

Bike Sharing Demand

데이터분석과 시각화, 머신러닝 알고리즘으로 시간당 자전거 대여량을 예측하기

(이 주피터 노트북은 다음의 링크 <https://bit.ly/ds-bike-0202> (<https://bit.ly/ds-bike-0202>) 에서 다운받을 수 있습니다.)

이번 캐글 경진대회는 시간당 자전거 대여량을 예측하는 [Bike Sharing Demand](https://www.kaggle.com/c/bike-sharing-demand) (<https://www.kaggle.com/c/bike-sharing-demand>) 입니다. 워싱턴 D.C 소재의 자전거 대여 스타트업 [Capital Bikeshare](https://www.capitalbikeshare.com/) (<https://www.capitalbikeshare.com/>)의 데이터를 활용하여, 특정 시간대에 얼마나 많은 사람들이 자전거를 대여하는지 예측하는 것이 목표입니다.

사람들이 자전거를 대여하는데는 많은 요소가 관여되어 있을 겁니다. 가령 시간(새벽보다 낮에 많이 빌리겠죠), 날씨(비가 오면 자전거를 대여하지 않을 겁니다), 근무일(근무 시간에는 자전거를 대여하지 않겠죠) 등. 이런 모든 요소를 조합하여 워싱턴 D.C의 자전거 교통량을 예측해주세요. 이번 경진대회에서는 기존까지 배웠던 프로그래밍 언어와 인공지능&머신러닝 능력 외에도, 자전거 렌탈 시장에 대한 약간의 전문지식, 그리고 일반인의 기초 상식을 총동원 할 수 있습니다.

저번 [Titanic: Machine Learning from Disaster](https://www.kaggle.com/c/titanic/) (<https://www.kaggle.com/c/titanic/>) 경진대회와 마찬가지로, 이번에도 프로그래밍 언어 파이썬([Python](https://www.python.org/) (<https://www.python.org/>)), 데이터 분석 패키지 판다스([Pandas](https://pandas.pydata.org/) (<https://pandas.pydata.org/>)), 그리고 머신러닝&인공지능 라이브러리인 싸이킷런([scikit-learn](https://scikit-learn.org/) (scikit-learn.org))을 사용합니다. 여기에 더불어, 이번에는 데이터 시각화 패키지 [matplotlib](https://matplotlib.org/) (<https://matplotlib.org/>)와 [Seaborn](https://seaborn.pydata.org/) (<https://seaborn.pydata.org/>)을 본격적으로 활용해볼 것입니다.

컬럼 설명

(데이터는 [다음의 링크](https://www.kaggle.com/c/bike-sharing-demand/data) (<https://www.kaggle.com/c/bike-sharing-demand/data>)에서 다운받으실 수 있습니다)

- **datetime** - 시간. 연-월-일 시:분:초 로 표현합니다. (가령 2011-01-01 00:00:00은 2011년 1월 1일 0시 0분 0초)
- **season** - 계절. 봄(1), 여름(2), 가을(3), 겨울(4) 순으로 표현합니다.
- **holiday** - 공휴일. 1이면 공휴일이며, 0이면 공휴일이 아닙니다.
- **workingday** - 근무일. 1이면 근무일이며, 0이면 근무일이 아닙니다.
- **weather** - 날씨. 1 ~ 4 사이의 값을 가지며, 구체적으로는 다음과 같습니다.
 - 1: 아주 깨끗한 날씨입니다. 또는 아주 약간의 구름이 끼어있습니다.
 - 2: 약간의 안개와 구름이 끼어있는 날씨입니다.
 - 3: 약간의 눈, 비가 오거나 천둥이 칩니다.
 - 4: 아주 많은 비가 오거나 우박이 내립니다.
- **temp** - 온도. 섭씨(Celsius)로 적혀있습니다.
- **atemp** - 체감 온도. 마찬가지로 섭씨(Celsius)로 적혀있습니다.
- **humidity** - 습도.
- **windspeed** - 풍속.
- **casual** - 비회원(non-registered)의 자전거 대여량.
- **registered** - 회원(registered)의 자전거 대여량.
- **count** - 총 자전거 대여량. 비회원(casual) + 회원(registered)과 동일합니다.

In [1]:

```
# 파이썬의 데이터 분석 패키지 Pandas(pandas.pydata.org) 를 읽어옵니다.  
# Pandas는 쉽게 말해 파이썬으로 엑셀을 다룰 수 있는 툴이라고 보시면 됩니다.  
# 이 패키지를 앞으로는 pd라는 축약어로 사용하겠습니다.  
import pandas as pd
```

Load Dataset

언제나처럼 모든 데이터 분석의 시작은 주어진 데이터를 읽어오는 것입니다. [판다스\(Pandas\)](https://pandas.pydata.org/) (<https://pandas.pydata.org/>)의 `read_csv` (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html)를 활용하여 [Bike Sharing Demand](https://www.kaggle.com/c/bike-sharing-demand) (<https://www.kaggle.com/c/bike-sharing-demand>) 경진대회에서 제공하는 두 개의 데이터(train, test)를 읽어오겠습니다. ([다운로드 링크 \(https://www.kaggle.com/c/bike-sharing-demand/data\)](https://www.kaggle.com/c/bike-sharing-demand/data))

앞서 [Titanic: Machine Learning from Disaster \(https://www.kaggle.com/c/titanic/\)](https://www.kaggle.com/c/titanic/) 경진대회와 마찬가지로, 여기에서도 파일의 경로를 지정하는 방법에 주의하셔야 합니다. 만일 `read_csv`를 실행할 때 (`FileNotFoundError`)라는 이름의 에러가 난다면 경로가 제대로 지정이 되지 않은 것입니다. **파일의 경로를 지정하는 법이 생각나지 않는다면 [다음의 링크 \(http://88240.tistory.com/122\)](http://88240.tistory.com/122)를 통해 경로를 지정하는 법을 복습한 뒤 다시 시도해주세요.**

In [2]:

```
# 판다스의 read_csv로 train.csv 파일을 읽어옵니다.  
# 여기서 datetime은 특별히 날짜로 해석하기 위해 parse_dates 옵션에 넣어줍니다.  
# 읽어온 데이터를 train이라는 이름의 변수에 할당합니다.  
train = pd.read_csv("data/train.csv")  
  
# train 변수에 할당된 데이터의 행렬 사이즈를 출력합니다.  
# 출력은 (row, column) 으로 표시됩니다.  
print(train.shape)  
  
# head()로 train 데이터의 상위 5개를 띄웁니다.  
train.head()
```

(10886, 12)

Out[2]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	cas
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	



In [3]:

```
# train.csv 파일을 읽어온 방식과 동일하게 test.csv를 읽어옵니다.
# 이후 이 데이터를 test라는 이름의 변수에 저장합니다.
test = pd.read_csv("data/test.csv")

# 마찬가지로 행렬(row, column) 사이즈를 출력하고
print(test.shape)

# 전체 test 데이터에서 상위 5개만 출력합니다.
test.head()
```

(6493, 9)

Out[3]:

	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

Preprocessing

데이터를 읽어왔으면, 이 데이터를 편하게 분석하고 머신러닝 알고리즘에 집어넣기 위해 간단한 전처리 (Preprocessing) 작업을 진행하겠습니다.

[Bike Sharing Demand \(https://www.kaggle.com/c/bike-sharing-demand\)](https://www.kaggle.com/c/bike-sharing-demand)는 편리하게도 대부분의 데이터가 전처리 되어 있습니다. (가령 season 컬럼은 봄을 spring이라 표현하지 않고 1이라고 표현합니다) 그러므로 [Titanic: Machine Learning from Disaster \(https://www.kaggle.com/c/titanic/\)](https://www.kaggle.com/c/titanic/) 경진대회와는 달리 간단한 전처리만 끝내면 바로 머신러닝 모델에 데이터를 집어넣을 수 있습니다.

Parse datetime

먼저 날짜(datetime) 컬럼을 전처리 하겠습니다.

날짜 컬럼은 얼핏 보면 여러개의 숫자로 구성되어 있습니다. (ex: 2011-01-01 00:00:00) 하지만 결론적으로 숫자는 아니며, 판다스에서는 문자열(object) 또는 날짜(datetime64)로 인식합니다. (값에 하이픈(-)과 콜론(:)이 있기 때문입니다) 그러므로 날짜(datetime) 컬럼을 사용하기 위해서는 머신러닝 알고리즘이 이해할 수 있는 방식으로 전처리를 해줘야 합니다.

날짜(datetime) 컬럼을 전처리하는 가장 쉬운 방법은 연, 월, 일, 시, 분, 초를 따로 나누는 것입니다. 가령 2011-01-01 00:00:00은 2011년 1월 1일 0시 0분 0초라고 볼 수 있으므로, 2011, 1, 1, 0, 0, 0으로 따로 나누면 총 6개의 숫자가 됩니다. 즉, 날짜(datetime) 컬럼을 여섯개의 다른 컬럼으로 나누어주는 것이 날짜 컬럼을 전처리하는 핵심입니다.

In [4]:

```
# 판다스(Pandas)의 to_datetime을 활용해 datetime 컬럼의 타입을 문자열(String)에서 날짜/시간(datetime) 데이터로 다룹니다.
train["datetime"] = pd.to_datetime(train["datetime"])

# train 데이터에 연, 월, 일, 시, 분, 초를 나타내는 새로운 컬럼을 생성합니다.
# 각각의 이름을 datetime-year/month/day/hour/minute/second라고 가정합니다.
# 이 컬럼에 날짜(datetime) 컬럼의 dt(datetime의 약자입니다) 옵션을 활용하여 연월일시분초를 따로 넣어줍니다.
train["datetime-year"] = train["datetime"].dt.year
train["datetime-month"] = train["datetime"].dt.month
train["datetime-day"] = train["datetime"].dt.day
train["datetime-hour"] = train["datetime"].dt.hour
train["datetime-minute"] = train["datetime"].dt.minute
train["datetime-second"] = train["datetime"].dt.second

# train 변수에 할당된 데이터의 행렬 사이즈를 출력합니다.
# 출력은 (row, column) 으로 표시됩니다.
print(train.shape)

# .head()로 train 데이터의 상위 5개를 띄우되,
# datetime과 이와 연관된 나머지 일곱 개의 컬럼만을 출력합니다.
train[["datetime", "datetime-year", "datetime-month", "datetime-day", "datetime-hour", "datetime-minute", "datetime-second"]].head()
```

(10886, 18)

Out[4]:

	datetime	datetime-year	datetime-month	datetime-day	datetime-hour	datetime-minute	datetime-second
0	2011-01-01 00:00:00	2011	1	1	0	0	0
1	2011-01-01 01:00:00	2011	1	1	1	0	0
2	2011-01-01 02:00:00	2011	1	1	2	0	0
3	2011-01-01 03:00:00	2011	1	1	3	0	0
4	2011-01-01 04:00:00	2011	1	1	4	0	0

In [5]:

```
# train 데이터와 마찬가지로 test 데이터의 datetime 컬럼도 마찬가지로 문자열(String)에서 날짜/시간(datetime) 데이터로 다룹니다.
test["datetime"] = pd.to_datetime(test["datetime"])

# test 데이터와 train 데이터와 동일하게 연, 월, 일, 시, 분, 초 컬럼을 생성합니다.
test["datetime-year"] = test["datetime"].dt.year
test["datetime-month"] = test["datetime"].dt.month
test["datetime-day"] = test["datetime"].dt.day
test["datetime-hour"] = test["datetime"].dt.hour
test["datetime-minute"] = test["datetime"].dt.minute
test["datetime-second"] = test["datetime"].dt.second

# test 변수에 할당된 데이터의 행렬 사이즈를 출력합니다.
# 출력은 (row, column) 으로 표시됩니다.
print(test.shape)

# .head()로 test 데이터의 상위 5개를 띄우되,
# datetime과 이와 연관된 나머지 일곱 개의 컬럼만을 출력합니다.
test[["datetime", "datetime-year", "datetime-month", "datetime-day", "datetime-hour",
      "datetime-minute", "datetime-second"]].head()
```

(6493, 15)

Out[5]:

	datetime	datetime-year	datetime-month	datetime-day	datetime-hour	datetime-minute	datetime-second
0	2011-01-20 00:00:00	2011	1	20	0	0	0
1	2011-01-20 01:00:00	2011	1	20	1	0	0
2	2011-01-20 02:00:00	2011	1	20	2	0	0
3	2011-01-20 03:00:00	2011	1	20	3	0	0
4	2011-01-20 04:00:00	2011	1	20	4	0	0

Explore

전처리(Preprocessing)를 끝냈으면 그 다음에는 데이터를 분석해보겠습니다.

