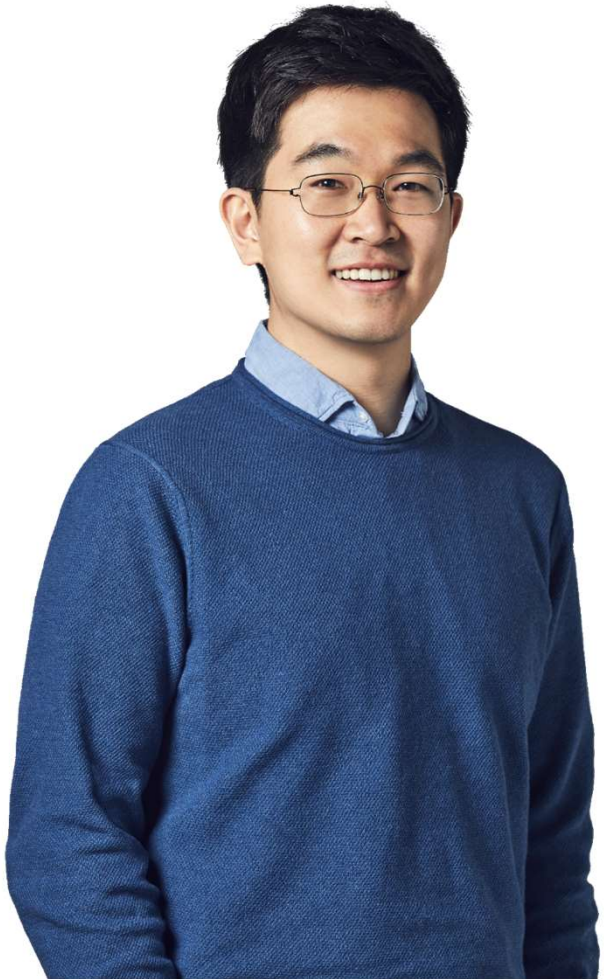
Python 데이터 전처리 기초

기초 문법부터 데이터 결합 및 파생 변수 생성까지

임 경 덕 x fastcampus



임 경 덕

데이터 분석가 | 강사

열심히 달려온 주요 이력

지식과 경험, 역량과 아이디어로
데이터 기술을 전달하고 있습니다.

통계학
학사 | 석사

삼성카드
회원마케팅

직무 교육
R | Python

분석 프로젝트

창업
컨설팅 | 컨텐츠

- L유통 : 마트/백화점 상품 추천
- H손보 : 유지 예측, 고객 등급화,
  FDS 등 모형 개발 코칭
- B카드 : 마이데이터 활용 상품 추천

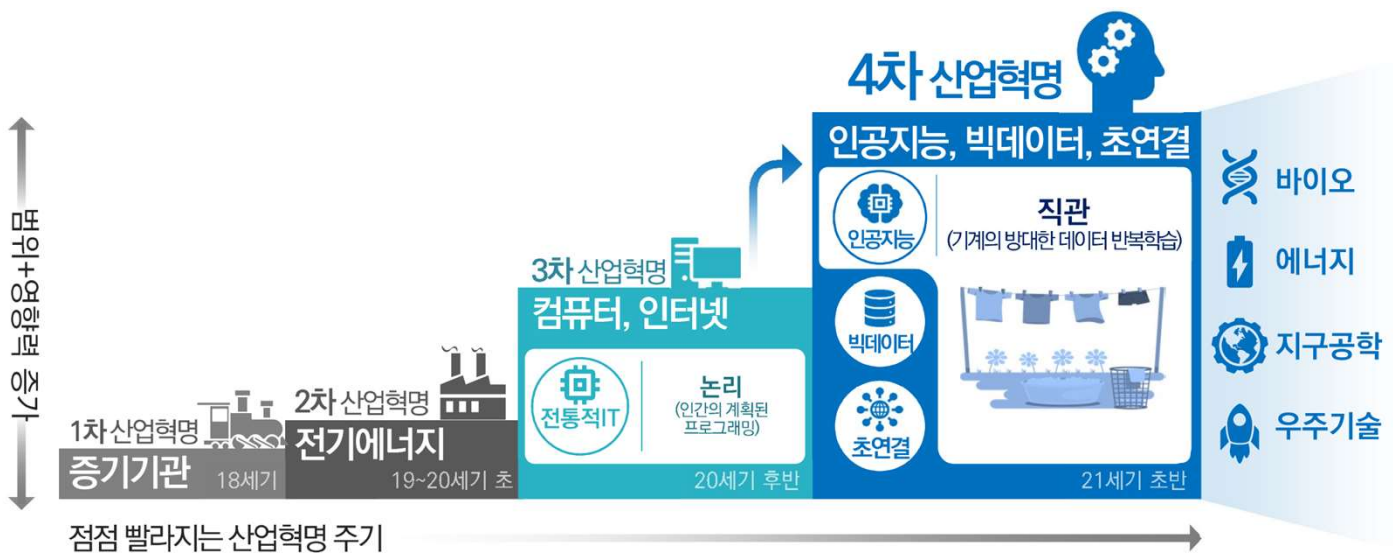
Contents
목차

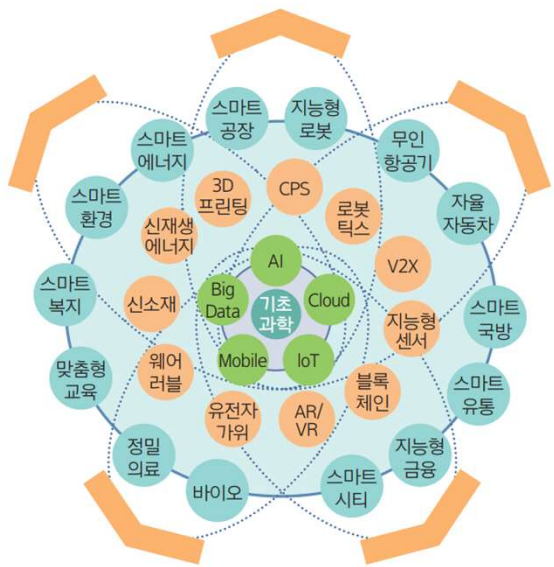
Python 데이터 처리 기초

1. 데이터 분석의 이해

달라진 데이터의 위상

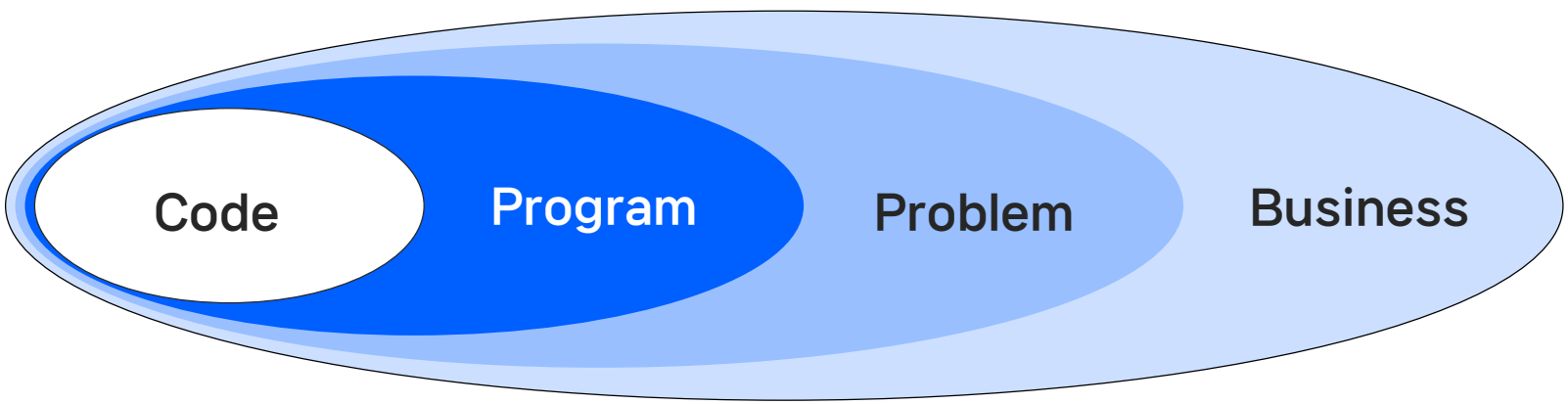
4차 산업혁명과 데이터





* 출처 : 대통령직속 4차산업혁명위원회

코딩과 프로그래밍, 그리고 비즈니스



1. 비즈니스와 업무에 대한 이해
2. 문제 정의, 문제 해결 방법 및 절차 제시
3. 적절한 도구와 방법과 절차로 문제 해결 Programming

Coding

목적에 따라 배워야 할 것이 다릅니다.

분석	vs	개발
현황 파악 / 요인 분석 AI	Dashboard RPA	IoT AI

요리와 닮은 데이터 분석 과정

재료

데이터

저장

적재/저장

손질

추출/전처리

조리

분석

음식

리포트/모형

분석 목표 설정의 중요성

"목표?"

"목표!"

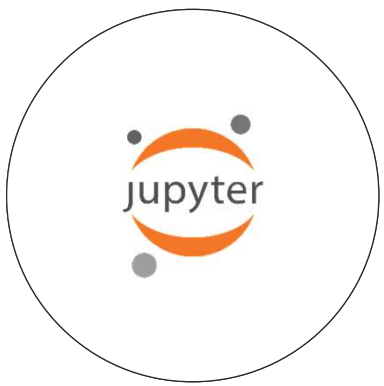
Python 분석 환경 설정

Python

데이터 집계와 시각화 뿐만 아니라
알고리즘 활용 등 가능

Anaconda

Python을 포함한 다양한 도구를
통합적으로 관리하는 프로그램



Jupyter

인터랙티브 Python 코딩을 지원하는
대표적인 웹 서비스

Python 데이터 처리 기초

2. Python 기초 문법

Python 기초 문법 – 타입(type)

데이터 타입(자료형, 형식)

객체를 표현하고 저장하는 형식
type( ) 함수를 활용해 데이터 타입의 확인 가능
각 형식에 따라 활용가능한 메서드(method, 함수)가 다름

정수(int): integer, 가장 대표적인 숫자

실수(float): 소수점을 포함한 숫자

문자열(str): string, 일반적인 글자, 큰따옴표("")나 작은따옴표('')를 활용

불(bool): 비교 연산 등의 결과로 True나 False 둘 중 하나의 값을 가짐

Python 기초 문법 – 여러 값의 묶음

list, tuple, dictionary

여러 개 값을 하나로 묶어서 활용 가능하며 상황에 따라 적절한 방법을 선택

리스트(list)
복수의 숫자나 문자열을 한데 모은 것
[ ]나 list( ) 함수를 활용해서 생성
목록의 추가, 수정, 삭제가 자유로우며 일반적인 데이터 분석에서 주로 활용

튜플(tuple)
리스트와 동일하게 복수의 값을 묶음
( )를 활용해서 생성, 변경 불가능

딕셔너리(dictionary)
key와 value를 :으로 결합한 쌍 여러 개를 묶음
json 형식과 비슷한 형식으로 값을 활용

Python 기초 문법 – 제어문

if 조건문과 for와 while 반복문

조건에 따라 수행할 작업 내용을 다르게 지정하거나 반복 작업 수행에 활용

if를 활용한 조건문
조건에 따라 달라지는 작업 내용을 설정
if, elif, else 를 활용하여 복수의 조건을 활용 가능
콜론(:)과 들여쓰기를 활용해서 실행할 코드블록을 지정

for와 while을 활용한 반복(loop)
for : 특정한 값 목록이나 범위를 활용한 반복
while : 특정한 조건을 만족하는 동안 계속해서 반복

Python 기초 문법 – 라이브러리 설치 및 관리

!pip

라이브러리 설치 및 업데이트, 제거 등에 활용

Python

```
# 라이브러리 설치
!pip install pandas matplotlib seaborn

# 라이브러리 제거
!pip uninstall pandas

# 라이브러리 목록 확인
!pip list
```

Python 기초 문법 – 라이브러리 불러오기

import, from, as

라이브러리를 불러올 때 impor를 활용
from으로 특정한 모듈이나 함수만 불러올 수 있고, as로 별명(alias) 지정 가능

Python

```
# 라이브러리 불러오기
import pandas

# 라이브러리 별명 지정하고 불러오기
import pandas as pd

# 라이브러리에서 특정 함수만 불러오기
from pandas import read_csv
```

Python 기초 문법 – 수치형과 문자열

1, '가'

수치형(int, float) 객체는 그대로 입력하고 사칙연산자 등 활용 가능
문자열(str) 객체는 따옴표(‘’, “”)를 활용하여 입력

Python

```
# 사칙연산
1+2*3/4

# 문자열의 입력
‘삼성전자’
“삼성전자”

# +를 활용한 문자열 결합
‘Python ’ + ‘데이터 전처리 기초’
```

Python 기초 문법 – 할당과 출력

=, print()

=을 활용하여 다양한 형식의 객체를 저장 가능
print()를 활용하여 jupyter 셀 하단 혹은 콘솔창에 값 등을 출력 가능

Python

```
# =을 활용한 할당
a=10
a

# print()를 활용한 출력
print(a)

# f-string을 활용한 출력
print(f'저장된 a의 값은 {a}입니다.')
```

Python 기초 문법 – 비교 연산과 bool 형식의 이해

True, False

숫자와 등호, 부등호 등을 활용하여 비교 연산 가능
다양한 연산자, 함수를 활용하여 여부를 확인하고 True, False의 값을 갖는 bool 형식 생성

Python

```
# 부등호를 활용한 비교
x=10
x>=9

# ==, != 활용 일치, 불일치 여부 확인
x==10
x!=10

# type()을 활용한 형식 확인
type(True)
```

Python 기초 문법 – 리스트의 생성과 활용

[], list(), append(), extend()

대괄호 []나 list()를 활용하여 리스트(list)를 생성 가능
append()를 활용하여 요소를 추가하거나 extend()로 또다른 리스트를 이어 붙이기 가능

Python

```
# []를 활용한 리스트 생성
x=[1,3,5,7,9]
x

# append()를 활용한 요소 추가
x.append(11)
x

# extend()를 활용한 리스트 결합
x.extend([13, 15])
x
```

Python 기초 문법 – index와 슬라이스의 활용

[], :

[]를 활용하여 생성된 리스트 등에서 일부를 선택가능
:를 활용하여 연속적인 일정 범위의 값을 선택 가능

Python

```
# []를 활용한 리스트 생성과 부분 선택
x=[1,3,5,7,9]
x[0]

# :의 활용
x[2:4]
x[:3]
```

Python 기초 문법 – 튜플의 활용

()

()를 활용하여 튜플(tuple) 생성 가능
리스트와 달리 객체 추가, 수정이 불가능

Python

```
# ()를 활용한 튜플 생성(생략 가능)
a=(10, 20, 30)
a=10, 20, 30
a

# []를 활용한 부분 선택
a[0]

# 튜플 분할 저장
x, y, z = a
```

Python 기초 문법 – 딕셔너리의 활용

{}

key와 value로 구성된 값의 쌍을 저장 가능

Python

```
# {}를 활용한 딕셔너리 생성
a={'name':'PARK', 'age':30}
a

# []를 활용한 값 추출
a['name']

# .keys()를 활용한 key 목록 확인
a.keys()
```

Python 기초 문법 – 조건문

if, else, elif

bool 형식의 True, False의 값을 갖는 조건을 활용하여 작업 내용을 지정 가능

Python

```
# if를 활용한 조건문
if x>0:
  print('x는 0보다 크다')

# if, elif와 else를 활용한 조건문
if x>=90:
  print('x는 90이상')
elif x>=60:
  print('x는 60이상 90미만')
else x>=60:
  print('x는 60미만')
```

Python 기초 문법 – 반복문

for, while

for 는 반복 대상을 지정하여 특정만 작업 내용을 반복
while은 조건문이 True인 동안 계속해서 반복하거나 break 등을 활용해 반복 종료

Python

```
# for를 활용한 반복문
for x in [1,2,3]:
    print(f'x는 2보다 크다: {x>2}')

# while을 활용한 반복문
x=0
while x<=10:
    print('아직 x는 10보다 작다')
    x+=1
```

Python 데이터 처리 기초

3. 데이터 처리 기초

데이터 처리 – pandas의 활용

DataFrame 형식의 이해

Python에서 정형 데이터를 처리, 집계할 때 주로 pandas 라이브러리를 활용
pandas에서 데이터는 DataFrame 형식으로 저장되며, DataFrame은 Series의 결합 형태

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

데이터 불러오기 – csv 파일 불러오기

pandas.read_csv()

pandas 라이브러리의 read_csv()를 활용하여 csv 파일을 불러오기 가능
불러온 데이터는 DataFrame 형식으로 저장되고 다양한 메서드 활용 가능

Python

```
# csv 파일 불러오기
df_ = pandas.read_csv('경로/파일이름.csv')
df_

# 간단히 데이터 살펴보기
df_.shape       # 관측치, 변수 개수 확인
df_.head(n=)    # 앞 n개 관측치 확인
df_.columns     # 변수 이름 목록 확인
df_.dtypes      # 변수 형식 확인
```

데이터 불러오기 – xlsx 파일 불러오기

pandas.read_excel()

pandas 라이브러리의 read_excel()를 활용하여 Excel의 xlsx 파일 불러오기 가능
sheet 이름 혹은 번호를 지정 가능

Python

```
# xlsx 파일 불러오기
df_ = pandas.read_excel('경로/파일이름.xslx', engine='openpyxl')
df_

# 두번째 시트 불러오기
df_ = pandas.read_excel('경로/파일이름.xslx', sheet_name=1, engine='openpyxl')
```

데이터 결합 – 복수 데이터의 행/열 결합

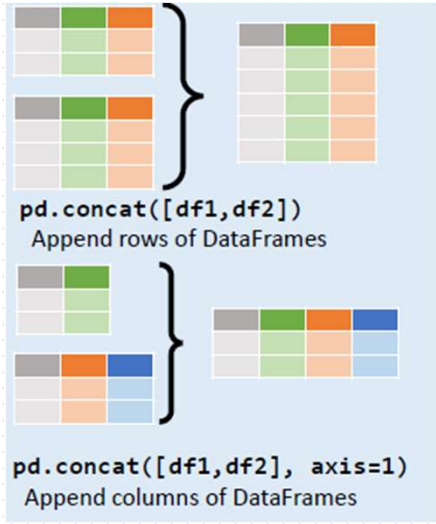
pandas.concat()

구조와 columns 등의 구성이 동일한 복수의 데이터를 행 결합 가능
axis=1 옵션을 활용하여 index가 동일한 복수 데이터를 열 결합 가능

Python

```
# 동일한 구조의 복수 DataFrame 행 결합(아래로 이어 붙이기)
pandas.concat([df1, df2, df3])

# 동일한 index를 갖는 복수 DataFrame 열 결합
pandas.concat([df1, df2, df3], axis=1)
```



데이터 결합 – key 활용 결합

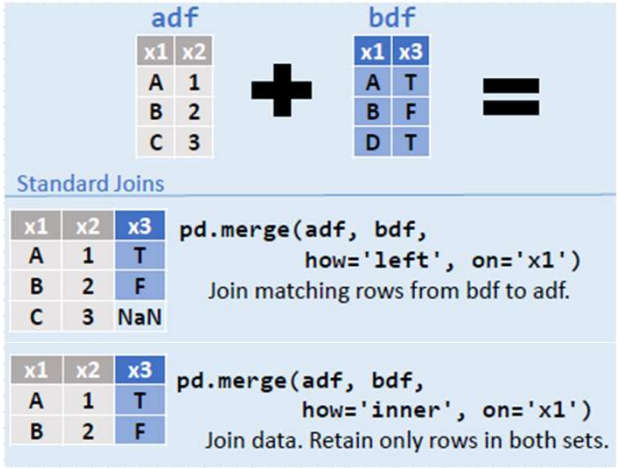
pandas.merge()

Excel의 VLOOKUP(), SQL의 JOIN과 같이 Key 변수를 활용한 데이터 결합
on으로 결합 기준 key 변수를 지정하고, how로 결합 방법을 지정

Python

```
# 두 데이터의 결합(inner join)
pd.merge(df_left, df_right, how='inner', on='key_name')

# 복수 key 변수를 갖는 두 데이터의 결합(left join)
pd.merge(df_left, df_right, how='left', on=['key1','key2'])
```



부분 선택 – 변수 선택

., [], select_dtypes()

.을 활용하여 DataFrame에서 하나의 변수를 선택 가능하나 변수 이름에 공백, 특수 문자가 포함된 경우 활용 불가능
[]에 변수 이름을 넣어 변수 선택 가능하며 select_dtypes() 등 다양한 함수 활용 가능

Python

```
# 이름이 x1인 변수 선택
df_.x1
df_['x1']

# 복수 변수 선택
df_[['x1', 'x2', 'x3']]

# 수치형 변수 전체 선택
df_.select_dtypes(include='number')
```

부분 선택 – 변수 이름 및 패턴 활용 선택

filter()

filter()와 정규표현식을 활용하면 변수 이름의 패턴을 활용한 변수 선택이 가능합니다.

Python

```
# 이름에 “score가 들어간” 변수 선택
df_.filter(regex=‘score’)

# 이름이 “sp로 시작하는” 변수 선택
df.filter(regex=‘^sp’)

# 이름이 “cd로 끝나는” 변수 선택
df.filter(regex=‘cd$’)
```

부분 선택 – 관측치, 변수 동시 선택

loc[], iloc[]

loc[]과 index, columns를 활용하여 관측치와 변수를 동시 선택 가능
.iloc[]과 관측치, 변수 순서를 활용하여 데이터 부분 선택 가능

Python

```
# index가 0~10인 관측치의 변수 'x1', 'x2', 'x3' 선택
df_.loc[0:10, ['x1','x2','x3']]

# 0~9번째 관측치의 4~6번째 변수 선택
df_.iloc[0:10, 3:6]
```

부분 선택 – 조건 일치 관측치 선택

True/False, &, |

변수 선택과 다양한 조건문을 활용하여 조건과 일치하는 일부 관측치 선택 가능
and, or를 활용하여 복수의 조건의 논리 연산 활용 가능

Python

```
# x1이 10이상인 관측치 선택
df_[df_['x1'] >= 10]

# x1이 10이상이고 x2는 'A' 혹은 'Z'의 값을 갖는 관측치 선택
df_[(df_['x1'] >= 10) & (df_['x1'].isin(['A', 'Z'])]
```

부분 선택 – 상위, 하위 관측치 선택

nlargest(), nsmallest()

특정 변수를 기준으로 상위 혹은 하위 n개 관측치 선택 가능

Python

```
# x1 기준 상위 10개 관측치 선택
df_.nlargest(10, 'x1')

# x1 기준 하위 10개 관측치 선택
df_.nsmallest(10, 'x1')
```

Python 데이터 처리 기초

4. 간단한 데이터 집계 및 시각화

집계값 계산 – 건수, 합계, 평균

count(), sum(), mean()

관심 대상 수치형 변수를 선택하고, count(), sum(), mean() 등의 함수를 활용하여 집계값 계산

Python

```
# x1의 건수, 합계, 평균 계산
df_['x1'].count()
df_['x1'].sum()
df_['x1'].mean()
```

집계값 계산 – 그룹별 집계

groupby()

groupby()를 활용하여 그룹 변수를 지정하고 그룹별 집계값 계산 가능

Python

```
# x2별 x1의 건수, 합계, 평균 계산
df_.groupby('x2')['x1'].count()
df_.groupby('x2')['x1'].sum()
df_.groupby('x2')['x1'].mean()
```

집계값 계산 – 피벗 테이블 생성

pandas.pivot_table()

pivot_table()을 활용하여 표형태로 집계값 계산 가능

Python

```
# x2, x3별 x1의 평균을 표형태로 계산
df_.pivot_table(index='x2', columns='x3', values='x1', aggfunc='mean')
```

그래프 작성 – 히스토그램과 막대그래프

seaborn.histplot(), seaborn.countplot()

일반적으로 matplotlib, seaborn 라이브러리를 활용
히스토그램, 막대그래프 등의 그래프를 생성하고 활용

Python

```
# x1의 히스토그램 생성
seaborn.histplot(data=df_, x='x1')

# x2의 막대그래프 생성
seaborn.countplot(data=df_, x='x2')
```

Python 데이터 처리 기초

5. 데이터 처리 심화

변수 처리 – 변수 이름 변경

rename()

rename()을 활용하여 기존 변수 이름을 새로운 이름으로 변경

Python

```
# ‘x1’의 이름을 ‘var1’으로 변경
df_.rename(columns={‘x1’:’var1’})
```

변수 처리 – 특정 변수 제거

drop()

drop()과 columns= 옵션으로 특정 변수 제거 가능

Python

```
# 변수 'x5', 'x6'을 제거
df_ = df_.drop(columns=['x5','x6'])
```

결측값 처리 – 결측치 제거 및 대체

dropna(), fillna()

결측값을 포함한 관측치를 dropna()로 제거 가능
fillna()를 활용하여 특정한 값으로 결측값을 대체

Python

```
# 하나라도 결측값이 있는 관측치 제거
df_.dropna()

# ‘x1’가 결측값인 관측치 제거
df_.dropna(subset=[‘x1’])

# 모든 변수의 결측값을 모두 0으로 대체
df_.fillna(value=0)

# 변수별 결측값 대체값을 지정 가능
df_.fillna(value={‘x1’:10, ‘x2’:‘NA’})
```

관측치 정렬 – 기준 변수 및 index 활용

sort_values(), sort_index(), reset_index()

sort_values()로 특정 변수 기준 관측치를 정렬 가능
sort_index()로 index 기준 관측치 정렬이 가능하고, reset_index()로 정렬 후 index 초기화 가능

Python

```
# 변수 'x1'의 내림차순으로 관측치 정렬
df_ = df_.sort_values('x1', ascending=False)

# DataFrame df_의 index를 초기화
df_ = df_.reset_index(drop=True)

# index 기준 관측치 정렬
df_ = df_.sort_index()
```

변수 형식 변환 – Series의 다양한 형식 확인

다양한 Series type(타입, 형식)

DataFrame은 여러 개 Series가 결합된 형태이고,
각 Series는 아래와 같이 다양한 형식 중 하나로 지정되어 있음
지정된 형식에 따라 활용 가능한 메서드의 차이가 있음

float:	실수(소수점을 포함한 숫자)
int:	정수(integer)
datetime:	날짜 혹은 날짜시간
bool:	불/불린(True 혹은 False)
category:	범주형
object:	문자형(string) 혹은 그 외

변수 형식 변환 – 날짜 등의 변수의 활용

astype(), pandas.to_datetime()

astype()을 활용하여 특정 변수의 형식을 변환 가능
문자열로 지정된 날짜의 경우 pandas의 to_datetime()을 활용하여 형식을 변환하고 활용

Python

```
# ‘x1’을 정수 형식으로 변환하여 업데이트
df_[‘x1’] = df_[‘x1’].astype(‘int’)

# ‘x4’를 날짜 형식으로 변환하고 업데이트
df_[‘x4’] = pandas.to_datetime(df_[‘x4’])

# 날짜 형식 ‘x4’에서 요일을 추출하고 파생변수 생성
df_[‘dow’] = df_[‘x4’].dt.weekday
```

변수 형식 변환 – 수치형 변수의 구간화

pandas.cut(), pandas.qcut()

적절한 구간값(breaks)을 활용하여 수치형 변수를 구간화, 범주화 가능
cut()으로 최솟값, 최솟값 사이 등간격으로 구간화하거나 qcut()으로 등비율로 구간화 가능

Python

```
# ‘x1’을 10개 등간격 구간으로 구간화
df_[‘x1_grp’] = pandas.cut(df_[‘x1’], bins=10)

# ‘x1’을 10개 등비율 구간으로 구간화
df_[‘x1_grp’] = pandas.qcut(df_[‘x1’], q=10)
```

파생변수 생성 – 이동, 누적, 순위

shift(), rank(), cumsum()

shift(), rank(), cumsum() 등을 groupby()와 함께 활용하여 그룹 파생변수 생성 가능

Python

```
# ‘x2’별 ‘x1’을 하나씩 아래로 밀어내기
df_[‘x1_prev’] = df_.groupby(‘x2’)[‘x1’].shift()

# ‘x2’별 ‘x1’을 하나씩 위로 당기기
df_[‘x1_next’] = df_.groupby(‘x2’)[‘x1’].shift(-1)

# ‘x2’별 ‘x1’의 순위 매기기
df_[‘x1_rank’] = df_.groupby(‘x2’)[‘x1’].rank(ascending=True, method='first’)

# ‘x2’별 ‘x1’의 누적합 계산하기
df_[‘x1_cumsum’] = df_.groupby(‘x2’)[‘x1’].cumsum()
```

Python 데이터 처리 기초

6. 프로젝트 실습

프로젝트 – 데이터 처리와 간단한 집계

생산 데이터의 처리와 외부 기상 데이터의 결합, 집계

주요 실습 내용

변수 제거
조건을 활용한 관측치 선택
날짜 등 변수 형식 변환
월, 요일 등 파생변수 생성
라인별 생산량, 불량률 집계
온도, 습도 등 외부 기상 데이터 결합
온도, 습도 그룹 변수 생성
외부 기상요인에 따른 불량률 집계

감사합니다.