
▣ CH05 데이터 분석 기초

- 데이터 파악하기, 다루기 쉽게 수정하기

[05-1] 데이터 파악하기

```
> head() #앞부분 출력  
> tail() #뒷부분 출력  
> shape() #행, 열 갯수 출력  
> info() #변수 속성 출력  
> describe() #요약 통계량 출력
```

※사전에 워킹 디렉토리에 **exam.csv** 파일을 위치시킴.

```
In [1]: ## Ch05-1 데이터 파악하기  
import pandas as pd  
exam = pd.read_csv('exam.csv')  
  
exam.head() # 앞에서부터 5행까지 출력
```

```
Out[1]:   id  nclass  math  english  science  
0    1       1     50      98      50  
1    2       1     60      97      60  
2    3       1     45      86      78  
3    4       1     30      98      58  
4    5       2     25      80      65
```

```
In [7]: exam.head(3) # 앞에서부터 3행까지 출력
```

```
Out[7]:   id  nclass  math  english  science  
0    1       1     50      98      50  
1    2       1     60      97      60  
2    3       1     45      86      78
```

```
In [3]: exam.tail() # 뒤에서부터 5행까지 출력
```

```
Out[3]:    id  nclass  math  english  science
```

15	16	4	58	98	65
16	17	5	65	68	98
17	18	5	80	78	90
18	19	5	89	68	87
19	20	5	78	83	58

```
In [6]: exam.tail(3) # 뒤에서부터 3행까지 출력
```

```
Out[6]:    id  nclass  math  english  science
```

17	18	5	80	78	90
18	19	5	89	68	87
19	20	5	78	83	58

```
In [9]: exam.info() # 속성 출력
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 5 columns):
 #   Column   Non-Null Count  Dtype  
--- 
 0   id        20 non-null    int64  
 1   nclass    20 non-null    int64  
 2   math      20 non-null    int64  
 3   english   20 non-null    int64  
 4   science   20 non-null    int64  
dtypes: int64(5)
memory usage: 928.0 bytes
```

```
In [8]: exam.shape # 데이터가 몇 행, 몇 열로 구성되는지 출력
```

```
Out[8]: (20, 5)
```

```
In [18]: exam.describe(include='all') # 요약 통계량 (전체) 출력
exam.describe() # 요약 통계량 (일부) 출력
```

```
Out[18]:    id  nclass  math  english  science
```

count	20.000000	20.000000	20.000000	20.000000	20.000000
mean	10.500000	3.000000	57.450000	84.900000	59.450000
std	5.916080	1.450953	20.299015	12.875517	25.292968
min	1.000000	1.000000	20.000000	56.000000	12.000000
25%	5.750000	2.000000	45.750000	78.000000	45.000000
50%	10.500000	3.000000	54.000000	86.500000	62.500000
75%	15.250000	4.000000	75.750000	98.000000	78.000000
max	20.000000	5.000000	90.000000	98.000000	98.000000

```
In [83]: exam['math'].describe() # Series에 대한 요약 통계 출력
```

```
Out[83]: count    20.000000
mean     57.450000
std      20.299015
min     20.000000
25%     45.750000
50%     54.000000
75%     75.750000
max     90.000000
Name: math, dtype: float64
```

```
In [ ]:
```

[] mpg 데이터 파악하기

mpg(mile per gallon)는 미국 환경보호국에서 공개한 데이터

1999~2008년 미국에 출시된 자동차 24종의 제원 정보

```
In [1]: ## mpg 데이터 불러오기
import pandas as pd
mpg = pd.read_csv('mpg.csv')
```

```
In [2]: ## mpg 데이터 확인
mpg.tail()
```

```
Out[2]:   manufacturer model  displ  year  cyl      trans  drv  cty  hwy  fl  category
  229  volkswagen  passat    2.0  2008    4  auto(s6)    f   19   28   p  midsize
  230  volkswagen  passat    2.0  2008    4  manual(m6)  f   21   29   p  midsize
  231  volkswagen  passat    2.8  1999    6  auto(l5)    f   16   26   p  midsize
  232  volkswagen  passat    2.8  1999    6  manual(m5)  f   18   26   p  midsize
  233  volkswagen  passat    3.6  2008    6  auto(s6)    f   17   26   p  midsize
```

```
In [3]: ## mpg 데이터 확인
mpg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 234 entries, 0 to 233
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype  
 ____ _____
 0   manufacturer    234 non-null    object 
 1   model          234 non-null    object 
 2   displ           234 non-null    float64
 3   year            234 non-null    int64  
 4   cyl             234 non-null    int64  
 5   trans           234 non-null    object 
 6   drv              234 non-null    object 
 7   cty             234 non-null    int64  
 8   hwy             234 non-null    int64  
 9   fl               234 non-null    object 
 10  category         234 non-null    object 
dtypes: float64(1), int64(4), object(6)
memory usage: 20.2+ KB
```

```
In [4]: mpg.head()
```

Out[4]:

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	category
0	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact
1	audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact
2	audi	a4	2.0	2008	4	manual(m6)	f	20	31	p	compact
3	audi	a4	2.0	2008	4	auto(av)	f	21	30	p	compact
4	audi	a4	2.8	1999	6	auto(l5)	f	16	26	p	compact

[05-2] 변수명 바꾸기

> 데이터 프레임 복사

> 데이터 프레임 열 변수명 바꾸기

```
In [4]: ## Ch05-2 변수명 바꾸기
# 데이터 프레임 복사
df = mpg.copy()
df.head(1)
```

Out[4]:

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	category
0	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact

```
In [5]: ## Ch05-2 변수명 바꾸기
# 데이터 프레임 열 변수명 바꾸기
df = df.rename(columns = {'cty' : 'city', 'hwy' : 'highway'})
df.head(1)
```

Out[5]:

	manufacturer	model	displ	year	cyl	trans	drv	city	highway	fl	category
0	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact

<> df = mpg.copy()와 df = mpg의 차이

> df = mpg.copy()는 mpg 데이터를 df에 복사하는 방식으로 처리

> df = mpg는 mpg 데이터를 df가 참조하는 방식으로 처리

```
In [6]: # copy()로 복사된 df의 행 값 변경
df.at[0, 'city'] = df.at[0, 'city'] + 100
```

```
In [7]: df.head(1) #copy()된 데이터 확인
```