주어진 데이터를 시각화나 분석 툴을 통해 다양한 관점에서 이해하는 과정을 탐험적 데이터 분석([Exploratory Data Analysis](https://en.wikipedia.org/wiki/Exploratory_data_analysis) (https://en.wikipedia.org/wiki/Exploratory_data_analysis))이라고 합니다. 저번 타이타닉 문제와 마찬가지로, 이번에도 파이썬의 데이터 시각화 패키지인 ([matplotlib](https://matplotlib.org/) (<https://matplotlib.org/>))와 [seaborn](https://seaborn.pydata.org/) (<https://seaborn.pydata.org/>) 을 활용해서 분석해보겠습니다.

In [6]:

```
# matplotlib로 실행하는 모든 시각화를 자동으로 주피터 노트북에 띄웁니다.  
# seaborn 도 결국에는 matplotlib를 기반으로 동작하기 때문에, seaborn으로 실행하는 모든 시각  
# 화도 마찬가지로 주피터 노트북에 자동적으로 띄워집니다.  
%matplotlib inline  
  
# 데이터 시각화 패키지 seaborn을 로딩합니다. 앞으로는 줄여서 sns라고 사용할 것입니다.  
import seaborn as sns  
  
# 데이터 시각화 패키지 matplotlib를 로딩합니다. 앞으로는 줄여서 plt라고 사용할 것입니다.  
import matplotlib.pyplot as plt
```

datetime

먼저 분석할 컬럼은 날짜(datetime) 컬럼입니다. 날짜 컬럼은 [Bike Sharing Demand](https://www.kaggle.com/c/bike-sharing-demand) (<https://www.kaggle.com/c/bike-sharing-demand>) 경진대회 of 핵심 컬럼이라고 볼 수 있으며, 이번 경진대회에서 상위 성적을 올리고 싶다면 날짜 컬럼을 완벽하게 이해하는 것이 무엇보다도 중요합니다.

먼저 연/월/일/시/분/초에 따른 자전거 대여량을 시각화 해보겠습니다.

Q1-1) 시각화를 하기 전에 어떤 그림이 나올 것으로 예상하시나요? 최소 3가지 아이디어를 생각해 보세요.

주의: 이 내용은 반드시 시각화를 하기 전에 작성하셔야 합니다. 그래야 시각화 결과와 본인의 아이디어를 비교해서 차이를 발견할 수 있습니다.

1. 일단 분(Dates-minute), 초(Dates-second)는 자전거 대여량을 판가름하는데 별 영향이 없을 것 같습니다. 가령 현재 시간이 37분이면 자전거를 대여하고, 43분이면 자전거를 대여하지 않는 행동을 하지는 않을 것입니다. 그러므로 countplot으로 시각화를 해보면, 마치 [Uniform Distribution](https://m.blog.naver.com/running_p/90179231685) (https://m.blog.naver.com/running_p/90179231685)과 같은 모양이 나올 것 같습니다.
2. 그리고 일(Dates-day)도 비슷합니다. 하지만 일(Dates-day)은 분과 초와는 다르게, 1) 2월에는 28일 이후가 존재하지 않기 때문에, 29, 30, 31일은 다른 날보다 데이터가 적을 수도 있습니다. (예외적으로 2012년은 2월 29일이 있습니다), 비슷하게 2) 31일의 경우에는 다른 날에 비해 데이터가 절반밖에 되지 않을 것입니다. 하지만 우리는 데이터의 갯수보다는 날짜별 자전거의 평균 대여량이 중요하기 때문에, 실제 분석에는 큰 영향을 미치지 않을 것으로 예상합니다.
3. 이런 사항 외에도, 사람의 행동 패턴 상으로 날짜나 시간이라는 개념이 자전거를 대여하는데 중요한 영향을 미칠 것 같습니다. 가령 1) 시간(hour)을 기준으로 새벽보다는 오후에 사람들이 자전거를 많이 빌릴 것이며, 2) 월(month)을 기준으로 추운 여름보다는 따뜻한 봄이나 가을, 내지는 더운 여름이 더 많이 빌릴 것 같습니다.

자, 그럼 위 예상과 실제 데이터가 일치하는지 데이터 시각화를 통해 살펴보도록 하겠습니다.

In [7]:

```
# matplotlib의 subplots를 사용합니다. 이 함수는 여러 개의 시각화를 한 화면에 띄울 수 있도록
# 합니다.
# 이번에는 2x3으로 총 6개의 시각화를 한 화면에 띄웁니다.
figure, ((ax1, ax2, ax3), (ax4, ax5, ax6)) = plt.subplots(nrows=2, ncols=3)

# 시각화의 전체 사이즈는 18x8로 설정합니다.
figure.set_size_inches(18, 8)

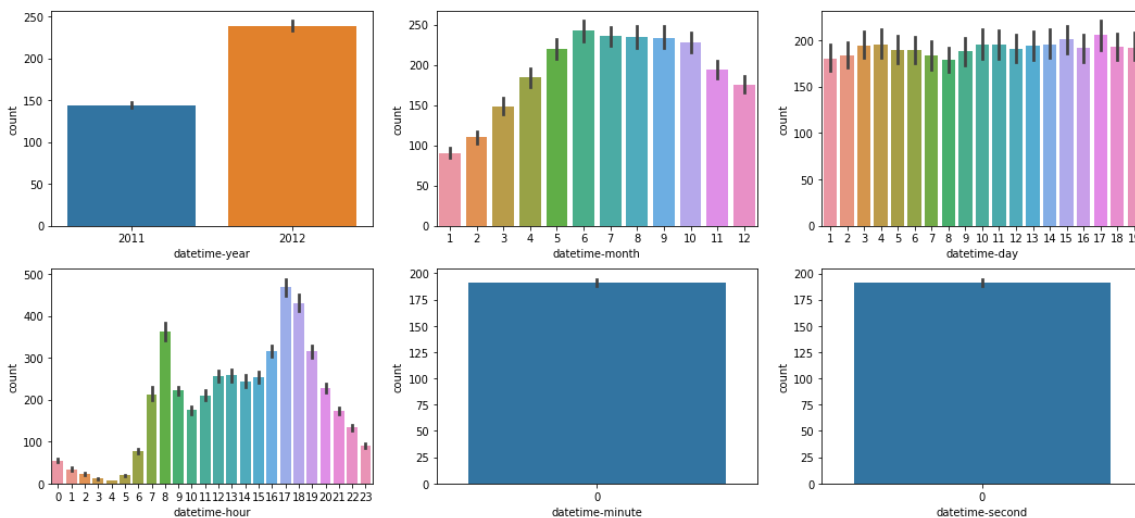
# seaborn의 barplot으로 subplots의 각 구역에
# 연, 월, 일, 시, 분, 초 별 자전거 대여량을 출력합니다.
sns.barplot(data=train, x="datetime-year", y="count", ax=ax1)
sns.barplot(data=train, x="datetime-month", y="count", ax=ax2)
sns.barplot(data=train, x="datetime-day", y="count", ax=ax3)
sns.barplot(data=train, x="datetime-hour", y="count", ax=ax4)
sns.barplot(data=train, x="datetime-minute", y="count", ax=ax5)
sns.barplot(data=train, x="datetime-second", y="count", ax=ax6)
```

