Presentation, 2021.06.04

“ 우리동네 상권에 치킨집을 차려보자 ”

유동인구와 추정매출을 활용한 치킨집 상권 추천

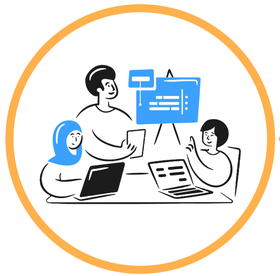
3조, 임재일, 민보라, 김휘성, 정의석, 권혁은

2021.05.31
~
2021.06.04

HYU | HANYANG UNIVERSITY SOFTWARE CONVERGENCE EDUCATION CENTER

미니프로젝트, openapi를 활용하여 pipeline을 구축.

CONTENTS.



팀원소개

- 팀 구성
- 역할분담

프로젝트 개요, 데이터에 대한 이해

- 프로젝트 명
- 프로젝트 주제 및 소개
- 활용 데이터

파이프라인 구축

- 개발환경 및 버전
- 데이터 흐름

데이터 전처리와 데이터 정제

- 사용 도구
- 필요 데이터 추출

시각화 및 분석

- 사용 도구
- 결과 도출

미니프로젝트, openapi를 활용하여 pipeline을 구축.

임재일

파이프라인 구축, 분석 및 시각화, 보고서작성

민보라

파이프라인 구축, 보고서작성

김휘성

데이터 수집 및 전처리, 분석 및 시각화, 보고서작성

정의석

데이터 수집 및 전처리, 분석, 보고서작성

권혁은

파이프라인 구축, 보고서작성

유동인구와 추정매출을 활용한 치킨집 상권 추천

주제 소개

치킨에 대한 수요가 높은 만큼 치킨에 대한 공급도 많은 상황 속에서,
치킨집 창업을 꿈꾸는 예비 사장님들을 위한 상권 추천 프로젝트

선정 이유

서울 지역의 유동인구와 추정매출을 활용하여 상관관계를 분석하고,
치킨집 창업에 적합한 상권지역을 파악하여 정보 제공

기대 효과

상권 분석 정보를 제공하여 예비 창업자들이 창업에 대한 참고 자료
로 활용

프로젝트 개발환경

tech-stacks	version
elasticsearch	7.12.0
logstash	7.12.0
kibana	7.12.0
kafka	2.12
APACHE Spark	3.1.1
	3.6.8

활용 데이터

◎ 활용 데이터

데이터 명	형식	출처
상권 – 추정유동인구	Open API ->CSV	서울열린데이터광장 (서울시 우리 마을 가게 상권 분석 서비스)
상권 – 추정매출	Open API ->CSV	서울열린데이터광장 (서울시 우리 마을 가게 상권 분석 서비스)
상권 – 점포	Open API ->CSV	서울열린데이터광장 (서울시 우리 마을 가게 상권 분석 서비스)
상권 – 상권영역	CSV	서울열린데이터광장 (서울시 우리 마을 가게 상권 분석 서비스)
행정동 코드	CSV	서울열린데이터광장

http://data.seoul.go.kr/dataList/3/literacyView.do

활용 데이터

샘플 URL

샘플 URL	2017 서울시 우리마을가게 상권분석서비스(상권배후지-추정유동인구) http://openapi.seoul.go.kr:8088/(인증키)/xml/VwsmTrdhlFlpopQq/1/5/2017

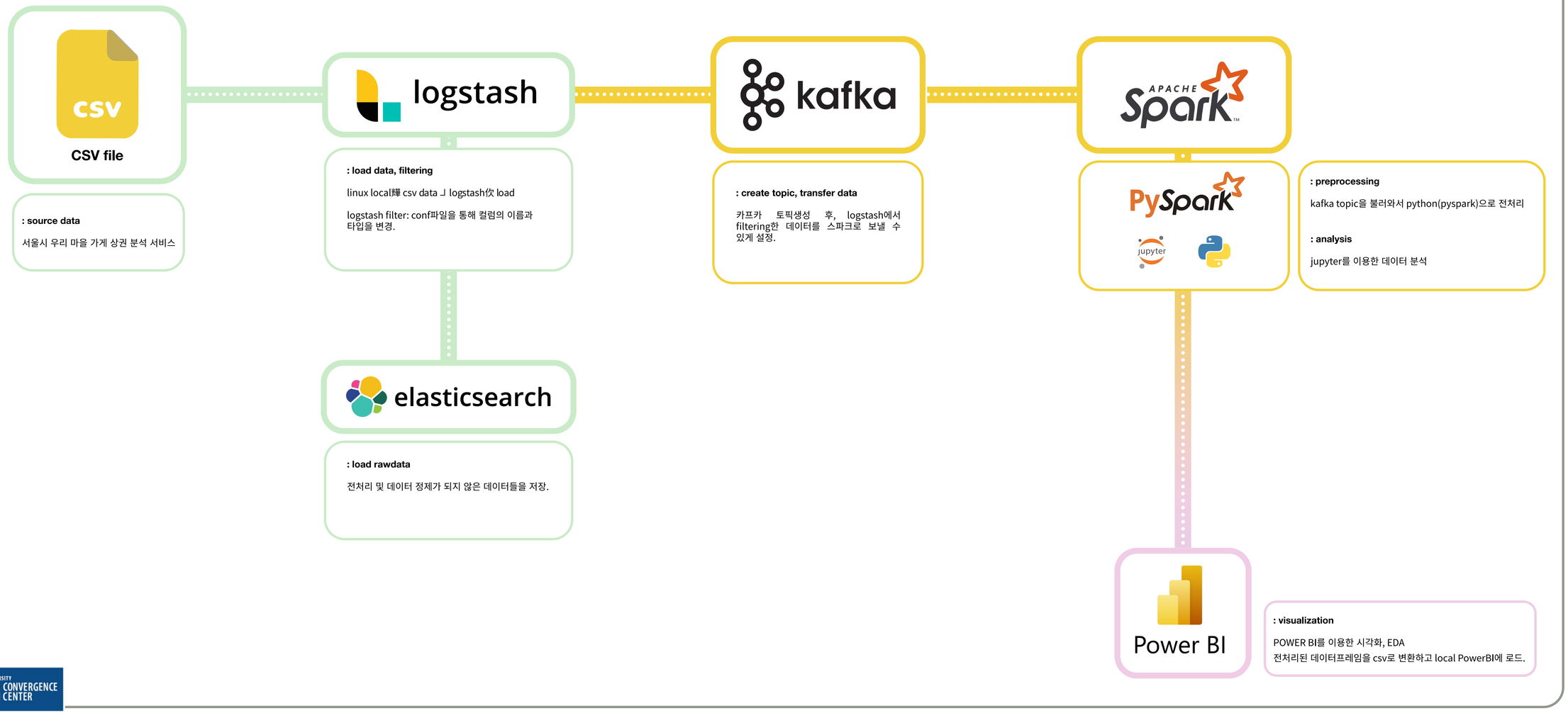
요청인자

변수명	타입	변수설명	값설명
KEY	String(필수)	인증키	OpenAPI 에서 발급된 인증키
TYPE	String(필수)	요청파일타입	xml : xml, xml파일 : xmlf, 엑셀파일 : xls, json파일 : json
SERVICE	String(필수)	서비스명	VwsmTrdhlFlpopQq
START_INDEX	INTEGER(필수)	요청시작위치	정수 입력 (페이징 시작번호 입니다 : 데이터 행 시작번호)
END_INDEX	INTEGER(필수)	요청종료위치	정수 입력 (페이징 끝번호 입니다 : 데이터 행 끝번호)
STDR_YY_CD	STRING(선택)	기준_년_코드	

http://data.seoul.go.kr/dataList/3/literacyView.do

파이프라인

: Pipeline



파이프라인-Logstash

```
input {
file {
path => "/root/sigungu.csv"
start_position => "beginning"
sincedb_path => "/dev/null"
}
```

input

```
filter {
        csv {
                separator => ","
                columns => ["상권코드","시군구코드"]
        }

        mutate{
                    rename => {
                        "상권코드" => "commercialcode"
                        "시군구코드" => "sigungucode"
                    }
        }
}
```

filter

```
output {
    kafka {
        bootstrap_servers => "http://localhost:9092"
        topic_id => "sigungu"
        codec => json
    }
}
```

output

파이프라인-Kafka

```
past_file_list = glob.glob(os.getcwd()+"/*.csv")

def new_file():
    global past_file_list
    file_list = glob.glob(os.getcwd()+"/*.csv")
    if len(file_list)!=len(past_file_list):
        for i in file_list:
            if i not in past_file_list:
                file = i
                file = file.split("/")[-1].replace('.csv','')
                break
        topic = "/usr/local/kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic "+file
        logstash = "/usr/share/logstash/bin/logstash -f /etc/logstash/conf.d/"+file+".conf"
        os.system(topic)
        time.sleep(60)
        os.system(logstash)
        time.sleep(300)
    past_file_list = file_list
    threading.Timer(10,new_file).start()

new_file()
```

upload data in kafka

파이프라인-pyspark01

```
spark = SparkSession
.builder
.master("local")
.appName("api")
.getOrCreate()
```

```
df3 = spark\
.read\
.format("kafka")\
.option("kafka.bootstrap.servers","localhost:9092")\
.option("subscribe", "people1820")\
.load()
```

create spark object, read topic

파이프라인-pyspark02

```
dfstr3 = df3.selectExpr("CAST(value AS STRING)")

df_struct3 = StructType()\
.add("year", StringType())\
.add("quarter", StringType())\
.add("divisioncode", StringType())\
.add("divisionname", StringType())\
.add("commercialcode", StringType())\
.add("commercialname", StringType())\
.add("totalpeople", StringType())\
.add("male", StringType())\
```

```
df_struct3 = dfstr3.select(from_json(col("value"),
df_struct3).alias("df3"))
dfnew3 = df_struct3.select("df3.*")
dfnew3 = dfnew3.toPandas()
```

	year	quarter	divisioncode	divisionname	commercialcode	commercialname	totalpeople	male	female	10s	20s	30s	40s	50s
0	기준년코드	기준분기코드	상권구분코드	상권구분코드명	상권코드	상권코드명	0	0	0	10	20	30	40	50
1	2018	4	A	골목상권	1000423	중산로15길	345620	169326	176294	83474	32414	41216	50745	47193
2	2018	4	A	골목상권	1000611	개봉로17다길	502769	236814	265956	75087	59098	78919	72346	71056
3	2018	4	A	골목상권	1000047	퇴계로56길	551472	269040	282432	29197	108351	106668	98149	88074
4	2018	4	A	골목상권	1000612	개봉로1길	492852	234137	258715	90500	53985	82917	75128	74283
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
17932	2020	1	A	골목상권	1000510	공달래로3길	868186	401170	467016	136291	93559	115088	130513	143454
17933	2020	1	D	발달상권	1001021	가산디지털단지역_1	106835	65651	41184	9114	20445	26975	23419	15365
17934	2020	1	D	발달상권	1001022	관악구 사당역_2	796907	386052	410855	64196	179543	141827	109628	115915
17935	2020	1	A	골목상권	1000511	공달래로5길	565967	265651	300316	85370	62295	72421	80390	99458
17936	2020	1	D	발달상권	1001023	가산디지털단지역_2	139650	84588	55060	5799	30103	38534	31736	18638

17937 rows × 15 columns

the value read by binary, tranfrom dataframe

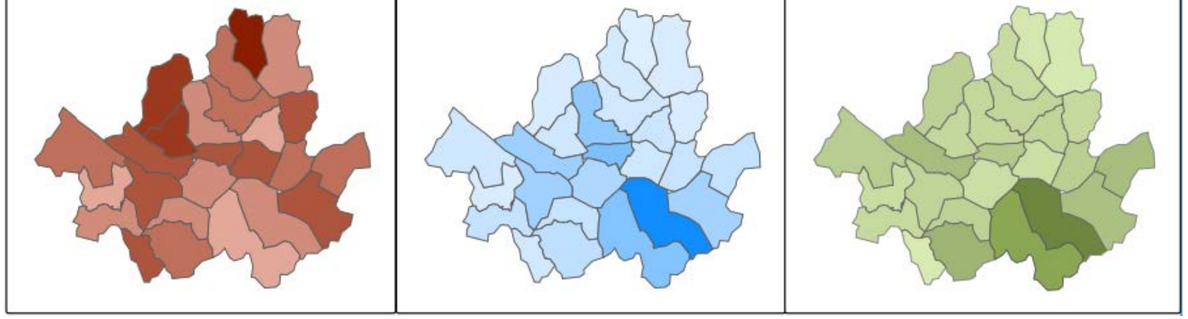
파이프라인-pyspark03

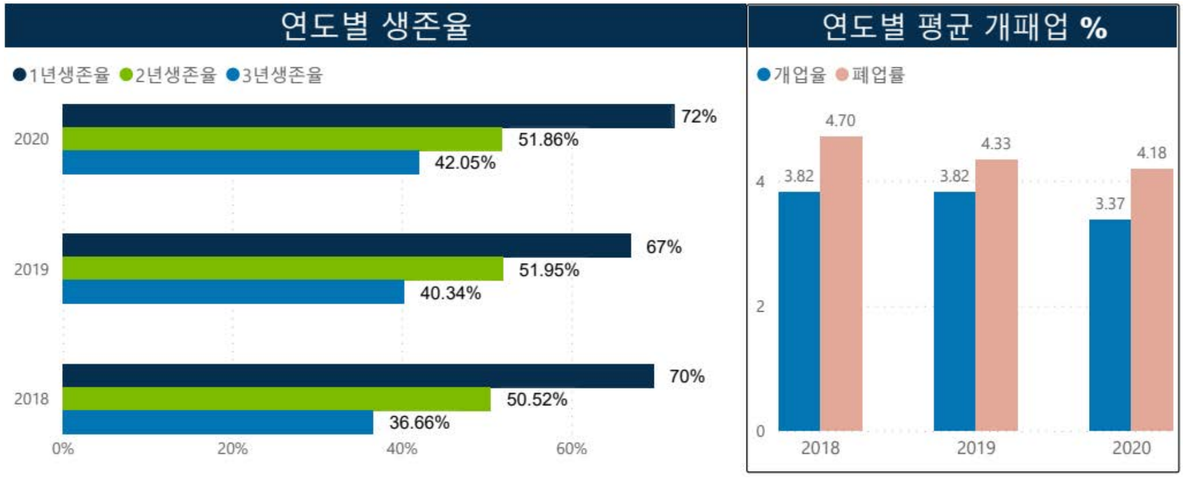
```
# 매출과 점포수 데이터 병합
inner_join = pd.merge(sales,store,on=
['year','quarter','divisioncode','divisionname','commercialcode','commerc
ialname', 'servicecode','servicename'])


# 유동인구 데이터와 병합
inner_join2 = pd.merge(inner_join,people,on=
['year','quarter','divisioncode','divisionname','commercialcode','commerc
ialname'])

# 행정동 코드와 결합
inner_join3 = pd.merge(inner_join2,sigungu,on=['commercialcode'])

# 매핑데이터에서 시 데이터가 서울인 데이터만 살리기
temp = mapping['sicode'] == '서울'
mapping = mapping[temp]
# 위 데이터와 병합하기 위해 sigungucode로 이름 변경
mapping=mapping.rename(columns={'code': 'sigungucode'})
# 시 컬럼 버리기
mapping=mapping.drop('sicode',axis=1)
# 지역구 병합
last_data = pd.merge(inner_join3,mapping,on=['sigungucode'])

# csv 파일로 내보내기
last_data.to_csv('./last_data.csv',encoding='utf-8',index=False)
```

join data, export to csv

시각화

Power BI



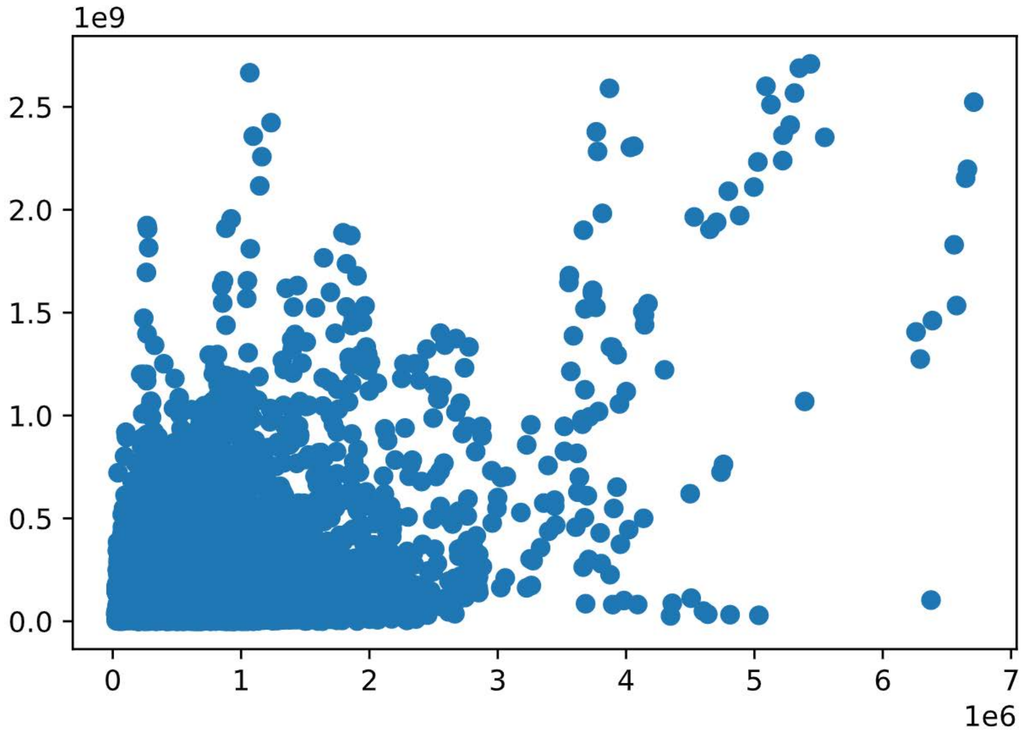


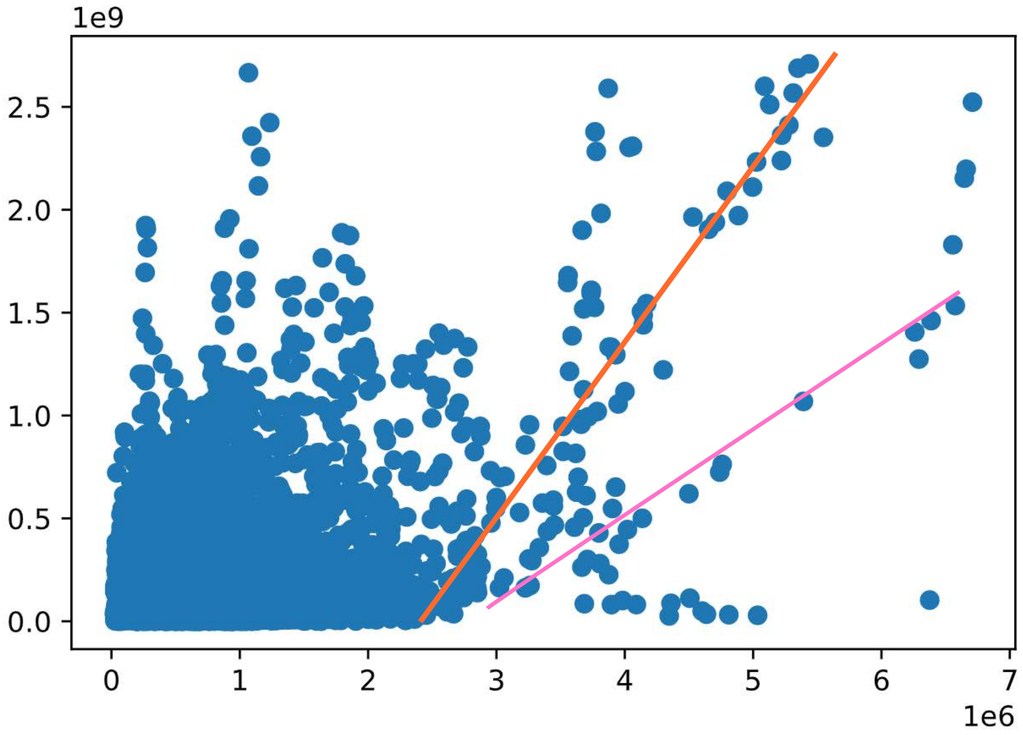
지역구	점포수	총유동인구	총매출
강남구	82,802	4,633,740,848	7,336,477,613,181
관악구	28,048	2,795,616,117	1,519,066,016,719
서초구	41,697	2,490,100,005	3,485,823,182,059
마포구	39,736	2,455,187,819	2,632,589,586,545
강동구	25,716	2,433,774,776	1,348,935,743,756
송파구	32,686	2,297,588,051	2,426,660,329,528
중구	50,566	1,952,030,969	3,724,297,429,425
강서구	19,423	1,825,941,884	944,075,162,659
광진구	21,841	1,783,174,745	1,217,917,979,640
은평구	15,429	1,686,990,523	666,163,763,416
동작구	20,834	1,624,570,843	1,272,091,903,768
영등포구	35,016	1,537,395,554	2,508,412,349,848
동대문구	22,216	1,512,929,389	1,126,415,338,135
성북구	18,439	1,483,456,649	911,269,246,560
종로구	43,915	1,420,634,998	3,088,997,002,403
서대문구	17,797	1,413,644,514	944,407,380,399
중랑구	16,278	1,364,927,779	665,487,825,115
구로구	23,738	1,338,477,767	1,319,702,507,376
강북구	15,553	1,226,370,260	778,870,276,670
양천구	13,539	1,192,999,389	644,135,483,557
용산구	27,054	1,155,174,761	2,061,519,291,604
성동구	17,841	1,116,506,957	995,318,189,630
도봉구	11,196	791,571,788	475,276,443,614
노원구	10,387	736,723,159	652,641,303,242
금천구	17,288	678,271,179	922,385,834,207
합계	**669,035**	**42,947,800,723**	**43,668,937,183,056**

데이터 분석

```
# 아래의 코드는 전체의 데이터에서 특정 지역구에 있는 상권들에서 음식점 별 상관관계 및 추이 확인하기 위한 것.

## packages
## 패키지 불러오기
import pandas as pd
from pandas.plotting import scatter_matrix

## 데이터 불러오기
test_df_area = pd.read_csv('./last_data.csv',encoding='utf-8',index=False)

korean_columns=['기준년', '기준분기코드', '상권구분코드', '상권구분코드명', '상권코드', '상권코드명', '서비스업종코드',
       '서비스업종코드명', '당월매출금액', '당월매출건수', '주중매출금액', '주말매출금액', '남성매출금액', '여성매출금액',
       '10대매출금액', '20대매출금액', '30대매출금액', '40대매출금액', '50대매출금액', '60대이상매출금액',
       '주중매출건수', '주말매출건수', '남성매출건수', '여성매출건수', '점포수', '개업율', '폐업률', '총유동인구',
       '남성유동인구', '여성유동인구', '10대유동인구', '20대유동인구', '30대유동인구', '40대유동인구',
       '50대유동인구', '60대이상유동인구', '시군구코드', '지역구']

test_df_area.columns = korean_columns
```

데이터 분석



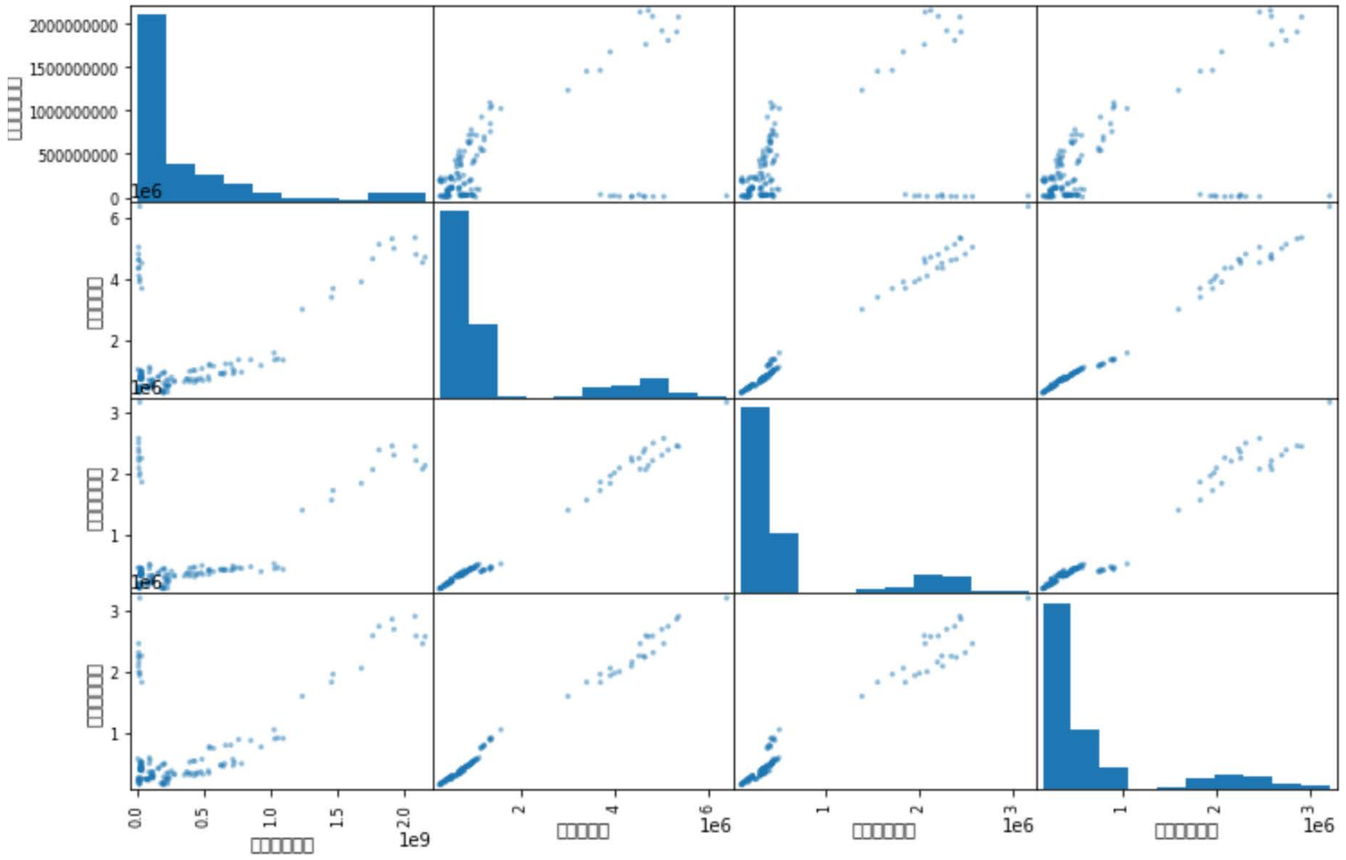


전체 데이터에서 '총유동인구'-'당월총매출금액'

데이터 분석

```
## correlations
## GRAPH
## 그래프로 나타내기
def location_corr(df, location_name, restaurant):
    # df: dataframe
    # location_name: 지역구
    # restaurant: 서비스업종코드명
    location = df['지역구'] == str(location_name)
    location_subset = df[location]
    location_food = location_subset['서비스업종코드명'] == str(restaurant)
    location_food_subset = location_subset[location_food]
    location_for_corr = location_food_subset.drop(columns=['기준년', '기준분기코드', '상권구분코드', '상권구분코드명', '상권코드', '상권코드명', '서비스업종코드', '서비스업종코드명', '시군구코드', '지역구'], axis=1)

    attributes = ['당월매출금액','총유동인구','남성유동인구','여성유동인구']
    return scatter_matrix(location_for_corr[attributes], figsize=(12, 8))

## correlations
## SCORE
## 점수로 나타내기
def location_corr_score(df, location_name, restaurant):
    # df: dataframe
    # location_name: 지역구
    # restaurant: 서비스업종코드명
    location = df['지역구'] == str(location_name)
    location_subset = df[location]
    location_food = location_subset['서비스업종코드명'] == str(restaurant)
    location_food_subset = location_subset[location_food]
    location_for_corr = location_food_subset.drop(columns=['기준년', '기준분기코드', '상권구분코드', '상권구분코드명', '상권코드', '상권코드명', '서비스업종코드', '서비스업종코드명', '시군구코드', '지역구'], axis=1)
    corr_location = location_for_corr.corr()
    return corr_location['당월매출금액'].sort_values(ascending=False)
```

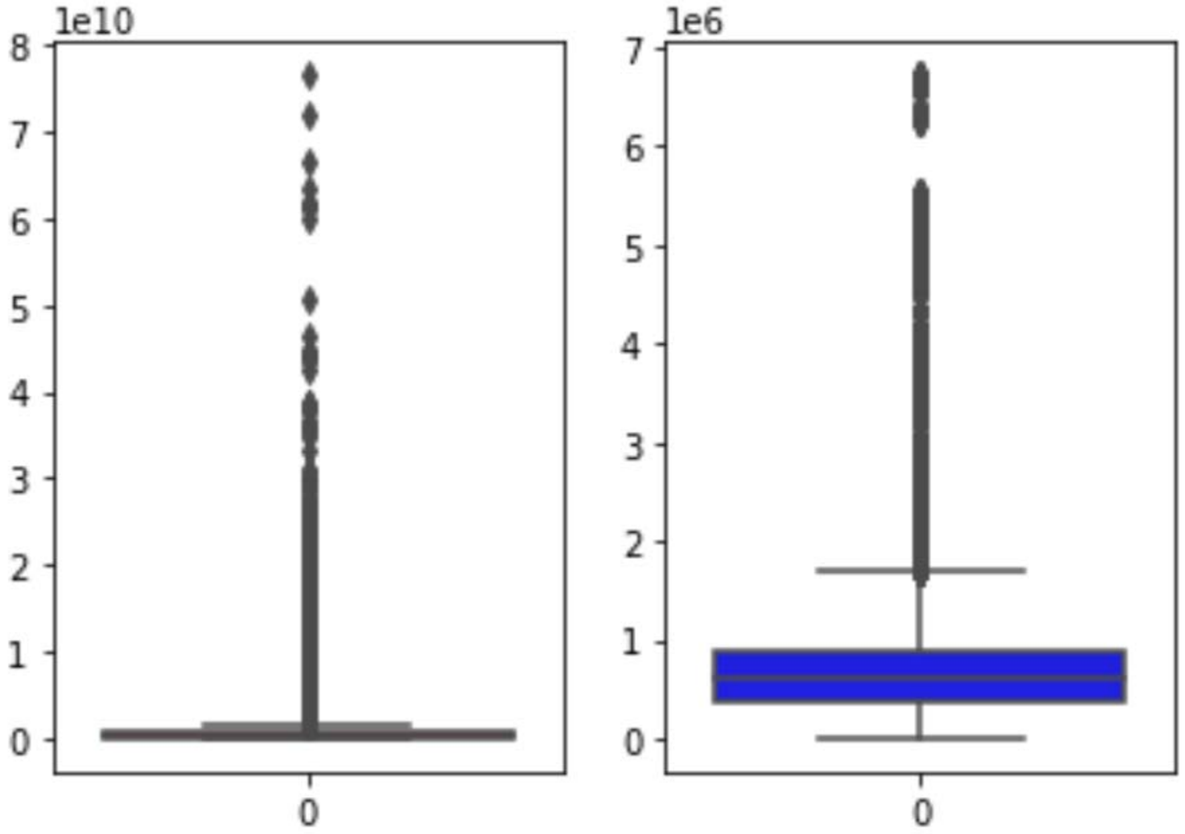
데이터 분석


2000000000
1500000000
1000000000
500000000
0
1e6
6
4
2
1e6
3
2
1
1e6
3
2
1
0.0
0.5
1.0
1.5
2.0
1e9
2
4
6
1e6
1
2
3
1e6
1
2
3
1e6

데이터 분석

```
## 결론적로 '이상치'로 보이는 분포가 많이 있다는 것을 확인할 수 있었다.
## 예상되는 이상치로 보이는 패턴들의 의미는 배달 및 음식점을 위한 유동인구가 아닐것으로 보인다.
## 이 예상치를 제거한다면 깨끗한 상태의 데이터를 확인할 수 있을 것이다.

## 이상치 제거 및 머신러닝을 돌릴 수 있는 패키지 불러오기
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression

## 치킨전문점에 해당하는 총유동인구와 당월매출금액의 이상치 제거

# 우선 boxplot으로 그 추세를 확인해본다.
fig, ax = plt.subplots(ncols=2)
sns.boxplot(data=test_df_area['당월매출금액'], color='red', ax=ax[0])
sns.boxplot(data=test_df_area['총유동인구'], color='blue', ax=ax[1])
# 이를 확인해 봤을때, 극단적으로 낮게 나타나는 값들이 너무 많았다.
# 예상하건데, 이 값들이 배달 및 음식점을 위해 나타난 인구가 아닐 것으로 보인다.
# 이 낮은 값들을 역으로 제거하여 평균을 올리고 컬럼에 있는 이상치를 제거한다.
```

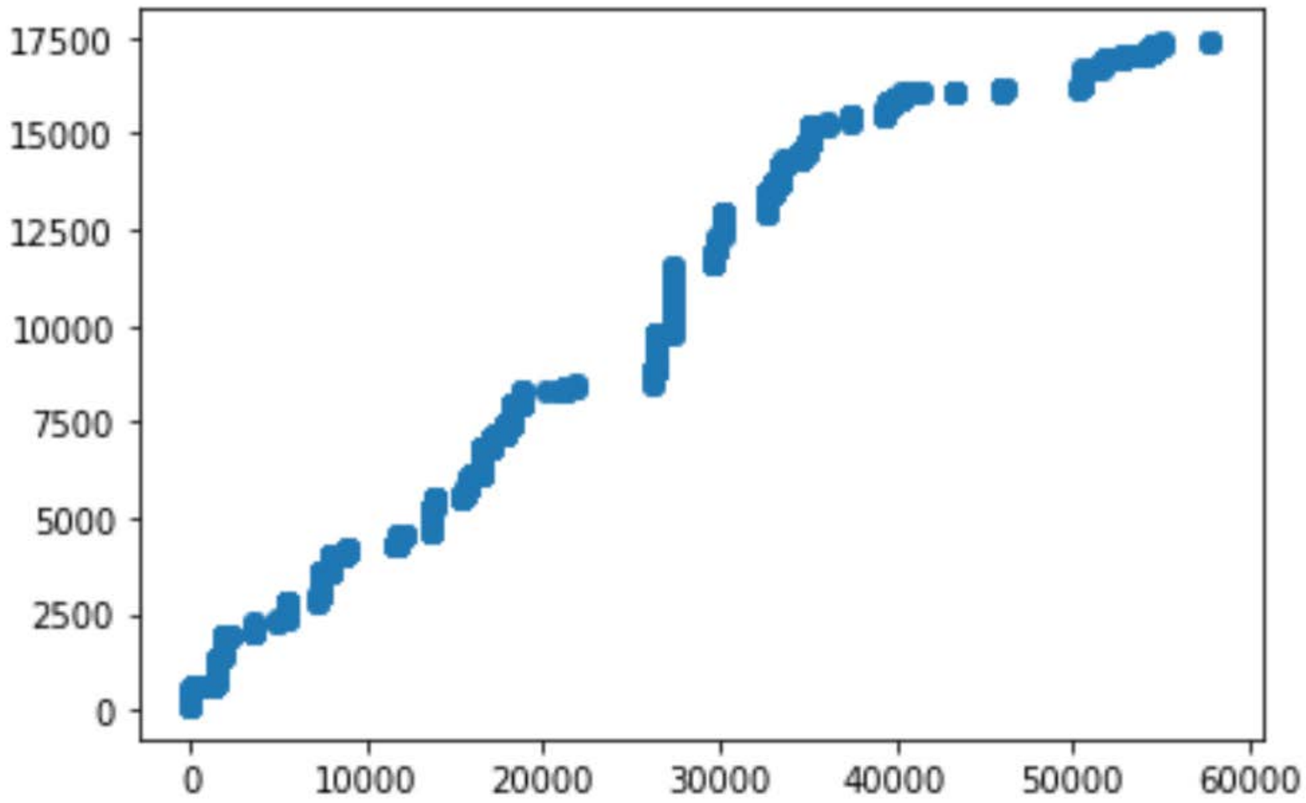
데이터 분석



데이터 분석

```
# 이상치를 포착하는 함수를 만들었다.
def outliear_iqr(data):
    q1, q3 = np.percentile(data, [25, 75])
    iqr = q3-q1
    lower_bound = q1 - (iqr * 1.5)
    upper_bound = q3 + (iqr * 1.5)
    return np.where((data>upper_bound)|(data<lower_bound))

# 이상치 포착함수를 적용하여 각 컬럼의 이상치를 변수에 지정했다.

money_outlier = outliear_iqr(test_df_area['당월매출금액'])[0]
people_outlier = outliear_iqr(test_df_area['총유동인구'])[0]

# 이상치의 개수를 보면 데이터의 당월 매출 금액이 훨씬 많은 것을 확인할 수 있다.
# 길이를 맞추지 않으면 그래프로 확인할 수 없기 때문에 우선 길이를 맞춰주었다.

people_ot = list(people_outlier)
money_ot = list(money_outlier[:2473])

# x,y에 값을 할당하고 산점도 그래프로 확인해보았다.
plt.plot(people_ot, money_ot, 'o')
```

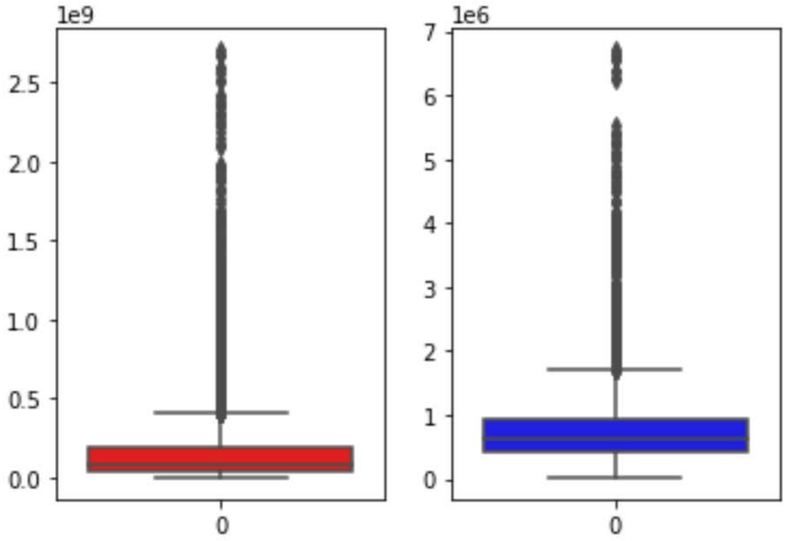
데이터 분석

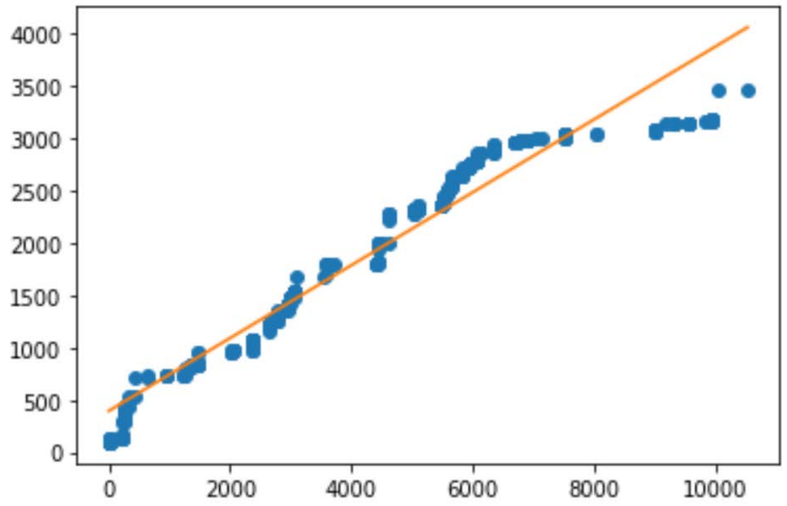

17500
15000
12500
10000
7500
5000
2500
0
0
10000
20000
30000
40000
50000
60000

데이터 분석

```
# 이 방식으로 러닝을 돌릴 수 있는지 확인해보기로 했다.
chickens_ = test_df_area['서비스업종코드명'] == '치킨전문점'
chicken_dataframe = test_df_area[chickens_]
chicken_money = chicken_dataframe['당월매출금액']
chicken_people = chicken_dataframe['총유동인구']

plt.plot(chicken_people, chicken_money, 'o')

fig, ax = plt.subplots(ncols=2)
sns.boxplot(data=chicken_money, color='red', ax=ax[0])
sns.boxplot(data=chicken_people, color='blue', ax=ax[1])

money_outlier = outliear_iqr(chicken_money)[0]
people_outlier = outliear_iqr(chicken_people)[0]
people_ot = list(people_outlier)
money_ot = list(money_outlier[:445])
plt.plot(people_ot, money_ot, 'o')

df_pm = pd.DataFrame(index=range(0, 445), columns=['people', 'money'])
df_pm['people'] = people_ot
df_pm['money'] = money_ot
linear_filter = LinearRegression()
linear_filter.fit(df_pm['people'].values.reshape(-1, 1), df_pm['money'])
plt.plot(df_pm['people'], df_pm['money'], 'o')
plt.plot(df_pm['people'], linear_filter.predict(df_pm['people'].values.reshape(-1, 1)))

nowon_chick = chicken_dataframe['지역구'] == '노원구'
nowon_chick_peole = chicken_dataframe[nowon_chick]
nowon_chk_peoaverage = nowon_chick_peole['총유동인구'].mean()

nowon_chk_peoaverage
linear_filter.predict([[959228]])
```

데이터 분석


1e9
2.5
2.0
1.5
1.0
0.5
0.0
0
1e6
7
6
5
4
3
2
1
0
0


4000
3500
3000
2500
2000
1500
1000
500
0
0
2000
4000
6000
8000
10000