Out[7]:

	manufacturer	model	displ	year	cyl	trans	drv	city	highway	fl	category
0	audi	a4	1.8	1999	4	auto(l5)	f	118	29	p	compact

```
In [8]: mpg.head(1) #변화 없는 원본 데이터 확인
```

Out[8]:

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	category
0	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact

```
In [9]: #mpg 데이터를 df가 참조하는 방식으로 처리  
df = mpg #mpg 데이터를 df가 참조하는 방식으로 처리  
df.head(1) #copy()된 데이터 확인
```

```
Out[9]:   manufacturer model  displ  year  cyl  trans  drv  cty  hwy  fl  category  
0       audi     a4    1.8  1999    4  auto(l5)   f   18   29  p  compact
```

```
In [10]: # mpg를 참조하는 df의 행 값 변경  
df.at[0, 'cty'] = df.at[0, 'cty'] + 100
```

```
In [11]: df.head(1)
```

```
Out[11]:   manufacturer model  displ  year  cyl  trans  drv  cty  hwy  fl  category  
0       audi     a4    1.8  1999    4  auto(l5)   f  118   29  p  compact
```

```
In [12]: mpg.head(1) #함께 바뀐 원본 데이터 확인
```

```
Out[12]:   manufacturer model  displ  year  cyl  trans  drv  cty  hwy  fl  category  
0       audi     a4    1.8  1999    4  auto(l5)   f  118   29  p  compact
```

```
In [ ]:
```

[05-3] 파생변수 만들기

- > 파생 변수(**derived variable**)는 기존 속성들로부터 새롭게 만들어낸 변수
- > 변수의 추가는 데이터 프레임에 새로운 변수명을 사용하여 연산식으로 대입하면 됨.

```
In [14]: ## Ch05-3 파생변수 만들기  
#데이터 프레임 생성  
df = pd.DataFrame({'var1' : [4, 3, 8],  
                   'var2' : [2, 6, 1]})  
df
```

```
Out[14]:   var1  var2  
0      4      2  
1      3      6  
2      8      1
```

```
In [15]: ## Ch05-3 파생변수 만들기  
#데이터 프레임에 파생변수 'var_sum' 추가  
df['var_sum'] = df['var1'] + df['var2'] # var_sum 파생변수 만들기  
df
```

```
Out[15]:    var1  var2  var_sum
```

	var1	var2	var_sum
0	4	2	6
1	3	6	9
2	8	1	9

```
In [16]: ## Ch05-3 파생변수 만들기
```

```
#데이터 프레임에 파생변수 'var_mean' 추가  
df['var_mean'] = (df['var1'] + df['var2']) / 2 # var_mean 파생변수 만들기  
df
```

```
Out[16]:    var1  var2  var_sum  var_mean
```

	var1	var2	var_sum	var_mean
0	4	2	6	3.0
1	3	6	9	4.5
2	8	1	9	4.5

[실습-1] mpg 통합연비 파생변수 만들기

1. 'total' 변수 추가하시오.

>> 'total'은 'cty'와 'hwy' 연비의 평균

2. 'total'로부터 전체 통합 평균 연비를 구하시오.

```
In [1]: ## mpg 데이터 불러오기
```

```
import pandas as pd  
mpg = pd.read_csv('mpg.csv')  
mpg.head()
```

```
Out[1]:    manufacturer  model  displ  year  cyl  trans  drv  cty  hwy  fl  category
```

0	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact
1	audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact
2	audi	a4	2.0	2008	4	manual(m6)	f	20	31	p	compact
3	audi	a4	2.0	2008	4	auto(av)	f	21	30	p	compact
4	audi	a4	2.8	1999	6	auto(l5)	f	16	26	p	compact

```
In [5]: ## [실습] mpg 통합연비 파생변수 만들기
```

```
# 1. mpg 통합연비 파생변수 만들기  
mpg['total'] = (mpg['cty'] + mpg['hwy']) / 2 # 통합 연비 변수 만들기  
mpg.head()
```

Out[5]:

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	category	total
0	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact	23.5
1	audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact	25.0
2	audi	a4	2.0	2008	4	manual(m6)	f	20	31	p	compact	25.5
3	audi	a4	2.0	2008	4	auto(av)	f	21	30	p	compact	25.5
4	audi	a4	2.8	1999	6	auto(l5)	f	16	26	p	compact	21.0

In [9]:

```
## [실습] mpg 통합연비 파생변수 만들기
# 2. 'total'로부터 전체 통합 평균 연비를 구하
sum(mpg['total']) / len(mpg)
```

Out[9]:

20.14957264957265

In [10]:

```
mpg['total'].mean() # 통합 연비 변수 평균
```

Out[10]:

20.14957264957265

[실습-2] 조건문을 활용해 파생변수 만들기

1. 기준값 정하기 : ('total' > 20) 이면 'pass', 아니면 'fail'
2. 합격 판정 변수 'test' 만들기
3. 빈도표로 합격 판정 자동차 수 살펴보기
4. 막대 그래프로 빈도 표현하기

In [6]:

```
## [실습] 조건문을 활용해 파생변수 만들기
#### 1. 기준값 정하기 : ('total' > 20) 이면 'pass', 아니면 'fail'
mpg[(mpg['total'] <= 20)] #조건부 행 추출
```

Out[6]:

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	category	total
11	audi	a4 quattro	2.8	1999	6	auto(l5)	4	15	25	p	compact	20.0
14	audi	a4 quattro	3.1	2008	6	manual(m6)	4	15	25	p	compact	20.0
15	audi	a6 quattro	2.8	1999	6	auto(l5)	4	15	24	p	midsize	19.5
17	audi	a6 quattro	4.2	2008	8	auto(s6)	4	16	23	p	midsize	19.5
18	chevrolet	c1500 suburban 2wd	5.3	2008	8	auto(l4)	r	14	20	r	suv	17.0
...
203	toyota	toyota tacoma 4wd	3.4	1999	6	manual(m5)	4	15	17	r	pickup	16.0
204	toyota	toyota tacoma 4wd	3.4	1999	6	auto(l4)	4	15	19	r	pickup	17.0
205	toyota	toyota tacoma 4wd	4.0	2008	6	manual(m6)	4	15	18	r	pickup	16.5
206	toyota	toyota tacoma 4wd	4.0	2008	6	auto(l5)	4	16	20	r	pickup	18.0
219	volkswagen	jetta	2.8	1999	6	auto(l4)	f	16	23	r	compact	19.5