/Users/shayne.kang/.pyenv/versions/3.6.7/envs/dsschool/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.

```
return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[7]:

<matplotlib.axes._subplots.AxesSubplot at 0x11d7ecbe0>



Q1-2) 이 시각화로 발견할 수 있는 사실은 어떤 게 있을까요? 그리고 앞서 우리의 예상과 어떤 차이가 있나요?

datetime-year

- 2011년도의 자전거 대여량보다 2012년도의 자전거 대여량이 더 높습니다. 이는 [Bike Sharing Demand](https://www.kaggle.com/c/bike-sharing-demand) (<https://www.kaggle.com/c/bike-sharing-demand>) 경진대회를 주최한 [Capital Bikeshare](https://www.capitalbikeshare.com/) (<https://www.capitalbikeshare.com/>)사가 꾸준히 성장하고 있다고 간주할 수 있습니다.

datetime-month

- 주로 여름(6~8월)에 자전거를 많이 빌리며, 겨울(12~2월)에는 자전거를 많이 빌리지 않습니다.
- 같은 겨울이라도 12월의 자전거 대여량이 1월의 자전거 대여량보다 두 배 가까이 높아 보입니다. 하지만 여기에는 숨겨진 비밀이 있는데, 다음에 나올 다른 시각화에서 자세히 살펴보겠습니다.

datetime-day

- x축을 자세히 보면 1일부터 19일까지밖에 없습니다. 20일은 어디에 있을까요? 바로 test 데이터에 있습니다. 이 시각화에서 알 수 있는 내용은, train 데이터와 test 데이터를 나누는 기준이 되는 컬럼이 바로 `datetime-day` 라는 것입니다. 그러므로 21일 이후의 자전거 대여량에 대해서는 우리도 알 수 없고, 머신러닝 알고리즘도 알지 못할 것입니다.

datetime-hour

- 새벽 시간에는 사람들이 자전거를 빌리지 않으며, 오후 시간에 상대적으로 자전거를 많이 빌립니다.
- 특이하게도 두 부분에서 사람들이 자전거를 특별히 많이 빌리는 현상이 있습니다. 바로 출근 시간(7~9시)과 퇴근 시간(16시~19시)입니다.
- 물론 출퇴근시간이 아닌 다른 시간대에 자전거를 빌리는 경우도 존재합니다. 이는 다음에 나올 다른 시각화에서 자세히 살펴보겠습니다.

datetime-minute & datetime-second

- 이 두 컬럼은 x축이 모두 0으로 되어있습니다. 즉, `datetime-minute`과 `datetime-second`은 기록되고 있지 않다는 사실을 알 수 있습니다.

자, 이제 더 중요한 사실에 대해서 고민해 보도록 하겠습니다.

우리에게 중요한건 데이터에 어떤 특징이 있는지 발견하는 것도 있지만, **이 특징을 활용해 앞으로 사용할 머신러닝 알고리즘을 개선시킬 수 있는가?**가 더 중요합니다. 또한 개선을 한다면 구체적으로 어떤 방식으로 개선하는지도 중요하겠죠.

Q1-3) 이 사실을 통해 어떻게 예측 모델을 개선할 수 있을까요? 최소 3가지 아이디어를 내보세요.

1. 먼저 분(datetime-minute)과 초(datetime-second)는 기록되지 않기 때문에 굳이 사용할 필요가 없을 것 같습니다. 차후에 머신러닝 알고리즘에 적용할 때, 이 부분은 feature에서 제거해도 될 것 같습니다.
2. 앞서 설명한대로, train 데이터와 test 데이터를 나누는 기준이 되는 컬럼이 바로 일(datetime-day) 컬럼입니다. 이런 경우 **datetime-day**를 feature로 집어넣으면 머신러닝 알고리즘이 과적합([overfitting](https://hyperdot.wordpress.com/2017/02/06/%EA%B3%BC%EC%A0%81%ED%95%A9overfitting/)) 되는 현상이 일어날 수 있습니다. 그러므로 train 데이터와 test 데이터를 나누는 기준이 되는 컬럼이 있으면, 이 컬럼은 feature로 사용하지 않는 것이 좋을 것 같습니다.
3. 이외에도 시(datetime-hour)컬럼을 보면 출퇴근시간에 사람들이 자전거를 많이 빌린다는 사실을 알 수 있습니다. 그렇다면, 만일 머신러닝 알고리즘이 출퇴근시간이라는 개념을 이해하지 못한다고 하면 이를 별도의 feature로 넣어주면 성능 향상을 꾀할 수 있을 듯 합니다. (다만 아쉽게도, workingday 라는 컬럼이 이 역할을 대신하고 있을 것입니다)

weather 컬럼 분석

그 다음 분석하고 싶은 컬럼은 날씨를 나타내는 weather 컬럼입니다. 이 컬럼을 다음의 값을 가지며, 구체적인 설명은 다음과 같습니다.

- 1: 아주 깨끗한 날씨입니다. 또는 아주 약간의 구름이 끼어있습니다.
- 2: 약간의 안개와 구름이 끼어있는 날씨입니다.
- 3: 약간의 눈, 비가 오거나 천둥이 칩니다.
- 4: 아주 많은 비가 오거나 우박이 내립니다.

이 데이터를 엑셀 분석, 내지는 시각화하여 weather에 따라 자전거 대여량이 어떻게 변하는지 살펴보도록 하겠습니다.

Q2-1) 시각화를 하기 전에 어떤 그림이 나올 것으로 예상하시나요? 최소 3가지 아이디어를 생각해 보세요.

주의: 이 내용은 반드시 시각화를 하기 전에 작성하셔야 합니다. 그래야 시각화 결과와 본인의 아이디어를 비교해서 차이를 발견할 수 있습니다.

1. 일단 당연하지만 안 좋은 날씨일수록 자전거 대여량이 낮아질 것 같습니다. 1(깨끗한 날씨)의 경우보다 4(아주 많은 비나 우박이 오는 날씨)인 경우에 자전거를 덜 빌릴 것입니다.
2. 그리고 값이 숫자(1, 2, 3, 4)로 되어있지만, 실제로는 수의 높고 낮은 관계가 존재하지 않을 것입니다. (이를 전문용어로 연속형(continuous) 데이터 vs 범주형(categorical) 데이터라고 합니다) 그러므로 보이는 것과는 다르게, 실제로는 범주형(categorical) 데이터로 처리해야 할 것입니다.
3. 아주 심하진 않겠지만, 날씨마다의 편차가 있을 것입니다. 가령 어떤 날은 날씨가 좋아도 안 빌리고, 어떤 날은 날씨가 안 좋아도 많이 빌릴 수도 있습니다.

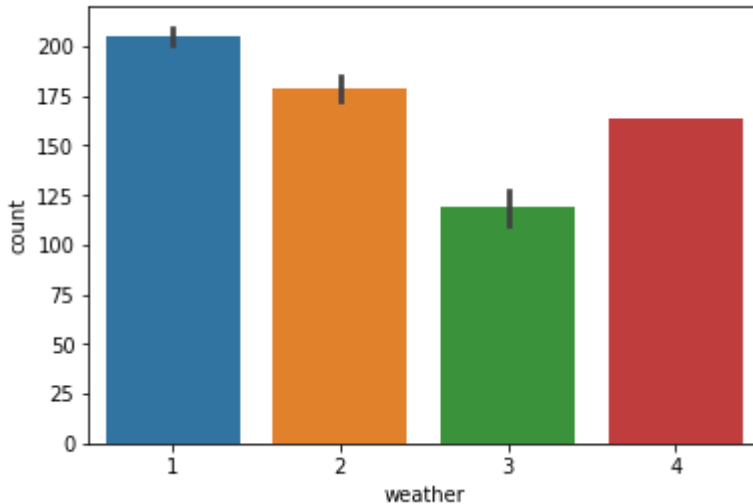
이번에도 위 예상과 실제 데이터가 일치하는지 데이터 시각화를 통해 살펴보도록 하겠습니다.

In [8]:

```
# seaborn의 barplot으로 날씨(weather) 데이터를 시각화합니다.  
# x축에는 날씨(weather) 데이터를, y축에는 자전거 대여량(count)을 출력합니다.  
sns.barplot(data=train, x="weather", y="count")
```

Out[8]:

<matplotlib.axes._subplots.AxesSubplot at 0x11d9cac50>



Q2-2) 이 시각화로 발견할 수 있는 사실은 어떤 게 있을까요? 그리고 앞서 우리의 예상과 어떤 차이가 있나요?

1. 앞서 생각한대로 날씨(weather)가 안 좋을수록 자전거 대여량이 낮아지는 현상을 발견할 수 있었습니다. 즉, 날씨(weather)값이 3보다 2가, 2보다 1이 더 자전거를 많이 빌리는 현상이 보입니다.
2. 하지만 굉장히 특이하게도 날씨가 4인 경우, 즉 아주 많은 비가 오거나 우박이 내리는 경우에 자전거를 많이 빌리는 현상이 보입니다. 심지어는 날씨가 2인 경우(약간의 안개나 구름)에 못지 않게 자전거를 많이 빌리는 사실을 알 수 있습니다.
3. 그리고 시각화에서 신뢰 구간(confidence interval)을 상징하는 검은색 세로 선이 날씨가 4인 경우에는 보이지 않습니다. 추측컨데 날씨가 4인 경우에는 일반적인 분포와는 다소 다른 현상이 일어나고 있다고 판단할 수 있습니다.

Q2-3) 이 사실을 통해 어떻게 예측 모델을 개선할 수 있을까요? 최소 3가지 아이디어를 내보세요.

1. 날씨(weather) 컬럼값이 1 ~ 3인 것만 봤을 때, 이 컬럼을 머신러닝 알고리즘에 feature로 넣으면 우리가 별도의 룰을 설정해주지 않아도 머신러닝 알고리즘이 알아서 날씨(weather)에 따른 자전거 대여량의 변화를 예측할 수 있을 것 같습니다. 아마도 날씨가 좋을 수록(1에 가까울수록) 자전거를 많이 빌리고, 안 좋을수록(3에 가까울수록) 자전거를 덜 빌릴 것 같습니다.
2. 하지만 신뢰 구간(confidence interval)을 보면 알 수 있듯이, 날씨(weather) 컬럼값이 4인 경우는 매우 부족합니다. (train 데이터 10,886개 기준 1개 밖에 없음) 그러므로 이 데이터를 제거하거나, 날씨(weather) 컬럼값이 4인 경우는 3으로 바꿔주면 머신러닝 알고리즘이 상대적으로 모수가 적은 데이터를 가지고 조건을 만들려는 시도를 하지 않기 때문에, 미묘하지만 성능 향상이 있을 것으로 예상합니다.
3. 또한 날씨(weather) 컬럼은 연속적(continuous)인 것 처럼 보이지만, 범주형(categorical) 데이터같은 느낌도 있습니다. 그렇기 때문에, 마치 타이타닉 경진대회에서 선착장(Embarked) 컬럼을 One Hot Encoding했던 것 처럼, 이 컬럼도 One Hot Encoding을 한다면 머신러닝 알고리즘이 이 데이터를 범주형(categorical) 값으로 인식하여 성능이 좋아질 가능성이 있습니다.

windspeed 컬럼 분석

그 다음 분석하고 싶은 컬럼은 날씨를 나타내는 풍속을 나타내는 windspeed 컬럼입니다. 이 컬럼은 0에서 56까지의 값을 가집니다. 이 데이터도 시각화 해보도록 하겠습니다.

Q3-1) 시각화를 하기 전에 어떤 그림이 나올 것으로 예상하시나요? 최소 3가지 아이디어를 생각해 보세요.

1. 이 데이터는 연속형(continuous) 자료이므로 분포를 시각화하면 전형적인 정규 분포 (https://ko.wikipedia.org/wiki/%EC%A0%95%EA%B7%9C_%EB%B6%84%ED%8F%AC)가 나올 것입니다.
2. 하지만 이 데이터는 현실 세계의 데이터이기 때문에, 이론처럼 완벽한 정규 분포가 나오지는 않을 것입니다. 아마도 추측컨데 1) 몇몇 아웃라이어(이상치)가 존재하거나, 2) 바람이 특별하게 많이 불어서 분포의 오른쪽이 길게 늘어지는 현상이 생길 것 같습니다.
3. 그리고 추측컨데 바람이 너무 많이 불면 사람들이 자전거를 덜 빌릴 것으로 예상합니다.

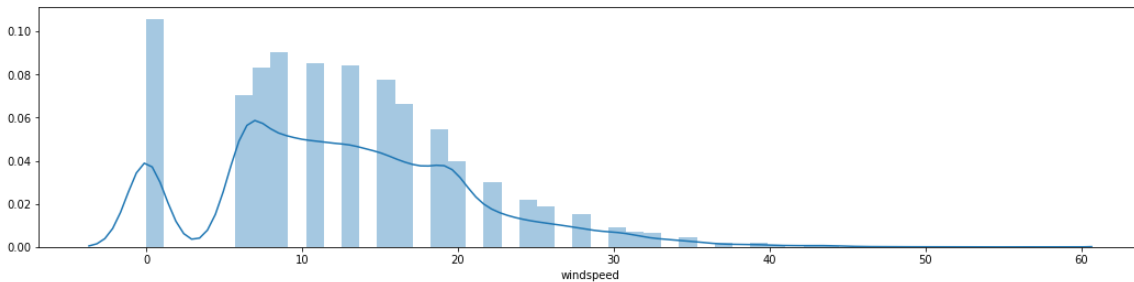
위 예상과 실제 데이터가 일치하는지 다시 한 번 살펴보도록 하겠습니다.

In [9]:

```
# matplotlib를 활용해 그래프의 사이즈를 가로로 길게(18) 늘리고, 세로로 짧게(4) 줄입니다.  
plt.figure(figsize = (18, 4))  
  
# seaborn의 distplot으로 풍속(windspeed) 데이터를 시각화합니다.  
# windspeed 데이터의 전체 분포(distribution)을 시각화합니다.  
sns.distplot(train["windspeed"])
```

Out[9]:

<matplotlib.axes._subplots.AxesSubplot at 0x11db7f048>

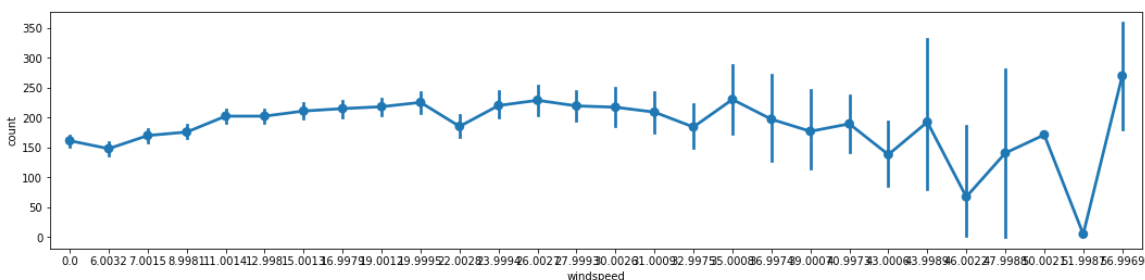


In [10]:

```
# matplotlib를 활용해 그래프의 사이즈를 가로로 길게(18) 늘리고, 세로로 짧게(4) 줄입니다.  
plt.figure(figsize = (18, 4))  
  
# seaborn의 pointplot으로 풍속(windspeed)에 따른 자전거 대여량의 변화를 시각화합니다.  
# x축에는 풍속(windspeed)을, y축에는 자전거 대여량(count)을 집어넣습니다.  
sns.pointplot(data = train, x="windspeed", y="count")
```

Out[10]:

<matplotlib.axes._subplots.AxesSubplot at 0x11db51f60>



In [11]:

```
# 풍속(windspeed) 데이터에서 소수점이 아닌 나머지를 제거합니다.
# 가령 풍속이 16.9979일 경우, 이를 정수형(int)으로 변환하여 16으로 만든 뒤 두 개를 뺍니다.
# (그러면 0.9979가 나옵니다)
# 이를 windspeed(point)라는 이름의 새로운 컬럼에 할당합니다.
train["windspeed(point)"] = train["windspeed"] - train["windspeed"].astype('int')

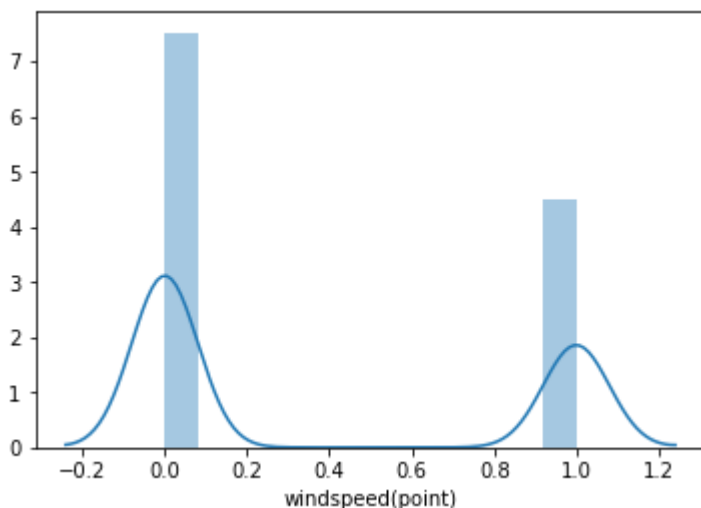
# windspeed(point) 컬럼의 중복을 제거하여 종류를 출력합니다.
# 간결한 형태로 출력하기 위해, 소수점 3자리로 줄입니다.
print(train["windspeed(point)"].unique().round(3))

# seaborn의 distplot으로 풍속(windspeed) 데이터를 시각화합니다.
# windspeed 데이터의 전체 분포(distribution)를 시각화합니다.
sns.distplot(train["windspeed(point)"])
```

```
[0.    0.003 0.998 0.001 1.    0.998 0.001 0.998 0.001 0.003 0.003 0.999
 0.999 0.003 0.002 0.998 0.997 0.001 0.001 0.001 0.999 0.997 0.999 0.002
 0.002 0.001 0.997 0.999]
```

Out[11]:

<matplotlib.axes._subplots.AxesSubplot at 0x11dfd0438>



Q3-2) 이 시각화로 발견할 수 있는 사실은 어떤 게 있을까요? 그리고 앞서 우리의 예상과 어떤 차이가 있나요?

1. 분포는 전반적으로 정규 분포가 나오지만, 1) 값이 연속적(continuous)이지 않고 듬성듬성 떨어져 있습니다. 즉, 이 데이터는 연속형(continuous) 데이터가 아닌 범주형(categorical) 데이터에 가까워 보입니다.
2. 더 특이한건, 풍속이 0인 경우가 굉장히 많으며, 정규 분포가 이상하게 보일 정도로 비중이 높습니다.
3. 또한 풍속이 과하게 높을수록 자전거를 덜 빌리는 현상이 보이는 것 같은데, 이는 전반적으로 모수가 부족한 듯 하여 신뢰도가 높지 않습니다. 다만 풍속이 낮을 경우에 전반적으로 자전거 대여량이 낮은 현상이 보입니다. (이는 우리가 예상하지 못한 현상입니다)

Q3-3) 이 사실을 통해 어떻게 예측 모델을 개선할 수 있을까요? 최소 3가지 아이디어를 내보세요.

1. 이 풍속(windspeed) 데이터를 머신러닝 알고리즘에 집어넣으면 머신러닝 알고리즘의 풍속에 따른 자전거 대여량의 변화를 스스로 판단할 수 있을 것 같습니다. 더 정확히는, 풍속이 낮거나 높을수록 자전거를 덜 빌리고, 풍속이 적당할 때 자전거를 더 많이 빌린다는 사실을 알 수 있습니다.
2. 풍속(windspeed)이 0인 경우는, 실제로는 풍속이 0이 아니라 값이 비어있을(NaN) 가능성이 있습니다. 이 경우에는 풍속(windspeed) 데이터를 머신러닝 알고리즘에 그대로 집어넣을 경우, 실제로는 바람이 불었지만 머신러닝 알고리즘 입장에서는 바람이 불지 않았다고 간주할 수 있습니다. 이 문제를 해결해준다면 머신러닝 알고리즘의 성능을 크게 끌어올릴 수 있을 것 같습니다.
3. 또한 풍속(windspeed)의 소수점을 자세히 보시면, 0.001xx, 0.002xx, 0.003xx나, 0.999xx, 0.998xx, 0.997xx가 대부분입니다. 즉, 이 값은 실수형(float)이 아닌 정수형(int)이라고 간주할 수 있습니다. 알고리즘에 따라 다르겠지만, 풍속(windspeed) 데이터를 그대로 넣지 않고 반올림한 뒤 정수형(int) 데이터로 넣어주면 더 좋은 결과가 나올 수도 있습니다.

temp

이번에 분석할 데이터는 온도(temp) 컬럼입니다. 여기서부터는 제가 직접 하지 않고, 여러분들을 위한 과제로 제공하겠습니다. 앞서 컬럼들을 분석했던 것 처럼, 온도(temp) 컬럼도 직접 분석해보세요.

힌트: 온도(temp) 컬럼만으로 좋은 분석 결과가 나오지 않는다면, 체감온도(atemp)를 포함한 다른 컬럼을 활용하여 시각화해보세요. 시각화는 [lmplot \(https://seaborn.pydata.org/generated/seaborn.lmplot.html?highlight=lmplot#seaborn.lmplot\)](https://seaborn.pydata.org/generated/seaborn.lmplot.html?highlight=lmplot#seaborn.lmplot)이나 [scatterplot \(https://seaborn.pydata.org/generated/seaborn.scatterplot.html?highlight=scatterplot#seaborn.scatterplot\)](https://seaborn.pydata.org/generated/seaborn.scatterplot.html?highlight=scatterplot#seaborn.scatterplot)을 사용하면 직관적인 시각화를 할 수 있을 것입니다. (단 scatterplot 은 seaborn의 버전이 낮으면 실행되지 않으니 이 점 주의해주세요. 이 경우는 버전을 업그레이드 한 뒤 사용하시면 됩니다)

Q4-1) 온도(temp) 컬럼을 시각화 하기 전에 어떤 그림이 나올 것으로 예상하시나요?

주의: 이 내용은 반드시 시각화를 하기 전에 작성하셔야 합니다. 그래야 시각화 결과와 본인의 아이디어를 비교해서 차이를 발견할 수 있습니다.

1. 풍속(windspeed) 데이터와 마찬가지로, 이 데이터는 연속형(continuous) 자료이므로 분포를 시각화하면 전형적인 정규 분포 (https://ko.wikipedia.org/wiki/%EC%A0%95%EA%B7%9C_%EB%B6%84%ED%8F%AC)가 나올 것입니다.
2. 또한 온도(temp)가 높으면 자전거를 덜 빌릴 것 같고, 정 반대로 온도가 낮으면 자전거를 많이 빌릴 것 같습니다. (하지만 온도가 너무 낮은 경우는 자전거를 덜 빌릴 것 같습니다)
3. 온도(temp)와 체감온도(atemp)는 서로 상관관계가 높으며, 사실상 거의 같은 성향의 데이터라고 간주할 수 있습니다.

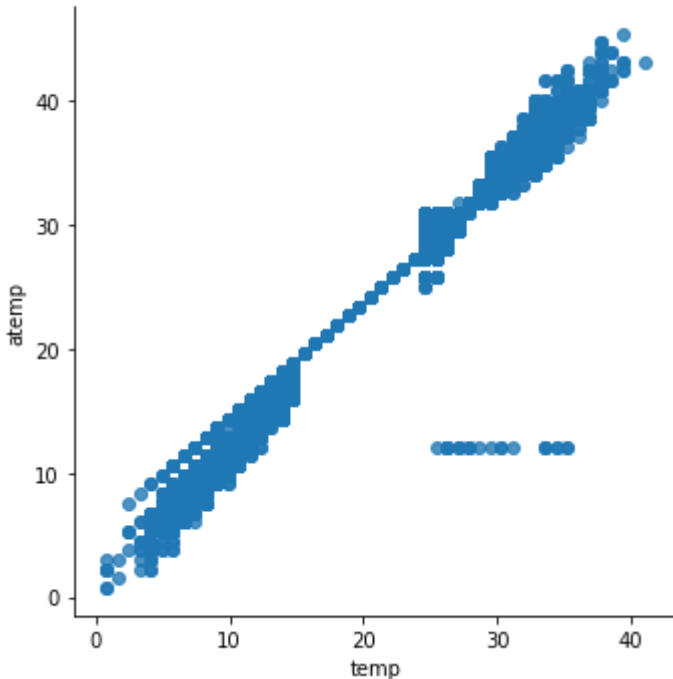
temp 컬럼 시각화하기

In [12]:

```
# seaborn의 lmpplot으로 온도(temp)와 체감온도(atemp)의 상관관계를 시각화합니다.  
# 여기서 추세선을 그리지 않기 위해 fit_reg 옵션은 False로 줍니다.  
sns.lmplot(data=train, x="temp", y="atemp", fit_reg=False)
```

Out[12]:

<seaborn.axisgrid.FacetGrid at 0x11df990b8>



In [13]:

```
# 온도(temp)의 종류를 출력합니다. 생각보다 종류가 많지 않습니다.  
train["temp"].unique()
```

Out[13]:

```
array([ 9.84,  9.02,  8.2 , 13.12, 15.58, 14.76, 17.22, 18.86, 18.04,  
       16.4 , 13.94, 12.3 , 10.66,  6.56,  5.74,  7.38,  4.92, 11.48,  
        4.1 ,  3.28,  2.46, 21.32, 22.96, 23.78, 24.6 , 19.68, 22.14,  
       20.5 , 27.06, 26.24, 25.42, 27.88, 28.7 , 30.34, 31.16, 29.52,  
       33.62, 35.26, 36.9 , 32.8 , 31.98, 34.44, 36.08, 37.72, 38.54,  
        1.64,  0.82, 39.36, 41.  ])
```

In [14]:

```
# 체감온도(atemp)의 종류를 출력합니다. 생각보다 종류가 많지 않습니다.  
train["atemp"].unique()
```

Out[14]:

```
array([14.395, 13.635, 12.88 , 17.425, 19.695, 16.665, 21.21 , 22.725,  
       21.97 , 20.455, 11.365, 10.605,  9.85 ,  8.335,  6.82 ,  5.305,  
        6.06 ,  9.09 , 12.12 ,  7.575, 15.91 ,  3.03 ,  3.79 ,  4.545,  
       15.15 , 18.18 , 25.   , 26.515, 27.275, 29.545, 23.485, 25.76 ,  
       31.06 , 30.305, 24.24 , 18.94 , 31.82 , 32.575, 33.335, 28.79 ,  
       34.85 , 35.605, 37.12 , 40.15 , 41.665, 40.91 , 39.395, 34.09 ,  
       28.03 , 36.365, 37.88 , 42.425, 43.94 , 38.635,  1.515,  0.76 ,  
        2.275, 43.18 , 44.695, 45.455])
```


In [15]:

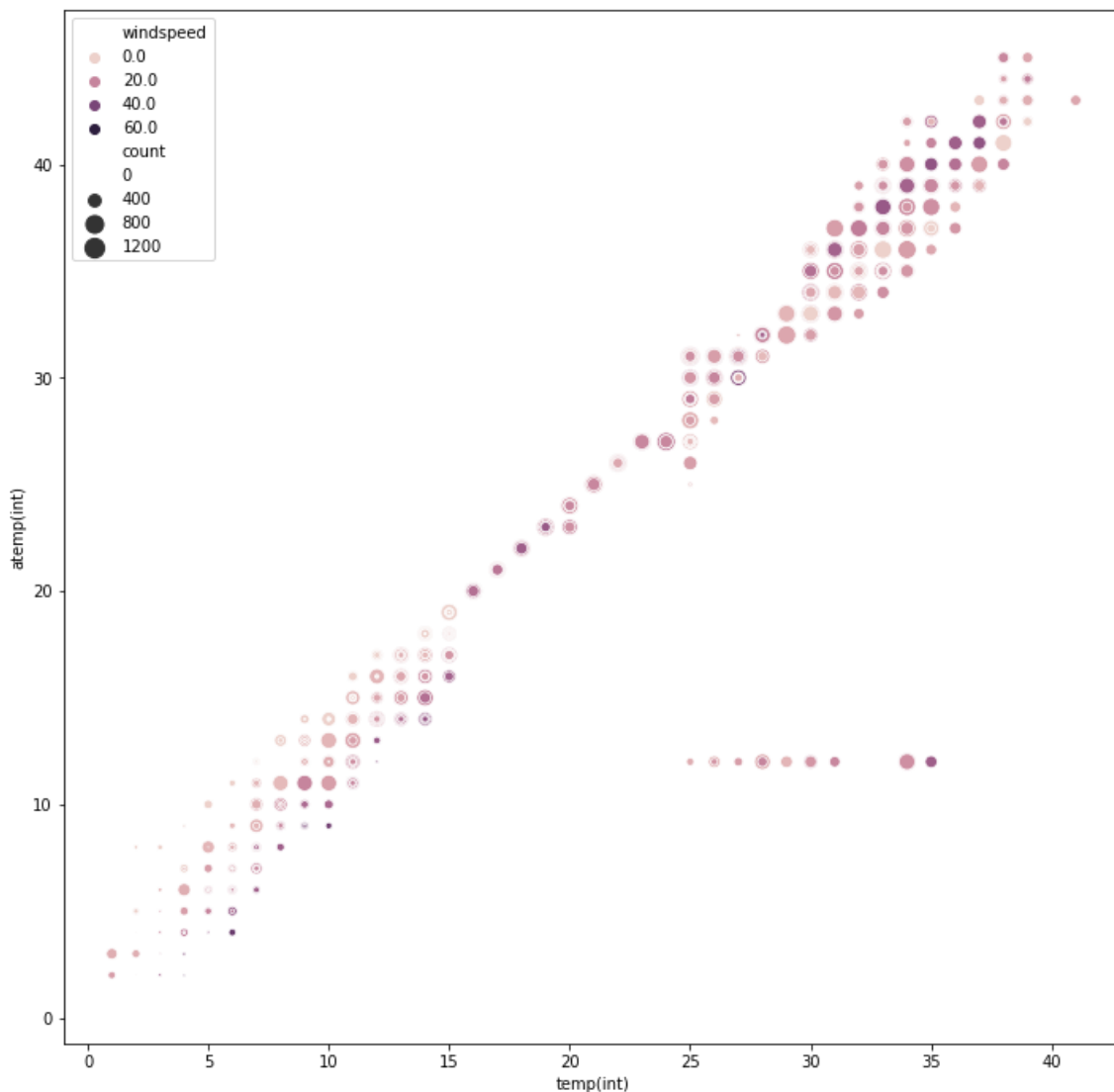
```
# 온도(temp)와 체감온도(atemp)는 반올림하여
# 각각 temp(int)와 atemp(int)라는 이름의 새로운 컬럼에 할당합니다.
train["temp(int)"] = train["temp"].round()
train["atemp(int)"] = train["atemp"].round()

# matplotlib를 활용해 그래프의 사이즈를 가로로 길게(18) 늘리고, 세로로 짧게(4) 줄입니다.
plt.figure(figsize = (12, 12))

# 이번에는 scatterplot을 활용해 온도(temp), 체감온도(atemp),
# 그리고 풍속(windspeed)과 자전거 대여량(count) 간의 상관관계를 시각화합니다.
# 주의: 이 코드는 seaborn의 버전이 0.9 이상이어야 합니다. 만일 실행이 되지 않는다면 seaborn의
# 버전을 업그레이드 해주세요.
sns.scatterplot(data=train, x="temp(int)", y="atemp(int)", hue="windspeed", size="count", sizes=(0, 150))
```

Out[15]:

<matplotlib.axes._subplots.AxesSubplot at 0x11e1cbef0>



In [16]:

```
# 정수형(int)으로 변환한 온도(temp)와 체감온도(atep)를 빼서
# 두 값의 차이를 나타내는 temp(difference)라는 새로운 컬럼을 추가합니다.
train["temp(difference)"] = train["temp(int)"] - train["atep(int)"]

# 정수형(int)으로 변환한 온도(temp)와 체감온도(atep), 그리고 temp(difference)를 출력합니다.
train[["temp(int)", "atep(int)", "temp(difference)"]].head()
```

Out[16]:

	temp(int)	atep(int)	temp(difference)
0	10.0	14.0	-4.0
1	9.0	14.0	-5.0
2	9.0	14.0	-5.0
3	10.0	14.0	-4.0
4	10.0	14.0	-4.0

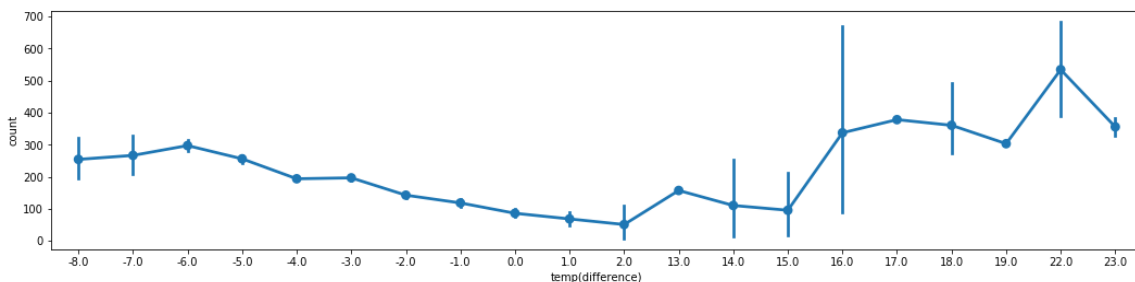
In [17]:

```
# matplotlib를 활용해 그래프의 사이즈를 가로로 길게(18) 늘리고, 세로로 짧게(4) 줄입니다.
plt.figure(figsize = (18, 4))

# seaborn의 point으로 temp(difference)에 따른 자전거 변화량을 시각화합니다.
sns.pointplot(data=train, x="temp(difference)", y="count")
```

Out[17]:

<matplotlib.axes._subplots.AxesSubplot at 0x11e1db780>



In [18]:

```
# temp(difference)의 종류와 그 개수를 출력합니다.  
# -6 부터 0을 제외한 나머지는 모수가 부족하므로, 이를 하나로 합쳐줘도 될 것 같습니다.  
train["temp(difference)"].value_counts()
```

Out[18]:

```
-4.0    5106  
-3.0    2097  
-2.0    1415  
-5.0    1049  
-1.0     509  
-6.0     405  
 0.0     219  
 1.0      32  
-7.0      21  
22.0       5  
-8.0       5  
14.0       5  
 2.0       4  
16.0       3  
15.0       3  
18.0       3  
23.0       2  
17.0       1  
13.0       1  
19.0       1
```

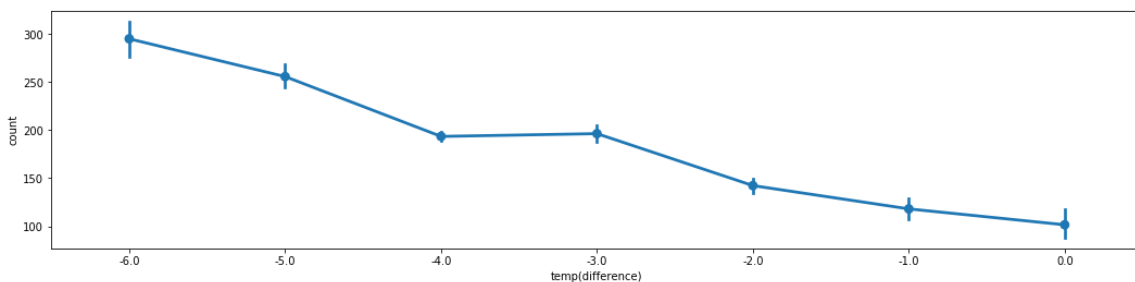
Name: temp(difference), dtype: int64

In [19]:

```
# temp(difference)가 -6보다 작은 데이터는 -6으로 통일시켜줍니다.  
train.loc[train["temp(difference)"] < -6, "temp(difference)"] = -6  
  
# 비슷하게, temp(difference)가 0보다 큰 데이터는 0으로 통일시켜줍니다.  
train.loc[train["temp(difference)"] > 0, "temp(difference)"] = 0  
  
# matplotlib를 활용해 그래프의 사이즈를 가로로 길게(18) 늘리고, 세로로 짧게(4) 줄입니다.  
plt.figure(figsize = (18, 4))  
  
# seaborn의 pointplot으로 temp(difference)에 따른 자전거 변화량을 시각화합니다.  
sns.pointplot(data=train, x="temp(difference)", y="count")
```

Out[19]:

<matplotlib.axes._subplots.AxesSubplot at 0x11e073b38>



Q4-2) 이 시각화로 발견할 수 있는 사실은 어떤 게 있을까요? 그리고 앞서 우리의 예상과 어떤 차이가 있나요?

1. 앞서 예상과 마찬가지로, 온도(temp)와 체감온도(atemp)는 연속형(continuous)인 데이터이며, 서로 상관관계가 높다는 사실을 알 수 있습니다.
2. 또한 온도(temp)나 체감온도(atemp)가 낮을수록 자전거를 덜 빌리고, 높을수록 자전거를 많이 빌린다는 사실을 알 수 있습니다.
3. 하지만 가끔 온도(temp)에 비해서 체감온도(atemp)가 상대적으로 높거나 낮은 경우가 있는데, 특히나 체감온도(atemp)가 온도(temp)보다 높을 경우 자전거를 더 많이 빌린다는 현상을 알 수 있습니다.

Q4-3) 이 사실을 통해 어떻게 예측 모델을 개선할 수 있을까요? 최소 3가지 아이디어를 내보세요.

1. 온도(temp) 컬럼이 자전거 대여량을 예측하는데 도움이 된다는 사실을 알 수 있습니다. 이 컬럼을 머신러닝 알고리즘에 넣어주면 머신러닝 알고리즘이 온도에 따른 자전거 대여량의 변화량을 예측하면서 성능이 올라갈 것으로 보입니다.
2. 또한 비슷하게 체감온도(atemp)도 자전거 대여량을 예측하는데 도움이 됩니다. 다만 온도(temp)와 체감온도(atemp)는 서로 비슷한 성향을 가지기 때문에, 두 컬럼 모두를 넣는건 둘 중 하나만 넣는 것에 비해 머신러닝 알고리즘의 성능을 향상시키는데 큰 도움을 주지는 못할 것입니다.
3. 또한 온도(temp)와 체감온도(atemp)의 차이가 자전거 대여량을 예측하는데 도움이 된다는 새로운 사실을 알 수 있었습니다. 이 정보를 머신러닝 알고리즘에 넣어주면 머신러닝의 성능 향상을 꾀할 수 있을 것으로 보입니다.

datetime-dayofweek

이번에 분석할 데이터는 요일을 나타내는 데이터, 일명 datetime-dayofweek 입니다. 이 데이터는 현재 feature로 가지고 있지만, 날짜와 시간을 나타내는 datetime 컬럼에서 추출할 수 있습니다. (ex: `train["datetime"].dt.day_name()`) 앞서 분석과 마찬가지로, 날짜 컬럼도 탐험적 데이터 분석을 통해 분석해보겠습니다.

In [20]:

```
# train 데이터에 요일을 나타내는 새로운 컬럼을 생성합니다.
# 이 컬럼의 이름을 datetime-dayofweek 이라고 가정합니다.
# 이 컬럼에 날짜(datetime) 컬럼의 dt(datetime의 약자입니다) 옵션을 활용하여 요일 정보를 따로
  넣어줍니다.
train["datetime-dayofweek"] = train["datetime"].dt.day_name()

# train 변수에 할당된 데이터의 행렬 사이즈를 출력합니다.
# 출력은 (row, column) 으로 표시됩니다.
print(train.shape)

# .head()로 train 데이터의 상위 5개를 띄우되,
# datetime 컬럼과 요일(datetime-dayofweek) 컬럼만 출력합니다.
train[["datetime", "datetime-dayofweek"]].head()
```

(10886, 23)

Out[20]:

	datetime	datetime-dayofweek
0	2011-01-01 00:00:00	Saturday
1	2011-01-01 01:00:00	Saturday
2	2011-01-01 02:00:00	Saturday
3	2011-01-01 03:00:00	Saturday
4	2011-01-01 04:00:00	Saturday

In [21]:

```
# train 데이터와 마찬가지로 test 데이터의 datetime 컬럼에서 요일(dayofweek) 정보를 뽑아냅니다.
test["datetime-dayofweek"] = test["datetime"].dt.day_name()

# test 변수에 할당된 데이터의 행렬 사이즈를 출력합니다.
# 출력은 (row, column) 으로 표시됩니다.
print(test.shape)

# .head()로 test 데이터의 상위 5개를 띄우되,
# datetime 컬럼과 요일(datetime-dayofweek) 컬럼만 출력합니다.
test[["datetime", "datetime-dayofweek"]].head()
```

(6493, 16)

Out[21]:

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

Q5-1) 요일(datetime-dayofweek) 컬럼을 시각화 하기 전에 어떤 그림이 나올 것으로 예상하시나요?

주의: 이 내용은 반드시 시각화를 하기 전에 작성하셔야 합니다. 그래야 시각화 결과와 본인의 아이디어를 비교해서 차이를 발견할 수 있습니다.

1. 사람의 생활 패턴은 주중이냐/주말이냐에 따라 극명하게 달라집니다. 그런 의미에서 요일(datetime-dayofweek) 컬럼을 시각화하면 어떤 요일이냐에 따라서 자전거 대여량이 다르게 시각화될 것 같습니다.
2. 하지만 비슷한 성향을 가진 근무일(workingday , 사실상 주중/주말을 나타냅니다) 컬럼과는 달리, 요일 (datetime-dayofweek) 컬럼은 같은 주말이라고 하더라도 토/일의 차이, 같은 주중이라고 하더라도 월/수/금의 차이가 드러날 것입니다.
3. 그리고 사람의 구체적인 생활 패턴에 따라 자전거의 대여량이 달라질 것으로 예상합니다. 가령 출퇴근을 하는 날이냐(출퇴근 시간에 많이 빌릴 것으로 예상합니다), 주말에 노는 날이냐(주중 낮에 많이 빌릴 것으로 예상합니다)에 따라 자전거 대여량이 달라질 것입니다.

위 예상과 실제 데이터가 일치하는지 시각화를 통해 분석해보도록 하겠습니다.

datetime-dayofweek 컬럼 시각화하기

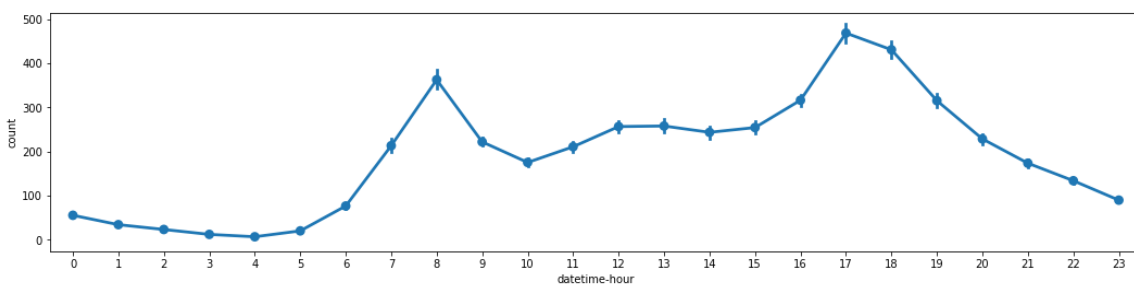
In [22]:

```
# matplotlib를 활용해 그래프의 사이즈를 가로로 길게(18) 늘리고, 세로로 짧게(4) 줄입니다.
plt.figure(figsize = (18, 4))

# seaborn의 pointplot으로 시간(datetime-hour)에 따른 자전거 변화량을 시각화합니다.
sns.pointplot(data=train, x="datetime-hour", y="count")
```

Out[22]:

<matplotlib.axes._subplots.AxesSubplot at 0x11eb1f898>



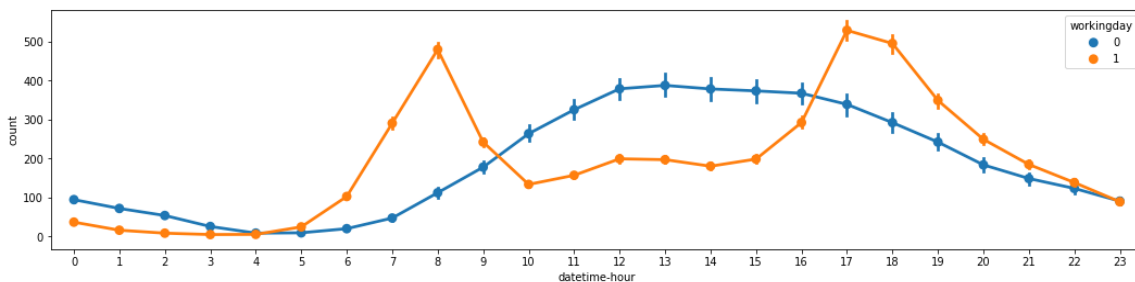
In [23]:

```
# matplotlib를 활용해 그래프의 사이즈를 가로로 길게(18) 늘리고, 세로로 짧게(4) 줄입니다.
plt.figure(figsize = (18, 4))

# seaborn의 pointplot으로 근무일 여부(workingday)와 시간(datetime-hour)에 따른 자전거 변화
# 양을 시각화합니다.
# 확실히 근무하는 날과 근무하지 않는 날의 자전거 대여량이 명확하게 차이가 납니다.
sns.pointplot(data=train, x="datetime-hour", y="count", hue="workingday")
```

Out[23]:

<matplotlib.axes._subplots.AxesSubplot at 0x11e78e518>



In [24]:

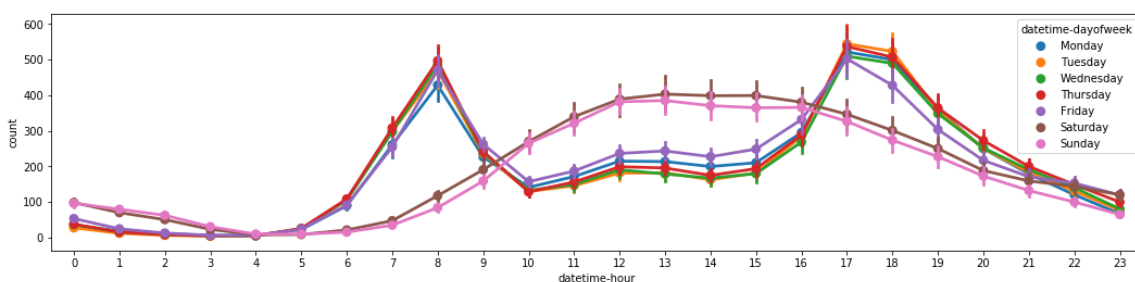
```
# matplotlib를 활용해 그래프의 사이즈를 가로로 길게(18) 늘리고, 세로로 짧게(4) 줄입니다.
plt.figure(figsize = (18, 4))

# 시각화할 때 요일의 순서를 명확하게 지정하기 위해, 월~일까지의 영어 호칭을 리스트로 담습니다.
# 이를 dayofweek_order라는 이름의 변수에 할당합니다.
dayofweek_order = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]

# seaborn의 pointplot으로 요일(datetime-dayofweek)과 시간(datetime-hour)에 따른 자전거 변
# 화량을 시각화합니다.
# 여기서 hue의 값을 dayofweek_order에 지정된 순서대로 출력하게 합니다.
sns.pointplot(data=train, x="datetime-hour", y="count", hue="datetime-dayofweek", hue_o
rder=dayofweek_order)
```

Out[24]:

<matplotlib.axes._subplots.AxesSubplot at 0x11ed46e48>



Q5-2) 이 시각화로 발견할 수 있는 사실은 어떤 게 있을까요? 그리고 앞서 우리의 예상과 어떤 차이가 있나요?

확실히 주중/주말을 나누는 `workingday` 컬럼에 비해, 구체적인 요일을 나타내는 `datetime-dayofweek` 컬럼이 더 디테일한 자전거 변화량을 나타낸다는 사실을 알 수 있습니다. 구체적으로는

1. 같은 주말이라고 하더라도 일요일이 토요일보다 자전거 대여량이 낮습니다. 이는 추측컨데 일요일은 다음 날이 근무일(월요일)이므로, 레크레이션이나 액티비티를 즐기고 싶다면 토요일에 즐기고 싶다는 사실이 드러난게 아닐까 생각합니다.
2. 비슷한 관점에서, 같은 주중이라고 금요일이 퇴근하는 시간대(18시 ~ 20시)에 자전거 대여량이 다른 주중에 비해 낮습니다. 이는 추측컨데 다음 날이 휴일(토요일)이므로, 금요일 밤을 즐기기 위해 차를 몰고 왔거나, 음주 후 차량이나 자전거를 이용할 수 없어서 다른 대중교통(버스, 지하철, 택시 등)을 이용한게 아닐까 생각합니다.
3. 또한 금요일은 같은 주중이라고 하더라도 오후 시간대(10시 ~ 16시)에 자전거 대여량이 높습니다. 그리고 그 다음 높은 주중이 월요일입니다. 이는 금요일과 월요일은 주중임에도 불구하고 주말과 끼여있는 요일이기 때문에, 주말의 성향을 어느정도 타는게 아닐까 생각합니다.

Q5-3) 이 사실을 통해 어떻게 예측 모델을 개선할 수 있을까요? 최소 3가지 아이디어를 내보세요.

1. 일단 요일(`datetime-dayofweek`) 컬럼을 머신러닝 모델에 집어넣으면, 머신러닝 모델이 요일에 따른 자전거 변화량의 차이를 이해하기 시작하면서 성능이 올라갈 것으로 예상합니다.
2. 구체적으로는 같은 주말이라고 하더라도 토요일과 일요일의 차이, 같은 주중이라고 하더라도 수요일과 월요일의 차이를 이해하기 시작할 것입니다. 이런 점에서 근무일(`workingday`) 컬럼만 넣는 것 보다 요일(`datetime-dayofweek`) 컬럼을 넣는 것이 더 좋은 성능을 낼 것입니다.
3. 그리고 이 데이터는 연속적(continuous) 데이터가 아닌 범주형(categorical) 데이터입니다. 그 의미는, 이 데이터를 그대로 넣지 않고 타이타닉 경진대회의 선착장(`Embarked`) 컬럼과 같이 One Hot Encoding을 해서 넣어주는게 머신러닝 알고리즘이 이해하기 쉽고, 더 좋은 성능을 낼 것으로 예상합니다.

casual & registered

이번에 분석할 데이터는 비회원과 회원을 나타내는 데이터, `casual` 과 `registered` 컬럼입니다. 이 컬럼을 자전거 대여량을 나타내는 `count` 컬럼과 거의 유사합니다. (실제로 `casual` 컬럼값과 `registered` 컬럼값을 더하면 `count` 컬럼값이 나옵니다)

하지만 아쉽게도 이 컬럼은 train 데이터에는 있지만 test 데이터에는 없습니다. 그러므로 이 데이터를 머신러닝 모델의 feature로 쓰는건 불가능합니다. 그렇다고 label로 쓰기에는, 우리가 맞춰야 하는 정답(=label)은 `count` 컬럼이죠. 하지만 그럼에도 불구하고, 이 두 개의 컬럼은 `count` 컬럼과 밀접한 연관이 있기 때문에, 잘 분석하고 이용한다면 머신러닝 모델의 성능을 크게 끌어올릴 수 있을 것 같습니다.

Q6-1) casual과 registered 컬럼을 시각화 하기 전에 어떤 그림이 나올 것으로 예상하시나요?

주의: 이 내용은 반드시 시각화를 하기 전에 작성하셔야 합니다. 그래야 시각화 결과와 본인의 아이디어를 비교해서 차이를 발견할 수 있습니다.

1. 일단 두 개의 컬럼은 자전거 대여량을 나타내는 count 컬럼과 유사하므로, 시각화를 하면 거의 유사한 패턴이 나올 것으로 생각합니다.
2. 하지만 두 회원간의 자전거 대여 성향이 다르기 때문에, 자전거를 대여하는 시기나 타이밍에 있어서 미묘하게 차이가 나지 않을까 합니다.
3. 그리고 회원의 자전거 대여량에 해당하는 registered 에 비해, 비회원의 자전거 대여량에 해당하는 casual 컬럼이 더 패턴을 찾기 어려울 것입니다. (비회원이라는건 그만큼 비정기적으로 자전거를 대여하는 것이기 때문에) 이 점을 감안해서 분석하는게 좋을 것 같습니다.

위 예상과 실제 데이터가 일치하는지 시각화를 통해 분석해보도록 하겠습니다.

casual과 registered 컬럼 시각화하기

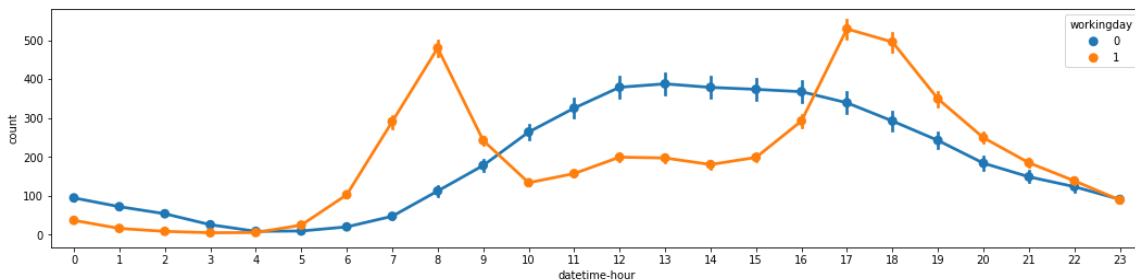
In [25]:

```
# matplotlib를 활용해 그래프의 사이즈를 가로로 길게(18) 늘리고, 세로로 짧게(4) 줄입니다.
plt.figure(figsize = (18, 4))

# seaborn의 pointplot으로 근무일 여부(workingday)와 시간(datetime-hour)에 따른 자전거 변화
# 양을 시각화합니다.
sns.pointplot(data=train, x="datetime-hour", y="count", hue="workingday")
```

Out[25]:

<matplotlib.axes._subplots.AxesSubplot at 0x11f0da588>



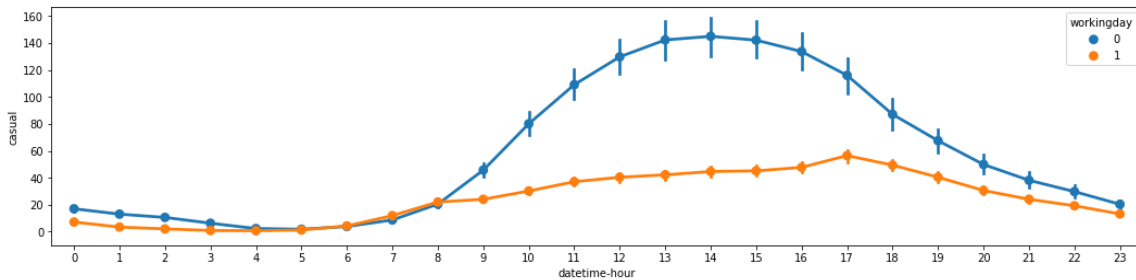
In [26]:

```
# matplotlib를 활용해 그래프의 사이즈를 가로로 길게(18) 늘리고, 세로로 짧게(4) 줄입니다.
plt.figure(figsize = (18, 4))

# seaborn의 pointplot으로 근무일 여부(workingday)와 시간(datetime-hour)에 따른 비회원의 자전거
# 변화량(casual)을 시각화합니다.
sns.pointplot(data=train, x="datetime-hour", y="casual", hue="workingday")
```

Out[26]:

<matplotlib.axes._subplots.AxesSubplot at 0x11f11fcc0>



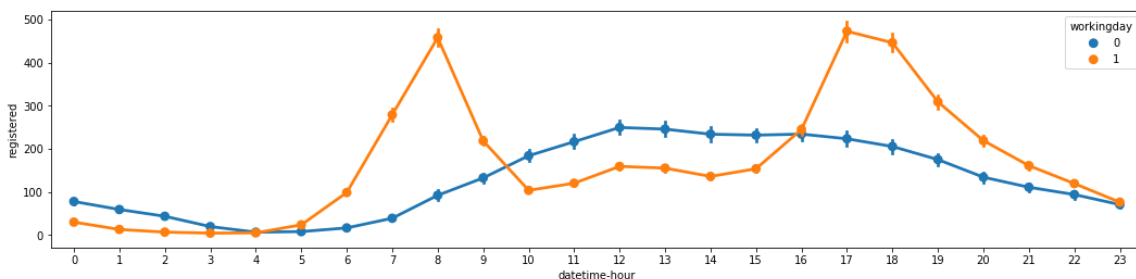
In [27]:

```
# matplotlib를 활용해 그래프의 사이즈를 가로로 길게(18) 늘리고, 세로로 짧게(4) 줄입니다.
plt.figure(figsize = (18, 4))

# seaborn의 pointplot으로 근무일 여부(workingday)와 시간(datetime-hour)에 따른 회원의 자전거
# 변화량(registered)을 시각화합니다.
sns.pointplot(data=train, x="datetime-hour", y="registered", hue="workingday")
```

Out[27]:

<matplotlib.axes._subplots.AxesSubplot at 0x11f344d30>



Q6-2) 이 시각화로 발견할 수 있는 사실은 어떤 게 있을까요? 그리고 앞서 우리의 예상과 어떤 차이가 있나요?

1. 시간(datetime-hour)에 따른 자전거 대여량의 변화량을 비교해보니, 비회원과 회원의 차이가 극명하게 난다는 사실을 알 수 있습니다.
2. 구체적으로는, 회원(registered)은 주로 출퇴근할때 자전거를 빌리고, 비회원(casual)은 주로 주말에 놀 때 자전거를 빌린다는 사실을 알 수 있습니다.
3. 그리고 시각화의 세로축(y축)을 보면 비회원(casual)보다는 회원(registered)이 자전거를 더 많이 빌린다는 사실을 알 수 있습니다.

Q6-3) 이 사실을 통해 어떻게 예측 모델을 개선할 수 있을까요? 최소 3가지 아이디어를 내보세요.

1. 일단 아쉽게도, 비회원(`casual`)과 회원(`registered`) 정보는 `test` 데이터에 넣을 수 없기 때문에 `feature`로 활용할 수 없습니다.
2. 하지만 다른 방식을 통해서 유사하게 이 데이터를 쓸 수 있을 것 같습니다. 가령 머신러닝 모델로 자전거 대여량을 나타내는(`count`) 컬럼을 예측하는게 아닌, 비회원(`casual`)과 회원(`registered`) 컬럼을 각각 `label`로 간주하고 예측한 뒤 이를 더하는 것도 방법입니다.
3. 내지는 비회원(`casual`) 컬럼의 값이 작고 패턴을 예측하기 어렵기 때문에, 비회원(`casual`), 회원(`registered`), 그리고 `count` 컬럼까지 총 3개를 동시에 예측해서 이를 섞어쓰는 것도 가능합니다.

Train

이제 분석을 통해 발견한 인사이트를 활용해보겠습니다.

이전 경진대회와 마찬가지로, 이번에도 머신러닝 알고리즘을 사용하겠습니다. 이번에도 변함없이 [지도학습 \(Supervised Learning\)](http://solarisailab.com/archives/1785), (<http://solarisailab.com/archives/1785>) 알고리즘을 사용할 계획이기 때문에, 데이터를 `Label`(맞춰야 하는 정답)과 `Feature`(`Label`을 맞추는데 도움이 되는 값들)로 나눌 필요가 있습니다.

이번 경진대회에서는 다음의 컬럼들을 `Feature`와 `Label`로 활용할 것입니다.

- **Feature:** 1) 계절(`season`), 2) 공휴일(`holiday`), 3) 근무일(`workingday`), 4) 날씨(`weather`), 5) 온도(`temp`), 6) 체감 온도(`atemp`), 7) 습도(`humidity`), 8) 풍속(`windspeed`), 9) 연(`datetime-year`), 10) 월(`datetime-month`), 11) 일(`datetime-day`), 12) 시(`datetime-hour`), 13) 분(`datetime-minute`), 14) 초(`datetime-second`) 입니다.
- **Label:** 자전거 대여량(`count`)을 사용합니다.

이를 통해 `train` 데이터와 `test` 데이터를 다음의 세 가지 형태의 값으로 나눌 것입니다.

- **X_train:** `train` 데이터의 `feature` 입니다. 줄여서 `X_train`이라고 부릅니다.
- **X_test:** `test` 데이터의 `feature` 입니다. 마찬가지로 줄여서 `X_test`라고 부릅니다.
- **y_train:** `train` 데이터의 `label` 입니다. 마찬가지로 줄여서 `y_train`이라고 부릅니다.

In [33]:

```
# Write your code here!
```

Use Decision Tree

X_train, X_test, y_train 데이터를 만들었으면, 이제 이 데이터를 활용하여 Decision Tree를 학습시켜 보겠습니다.

파이썬(Python) 프로그래밍 언어에서 인공지능&머신러닝에 관련된 기능을 가장 많이 보유하고 있는 건 [scikit-learn](https://scikit-learn.org/stable/) (<https://scikit-learn.org/stable/>) 입니다. scikit-learn에 있는 머신러닝 알고리즘에는 크게 두 가지 기능이 있습니다.

- **fit**: 머신러닝 알고리즘을 학습시킵니다. 전문용어로 fitting한다고 하기 때문에 fit이라는 표현을 사용합니다. fit을 하기 위해서는 train 데이터가 필요하며, 정확히는 train 데이터의 feature(X_train)와 label(y_train)이 필요합니다.
- **predict**: fit이 끝나면, 이후에 **predict**를 통해 예측을 할 수 있습니다. predict를 하기 위해서는 test 데이터가 필요하며, 정확히는 test 데이터의 feature(X_test)가 필요합니다.

이 두 가지 기능만 숙지하고 있으면 언제 어디서든 머신러닝 알고리즘을 활용할 수 있습니다.

In [29]:

```
# Write your code here!
```

Fit

머신러닝 모델을 가져왔으면, 이제 가져온 머신러닝 모델을 학습할 시간입니다.

앞서 설명한대로 머신러닝 모델의 fit 함수로 학습을 시킵니다. 학습을 할 때는 1) train 데이터의 feature인 X_train , 그리고 2) train 데이터의 label인 y_train 이 필요합니다.

In [30]:

```
# Write your code here!
```

Predict

머신러닝 알고리즘을 성공적으로 학습시켰다면, 남은 것은 이 알고리즘을 활용해 test 데이터에 있는 자전거 대여량을 예측하는 것입니다. 예측은 model.predict 로 할 수 있으며, 이 때 test 데이터의 feature인 X_test 가 필요합니다.

In [31]:

```
# Write your code here!
```

Submit

머신러닝 알고리즘을 활용해서 우리는 test 데이터에 있는 자전거 대여량(count)을 예측하였습니다. 이제 우리에게 남은 건 이를 캐글([kaggle \(http://kaggle.com/\)](http://kaggle.com/))이 권장하는 제출(submission) 포맷에 맞게 정리한 뒤 파일로 저장하는 것입니다.

캐글의 [Bike Sharing Demand \(https://www.kaggle.com/c/bike-sharing-demand\)](https://www.kaggle.com/c/bike-sharing-demand) 경진대회에서는 **sampleSubmission.csv**라는 제출 포맷을 제공합니다. ([다운로드 링크 \(https://www.kaggle.com/c/bike-sharing-demand/data\)](https://www.kaggle.com/c/bike-sharing-demand/data)) 우리는 우리가 예측한 값을 이 제출 포맷에 맞게 집어넣고 저장할 것입니다.

In [32]:

```
# Write your code here!
```

나머지 데이터를 활용해 머신러닝 알고리즘의 성능을 끌어올리기

앞서 구현한 기본 코드를 바탕으로, 다양한 방식으로 데이터를 수정하거나 모델을 고치는 등 머신러닝 모델의 성능을 개선할 수 있습니다. 아직 우리는 분석한 데이터를 다 넣지 않았고(ex: `datetime-dayofweek`), 랜덤 포레스트라는 알고리즘에 대해서도 완벽하게 이해하지 못했습니다. 그런 관점에서, 이 코드를 바탕으로 여러분들이 스스로 모델을 개선해서 성능을 끌어올려 주세요.

몇몇 힌트를 드리자면

- 1주차 수업에서 탐험적 데이터 분석(EDA)으로 분석한 결과를 바탕으로 feature를 새롭게 추가하거나, 고치거나, 아니면 제거할 수도 있습니다. 가령 앞서 분석한 결과를 바탕으로 요일(`datetime-dayofweek`)이나 온도와 체감온도의 차이(`temp(difference)`)를 새로운 feature로 추가하는 것도 가능합니다.
- 또는 정 반대로, 이미 존재하는 feature를 제거하는 것도 가능합니다. 앞서 분석한 내용대로 일(`datetime-day`)이나 분(`datetime-minute`), 초(`datetime-second`) 컬럼을 제거하는 것도 가능하고, 그 외에도 직접 판단하기에 필요 없다고 생각되는 컬럼을 제거한 다음에 캐글에 올려보세요.
- 내지는 이미 존재하는 컬럼의 값을 수정해서 적용하는 것도 가능합니다. 계절(`season`)이나 날씨(`weather`)를 One Hot Encoding해서 머신러닝 모델에 넣는다던가, 풍속(`windspeed`)의 빈 값을 모종의 제대로된 값으로 채워서 머신러닝 알고리즘에 넣는 것도 방법입니다.
- 아니면 아예 다른 관점에서 머신러닝 모델의 성능을 개선할 수 있습니다. 가령 우리가 사용한 머신러닝 알고리즘(`RandomForestRegressor` (<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>))의 여러 옵션들을 살펴본 뒤 이를 고치는 것도 방법이고, 아예 새로운 머신러닝 알고리즘을 적용하는 것도 좋은 아이디어입니다. (새로운 머신러닝 알고리즘에 관심이 있는 분들은, [Gradient Boosting](https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d) (<https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d>) 또는 [XGBoost](https://xgboost.readthedocs.io/en/latest/) (<https://xgboost.readthedocs.io/en/latest/>)나 [LightGBM](https://lightgbm.readthedocs.io/en/latest/) (<https://lightgbm.readthedocs.io/en/latest/>))을 살펴보세요.

위의 힌트, 또는 본인이 생각하기에 중요하다고 생각되는 부분을 머신러닝 알고리즘에 적용해서 개선해보세요. 주어진 형식에 구애받지 않고 자유롭게 데이터와 모델을 고쳐나가면 됩니다. 하지만 모델을 개선하는 과정에서 몇몇 도움이 되는 노하우를 공유하자면

- 앞서 1주차 과제와 마찬가지로, 지금 상황에서 중요한건 창의성보다는 검색과 벤치마킹입니다. 유사 솔루션, 경진대회, 데이터셋을 벤치마킹하는 실력도 매우 중요합니다. 캐글에서는 보통 [Kernel](https://www.kaggle.com/c/bike-sharing-demand/kernels) (<https://www.kaggle.com/c/bike-sharing-demand/kernels>) 탭에서 사람들이 본인들만의 분석 결과와 솔루션을 올리고, [Discussion](https://www.kaggle.com/c/bike-sharing-demand/discussion) (<https://www.kaggle.com/c/bike-sharing-demand/discussion>) 탭에서 경진대회에 대한 토론을 합니다. 이 탭을 집중적으로 살펴보고 벤치마킹 해주세요. 심지어 [이런](https://www.kaggle.com/viveksrinivasan/eda-ensemble-model-top-10-percentile/notebook) (<https://www.kaggle.com/viveksrinivasan/eda-ensemble-model-top-10-percentile/notebook>) 페이지에는 경진대회 상위 10%에 도달하는 노하우가 그대로 공유되어 있습니다. 이 노하우만 잘 이해해도 충분합니다.
- 그리고 비슷하게, [Bike Sharing Demand](https://www.kaggle.com/c/bike-sharing-demand) (<https://www.kaggle.com/c/bike-sharing-demand>) 경진대회의 다양한 솔루션들을 구글에서 찾을 수도 있습니다. [다음의 링크](https://www.analyticsvidhya.com/blog/2015/06/solution-kaggle-competition-bike-sharing-demand/) (<https://www.analyticsvidhya.com/blog/2015/06/solution-kaggle-competition-bike-sharing-demand/>)나 [다음의 링크](https://medium.com/@viveksrinivasan/how-to-finish-top-10-percentile-in-bike-sharing-demand-competition-in-kaggle-part-1-c816ea9c51e1) (<https://medium.com/@viveksrinivasan/how-to-finish-top-10-percentile-in-bike-sharing-demand-competition-in-kaggle-part-1-c816ea9c51e1>)처럼 이 경진대회에 대해 자세히 분석하고 솔루션을 제시하는 곳도 있습니다. 이런 솔루션을 구글에서 찾아서 적극적으로 벤치마킹 해주세요.
- 마지막으로, 데이터는 많이 분석하면 분석할수록 노하우가 쌓입니다. 그리고 캐글 경진대회도 많이 참여할수록 점점 노하우가 쌓이게 됩니다. 그런 의미에서, 이전에 참여한 경진대회에서 먹었던 분석 노하우가 전략을 적극적으로 활용해보세요. 가령 [Titanic](https://www.kaggle.com/c/titanic) (<https://www.kaggle.com/c/titanic>) 경진대회에서 먹었던 전략을 그대로 활용하는 것도 가능합니다.

제출

이번 2주차 과제의 목표는 [bike sharing demand \(https://www.kaggle.com/c/bike-sharing-demand\)](https://www.kaggle.com/c/bike-sharing-demand) 경진대회에서 최소 상위 40% 안에 드는 것입니다. (캐글 점수 기준 0.48368 이내) 만일 상위 30% 안에 들 수 있다면 수업을 굉장히 잘 따라왔다고 간주할 수 있고, (캐글 점수 기준 0.46206 이내) 상위 25% 안에 들었다고 한다면 데이터 사이언스를 처음 공부하는 사람 입장에서는 굉장히 풍부한 소질이 있다고 간주할 수 있습니다. (캐글 점수 기준 0.44727 이내)

만일 상위 40% 이내에 들었다면, 전담 튜터에게 **다이렉트 메시지(Direct Message)**로 1) 캐글 점수 캡처 파일과 2) 풀이를 완료한 과제 파일(.ipynb)을 제출해주세요!

수고하셨습니다!

In []: