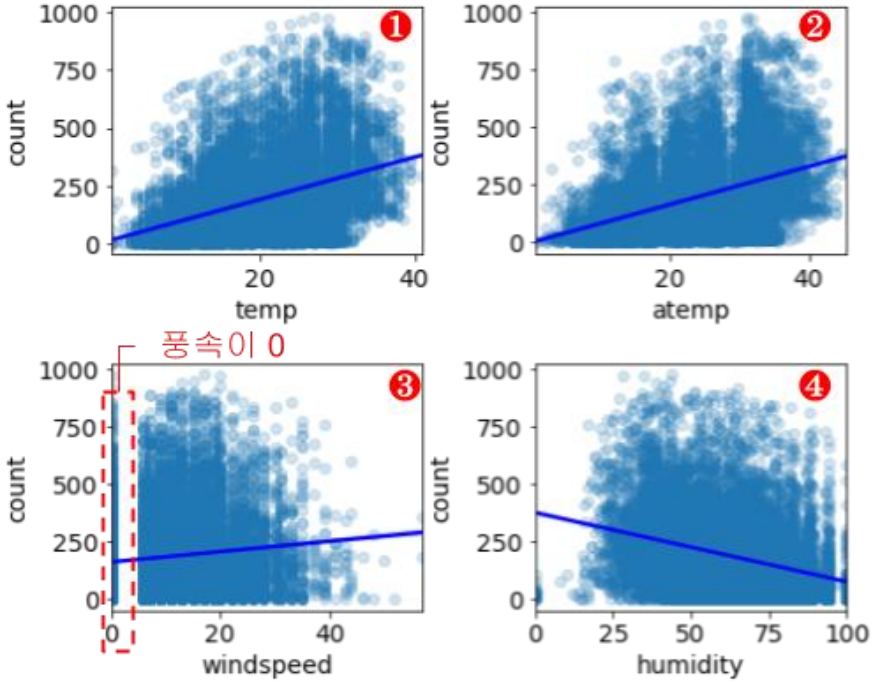
- 회귀선을 포함한 산점도 그래프는 수치형 데이터 간 상관관계를 파악하는 데 활용

```
# 스텝 1 : m행 n열 Figure 준비
mpl.rc('font', size=15)
figure, axes = plt.subplots(nrows=2, ncols=2) # 2행 2열
plt.tight_layout()
figure.set_size_inches(7, 6)

# 스텝 2 : 서브플롯 할당
# 온도, 체감 온도, 풍속, 습도 별 대여 수량 산점도 그래프
sns.regplot(x='temp', y='count', data=train, ax=axes[0, 0],
            scatter_kws={'alpha': 0.2}, line_kws={'color': 'blue'})
sns.regplot(x='atemp', y='count', data=train, ax=axes[0, 1],
            scatter_kws={'alpha': 0.2}, line_kws={'color': 'blue'})
sns.regplot(x='windspeed', y='count', data=train, ax=axes[1, 0],
            scatter_kws={'alpha': 0.2}, line_kws={'color': 'blue'})
sns.regplot(x='humidity', y='count', data=train, ax=axes[1, 1],
            scatter_kws={'alpha': 0.2}, line_kws={'color': 'blue'});
```

6.3 탐색적 데이터 분석

▪ 6.3.4 데이터 시각화 | 회귀선을 포함한 산점도 그래프

- 회귀선을 포함한 산점도 그래프 결과
  - ▪ ①과 ② 온도, 체감 온도와 대여 수량 관계
    - 온도, 체감 온도가 높을수록 대여 수량이 많음
  - ▪ ③ 풍속과 대여 수량 관계
    - 풍속이 셀수록 대여 수량이 많음(?)
    - 결측치가 많기 때문
    - 결측치가 많은 windspeed 피처는 제거
  - ▪ ④ 습도와 대여 수량 관계
    - 습하지 않을수록 대여 수량이 많음



분석 결과

windspeed 피처 제거

6.3 탐색적 데이터 분석

▪ 6.3.4 데이터 시각화 | 히트맵

- 히트맵heatmap은 데이터 간 상관관계를 색상으로 시각화하는 그래프
- 수치형 데이터인 temp, atemp, humidity, windspeed, count 간 상관계수 계산

```
train[['temp', 'atemp', 'humidity', 'windspeed', 'count']].corr()
```

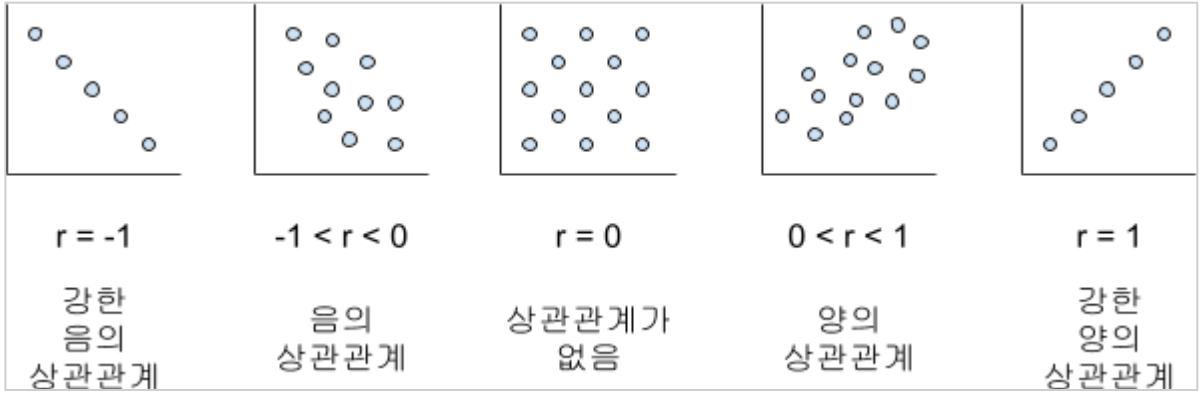
	temp	atemp	humidity	windspeed	count
temp	1.000000	0.984948	-0.064949	-0.017852	0.394454
atemp	0.984948	1.000000	-0.043536	-0.057473	0.389784
humidity	-0.064949	-0.043536	1.000000	-0.318607	-0.317371
windspeed	-0.017852	-0.057473	-0.318607	1.000000	0.101369
count	0.394454	0.389784	-0.317371	0.101369	1.000000

- 조합이 많아 피처간 관계가 한눈에 들어오지 않음 → 히트맵이 필요한 순간!

6.3 탐색적 데이터 분석

▪ 6.3.4 데이터 시각화 | 히트맵

- 피어슨 상관계수pearson correlation coefficient : 여러 상관계수 중 가장 기본적으로 사용되는 상관계수
  - 선형 상관관계의 강도strength와 방향direction을 나타냄
  - -1부터 1 사이의 값을 갖음
  - 상관계수가 음수면 음의 상관관계가 있다고 하고, 양수면 양의 상관관계가 있다고 함

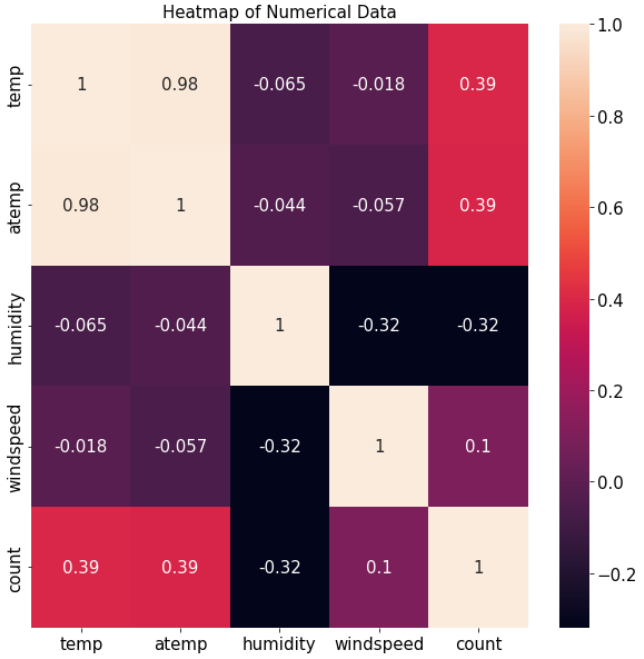


6.3 탐색적 데이터 분석

▪ 6.3.4 데이터 시각화 | 히트맵

- 히트맵heatmap 시각화

```
# 피처 간 상관관계 매트릭스
corrMat = train[['temp', 'atemp', 'humidity', 'windspeed', 'count']].corr()
fig, ax= plt.subplots()
fig.set_size_inches(10, 10)
sns.heatmap(corrMat, annot=True) # 상관관계 히트맵 그리기
ax.set(title='Heatmap of Numerical Data');
```



- ▪ 타깃값인 count와의 상관관계가 중요!
- ▪ 온도(temp)와 대여 수량(count) 간 상관계수는 0.39 (양의 상관관계)
- ▪ 풍속(windspeed)과 대여 수량의 상관계수는 0.1 (상관관계가 매우 약함)
  - windspeed 피처는 대여 수량 예측에 별 도움을 주지 못하므로 피처 제거
  - '회귀선을 포함한 산점도 그래프' 결과에서도 결측치가 많다는 이유로 같은 결론에 도달한 바 있음

분석 결과

windspeed 피처 제거

6.3 탐색적 데이터 분석

▪ 분석 정리 및 모델링 전략

- 분석 정리

1. **타깃값 변환** : 타깃값을 count가 아닌 log(count)로 변환해 사용(마지막에 다시 지수변환해 count로 복원해야 함)
2. **파생 피처 추가** : datetime 피처를 분리해 year, month, day, hour, minute, second 피처 생성 가능
3. **파생 피처 추가** : datetime에 숨어 있는 또 다른 정보인 요일(weekday) 피처 추가
4. **피처 제거** : 테스트 데이터에 없는 피처는 훈련에 사용해도 의미 없음. 따라서 훈련 데이터에만 있는 casual, registered 피처 제거
5. **피처 제거** : datetime 피처는 인덱스 역할만 하므로 타깃값 예측에 아무런 도움이 되지 않음
6. **피처 제거** : date 피처가 제공하는 정보는 year, month, day 피처에 담겨 있음
7. **피처 제거** : month는 season 피처의 세부 분류로 볼 수 있어 month 피처 제거
8. **피처 제거** : 막대 그래프 확인 결과, 파생 피처인 day는 분별력이 없음
9. **피처 제거** : 막대 그래프 확인 결과, 파생 피처인 minute와 second에는 아무런 정보가 담겨 있지 않음
10. **이상치 제거** : 포인트 플롯 확인 결과, weather가 4인 데이터는 이상치
11. **피처 제거** : 산점도와 히트맵 확인 결과, windspeed 피처에는 결측값이 많고 대여 수량과의 상관관계가 매우 약함