111 rows × 12 columns

In [12]:

```
## [실습] 조건문을 활용해 파생변수 만들기
# 20 이상이면 pass, 그렇지 않으면 fail 부여
import numpy as np #수치연산 지원 패키지 numpy
x = np.where(mpg['total'] >= 20, 'pass', 'fail') #np.where(조건식, True 시 실행문, False 시 실행문)
x
```

```
In [7]: ## [실습] 조건문을 활용해 파생변수 만들기
##### 2. 합격 판정 변수 'test' 만들기
# 20 이상이면 pass, 그렇지 않으면 fail 부여
import numpy as np    #수치연산 지원 패키지 numpy
mpg['test'] = np.where(mpg['total'] >= 20, 'pass', 'fail')
mpg.head(20)
```

Out[7]:

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	category	total	tes
0	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact	23.5	pas
1	audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact	25.0	pas
2	audi	a4	2.0	2008	4	manual(m6)	f	20	31	p	compact	25.5	pas
3	audi	a4	2.0	2008	4	auto(av)	f	21	30	p	compact	25.5	pas
4	audi	a4	2.8	1999	6	auto(l5)	f	16	26	p	compact	21.0	pas
5	audi	a4	2.8	1999	6	manual(m5)	f	18	26	p	compact	22.0	pas
6	audi	a4	3.1	2008	6	auto(av)	f	18	27	p	compact	22.5	pas
7	audi	a4 quattro	1.8	1999	4	manual(m5)	4	18	26	p	compact	22.0	pas
8	audi	a4 quattro	1.8	1999	4	auto(l5)	4	16	25	p	compact	20.5	pas
9	audi	a4 quattro	2.0	2008	4	manual(m6)	4	20	28	p	compact	24.0	pas
10	audi	a4 quattro	2.0	2008	4	auto(s6)	4	19	27	p	compact	23.0	pas
11	audi	a4 quattro	2.8	1999	6	auto(l5)	4	15	25	p	compact	20.0	pas
12	audi	a4 quattro	2.8	1999	6	manual(m5)	4	17	25	p	compact	21.0	pas
13	audi	a4 quattro	3.1	2008	6	auto(s6)	4	17	25	p	compact	21.0	pas
14	audi	a4 quattro	3.1	2008	6	manual(m6)	4	15	25	p	compact	20.0	pas
15	audi	a6 quattro	2.8	1999	6	auto(l5)	4	15	24	p	midsize	19.5	fa
16	audi	a6 quattro	3.1	2008	6	auto(s6)	4	17	25	p	midsize	21.0	pas
17	audi	a6 quattro	4.2	2008	8	auto(s6)	4	16	23	p	midsize	19.5	fa
18	chevrolet	c1500 suburban 2wd	5.3	2008	8	auto(l4)	r	14	20	r	suv	17.0	fa
19	chevrolet	c1500 suburban 2wd	5.3	2008	8	auto(l4)	r	11	15	e	suv	13.0	fa

In [8]: ## [실습] 조건문을 활용해 파생변수 만들기

3. 빈도표로 합격 판정 자동차 수

mpg['test'].value_counts() # 값별 빈도수 출력; 결과는 Series형

Out[8]:

```
test
pass ... 128
fail ... 106
Name: count, dtype: int64
```

In [15]: ## [실습] 조건문을 활용해 파생변수 만들기

4. 막대 그래프로 빈도 표현하기

```
count_test = mpg['test'].value_counts() # 연비 합격 빈도표를 변수에 할당
type(count_test) # 결과는 Series형
count_test
```

```
Out[15]: test
          pass ... 128
          fail ... 106
          Name: count, dtype: int64
```

```
In [10]: # Series를 DataFrame으로 바꾸기 : .to_frame('col_name')
          mpg['test'].value_counts().to_frame('cnt')
```

```
Out[10]:      cnt
```

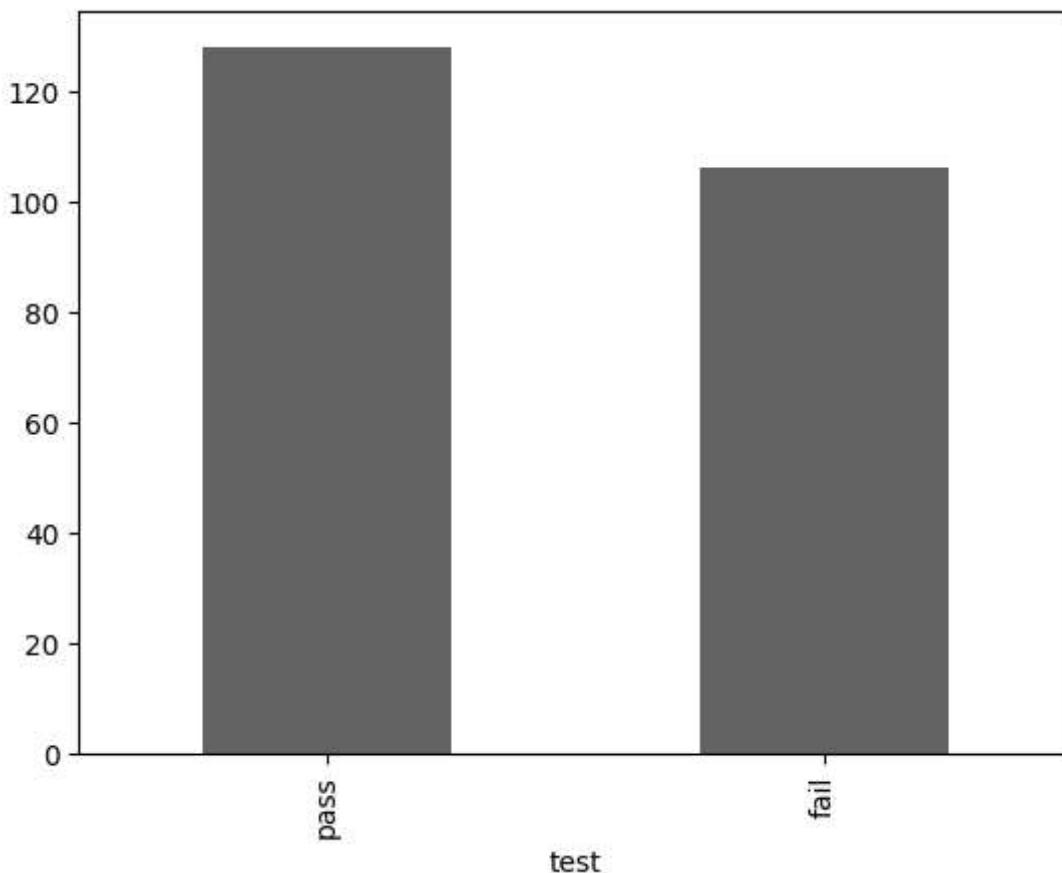
test
pass 128
fail 106

막대 그래프 그리기

```
> .bar()
```

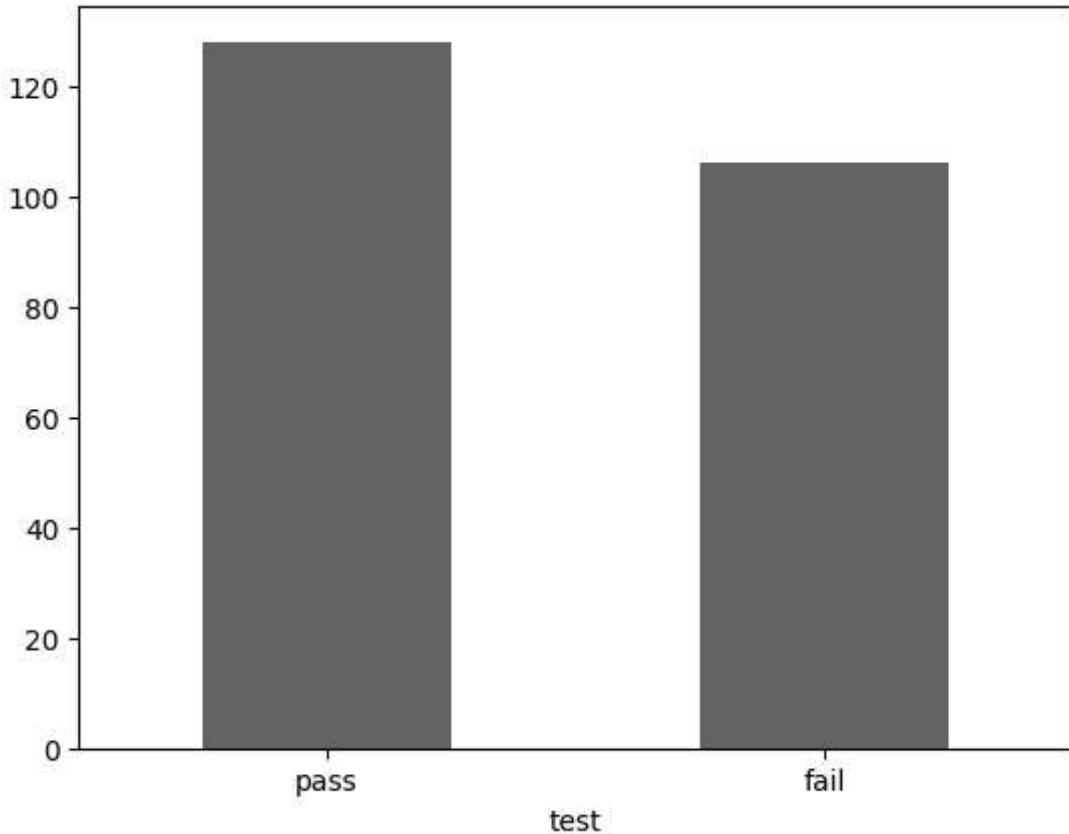
```
In [16]: count_test.plot.bar() # plot.bar()는 Series 자료형에서 제공되는 메서드
```

```
Out[16]: <Axes: xlabel='test'>
```



```
In [17]: ## 다양한 plot.bar() 매개변수 사용
          count_test.plot.bar(rot = 0) # rot = 0
```

```
Out[17]: <Axes: xlabel='test'>
```



▶ Pandas 라이브러리의 'Series' 객체에서 'plot.bar()' 메서드의 매개변수 확인

```
> import pandas as pd  
> bar_plot_params = pd.Series.plot.bar.__doc__  
> print(bar_plot_params)
```

In [18]: # plot.bar() 메서드의 매개변수 확인

```
import pandas as pd  
bar_plot_params = pd.Series.plot.bar.__doc__  
print(bar_plot_params)
```

Vertical bar plot.

A bar plot is a plot that presents categorical data with rectangular bars with lengths proportional to the values that they represent. A bar plot shows comparisons among discrete categories. One axis of the plot shows the specific categories being compared, and the other axis represents a measured value.

Parameters

x : label or position, optional
.... Allows plotting of one column versus another. If not specified,
.... the index of the DataFrame is used.

y : label or position, optional
.... Allows plotting of one column versus another. If not specified,
.... all numerical columns are used.

color : str, array-like, or dict, optional
.... The color for each of the DataFrame's columns. Possible values are:

.... - A single color string referred to by name, RGB or RGBA code,
..... for instance 'red' or '#a98d19'.

.... - A sequence of color strings referred to by name, RGB or RGBA
..... code, which will be used for each column recursively. For
..... instance ['green','yellow'] each column's bar will be filled in
..... green or yellow, alternatively. If there is only a single column to
..... be plotted, then only the first color from the color list will be
..... used.

.... - A dict of the form {column name : color}, so that each column will be
..... colored accordingly. For example, if your columns are called 'a' and
..... 'b', then passing {'a': 'green', 'b': 'red'} will color bars for
..... column 'a' in green and bars for column 'b' in red.

**kwargs

.... Additional keyword arguments are documented in
.... :meth:`DataFrame.plot`.

Returns

matplotlib.axes.Axes or np.ndarray of them
.... An ndarray is returned with one :class:`matplotlib.axes.Axes`
.... per column when ``subplots=True``.

See Also

.... DataFrame.plot.bart : Horizontal bar plot.
.... DataFrame.plot : Make plots of a DataFrame.
.... matplotlib.pyplot.bar : Make a bar plot with matplotlib.

Examples

.... Basic plot.

```
.... .. plot::  
....     :context: close-figs  
  
....     >>> df = pd.DataFrame({'lab':['A', 'B', 'C'], 'val':[10, 30, 20]})  
....     >>> ax = df.plot.bar(x='lab', y='val', rot=0)  
  
.... Plot a whole dataframe to a bar plot. Each column is assigned a  
.... distinct color, and each row is nested in a group along the  
.... horizontal axis.
```

```

..... .. plot::
..... :context: close-figs

..... >>> speed = [0.1, 17.5, 40, 48, 52, 69, 88]
..... >>> lifespan = [2, 8, 70, 1.5, 25, 12, 28]
..... >>> index = ['snail', 'pig', 'elephant',
.....             'rabbit', 'giraffe', 'coyote', 'horse']
..... >>> df = pd.DataFrame({'speed': speed,
.....                      'lifespan': lifespan}, index=index)
..... >>> ax = df.plot.bar(rot=0)

..... Plot stacked bar charts for the DataFrame

..... .. plot::
..... :context: close-figs

..... >>> ax = df.plot.bar(stacked=True)

..... Instead of nesting, the figure can be split by column with
..... ``subplots=True``. In this case, a :class:`numpy.ndarray` of
..... :class:`matplotlib.axes.Axes` are returned.

..... .. plot::
..... :context: close-figs

..... >>> axes = df.plot.bar(rot=0, subplots=True)
..... >>> axes[1].legend(loc=2) # doctest: +SKIP

..... If you don't like the default colours, you can specify how you'd
..... like each column to be colored.

..... .. plot::
..... :context: close-figs

..... >>> axes = df.plot.bar(
.....     rot=0, subplots=True, color={"speed": "red", "lifespan": "green"})
..... ...
..... >>> axes[1].legend(loc=2) # doctest: +SKIP

..... Plot a single column.

..... .. plot::
..... :context: close-figs

..... >>> ax = df.plot.bar(y='speed', rot=0)

..... Plot only selected categories for the DataFrame.

..... .. plot::
..... :context: close-figs

..... >>> ax = df.plot.bar(x='lifespan', rot=0)

```

[실습-3] 중첩 조건문 활용하기

1. 'grade' 파생변수 추가

>> 'grade'는 'total' ≥ 30 이면 'A' 등급, 아니고 'total' ≥ 20 이면 'B' 등급,
나머지는 'C' 등급

2. 등급 빈도표 만들기

3. 빈도표 막대 그래프로 그리기

```
In [11]: ## [실습] 조건문을 활용해 파생변수 만들기
# 'total' >=30 이면 'A' 등급, 아니고 'total' >=20 이면 'B' 등급, 나머지는 'C' 등급
import numpy as np #수치연산 지원 패키지 numpy
x = np.where(mpg['total'] >= 30, 'A', np.where(mpg['total'] >=20, 'B', 'C')) #np.where(조건식, True 시 실행, False 시 실행)
x
```

```
Out[11]: array(['B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'C', 'B', 'C', 'C', 'C', 'C', 'C', 'C', 'B', 'C',
       'B', 'C', 'C', 'C', 'C', 'B', 'B', 'B', 'B', 'B', 'B',
       'C', 'C', 'B', 'B', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C',
       'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C'],
      dtype='|<U1')
```

```
In [12]: ## [실습] 조건문을 활용해 파생변수 만들기
#### 1. 'grade' 파생변수 추가
#### >> 'grade'는 'total' >=30 이면 'A' 등급, 아니고 'total' >=20 이면 'B' 등급, 나머지는 'C' 등급
import numpy as np #수치연산 지원 패키지 numpy
mpg['grade'] = np.where(mpg['total'] >= 30, 'A', np.where(mpg['total'] >=20, 'B', 'C')) #np.where(조건식, True 시 실행, False 시 실행)
mpg.head()
```

```
Out[12]:   manufacturer model  displ  year   cyl  trans  drv   cty   hwy   fl category total test  c
0        audi    a4     1.8  1999     4  auto(l5)   f   18   29   p  compact  23.5  pass
1        audi    a4     1.8  1999     4 manual(m5)   f   21   29   p  compact  25.0  pass
2        audi    a4     2.0  2008     4 manual(m6)   f   20   31   p  compact  25.5  pass
3        audi    a4     2.0  2008     4  auto(av)   f   21   30   p  compact  25.5  pass
4        audi    a4     2.8  1999     6  auto(l5)   f   16   26   p  compact  21.0  pass
```

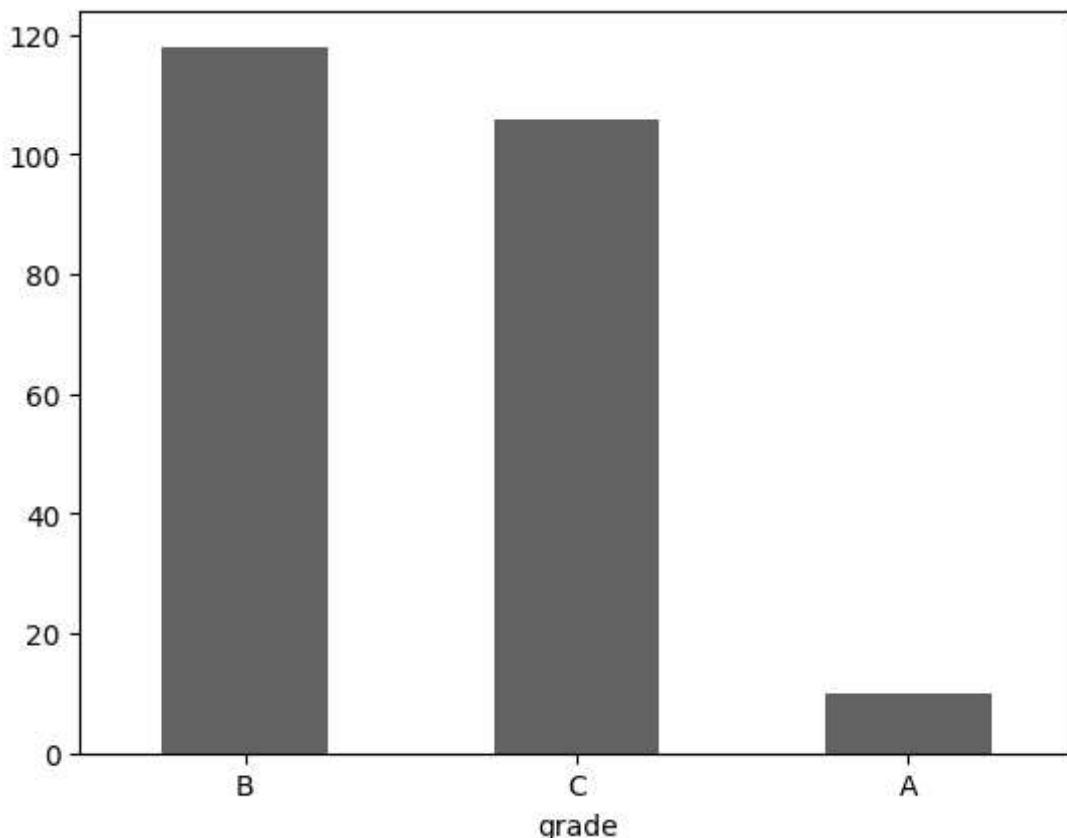
<> 데이터프레임 개별 항목 값 조작

```
In [18]: ## [실습] 조건문을 활용해 파생변수 만들기
#### 2. 등급 빈도표 만들기
count_grade = mpg['grade'].value_counts() # 등급 빈도표 만들기 (default는 빈도수 높은 순으로 정렬)
```

```
Out[18]: grade  
B ... 118  
C ... 106  
A ... 10  
Name: count, dtype: int64
```

```
In [19]: ## 빈도 막대 그래프 그리기  
count_grade.plot.bar(rot = 0) # 등급 빈도 막대 그래프 만들기
```

```
Out[19]: <Axes: xlabel='grade'>
```



```
In [20]: count_grade = mpg['grade'].value_counts().sort_index() # 등급 빈도표 만들기(index는  
count_grade
```

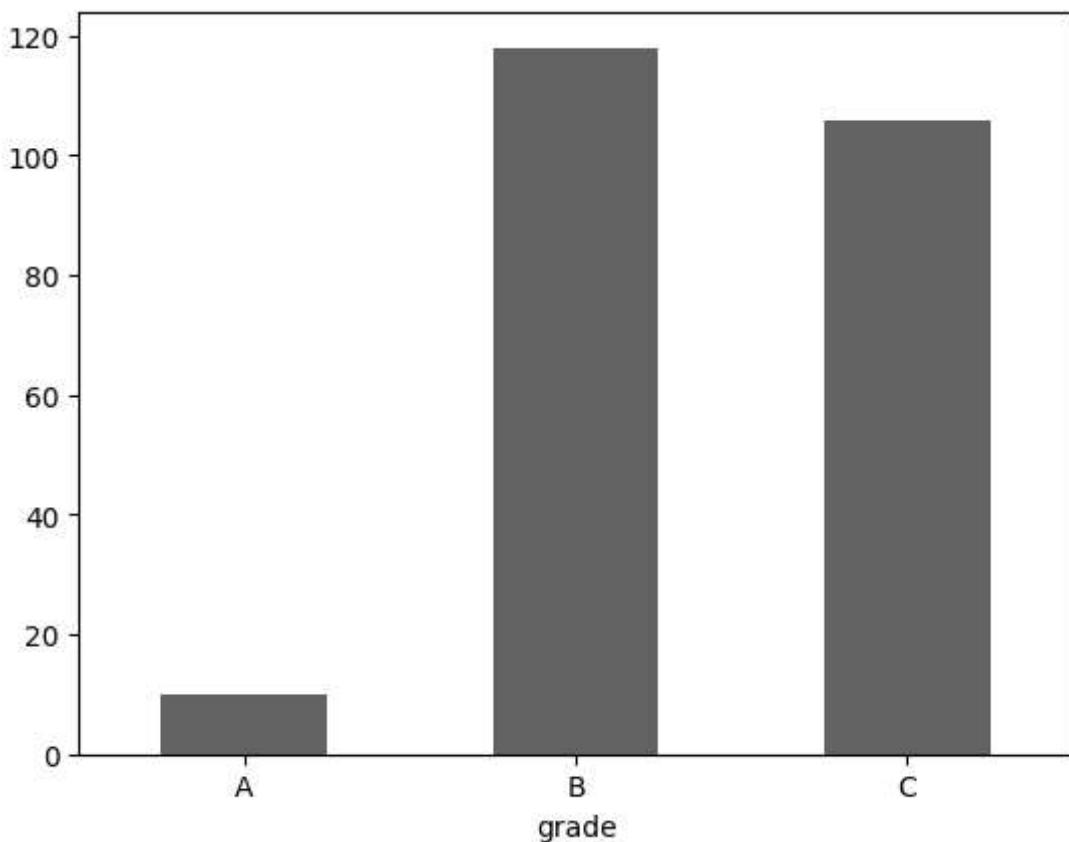
```
Out[20]: grade  
A ... 10  
B ... 118  
C ... 106  
Name: count, dtype: int64
```

```
In [21]: count_grade = mpg['grade'].value_counts()  
new_order = ['C', 'B', 'A'] #List형으로 정렬 순서 지정  
count_grade = count_grade[new_order] # 등급 빈도표 만들기(지정한 순 정렬 적용)  
count_grade
```

```
Out[21]: grade  
C ... 106  
B ... 118  
A ... 10  
Name: count, dtype: int64
```

```
In [22]: ## [실습] 조건문을 활용해 파생변수 만들기  
#### 3. 빈도표 막대 그래프로 그리기  
count_grade = mpg['grade'].value_counts() # 등급 빈도표 만들기 (default는 빈도수 높  
new_order = ['A', 'B', 'C'] #List형으로 정렬 순서 지정  
count_grade = count_grade[new_order] # 등급 빈도표 만들기(지정한 순 정렬 적용)  
count_grade.plot.bar(rot = 0) # 등급 빈도 막대 그래프 만들기
```

```
Out[22]: <Axes: xlabel='grade'>
```



```
In [28]: ## [실습] 조건문을 활용해 파생변수 만들기
##### 3. 빈도표 막대 그래프로 그리기
##### >> 'grade' 순으로 표현
##### >> color=['red', 'green', 'blue'] 순으로 사용
count_grade = mpg['grade'].value_counts().sort_index() # 등급 빈도표 만들기(index는
count_grade.plot.bar(rot = 0) # count_grade는 Series 자료형

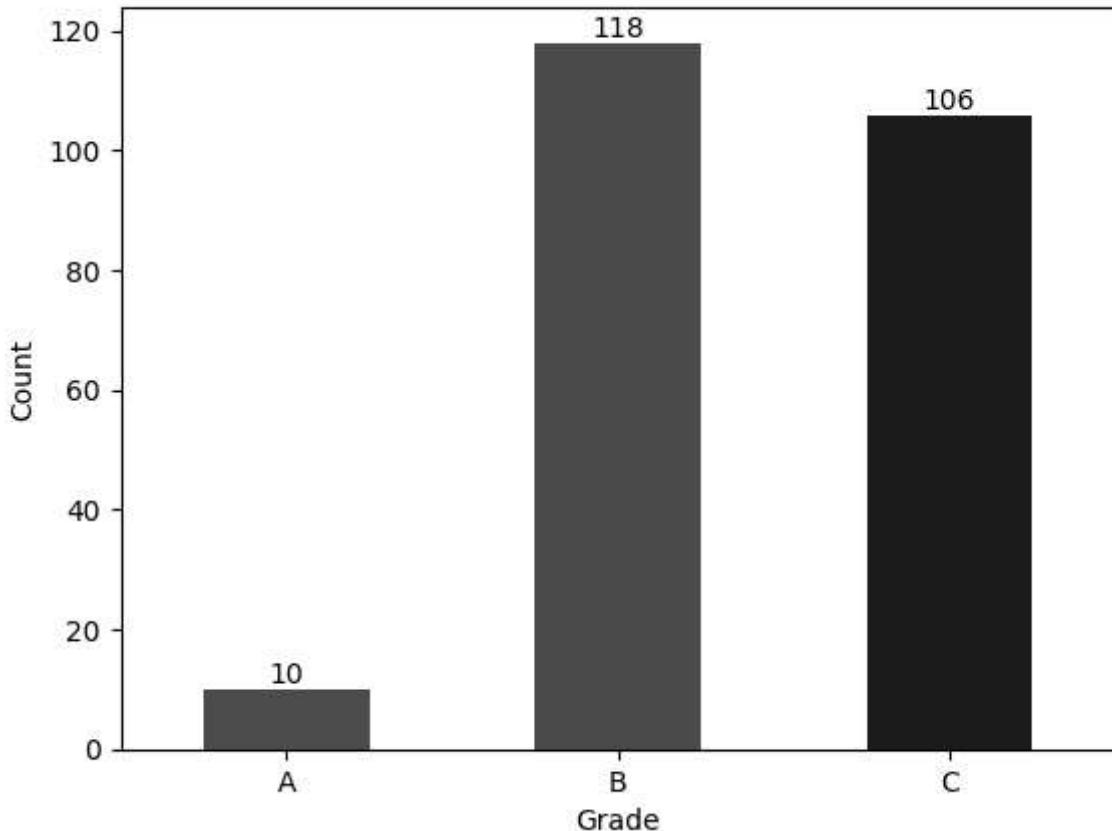
# 막대 그래프 그리기
ax = count_grade.plot.bar(rot = 0, color=['red', 'green', 'blue'])

# 막대 위에 값 표기
for idx, value in enumerate(count_grade):
    ax.text(idx, value + 1, str(value), ha='center')

# 그래프에 제목과 레이블 추가
ax.set_title('Bar Chart of Grade Count')
ax.set_xlabel('Grade')
ax.set_ylabel('Count')

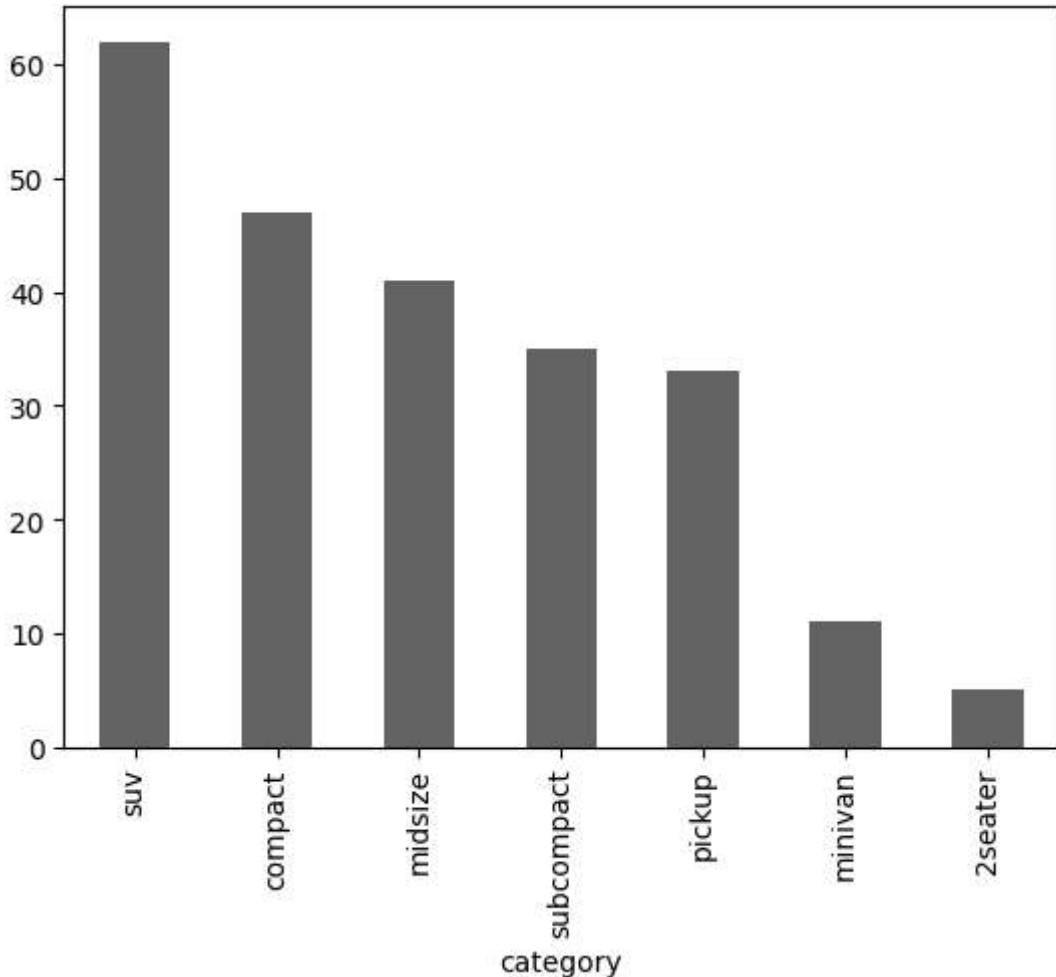
# 그래프 표시
import matplotlib.pyplot as plt
plt.show()
```

Bar Chart of Grade Count



```
In [28]: ## [실습] 조건문을 활용해 파생변수 만들기  
count_cat = mpg['category'].value_counts() # 등급 빈도표 만들기  
count_cat  
count_cat.plot.bar(rot = 90) # rot는 반시계방향 각도
```

```
Out[28]: <Axes: xlabel='category'>
```



In []:

[응용 실습] excel > csv > Data Frame > Data Frame 파생변수 추가 > 그래프 그리기

1. excel > csv 파일 형식 변환

> LMS에 제시된 엑셀 파일 Test_result.xlsx를 CSV 타입 파일로 변환

2. csv > Data Frame 데이터 변환

> 변환한 CSV 파일을 읽어 데이터 프레임으로 변환

> 변환된 데이터 프레임의 정보 확인

> Index 컬럼 'Unnamed: 0' 제거

3. Data Frame에 'Total' 파생변수를 추가

> 'Total'은 총점으로 (Attendance + Homework + Midterm + Final)

4. Data Frame에 'grade' 파생변수를 추가

> 'Grade'는 총점에 대한 등급 : "A+" : 100-95, "A" : 94-90, "B+" : 89-85, "B" : 84-80, "C+" : 79-75, "C" : 74-70, "D+" : 69-65, "D" : 64-60, "F" : 59-0

5. 파생 변수로 그래프 그리기

- > 'Grade'별 인원 수 분포 구하고, 확인하기
- > 'Grade'별 인원 수 막대 그래프 그리기
- > x-축을 높은 등급 순으로 그리기
- >>[도전] 막대에 빈도 값, 표 타이틀, 축 레이블 표시하기

6. 데이터 프레임을 **Test_result2.csv** 파일로 변환

- > **index**는 제외하고 변환

In []:

정리하기

```
In [1]: # 1. 패키지 로드
import pandas as pd
import numpy as np

# 2. 데이터 불러오기
mpg = pd.read_csv('mpg.csv')

# 3. 데이터 파악하기
mpg.head()      # 데이터 앞부분
mpg.tail()      # 데이터 뒷부분
mpg.shape       # 행, 열 수
mpg.info()       # 속성
mpg.describe()   # 요약 통계량

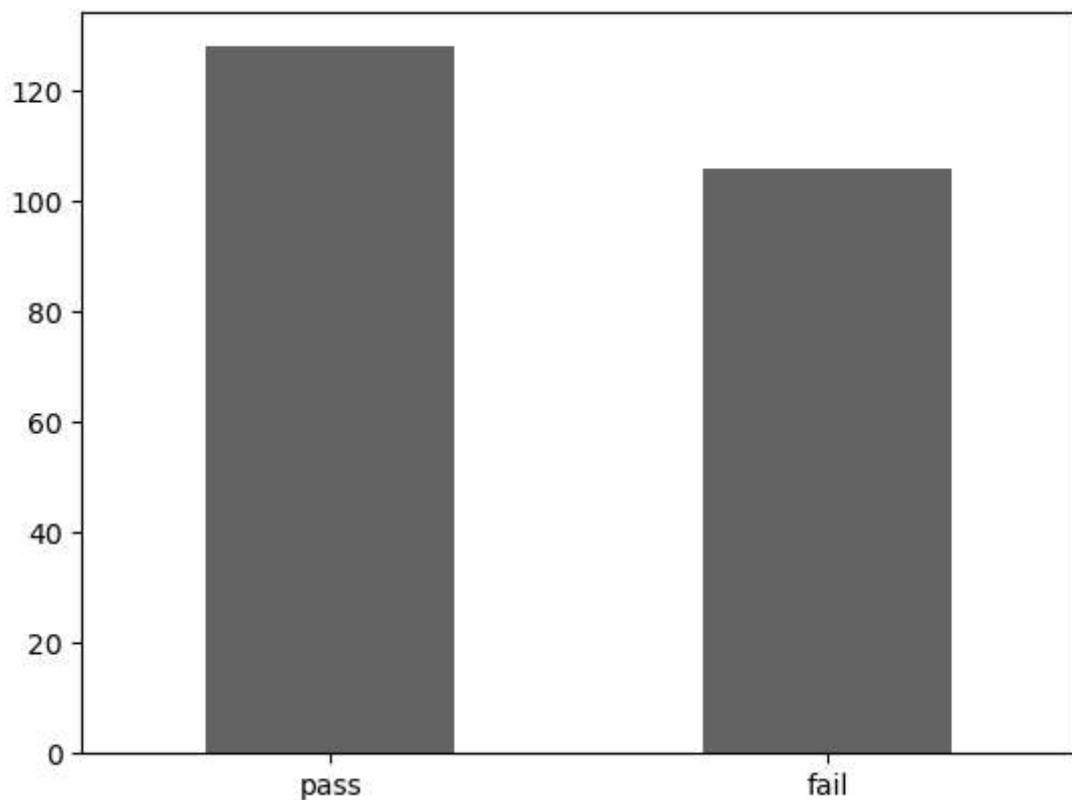
# 4. 변수명 바꾸기
mpg = mpg.rename(columns = {'manufacturer' : 'company'})

# 5. 파생변수 만들기
mpg['total'] = (mpg['cty'] + mpg['hwy'])/2          # 변수 조합
mpg['test'] = np.where(mpg['total'] >= 20, 'pass', 'fail') # 조건문 활용

# 6. 빈도 확인하기
count_test = mpg['test'].value_counts() # 빈도표 만들기
count_test.plot.bar(rot = 0)           # 빈도 막대 그래프 만들기
```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 234 entries, 0 to 233
Data columns (total 11 columns):
 # ... Column ... Non-Null Count Dtype ...
--- ... --- ...
 0 ... manufacturer ... 234 non-null ... object ...
 1 ... model ... 234 non-null ... object ...
 2 ... displ ... 234 non-null ... float64 ...
 3 ... year ... 234 non-null ... int64 ...
 4 ... cyl ... 234 non-null ... int64 ...
 5 ... trans ... 234 non-null ... object ...
 6 ... drv ... 234 non-null ... object ...
 7 ... cty ... 234 non-null ... int64 ...
 8 ... hwy ... 234 non-null ... int64 ...
 9 ... fl ... 234 non-null ... object ...
 10 ... category ... 234 non-null ... object ...
dtypes: float64(1), int64(4), object(6)
memory usage: 20.2+ KB

Out[1]: <AxesSubplot:>



In []: