빅데이터 분석

Big Data Analysis

P15

Covid – 19 Analysis

이재근 홍문기 최학준

빅데이터 분석

Big Data Analysis

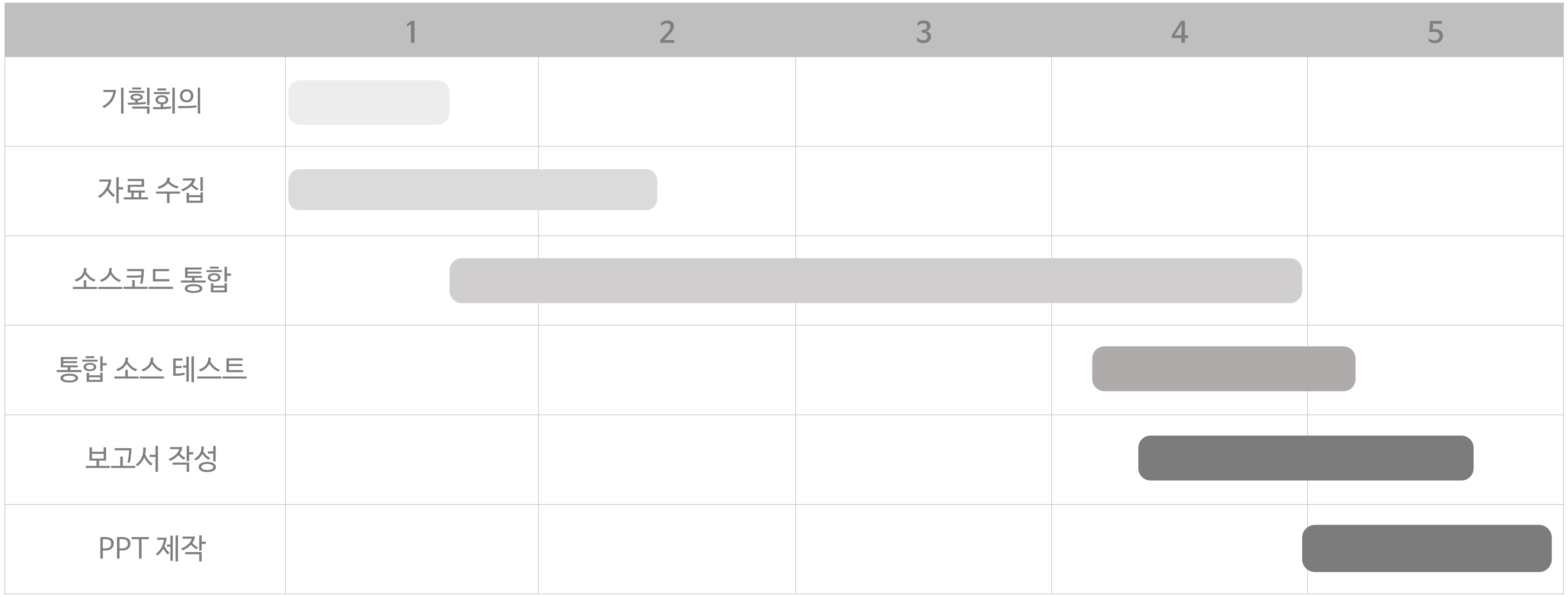
1. 계획서

   - 사용 데이터

   - 분석 및 결과

2. 소감문

PROJECT TIMELINE 2020.06.05 – 2020.06.15



이번 코로나 사태를 생각하며 어떤 데이터들이 코로나 확산에 얼마나 영향을 미쳤는 지를 시계열 데이터를 통해 상관관계를 분석하고 그래프로 시각화 하여 코로나 분석을 예측 및 분석 해보았습니다.

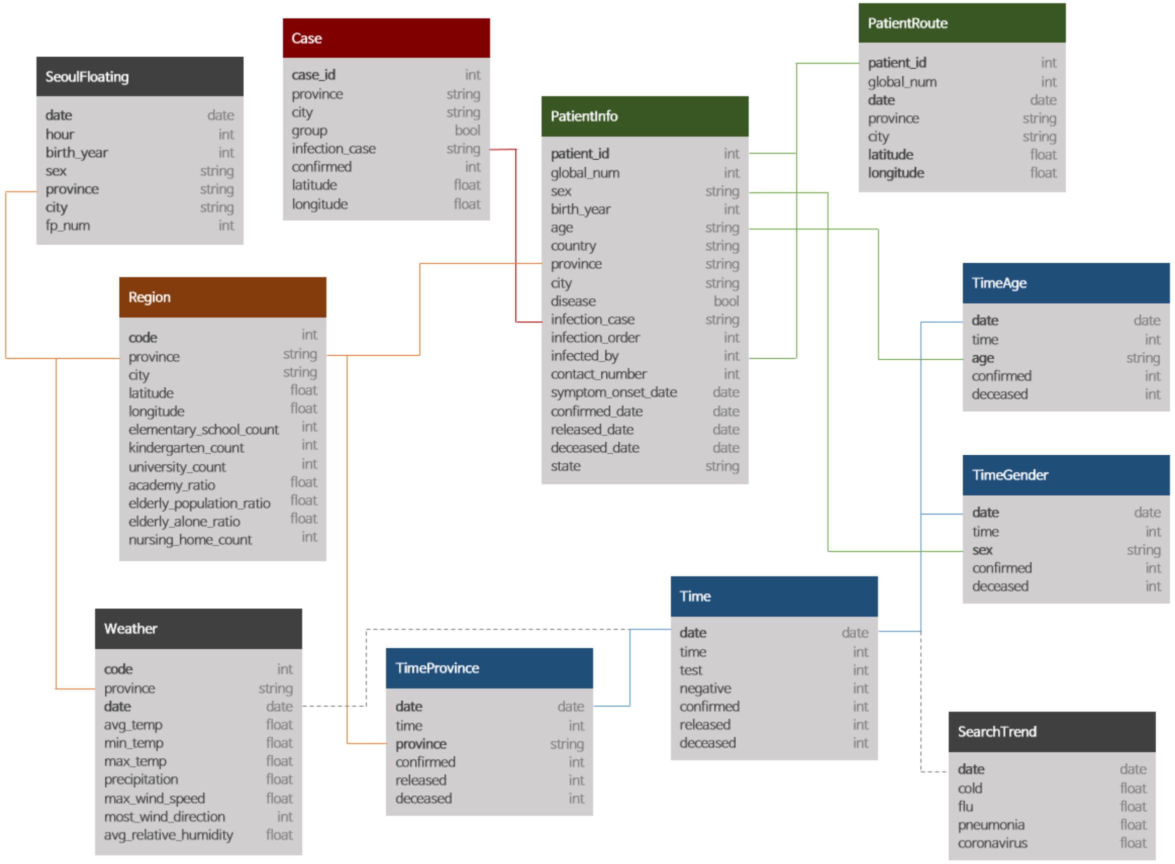
Used Data

Data Science for COVID-19 (DS4C)

DS4C: Data Science for COVID-19 in South Korea

https://www.kaggle.com/kimjihoo/coronavirusdataset

The Structure of our Dataset



비슷한 속성을 가진 데이터 = 같은 색상의 선

행이 열 사이에 연결된 경우 열의 값이 부분적으로 공유됨

점선은 약한 관련성을 의미

Used Data

Data Science for COVID-19 (DS4C)

DS4C: Data Science for COVID-19 in South Korea

https://www.kaggle.com/kimjihoo/coronavirusdataset

- Data Sources
  - Case.csv
  - PatientInfo.csv
  - PatientRoute.csv
  - Policy.csv
  - Region.csv
  - SearchTrend.csv
  - SeoulFloating.csv
  - Time.csv
  - TimeAge.csv
  - TimeGender.csv
  - TimeProvince.csv
  - Weather.csv

Age

```
In [21]: age_raw = get_data(file_paths[0])
         data_range(age_raw, 'date')
         age_list = age_raw.age.unique()
         print('Age groups:', age_list)
         print('# 80s == 80s and older')
```

[Sample data]

	date	time	age	confirmed	deceased
0	2020-03-02	0	0s	32	0
1	2020-03-02	0	10s	169	0
2	2020-03-02	0	20s	1235	0
816	2020-05-31	0	60s	1405	39
817	2020-05-31	0	70s	725	80
818	2020-05-31	0	80s	498	131

```
Date range: 91 days
# 2020-03-02 to 2020-05-31
Age groups: ['0s' '10s' '20s' '30s' '40s' '50s' '60s' '70s' '80s']
# 80s == 80s and older
```

Age

In [22]:

```
fig, ax = plt.subplots(figsize=(13, 7))
plt.title(f'Confirmed Cases by Age ({last_update})', fontsize=17)
sns.barplot(age_list, age_raw.confirmed[-9:])
ax.set_xlabel('age', size=13)
ax.set_ylabel('number of cases', size=13)
plt.show()
```

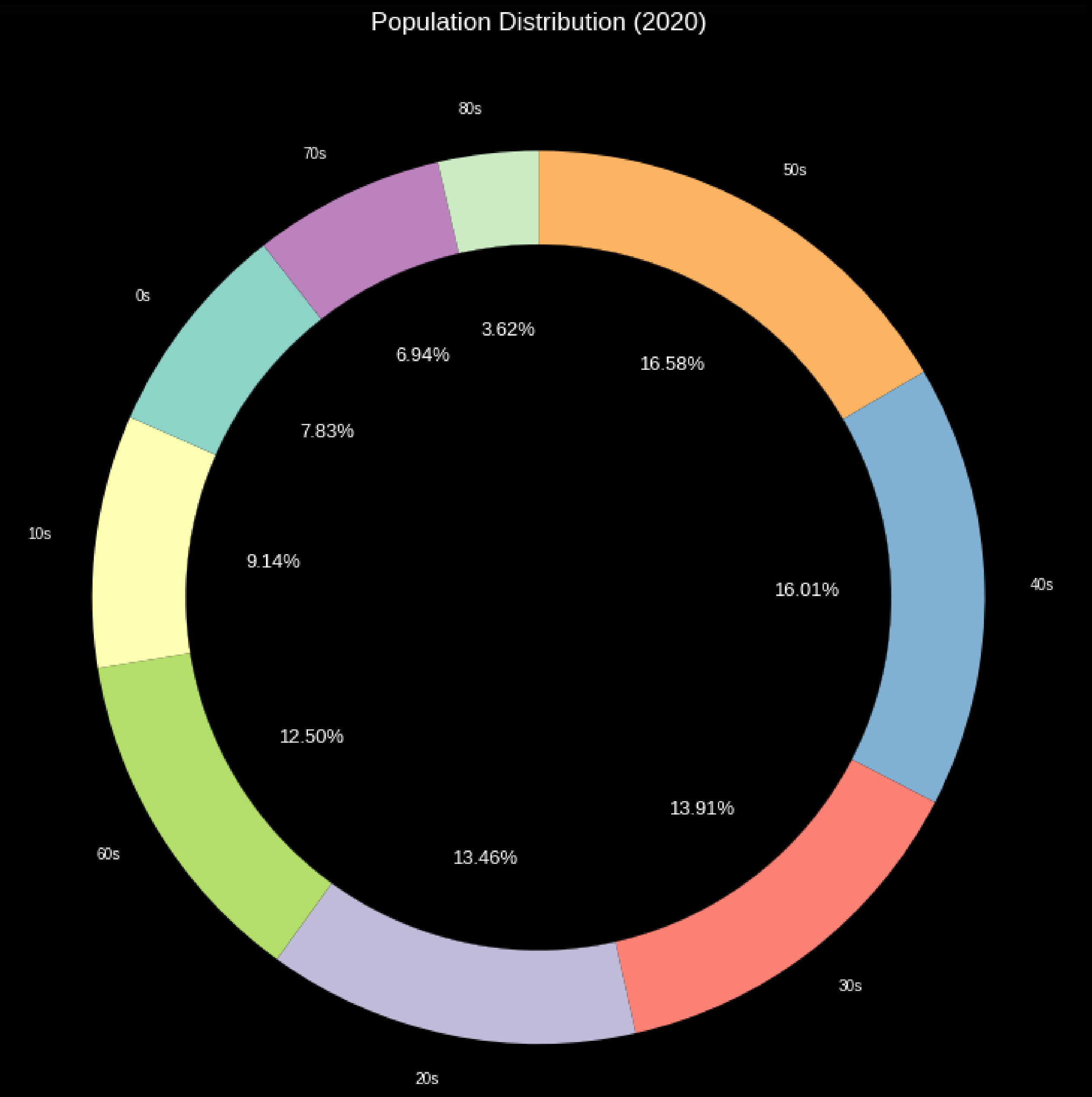
Age

In [23]:
```
pop_order = pd.DataFrame()
pop_order['age'] = age_list
pop_order['population'] = (4055740, 4732100, 6971785, 7203550, 8291728, 8587047, 6472987, 3591533, 1874109)
pop_order['proportion'] = round(pop_order['population']/sum(pop_order['population']) * 100, 2)
pop_order = pop_order.sort_values('population', ascending=False)
pop_order.set_index(np.arange(1, 10), inplace=True)
display(pop_order)
```

	age	population	proportion
1	50s	8587047	16.58
2	40s	8291728	16.01
3	30s	7203550	13.91
4	20s	6971785	13.46
5	60s	6472987	12.50
6	10s	4732100	9.14
7	0s	4055740	7.83
8	70s	3591533	6.94
9	80s	1874109	3.62

Age

```
In [24]: color_pie = [color_list[5], color_list[4], color_list[3]
                      , color_list[2], color_list[6], color_list[1]
                      , color_list[0], color_list[7], color_list[8]]
         fig, ax = plt.subplots(figsize=(11, 11))
         plt.title('Population Distribution (2020)', fontsize=17)
         pop_circle = plt.Circle((0,0), 0.79, color='black')
         plt.pie(pop_order.proportion
                 , labels = pop_order.age
                 , autopct = '%.2f%%'
                 , colors = color_pie
                 , startangle=90
                 , counterclock=False)
         p=plt.gcf()
         p.gca().add_artist(pop_circle)
         plt.show()
```



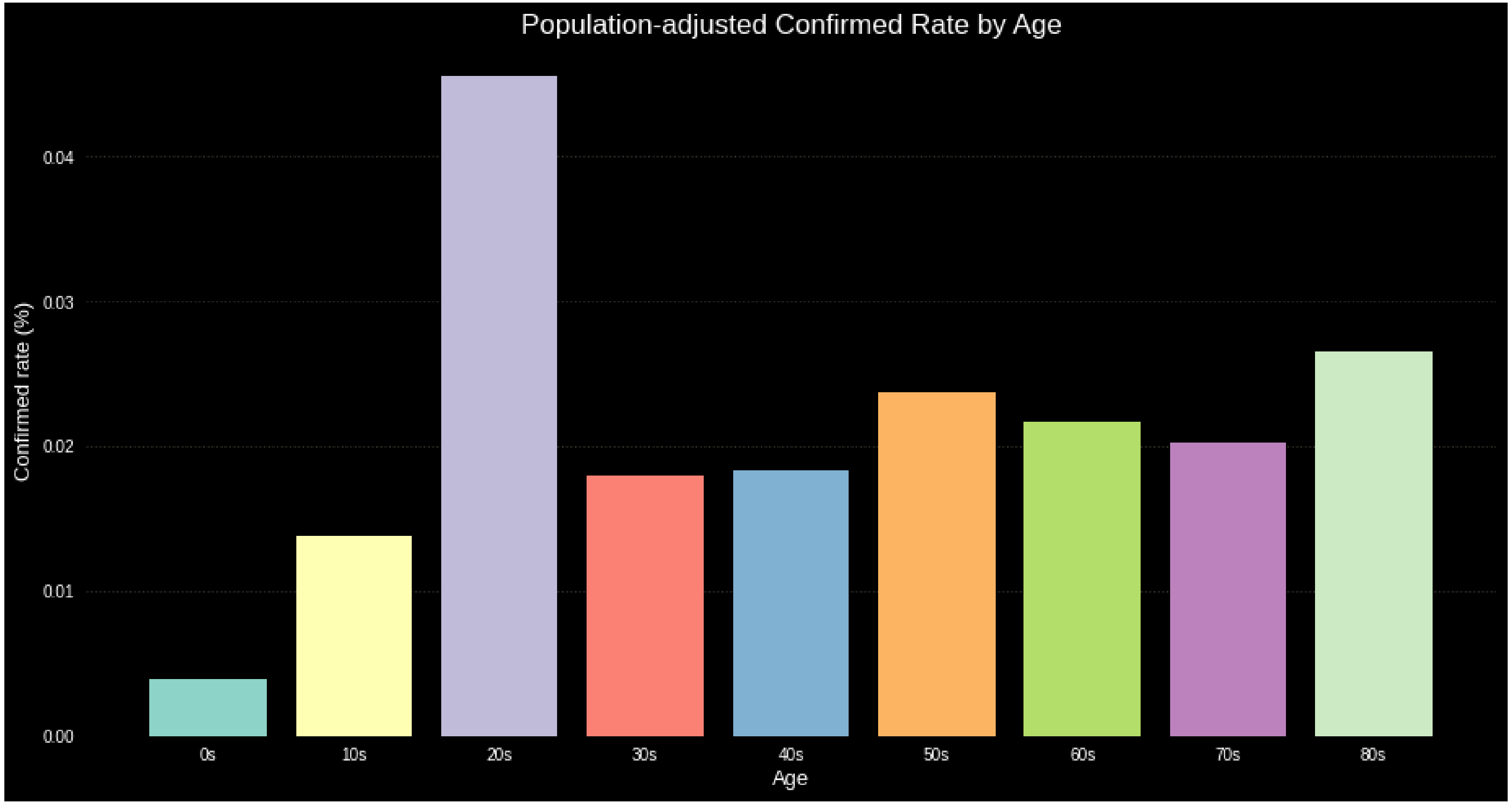
Age

```
In [26]: confirmed_by_population['confirmed_rate'] = confirmed_by_population['confirmed']/confirmed_by_population['population'] * 100;
display(confirmed_by_population)
```

	age	population	proportion	confirmed	confirmed_rate
7	0s	4055740	7.83	157	0.003871
6	10s	4732100	9.14	655	0.013842
4	20s	6971785	13.46	3176	0.045555
3	30s	7203550	13.91	1292	0.017936
2	40s	8291728	16.01	1521	0.018344
1	50s	8587047	16.58	2039	0.023745
5	60s	6472987	12.50	1405	0.021706
8	70s	3591533	6.94	725	0.020186
9	80s	1874109	3.62	498	0.026573

Age

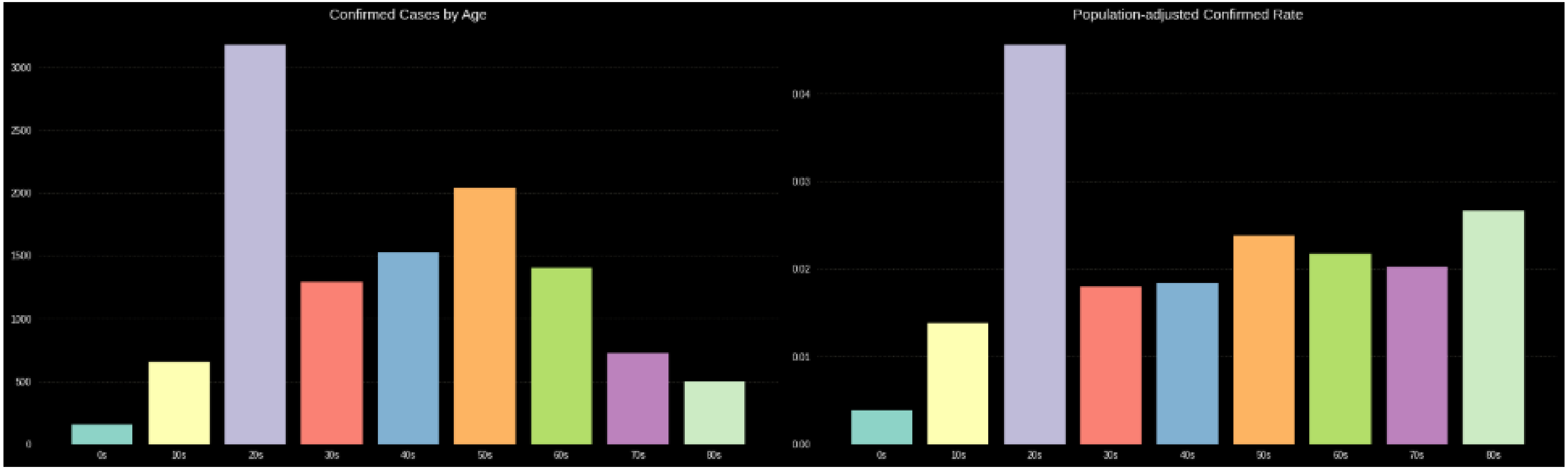
In [27]:
```
fig, ax = plt.subplots(figsize=(13, 7))
plt.title('Population-adjusted Confirmed Rate by Age', fontsize=17)
ax.bar(age_list, confirmed_by_population.confirmed_rate[-9:], color=color_list)
ax.set_xlabel('Age', size=13)
ax.set_ylabel('Confirmed rate (%)', size=13)
plt.show()
```



Age

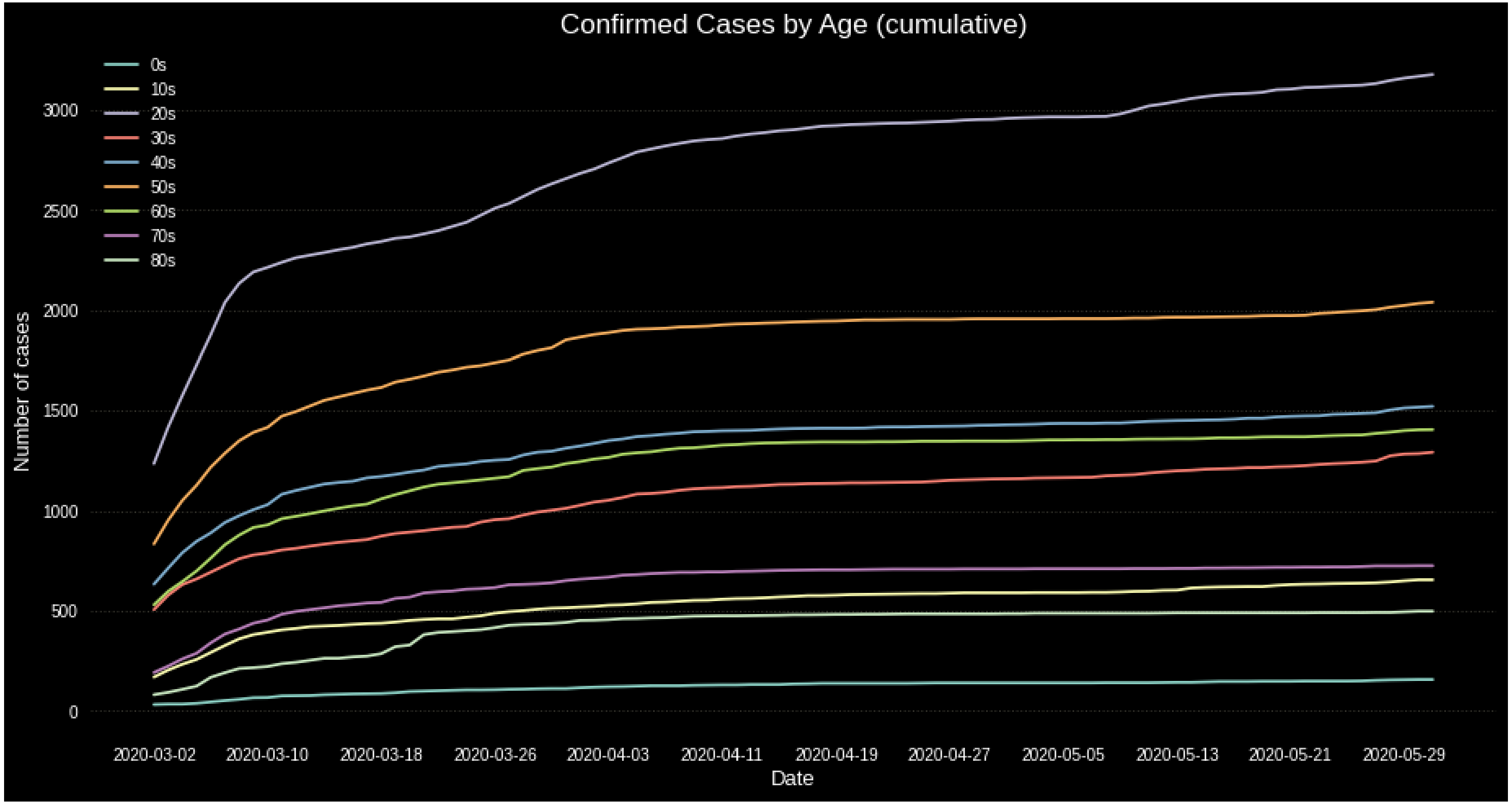
In [28]:

```python
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(23, 7))

## 1. Absolute numbers
axes[0].set_title('Confirmed Cases by Age', fontsize=15)
axes[0].bar(age_list, confirmed_by_population.confirmed, color=color_list)

## 2. Confirmed rate
axes[1].set_title('Population-adjusted Confirmed Rate', fontsize=15)
axes[1].bar(age_list, confirmed_by_population.confirmed_rate, color=color_list)

plt.show()
```



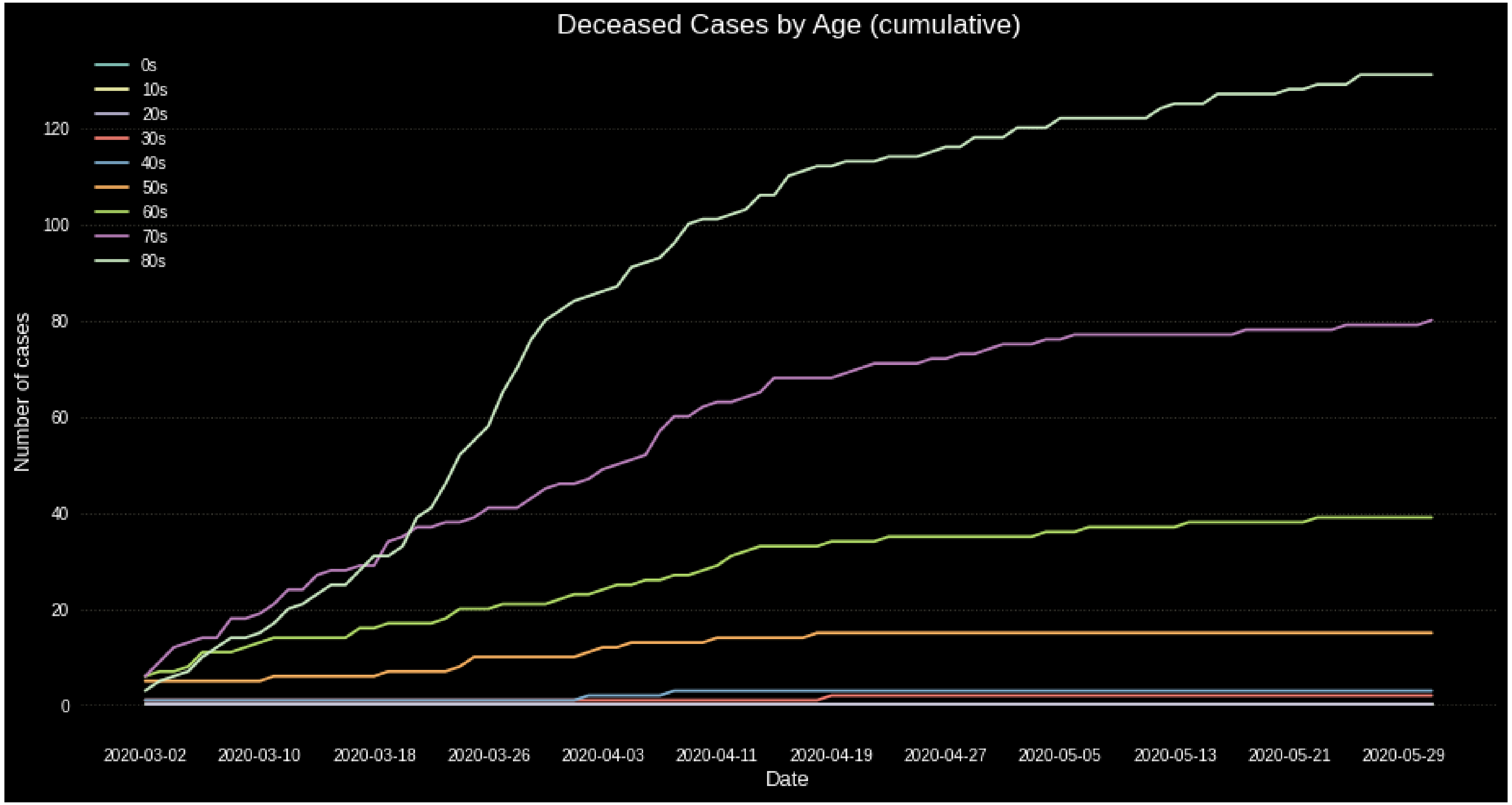
Age

In [29]:
```
## Plot time series of confirmed cases
plot_groupby(age_raw, 'age', 'confirmed', 'Confirmed Cases by Age (cumulative)')
```



Age

```
In [30]: plot_groupby(age_raw, 'age', 'deceased', 'Deceased Cases by Age (cumulative)')

age_deceased = age_raw.tail(9)[['age', 'deceased']]
age_deceased.set_index(np.arange(0, len(age_raw.age.unique())), inplace=True)
print('[Latest deceased cases]')
display(age_deceased.T)
```

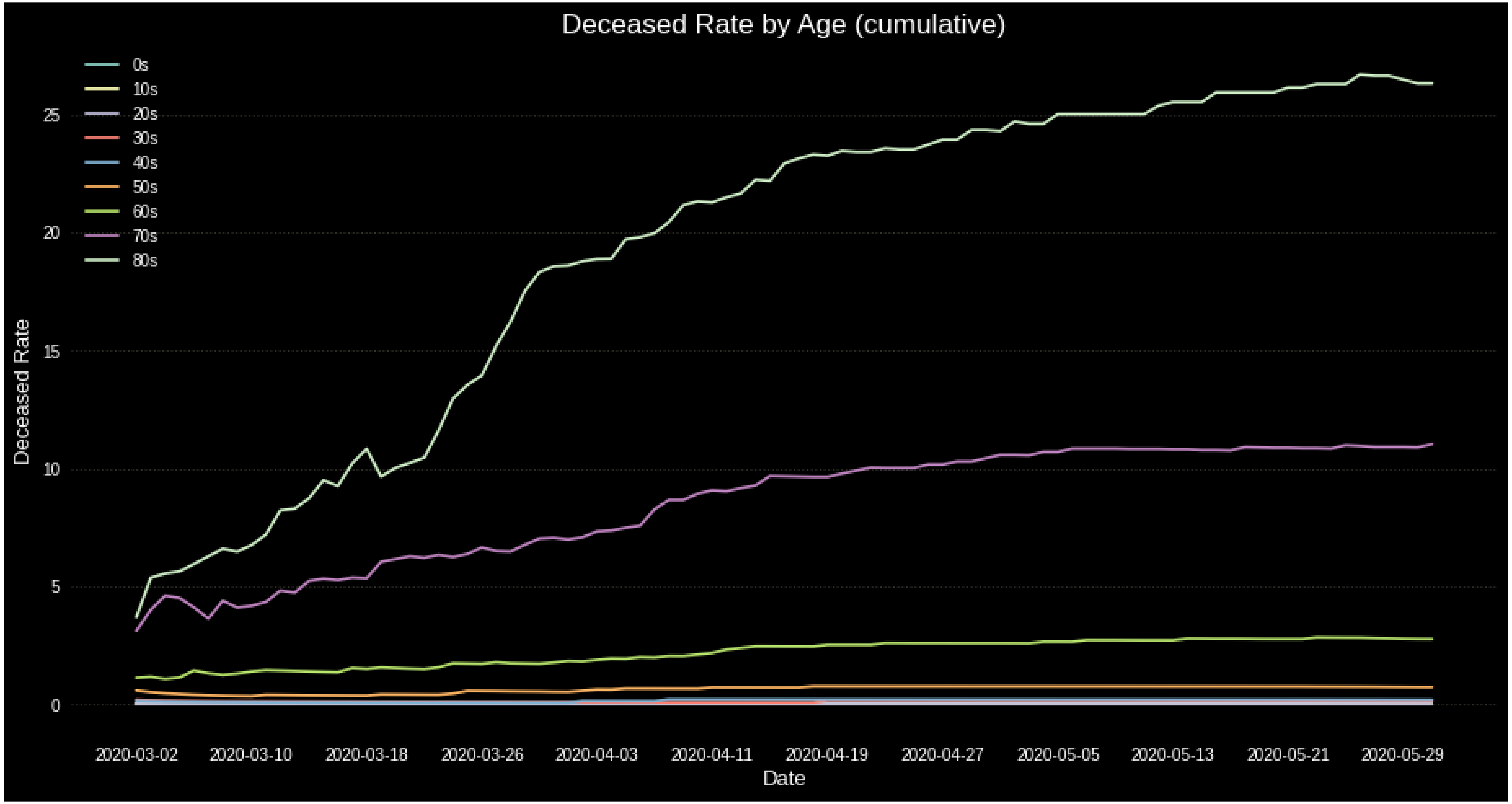


[Latest deceased cases]

	0	1	2	3	4	5	6	7	8
age	0s	10s	20s	30s	40s	50s	60s	70s	80s
deceased	0	0	0	2	3	15	39	80	131

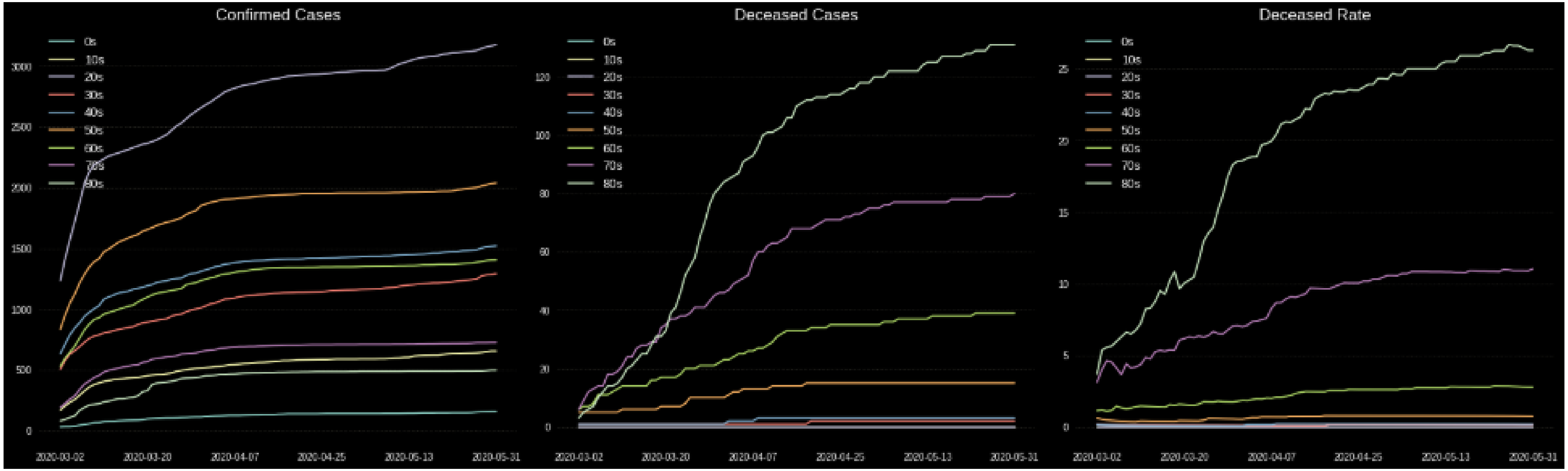
Age

In [31]:
```python
age_raw['deceased_rate'] = age_raw.deceased/age_raw.confirmed * 100.0

plot_groupby(age_raw, 'age', 'deceased_rate', 'Deceased Rate by Age (cumulative)', 'Deceased Rate')
```



Age

In [32]:
```
if not fast:
    fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(23, 7))
    sub_list = [age_raw.confirmed, age_raw.deceased, age_raw.deceased_rate]
    title_list = ['Confirmed Cases', 'Deceased Cases', 'Deceased Rate']
    for sub, i, title in zip(sub_list, range(len(sub_list)), title_list):
        confirmed_set = sub.groupby(age_raw.age)
        for confirmed_each, age_each in zip(confirmed_set, age_list):
            axes[i].plot(age_raw.date.unique(), confirmed_each[1], label=age_each)
            axes[i].set_title(title, size=17)
        axes[i].set_xticks(axes[i].get_xticks()[::int(len(age_raw.date.unique())/5)])
        axes[i].legend(fontsize=13)
```



Region

```
In [33]: region_raw = get_data(file_paths[1])
         data_range(region_raw, 'date')
```

[Sample data]

	date	time	province	confirmed	released	deceased
0	2020-01-20	16	Seoul	0	0	0
1	2020-01-20	16	Busan	0	0	0
2	2020-01-20	16	Daegu	0	0	0
2258	2020-05-31	0	Gyeongsangbuk-do	1379	1295	54
2259	2020-05-31	0	Gyeongsangnam-do	123	121	0
2260	2020-05-31	0	Jeju-do	15	13	0

Date range: 133 days
2020-01-20 to 2020-05-31

Region

In [34]:
```
print('Number of regions:', len(region_raw.province.unique()))
print('Number of logs per region:', len(region_raw[region_raw.province=='Jeju-do']))
print('regions * logs:', len(region_raw.province.unique()) * len(region_raw[region_raw.province=='Jeju-do']))
print('Number of rows:', len(region_raw))
```

```
Number of regions: 17
Number of logs per region: 133
regions * logs: 2261
Number of rows: 2261
```

In [35]:
```
region_raw.describe().iloc[1:, 1:]
```

Out [35]:

	confirmed	released	deceased
mean	391.320212	255.985847	7.641751
std	1291.888948	993.309392	28.890377
min	0.000000	0.000000	0.000000
25%	3.000000	0.000000	0.000000
50%	30.000000	12.000000	0.000000
75%	117.000000	56.000000	1.000000
max	6883.000000	6607.000000	185.000000

Region

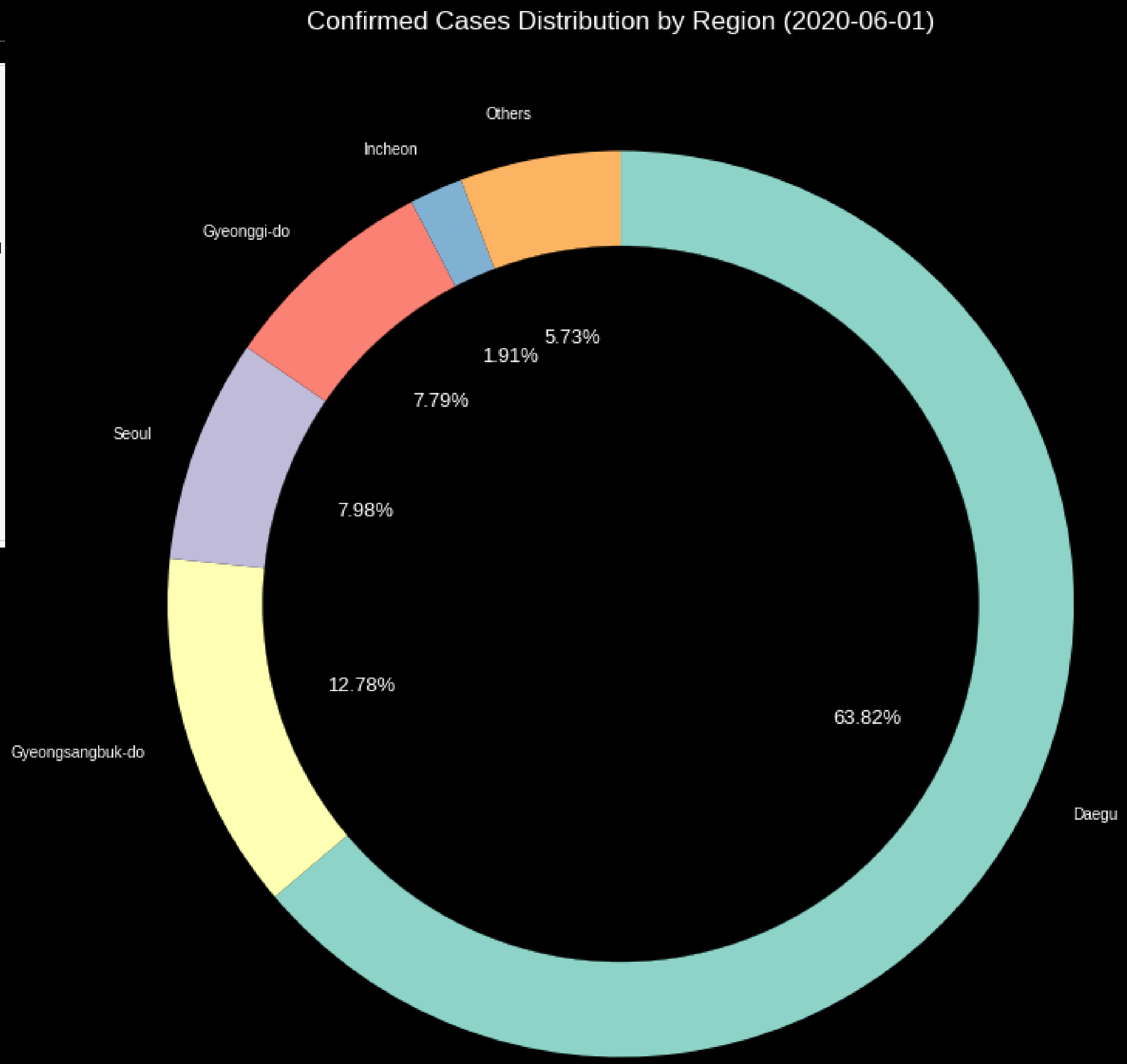
```
In [36]: loc_latest = region_raw[region_raw.date==region_raw.date.iloc[-1]]
del loc_latest['date']
del loc_latest['time']
loc_latest = loc_latest.iloc[:, :2]
loc_latest['proportion'] = round(loc_latest.confirmed / sum(loc_latest.confirmed) * 100, 2)
loc_latest = loc_latest.sort_values('proportion', ascending=False)
loc_latest.set_index(np.arange(1, len(loc_latest)+1), inplace=True)
loc_latest_all = loc_latest.copy()
loc_latest_all
```

Out[36]:

	province	confirmed	proportion
1	Daegu	6883	62.95
2	Gyeongsangbuk-do	1379	12.61
3	Seoul	861	7.87
4	Gyeonggi-do	840	7.68
5	Incheon	206	1.88
6	Busan	147	1.34
7	Chungcheongnam-do	146	1.34
8	Gyeongsangnam-do	123	1.12
9	Chungcheongbuk-do	60	0.55
10	Gangwon-do	57	0.52
11	Ulsan	52	0.48
12	Sejong	47	0.43
13	Daejeon	46	0.42
14	Gwangju	32	0.29
15	Jeollabuk-do	21	0.19
16	Jeollanam-do	19	0.17
17	Jeju-do	15	0.14

Region

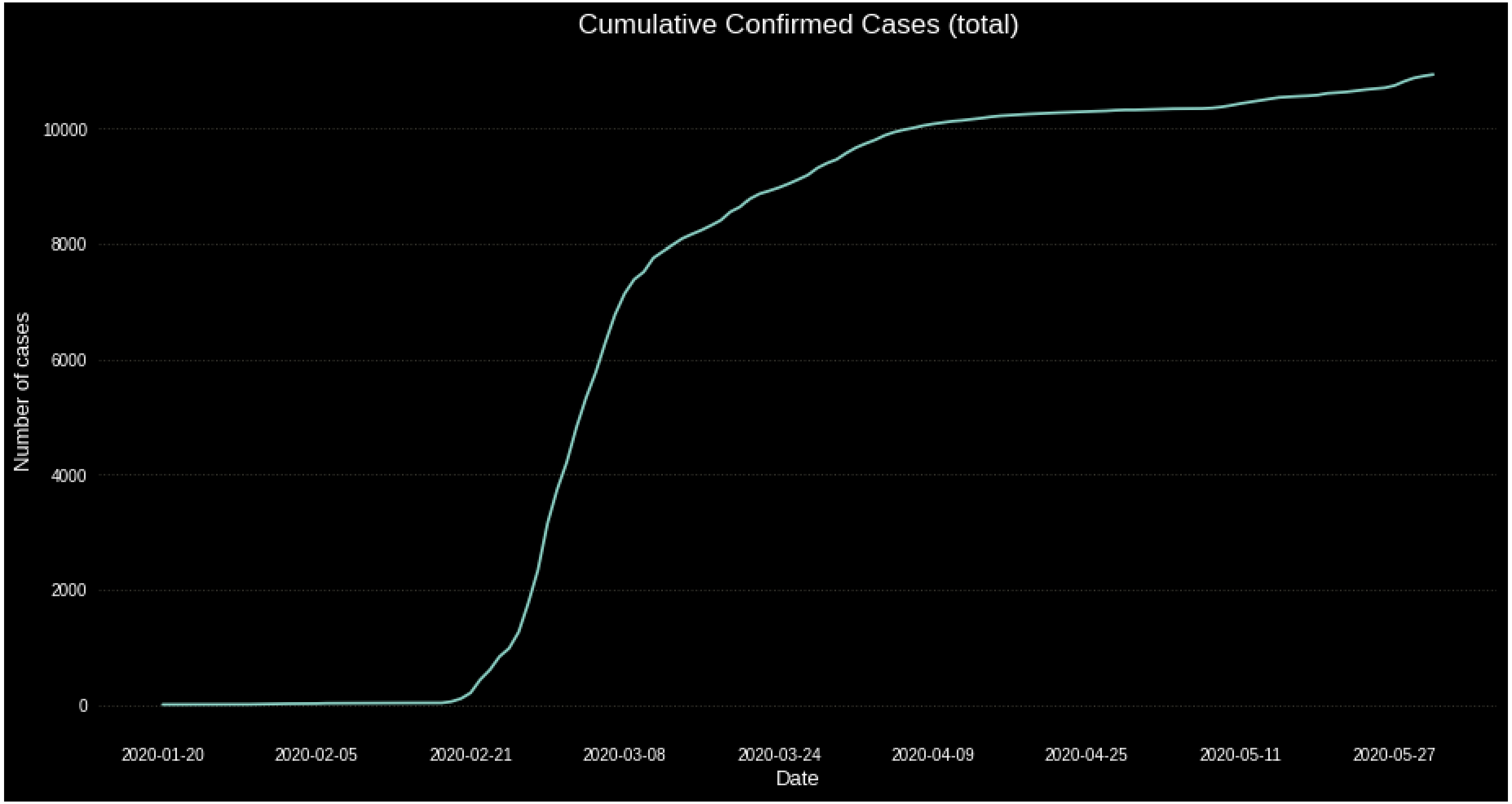
```
In [37]: loc_latest.loc['18',:] = loc_latest.iloc[6:, :].sum()
loc_latest.loc['18','province'] = 'Others'
loc_latest = loc_latest[loc_latest.proportion > loc_latest.iloc[5, 2]]
loc_latest

fig, ax = plt.subplots(figsize=(11, 11))
plt.title(f'Confirmed Cases Distribution by Region ({last_update})', fontsize=17)
pop_circle=plt.Circle((0,0), 0.79, color='black')
plt.pie(loc_latest.proportion
        , labels=loc_latest.province
        , autopct='%.2f%%'
        , startangle=90
        , counterclock=False
       )
p=plt.gcf()
p.gca().add_artist(pop_circle)
plt.show()
```



Region

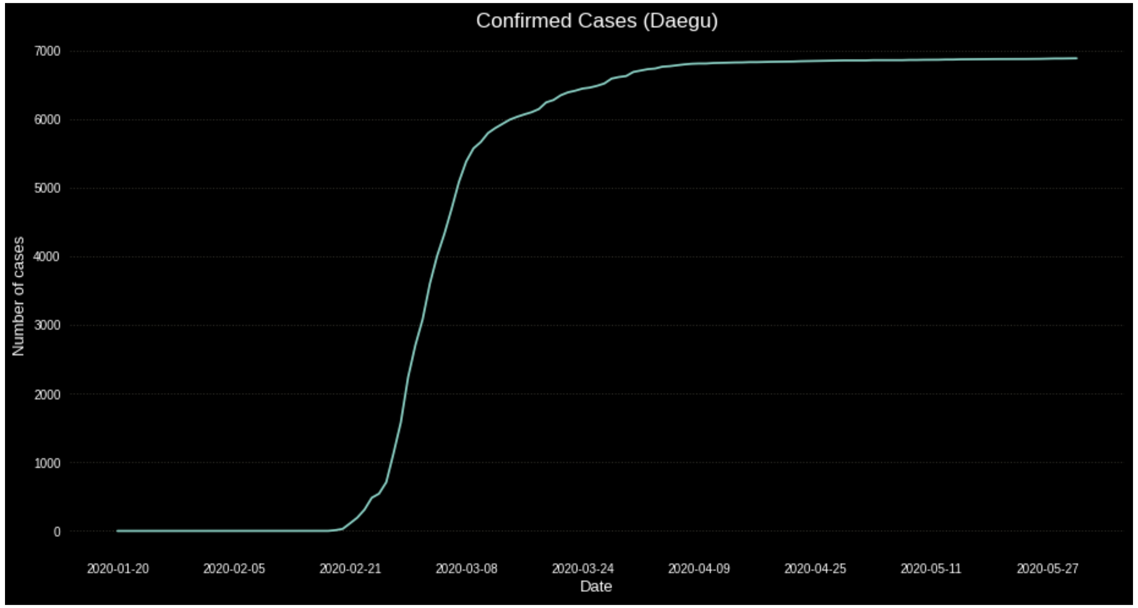
In [38]:

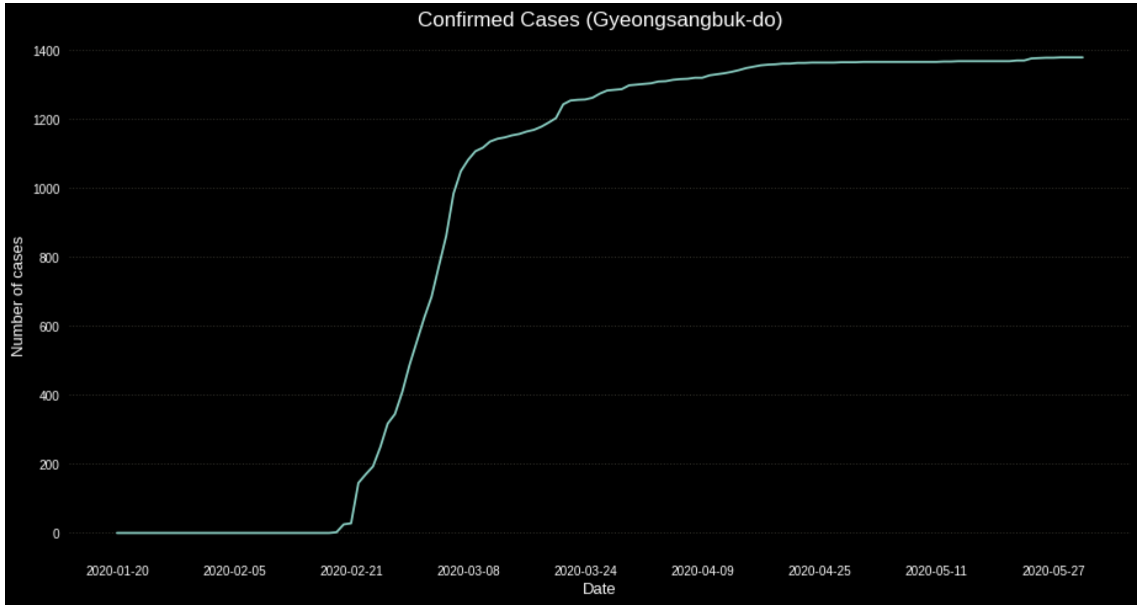
```python
total_list = region_raw.groupby('date').sum().confirmed

fig, ax = plt.subplots(figsize=(13, 7))
plt.title('Cumulative Confirmed Cases (total)', fontsize=17)
ax.set_xlabel('Date', size=13)
ax.set_ylabel('Number of cases', size=13)
plt.plot(region_raw.date.unique()
         , region_raw.groupby('date').sum().confirmed)
ax.set_xticks(ax.get_xticks()[::int(len(region_raw.date.unique())/8)])
plt.show()
```



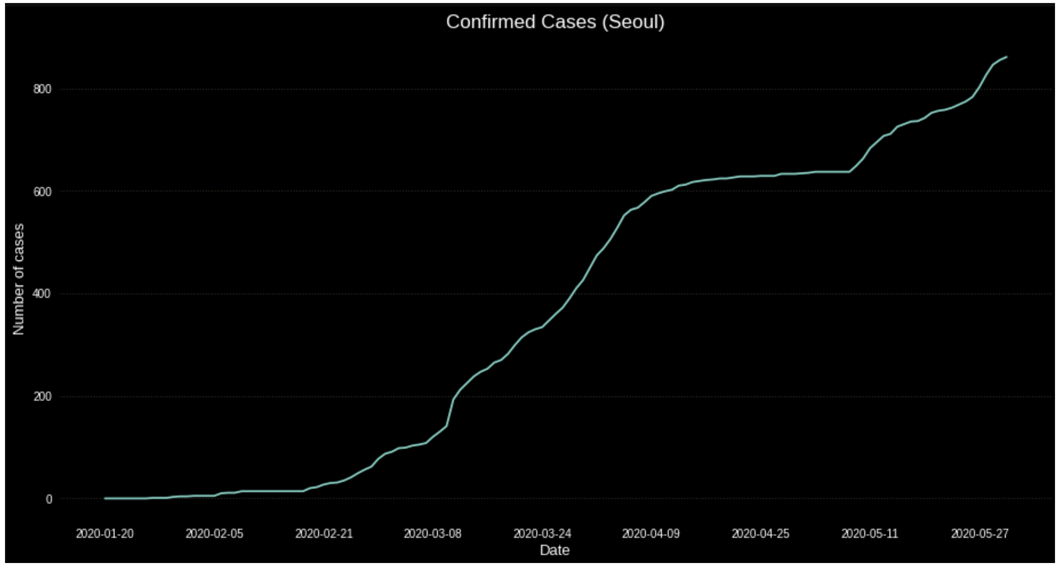
Region

```
In [41]: if not fast:
    print('[Confirmed Cases in each region (most to least)]')
    region_list = region_raw[region_raw.date==region_raw.date.iloc[-1]]\
                            .sort_values('confirmed', ascending = False)\
                            .province

    for region_name in region_list:
        fig, ax = plt.subplots(figsize=(13, 7))
        plt.title(f'Confirmed Cases ({region_name})', fontsize=17)
        ax.set_xlabel('Date', size=13)
        ax.set_ylabel('Number of cases', size=13)
        region_each = region_raw[region_raw.province==region_name]
        plt.plot(region_each.date.unique(), region_each.confirmed)
        ax.set_xticks(ax.get_xticks()[::int(len(region_each.date.unique())/8)])
        plt.show()
```

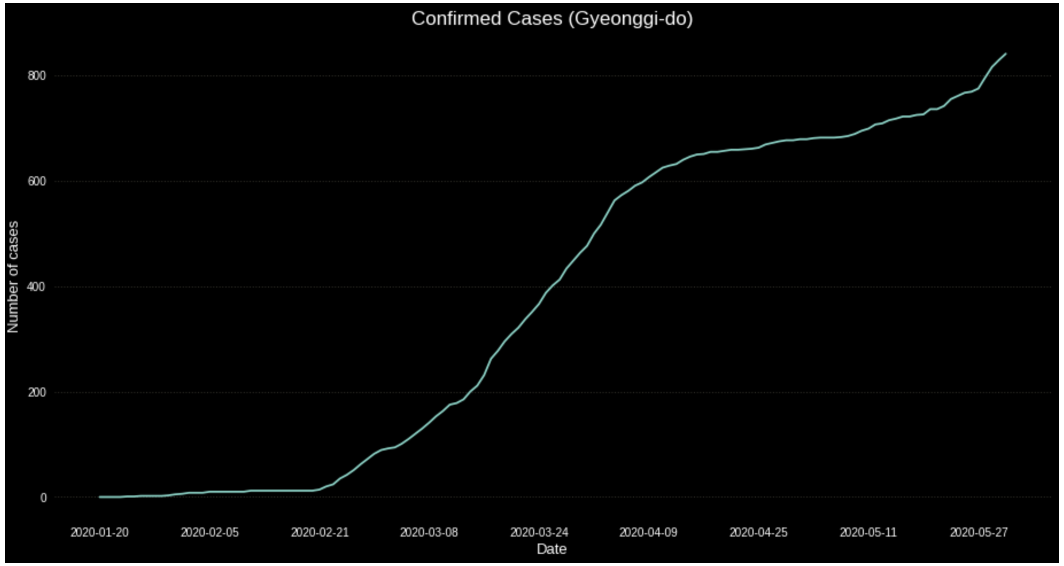




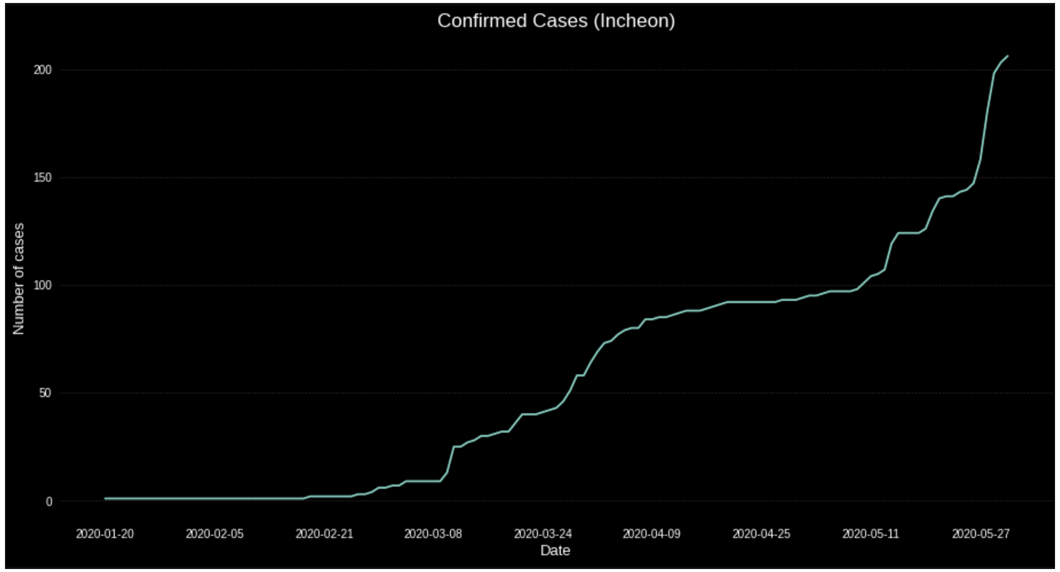
Region



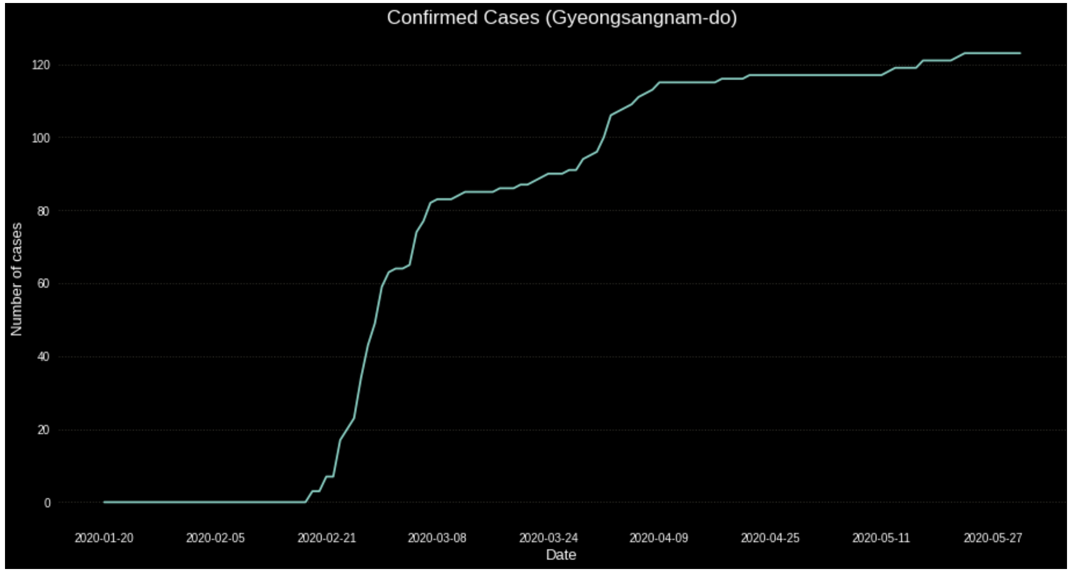
Confirmed Cases (Seoul)
Number of cases
Date



Confirmed Cases (Gyeonggi-do)
Number of cases
Date

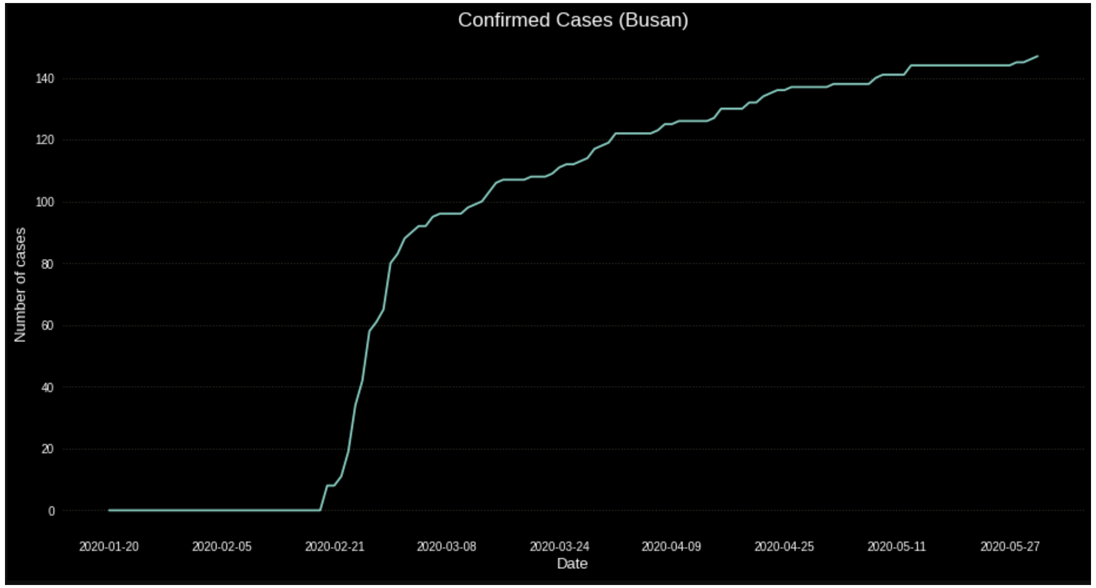


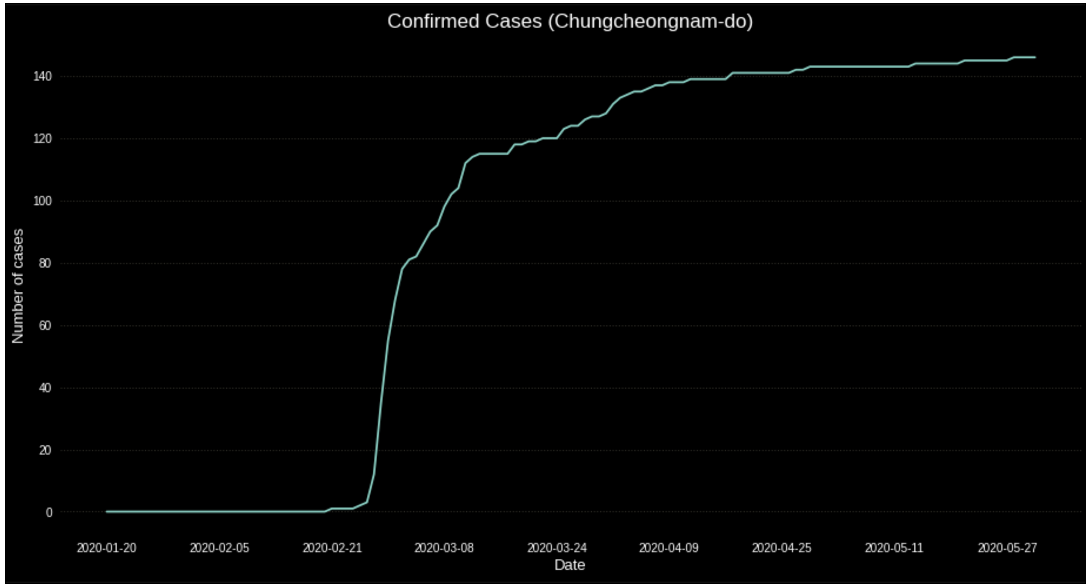
Confirmed Cases (Incheon)
Number of cases
Date

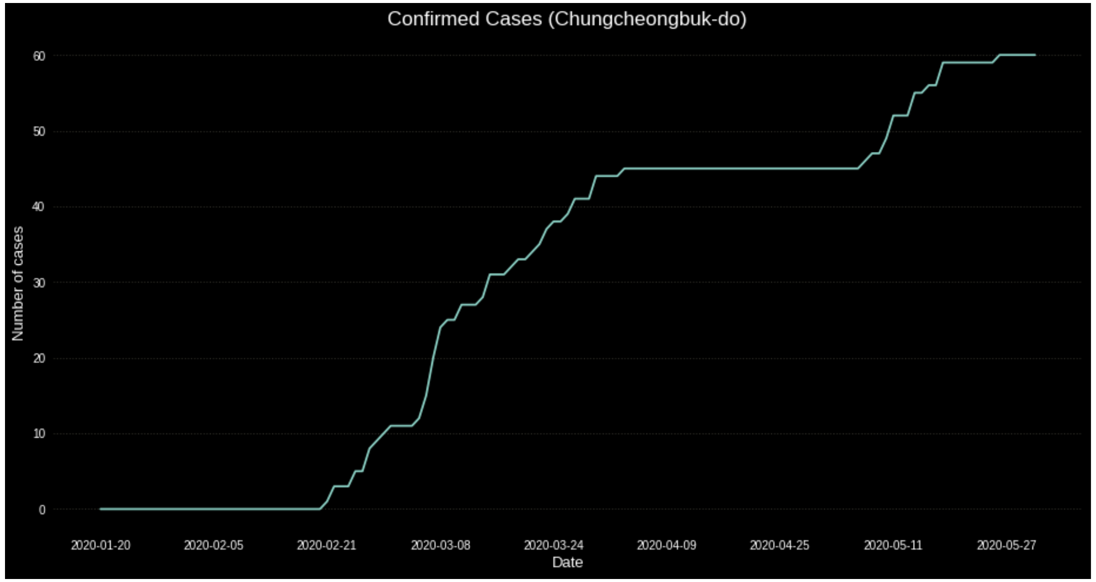


Confirmed Cases (Gyeongsangnam-do)
Number of cases
Date

Region

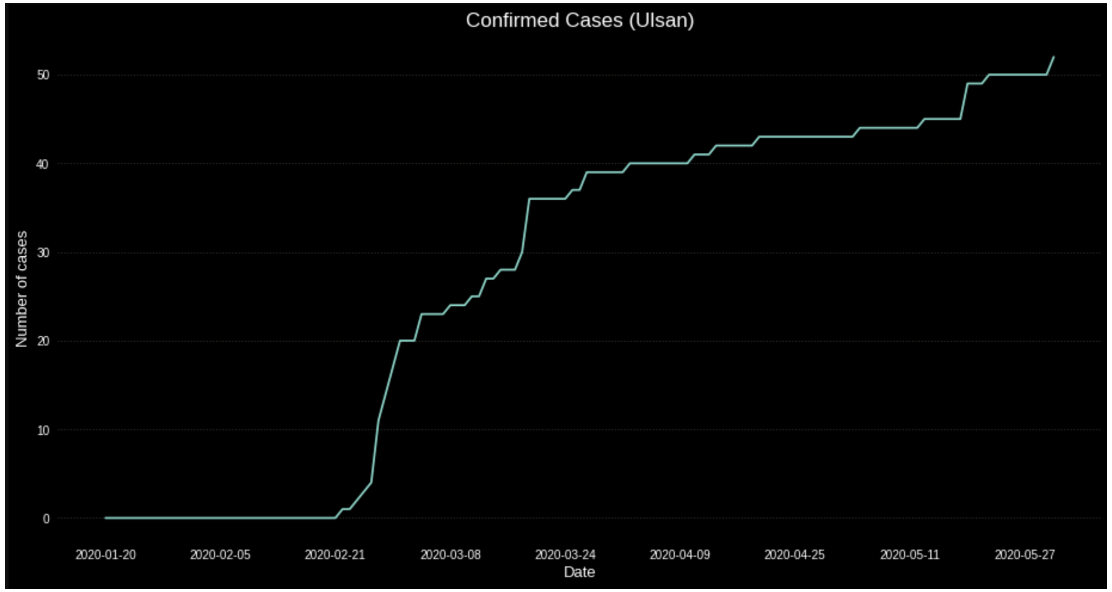

Confirmed Cases (Busan)
Number of cases
0
20
40
60
80
100
120
140
2020-01-20
2020-02-05
2020-02-21
2020-03-08
2020-03-24
2020-04-09
2020-04-25
2020-05-11
2020-05-27
Date

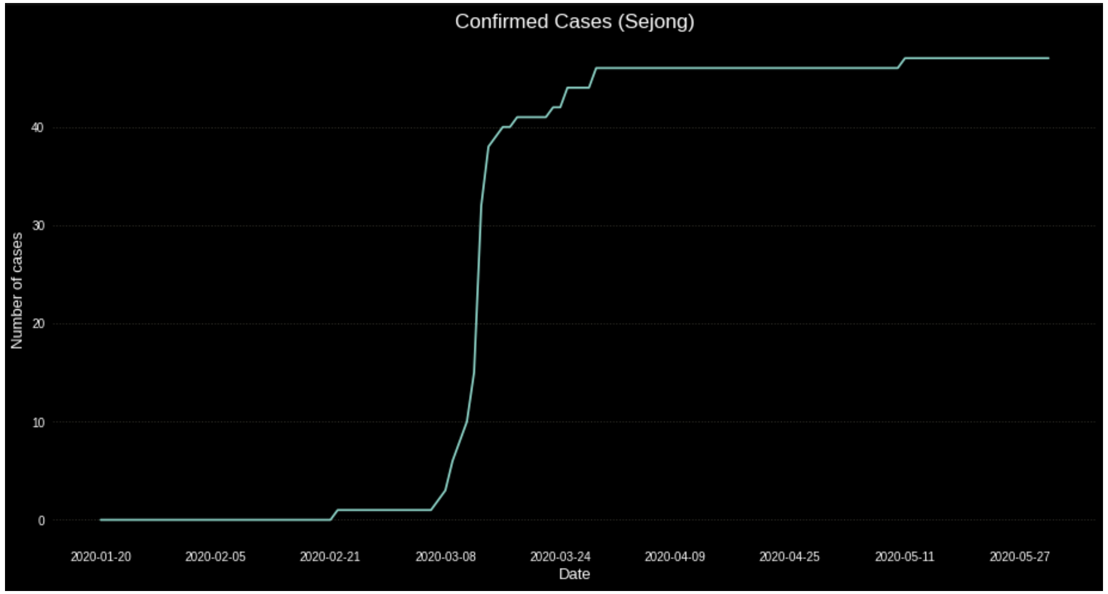

Confirmed Cases (Chungcheongnam-do)
Number of cases
0
20
40
60
80
100
120
140
2020-01-20
2020-02-05
2020-02-21
2020-03-08
2020-03-24
2020-04-09
2020-04-25
2020-05-11
2020-05-27
Date

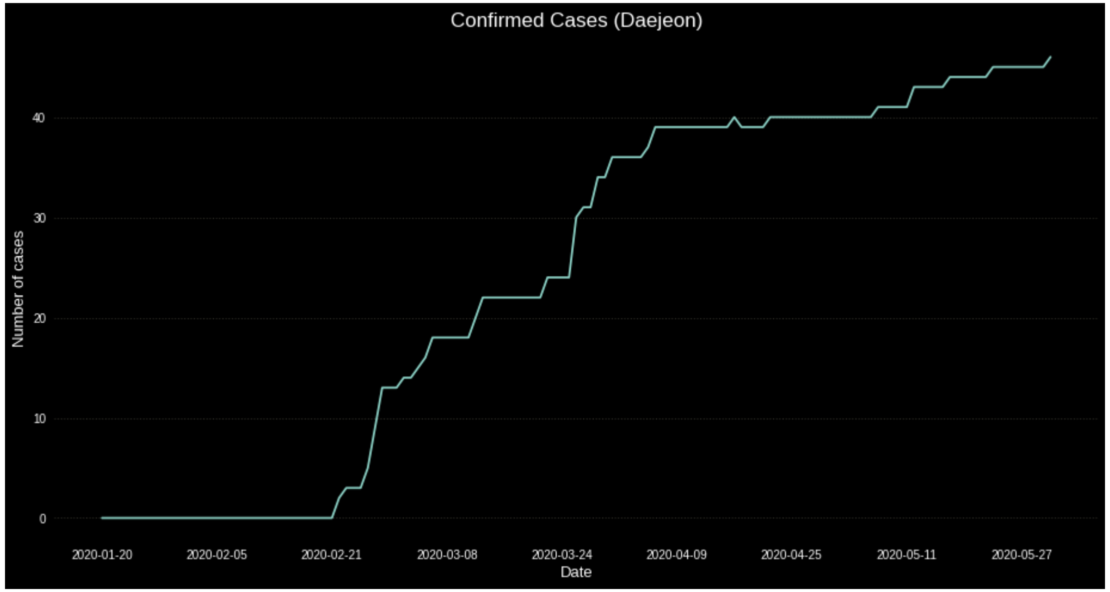

Confirmed Cases (Chungcheongbuk-do)
Number of cases
0
10
20
30
40
50
60
2020-01-20
2020-02-05
2020-02-21
2020-03-08
2020-03-24
2020-04-09
2020-04-25
2020-05-11
2020-05-27
Date

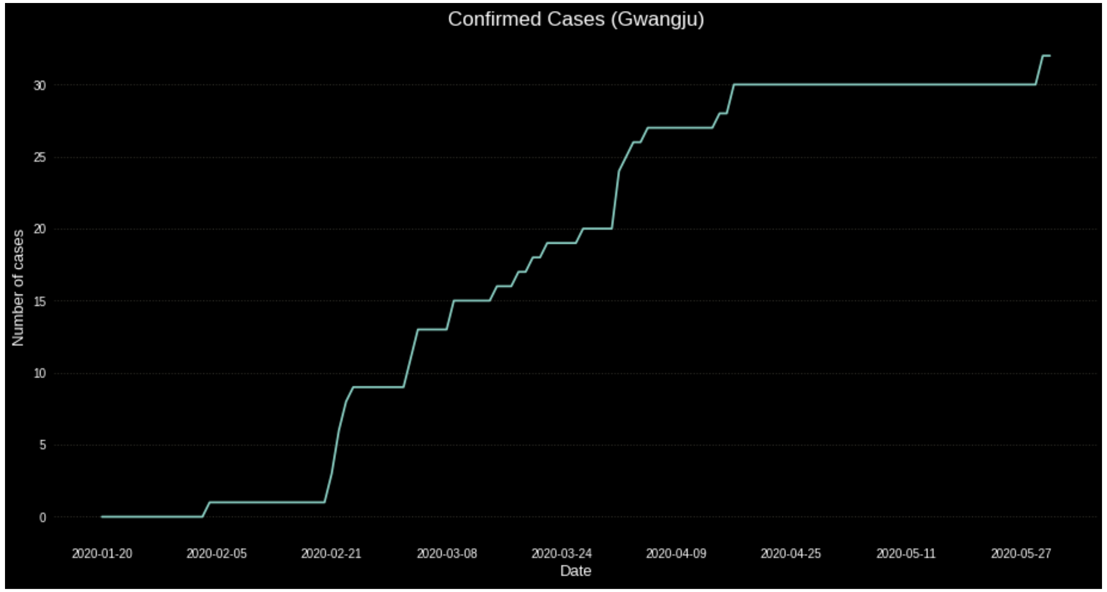

Confirmed Cases (Gangwon-do)
Number of cases
0
10
20
30
40
50
2020-01-20
2020-02-05
2020-02-21
2020-03-08
2020-03-24
2020-04-09
2020-04-25
2020-05-11
2020-05-27
Date

Region

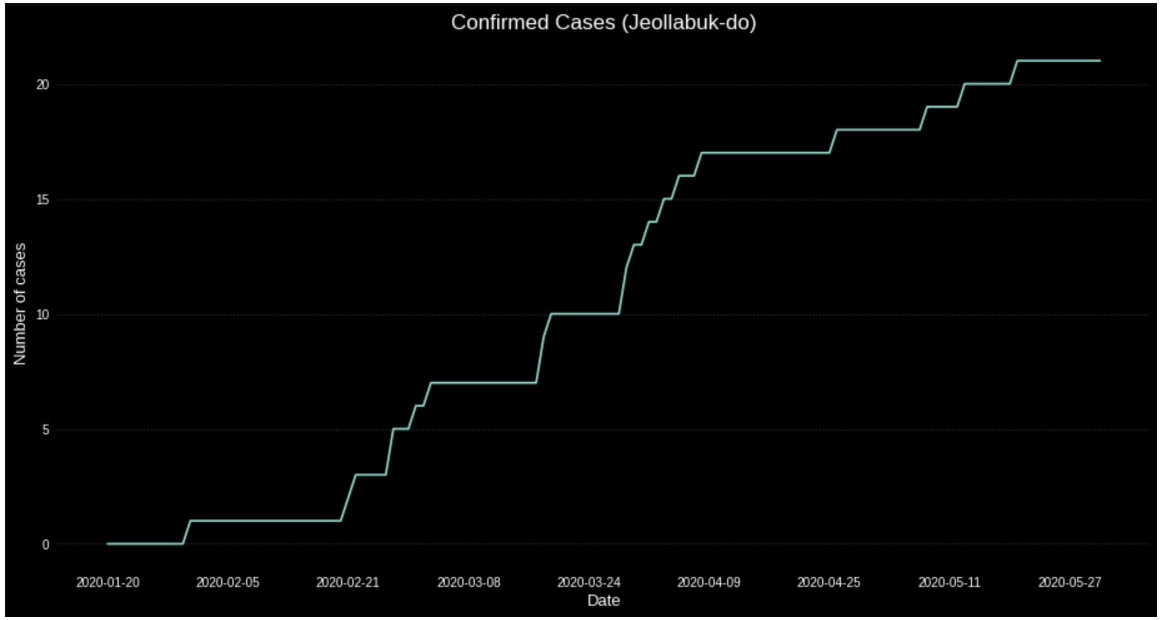

Confirmed Cases (Ulsan)
Number of cases
0
10
20
30
40
50
2020-01-20
2020-02-05
2020-02-21
2020-03-08
2020-03-24
2020-04-09
2020-04-25
2020-05-11
2020-05-27
Date


Confirmed Cases (Sejong)
Number of cases
0
10
20
30
40
2020-01-20
2020-02-05
2020-02-21
2020-03-08
2020-03-24
2020-04-09
2020-04-25
2020-05-11
2020-05-27
Date

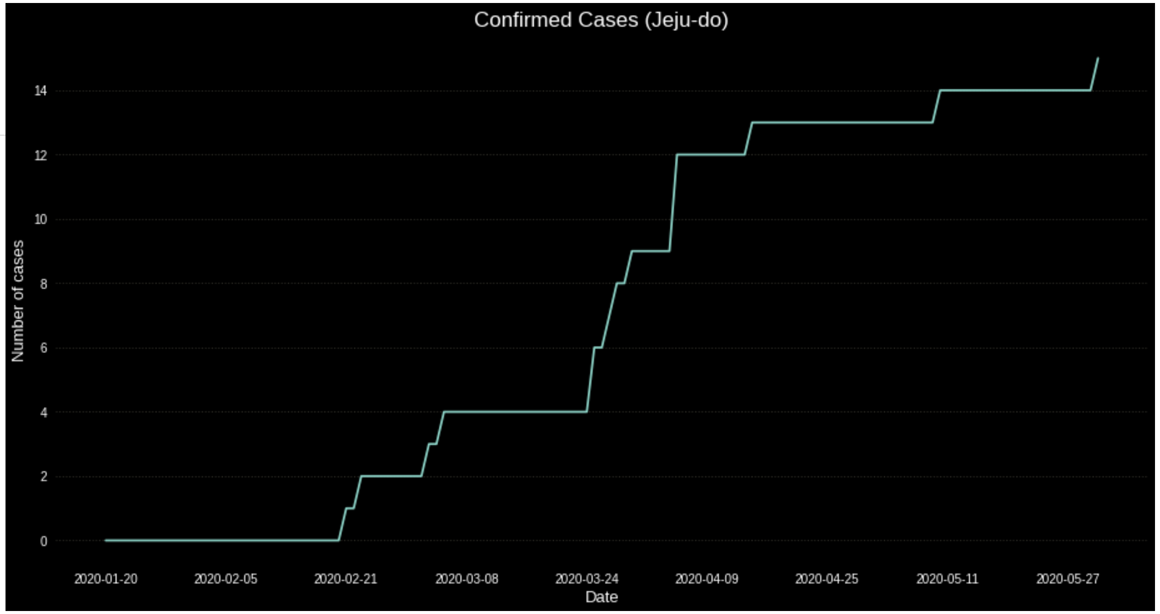

Confirmed Cases (Daejeon)
Number of cases
0
10
20
30
40
2020-01-20
2020-02-05
2020-02-21
2020-03-08
2020-03-24
2020-04-09
2020-04-25
2020-05-11
2020-05-27
Date


Confirmed Cases (Gwangju)
Number of cases
0
5
10
15
20
25
30
2020-01-20
2020-02-05
2020-02-21
2020-03-08
2020-03-24
2020-04-09
2020-04-25
2020-05-11
2020-05-27
Date

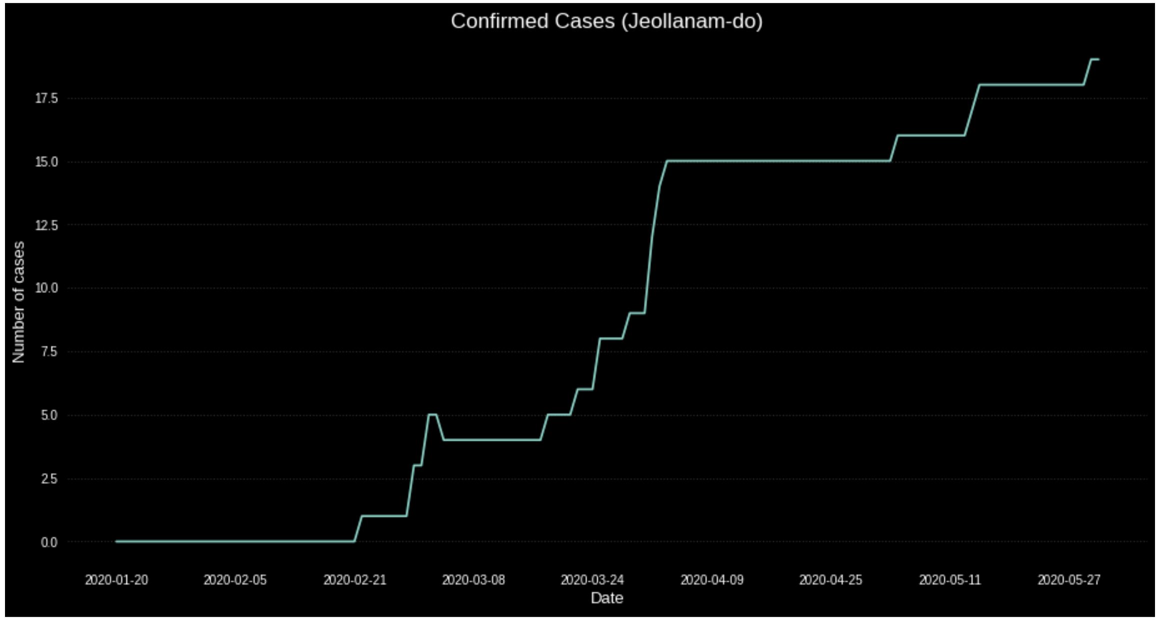
Region



Confirmed Cases (Jeollabuk-do)
Number of cases
Date
0
5
10
15
20
2020-01-20
2020-02-05
2020-02-21
2020-03-08
2020-03-24
2020-04-09
2020-04-25
2020-05-11
2020-05-27



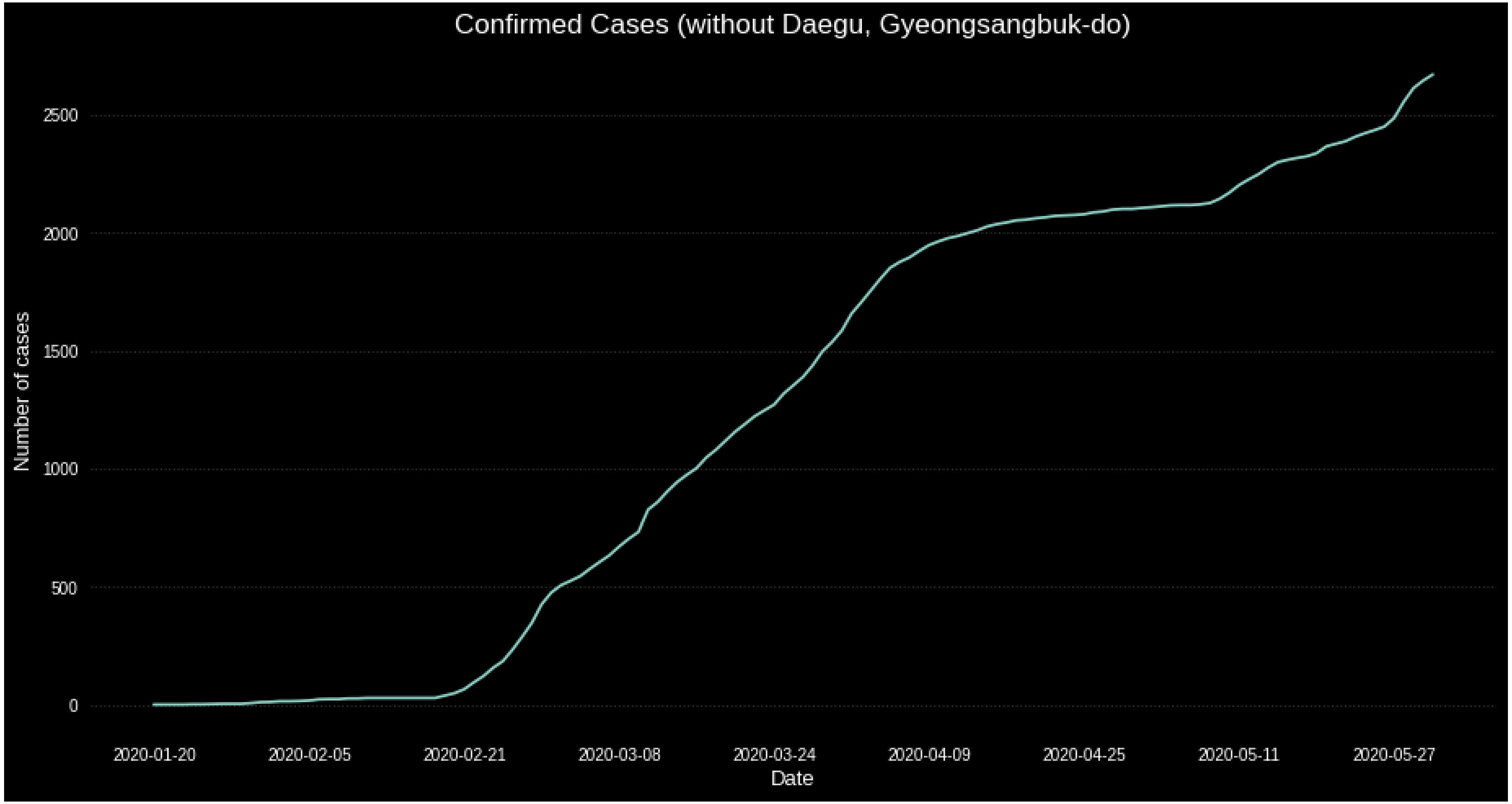
Confirmed Cases (Jeju-do)
Number of cases
Date
0
2
4
6
8
10
12
14
2020-01-20
2020-02-05
2020-02-21
2020-03-08
2020-03-24
2020-04-09
2020-04-25
2020-05-11
2020-05-27



Confirmed Cases (Jeollanam-do)
Number of cases
Date
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
2020-01-20
2020-02-05
2020-02-21
2020-03-08
2020-03-24
2020-04-09
2020-04-25
2020-05-11
2020-05-27

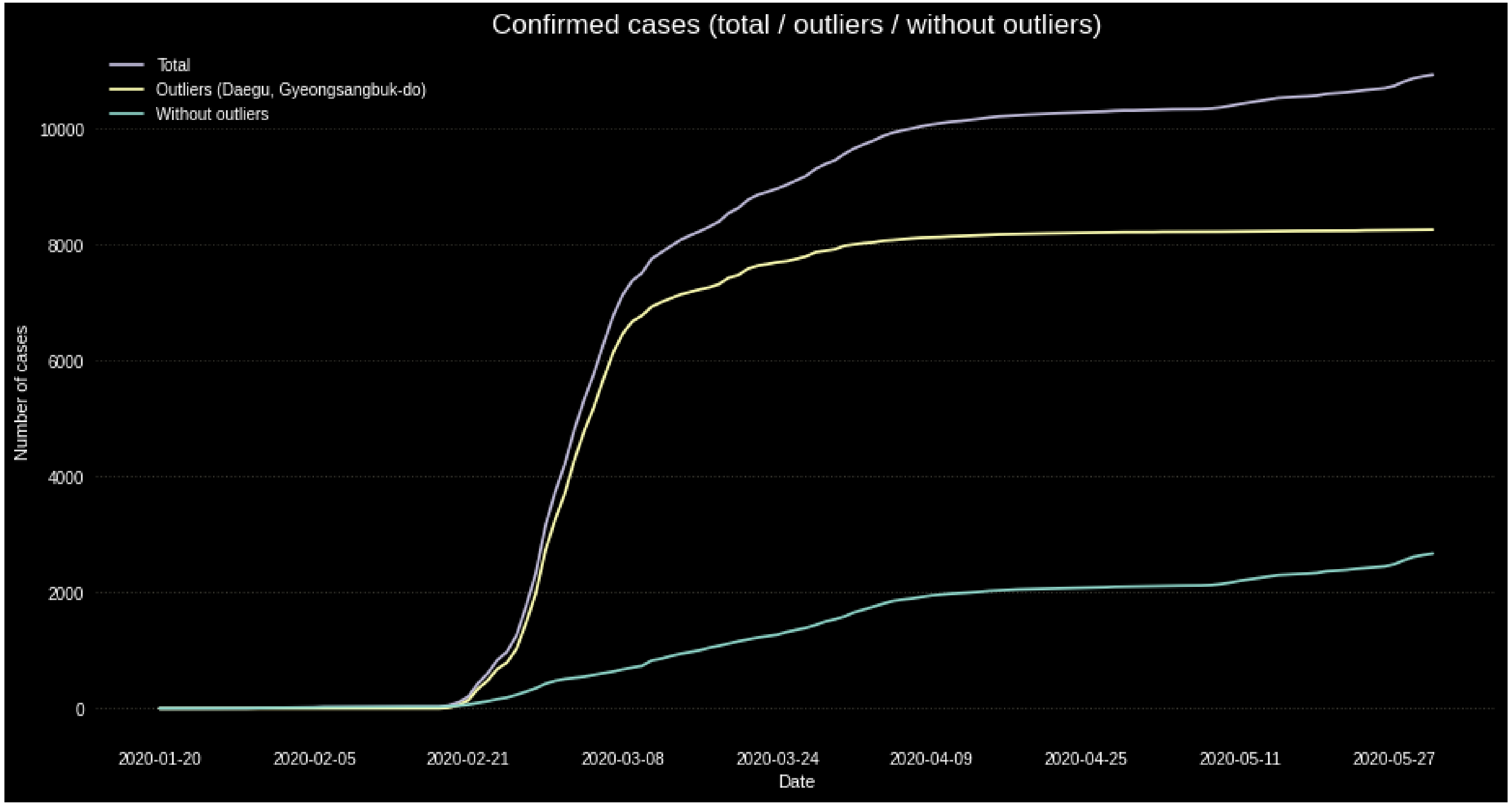
Region

In [42]:
```
wo_outliers = region_raw[(region_raw.province!='Daegu')
                          & (region_raw.province!='Gyeongsangbuk-do')
                         ].groupby('date').sum().confirmed

fig, ax = plt.subplots(figsize=(13, 7))
plt.title('Confirmed Cases (without Daegu, Gyeongsangbuk-do)', fontsize=17)
ax.set_xlabel('Date', size=13)
ax.set_ylabel('Number of cases', size=13)
plt.plot(region_raw.date.unique(), wo_outliers)
ax.set_xticks(ax.get_xticks()[::int(len(region_raw.date.unique())/8)])
plt.show()
```



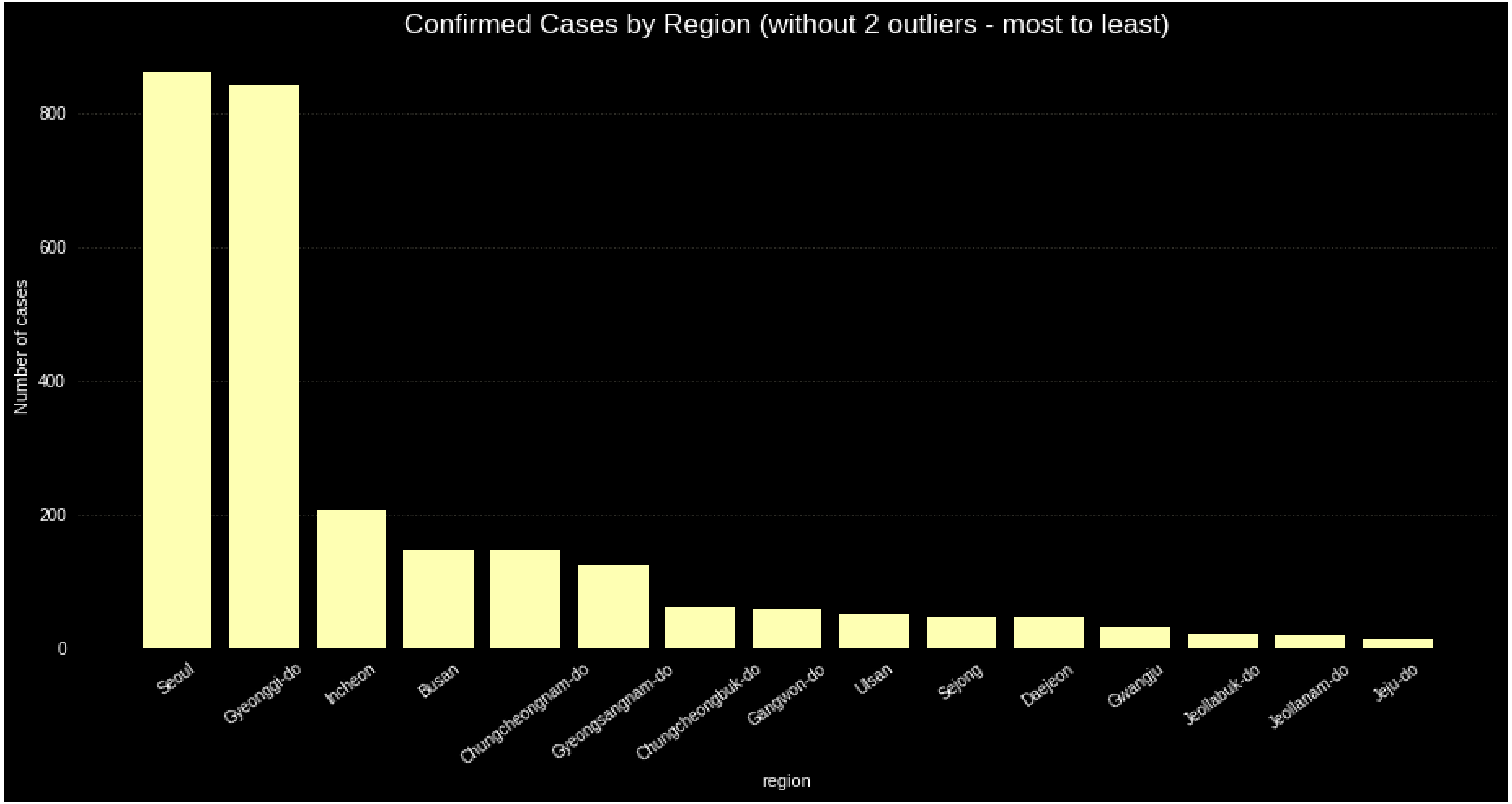
Region

```
In [43]: outliers = region_raw[(region_raw.province=='Daegu')#
                              | (region_raw.province=='Gyeongsangbuk-do')
                             ].groupby('date').sum().confirmed
fig, ax = plt.subplots(figsize=(13, 7))
plt.title('Confirmed cases (total / outliers / without outliers)', fontsize=17)
plt.plot(region_raw.date.unique(), total_list, color=color_list[2])
plt.plot(region_raw.date.unique(), outliers, color=color_list[1])
plt.plot(region_raw.date.unique(), wo_outliers, color=color_list[0])
ax.set_xticks(ax.get_xticks()[::int(len(region_raw.date.unique())/8)])
plt.xlabel('Date')
plt.ylabel('Number of cases')
plt.legend(['Total', 'Outliers (Daegu, Gyeongsangbuk-do)', 'Without outliers'])

plt.show()
```



Sex

```
fig, ax = plt.subplots(figsize=(13, 7))
plt.title('Confirmed Cases by Region (without 2 outliers - most to least)', fontsize=17)
plt.xticks(rotation=37)
plt.bar(loc_latest_all[2:].province
        , loc_latest_all[2:].confirmed
        , color=color_list[1])
plt.xlabel('region')
plt.ylabel('Number of cases')
plt.show()
```



Sex

```
In [54]: sex_raw = get_data(file_paths[3])
         data_range(sex_raw, 'date')
```

[Sample data]

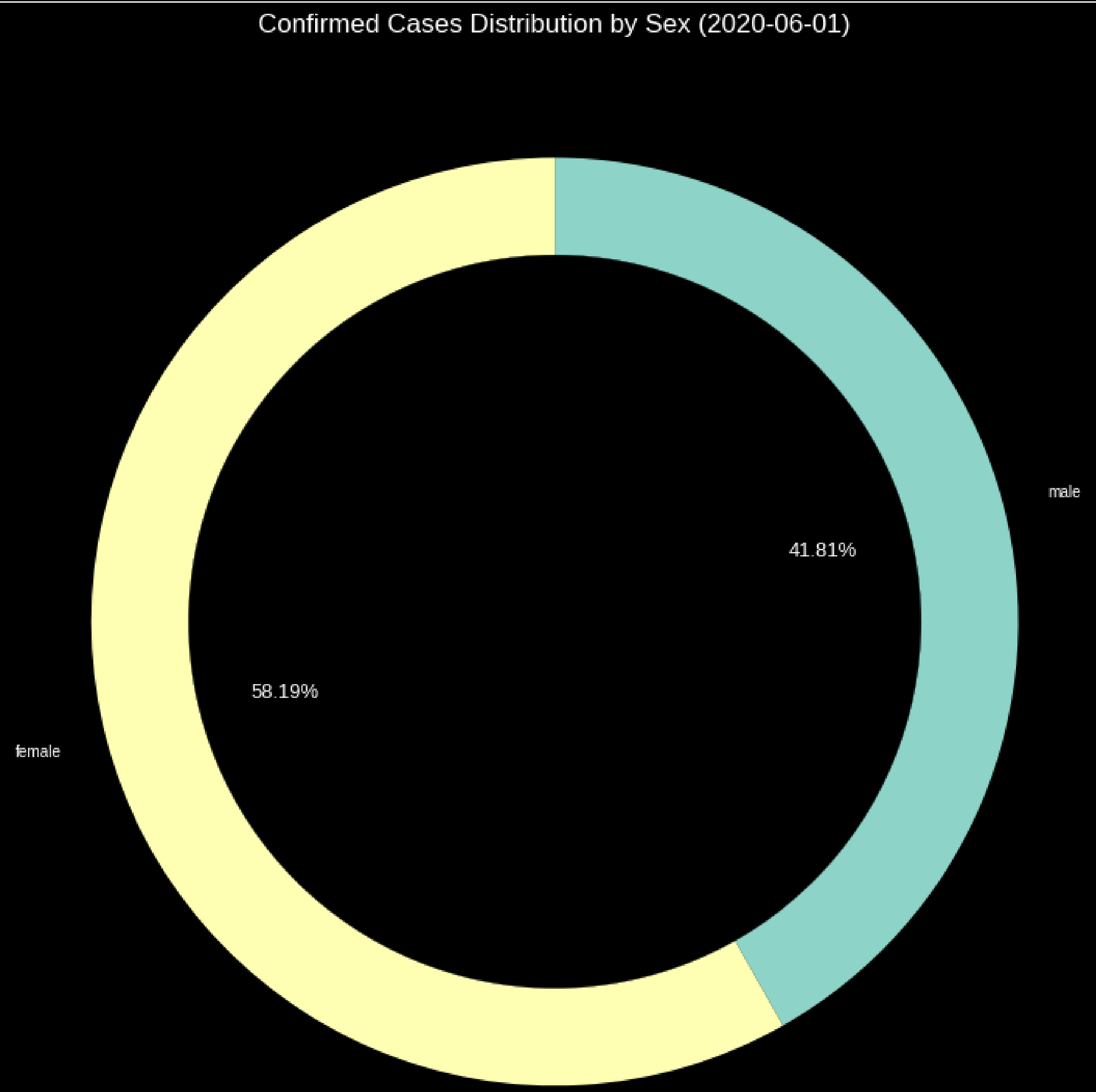
	date	time	sex	confirmed	deceased
0	2020-03-02	0	male	1591	13
1	2020-03-02	0	female	2621	9
2	2020-03-03	0	male	1810	16
179	2020-05-30	0	female	6661	127
180	2020-05-31	0	male	4795	143
181	2020-05-31	0	female	6673	127

```
Date range: 91 days
# 2020-03-02 to 2020-05-31
```

Sex

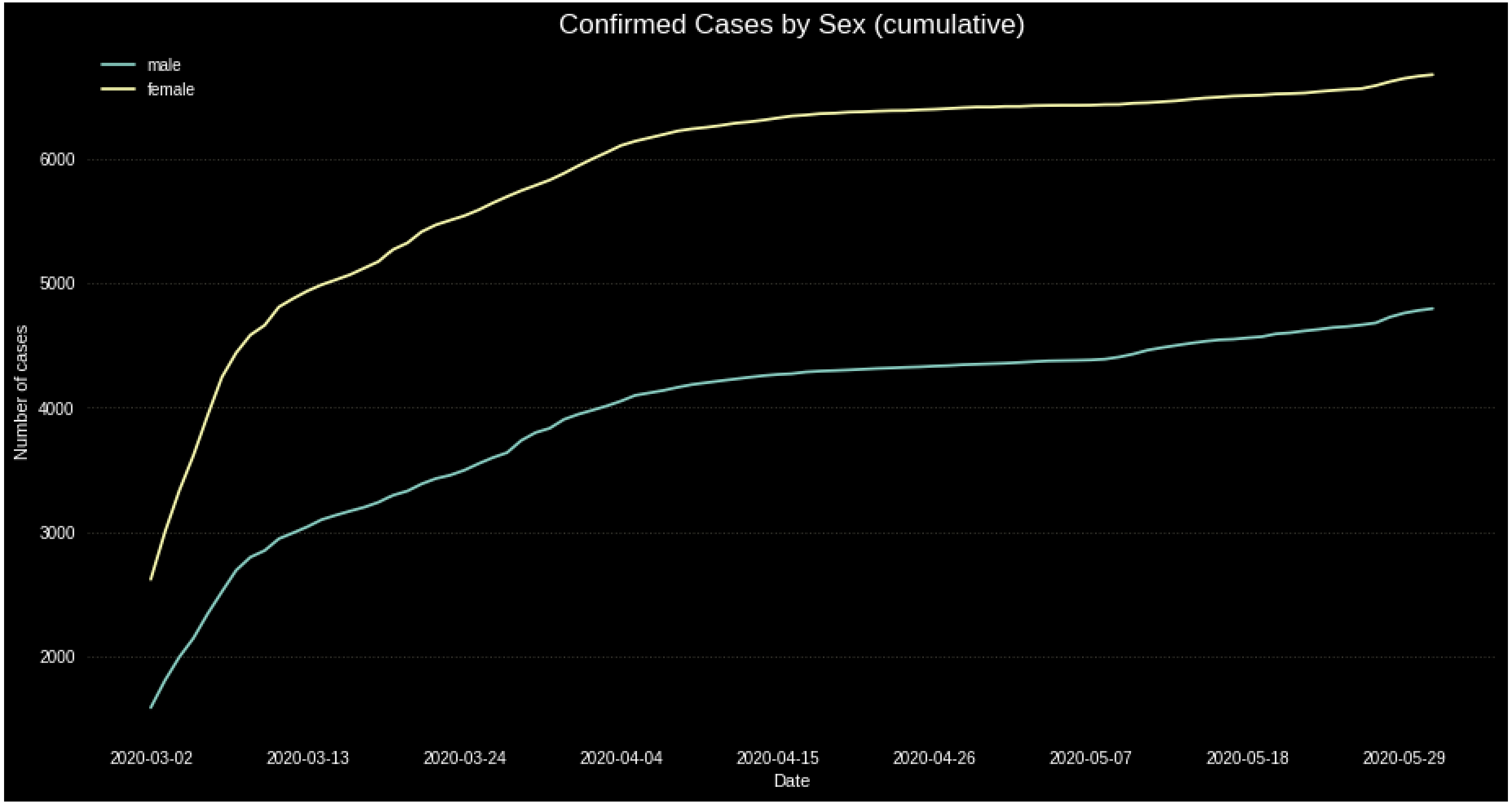
In [55]:

```
fig, ax = plt.subplots(figsize=(11, 11))
plt.title(f'Confirmed Cases Distribution by Sex ({last_update})', fontsize=17)
pop_circle=plt.Circle((0,0), 0.79, color='black')
plt.pie(sex_raw.confirmed[-2:]
        , labels=['male', 'female']
        , autopct='%.2f%%'
        , startangle=90
        , counterclock=False)
p=plt.gcf()
p.gca().add_artist(pop_circle)
plt.show()
```



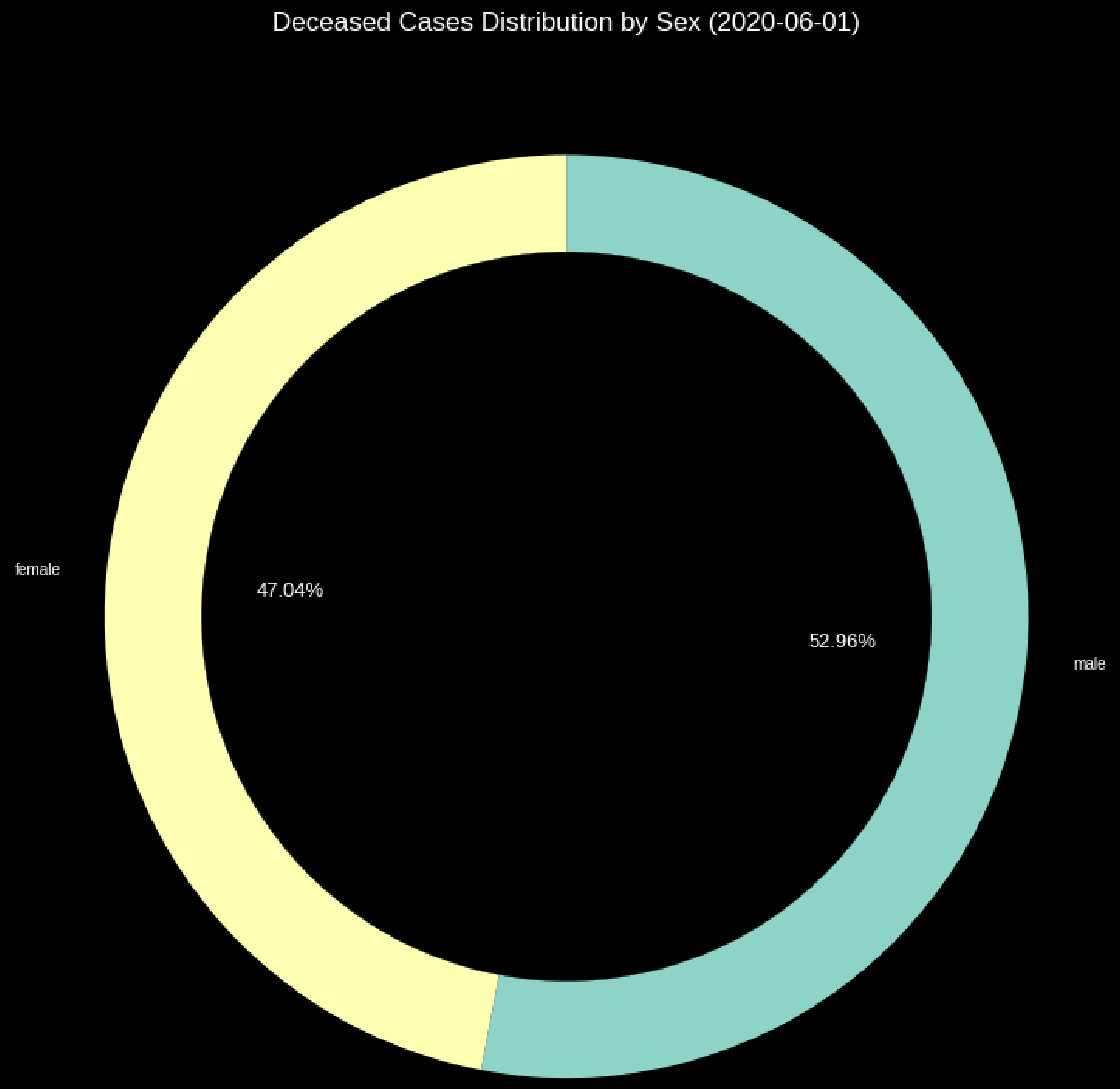
Sex

```
fig, ax = plt.subplots(figsize=(13, 7))
plt.title('Confirmed Cases by Sex (cumulative)', fontsize=17)
sex_confirmed = (sex_raw[sex_raw.sex=='male'].confirmed, sex_raw[sex_raw.sex=='female'].confirmed)
for sex_each, sex_label in zip(sex_confirmed, ['male', 'female']):
    plt.plot(sex_raw.date.unique(), sex_each, label=sex_label)
ax.set_xticks(ax.get_xticks()[::int(len(sex_raw.date.unique())/8)])
plt.xlabel('Date')
plt.ylabel('Number of cases')
ax.legend()
plt.show()
```



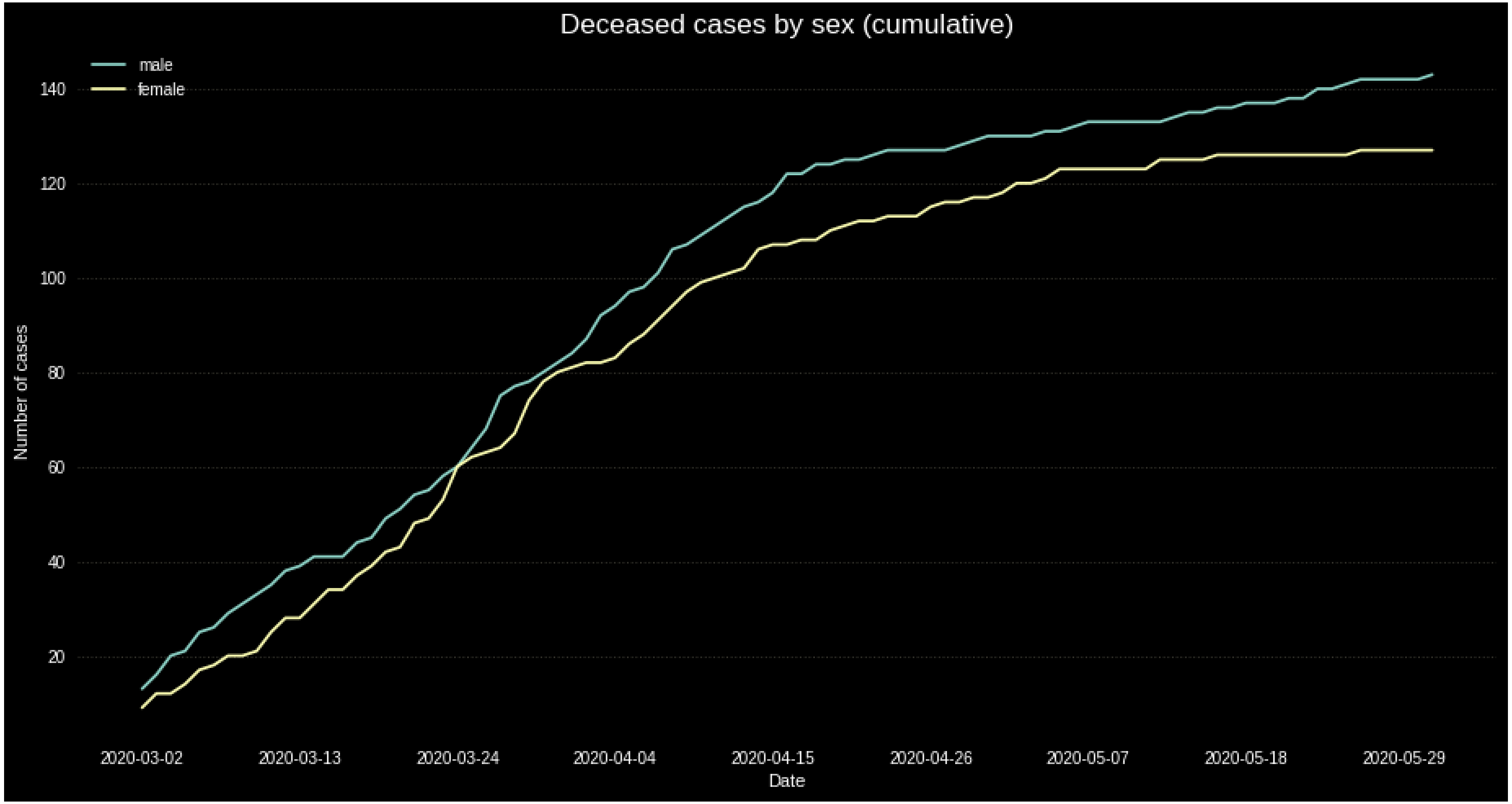
Sex

```
fig, ax = plt.subplots(figsize=(11, 11))
plt.title(f'Deceased Cases Distribution by Sex ({last_update})', fontsize=17)
pop_circle=plt.Circle((0,0), 0.79, color='black')
plt.pie(sex_raw.deceased[-2:]
        , labels=['male', 'female']
        , autopct='%.2f%%'
        , startangle=90
        , counterclock=False)
p=plt.gcf()
p.gca().add_artist(pop_circle)
plt.show()
```



Sex

```
fig, ax = plt.subplots(figsize=(13, 7))
plt.title('Deceased cases by sex (cumulative)', fontsize=17)
sex_deceased = (sex_raw[sex_raw.sex=='male'].deceased, sex_raw[sex_raw.sex=='female'].deceased)

for sex_each, sex_label in zip(sex_deceased, ['male', 'female']):
    plt.plot(sex_raw.date.unique(), sex_each, label=sex_label)
ax.set_xticks(ax.get_xticks()[::int(len(sex_raw.date.unique())/8)])
plt.xlabel('Date')
plt.ylabel('Number of cases')
ax.legend()
plt.show()
```



Sex

```
fig, ax = plt.subplots(figsize=(11, 11))
plt.title('Population Sex Balance (2020-02)', fontsize=17)
pop_circle=plt.Circle((0,0), 0.79, color='black')
plt.pie([25984136, 25860491]
        , labels=['male', 'female']
        , autopct='%.2f%%'
        , startangle=90
        , counterclock=False)
p=plt.gcf()
p.gca().add_artist(pop_circle)
plt.show()
```

Weather

```
weather_raw = get_data(file_paths[6], transpose=True)
data_range(weather_raw, 'date')
```

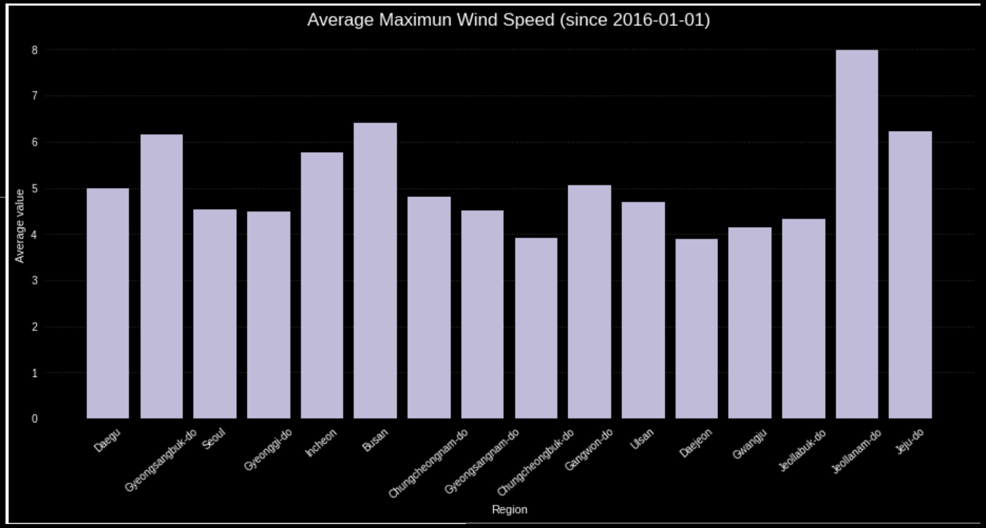
```
[Sample data]
```

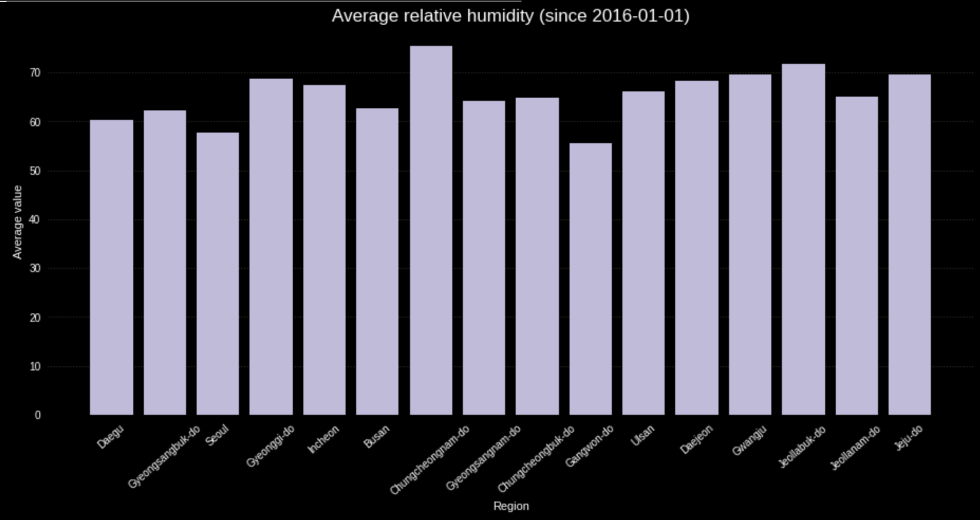
	0	1	2	25804	25805	25806
code	10000	11000	12000	60000	61000	70000
province	Seoul	Busan	Daegu	Gyeongsangbuk-do	Gyeongsangnam-do	Jeju-do
date	2016-01-01	2016-01-01	2016-01-01	2020-05-31	2020-05-31	2020-05-31
avg_temp	1.2	5.3	1.7	18.5	19.3	19.7
min_temp	-3.3	1.1	-4	11.2	14.6	17.1
max_temp	4	10.9	8	24.7	24.6	23.6
precipitation	0	0	0	0	0	0
max_wind_speed	3.5	7.4	3.7	5	4	6.2
most_wind_direction	90	340	270	180	140	70
avg_relative_humidity	73	52.1	70.5	77.4	71.5	88.1

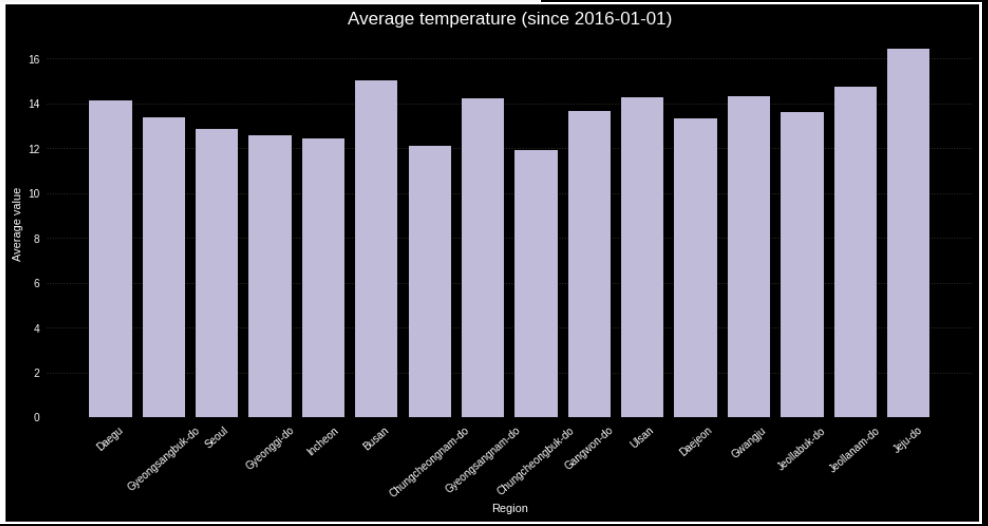
```
Date range: 1613 days
# 2016-01-01 to 2020-05-31
```

Weather

```python
weather_avg = pd.DataFrame(
    [weather_stat.index
     , weather_stat['avg_temp']
     , weather_stat['precipitation']
     , weather_stat['max_wind_speed']
     , weather_stat['avg_relative_humidity']]
    ).T
weather_avg.columns = ['region', 'temperature', 'precipitation'
                       , 'max_wind_speed', 'relative_humidity']

sorter = list(pop_meta.region[pop_meta.region != 'Sejong'].values)
weather_avg.region = weather_avg.region.astype('category')
weather_avg.region.cat.set_categories(sorter, inplace=True)
weather_avg = weather_avg.sort_values(['region'])
weather_avg.index = range(len(weather_raw.province.unique()))

title_list = ['Average temperature', 'Average Maximun Wind Speed', 'Average relative humidity']
for col, title in zip(weather_avg.columns[[1, 3, 4]], title_list):
    plt.figure(figsize=(13, 7))
    plt.title(f'{title} (since 2016-01-01)', fontsize=17)
    plt.xticks(rotation=41)
    plt.bar(weather_avg.region, weather_avg[col], color=color_list[2])
    plt.xlabel('Region')
    plt.ylabel('Average value')
    plt.show()
```

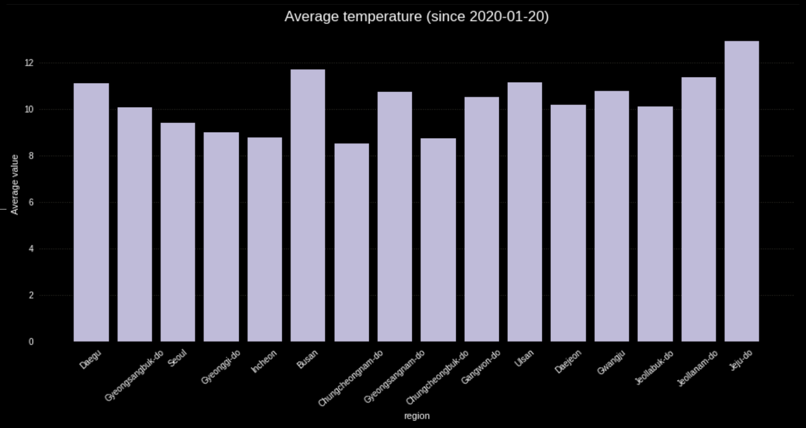


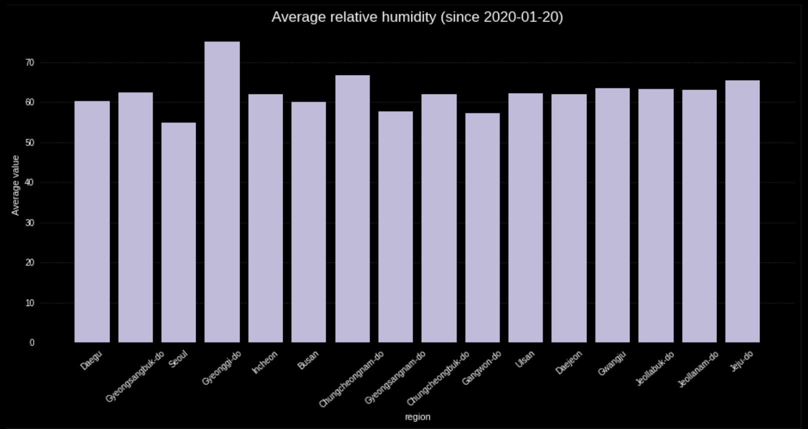


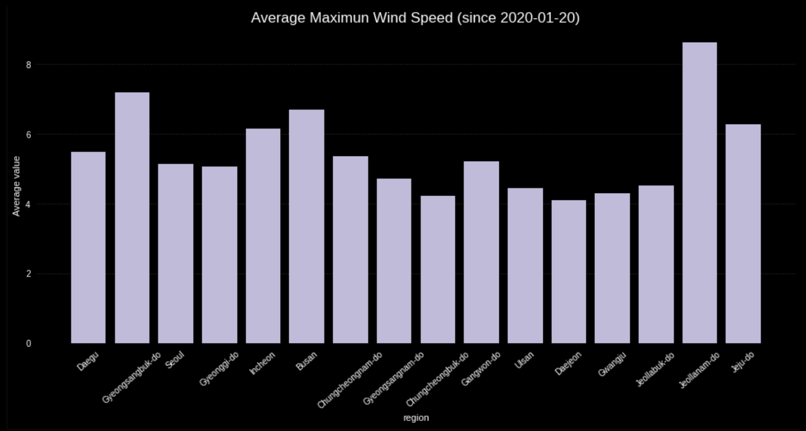


Weather

```python
## 1. Create a dataframe
weather_covid = weather_raw[weather_raw.date >= '2020-01-20']
weather_cov_stat = weather_covid.loc[:, 'province':].groupby('province').mean()
weather_cov_avg = pd.DataFrame(
    [weather_cov_stat.index
     , weather_cov_stat['avg_temp']
     , weather_cov_stat['precipitation']
     , weather_cov_stat['max_wind_speed']
     , weather_cov_stat['avg_relative_humidity']]
    ).T

## 2. Order by confirmed cases
weather_cov_avg.columns = ['region', 'temperature', 'precipitation'
                           , 'max_wind_speed', 'relative_humidity']
weather_cov_avg.region = weather_cov_avg.region.astype('category')
weather_cov_avg.region.cat.set_categories(sorter, inplace=True)
weather_cov_avg = weather_cov_avg.sort_values(['region'])

## 3. Plot values
title_list = ['Average temperature', 'Average Maximun Wind Speed', 'Average relative humidity']
for col, title in zip(weather_cov_avg.columns[[1, 3, 4]], title_list):
    plt.title(f'{title} (since 2020-01-20)', fontsize=17)
    plt.xticks(rotation=41)
    plt.bar(weather_avg.region, weather_cov_avg[col], color=color_list[2])
    plt.xlabel('region')
    plt.ylabel('Average value')
    plt.show()
```







Insight

Age(나이)
- 20대가 가장 감연된 연령대로 총 27.8%가 20대로 구성되어 있다.
- 70세 이상이 가장 사망한 연령그룹으로 전체의 약 48%를 차지한다.

Region(지역)
- 당연한 것일 수 있지만 인구가 많을수록 확진자 수가 증가한다.
- outlier 대구, 경북에 대해서 이상치가 발생한것을 확인 (신천지)

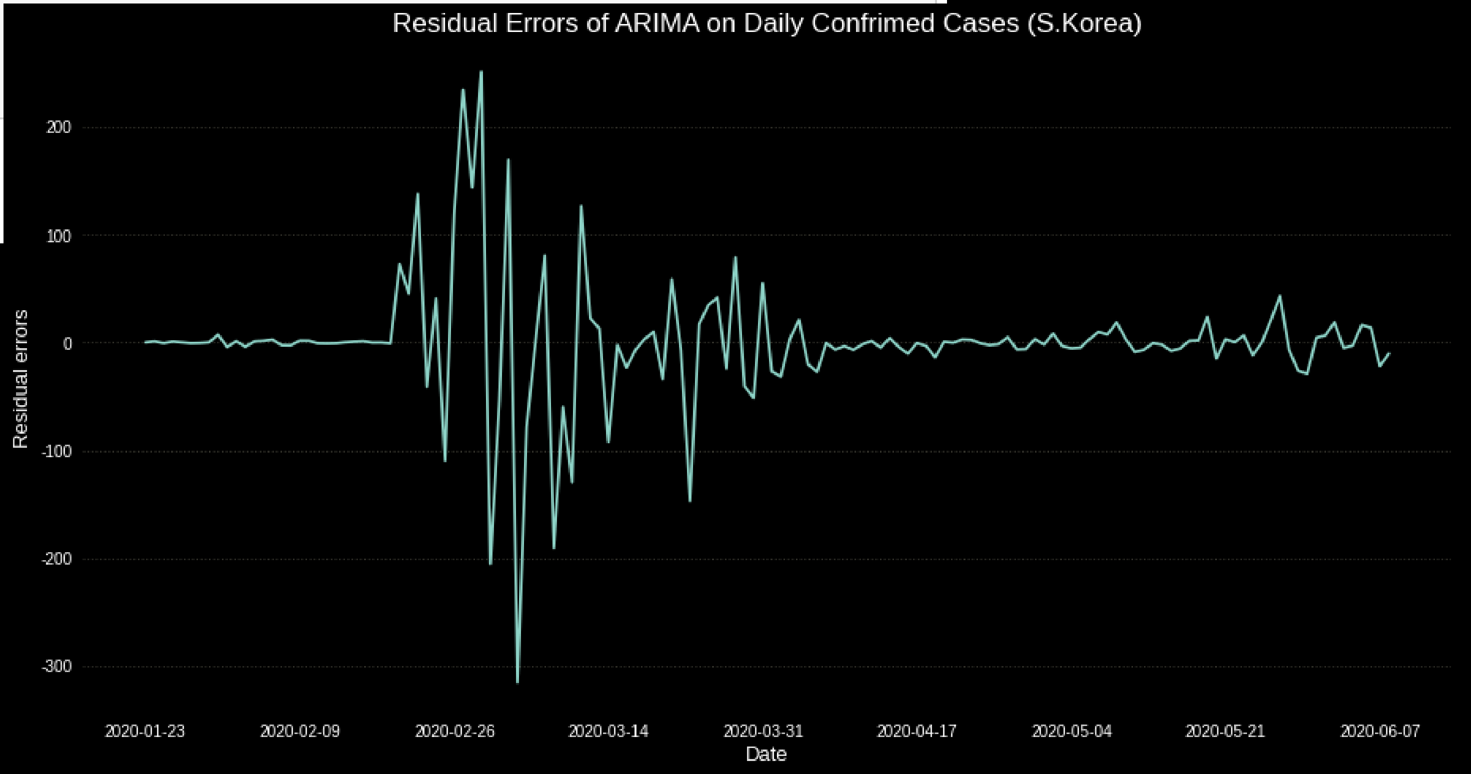
Sex(성별)
- 성별을 보면 남녀가의 유의할 정도로 큰 차이가 있지는 않다.
- 감염에 대해서는 여성이 59%를 남성이 41%를 차지한다.
- 사망에 대해서는 남성이 약 52%, 여성이 48%로 구성된다.

Weather(날씨)
- 날씨와 관련해서는 명확한 상관 관계가 없다.
- 온도에 대해서는 약한 음의 상관관계가 구성되어 있다.
- 풍속에 대해서는 약한 양의 상관관계가 구성되어 있다.

ARIMA

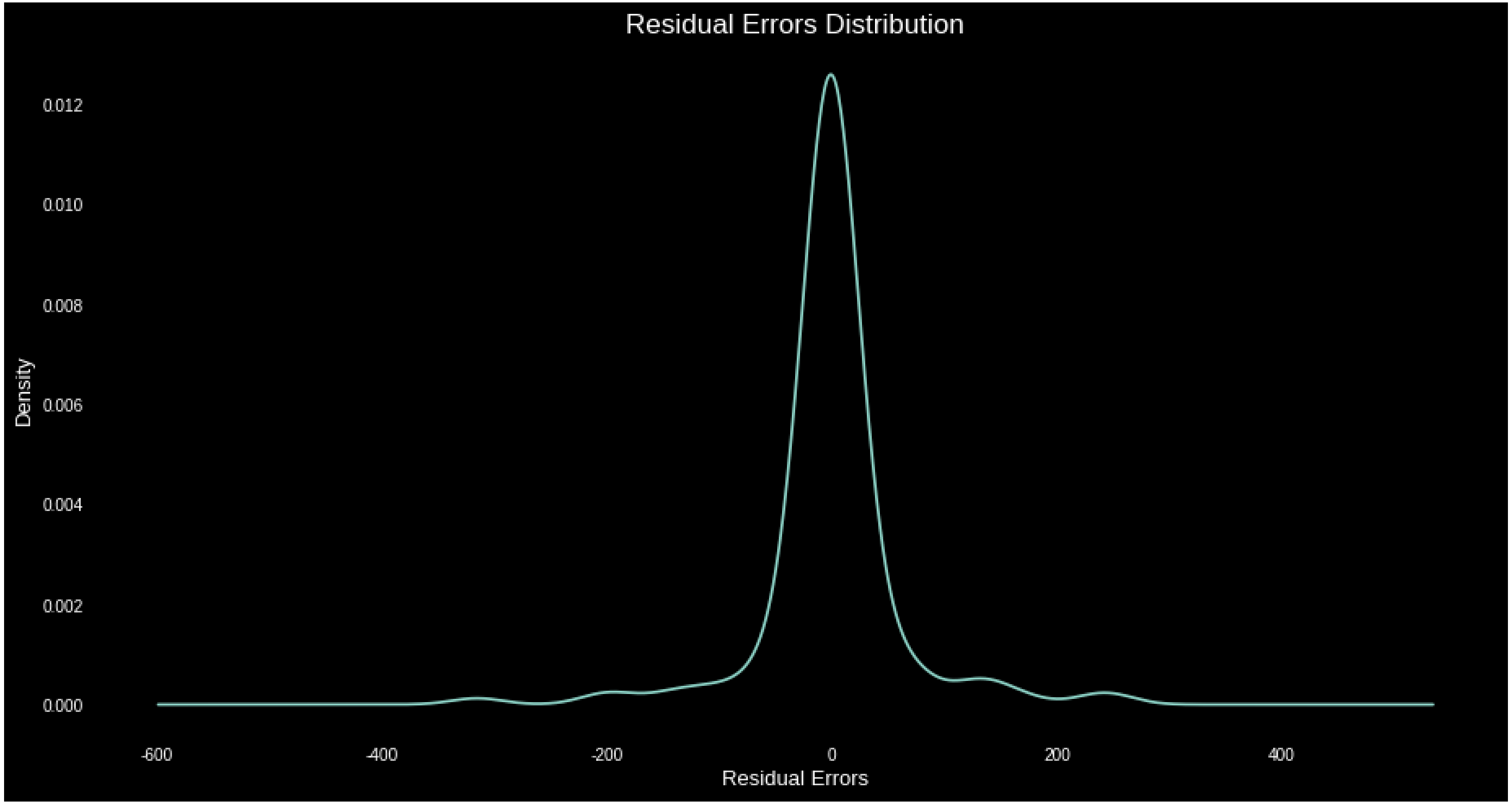
```
p, d, q = 10, 1, 0
date_list = daily_korea.Date.apply(lambda x: datetime.strptime(x, '%Y-%m-%d').date())
arima = ARIMA(daily_korea.TargetValue
              , dates= date_list
              , order=(p, d, q)
              , freq="D").fit()
print(f'# ARIMA model fitted₩n[Parameters]₩np(AR part): {p}, d(I part): {d}, q(MA part): {q}')

fig, ax = plt.subplots(figsize=(13, 7))
plt.plot(daily_korea.Date, arima.resid)
plt.title('Residual Errors of ARIMA on Daily Confrimed Cases (S.Korea)', size=17)
plt.xlabel('Date', size=13)
plt.ylabel('Residual errors', size=13)
ax.set_xticks(ax.get_xticks()[::int(len(daily_korea.Date)/8)])
plt.show()
```

```
# ARIMA model fitted
[Parameters]
p(AR part): 10, d(I part): 1, q(MA part): 0
```



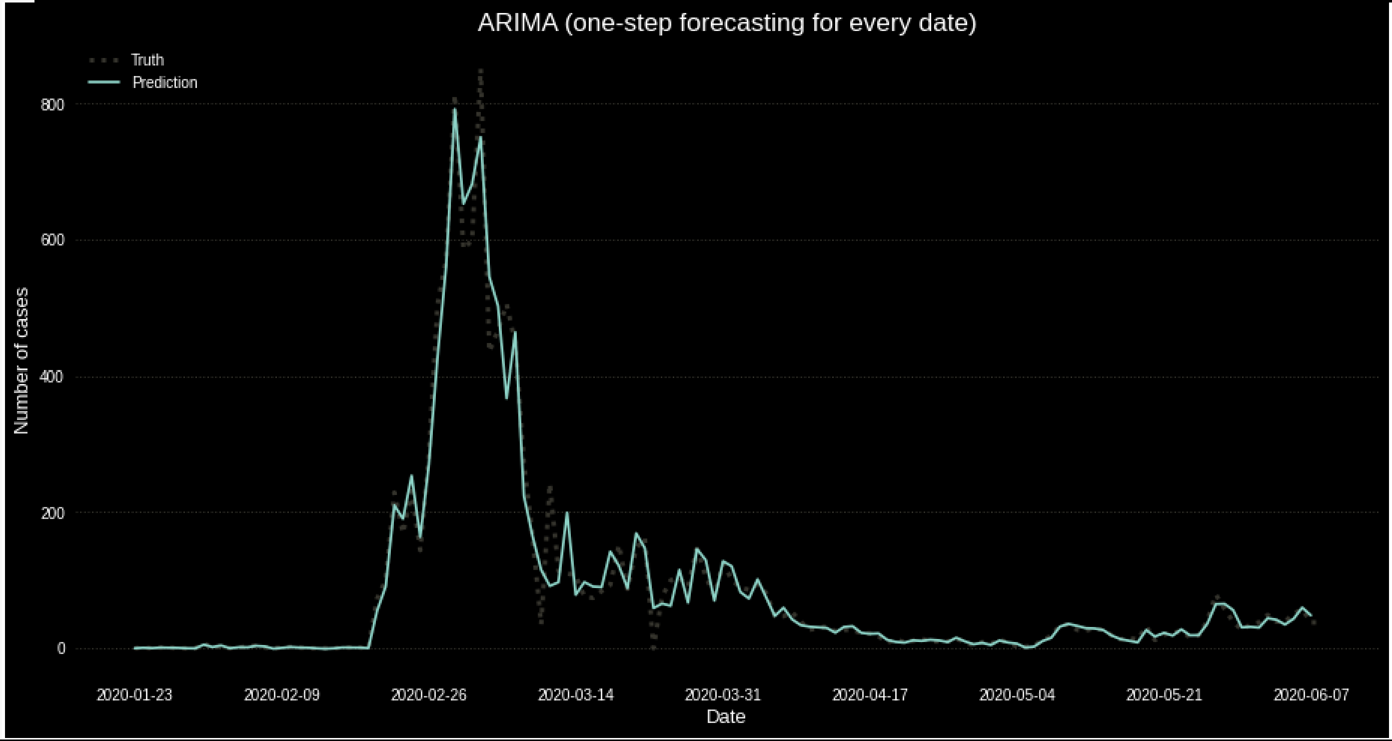
ARIMA

```python
## 1. Check distribution of residual errors
arima.resid.plot(kind='kde'
                 , grid=False)
plt.title('Residual Errors Distribution', size=17)
plt.xlabel('Residual Errors', size=13)
plt.ylabel('Density', size=13)
plt.show()
## 2. Check statistics
print('[Basic statistics]')
print(arima.resid.describe())
```



ARIMA

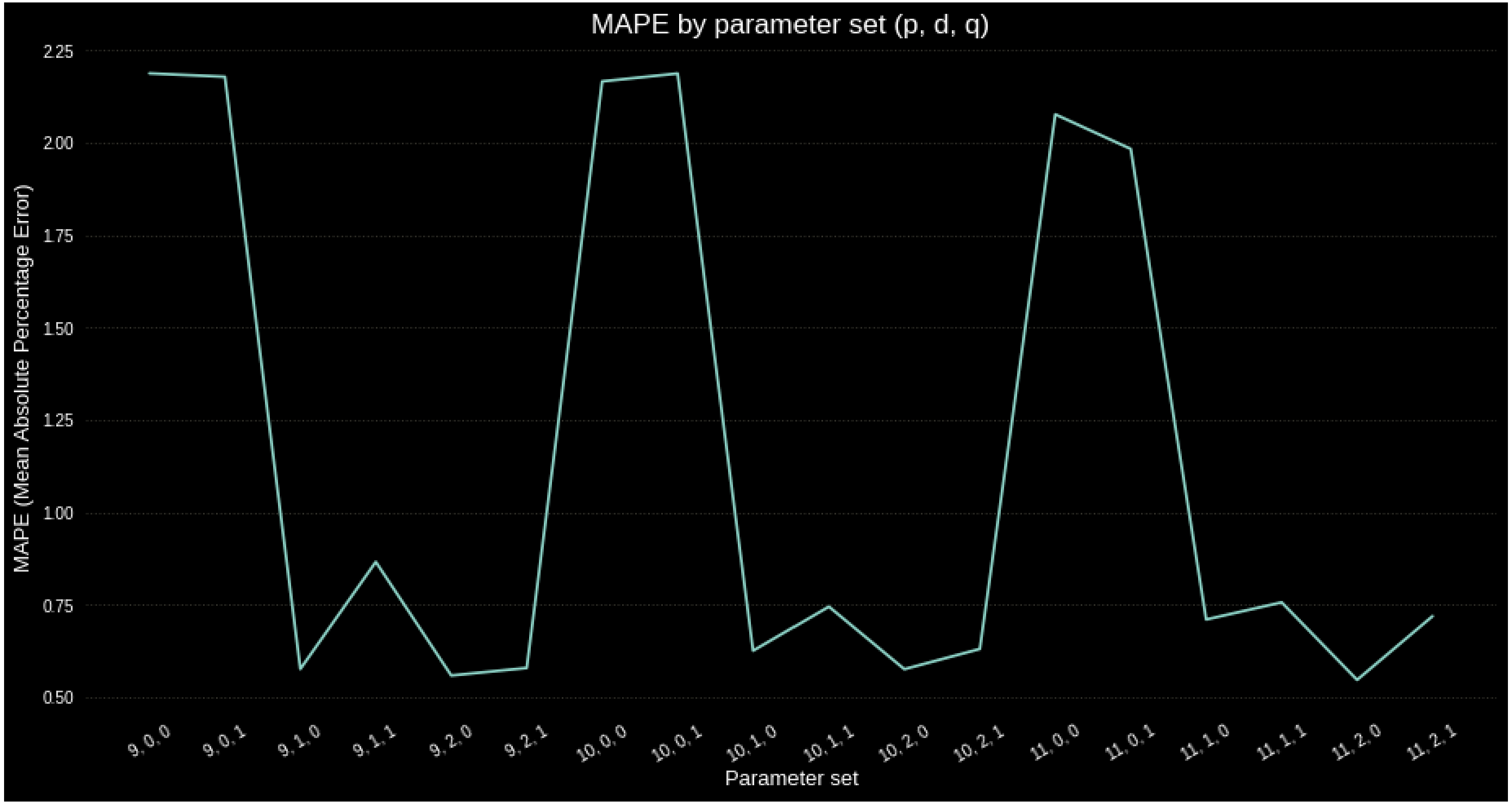
```
## 1. Overlap predictions(+1 step to the last observation) onto the truth
fig, ax = plt.subplots(figsize=(13, 7))

plt.plot(daily_korea.Date
         , daily_korea.TargetValue
         , color='#33322B', ls=':' , lw=3)
plt.plot(daily_korea.Date[:-1]
        , arima.predict()[1:])
plt.title('ARIMA (one-step forecasting for every date)', size=17)
plt.xlabel('Date', size=13)
plt.ylabel('Number of cases', size=13)
ax.set_xticks(ax.get_xticks()[::int(len(daily_korea.Date)/8)])
plt.legend(['Truth', 'Prediction'], loc='upper left')
plt.show()

## 2. Check scores
diff, rmse, mae, mape = diff_metrics(daily_korea.TargetValue[:-1], arima.predict()[1:])
scores = pd.DataFrame(
    {'rmse': rmse
     , 'mae': mae
     , 'mape': mape}
    , index=['score']
)
display(scores)
print('- RMSE: Root Mean Sqaure Error₩
    ₩n- MAE: Mean Absolute Error₩
    ₩n- MAPE: Mean Absolute Percentage Error₩
      ')
```



ARIMA

```python
p, d, q = 10, 1, 0
param_list, mape_list = arima_grid(daily_korea, p, d, q, 1);
print(f'Minium MAPE: {round(min(mape_list), 4)} by {param_list[np.argmin(mape_list)]} (p, d, q)')
```

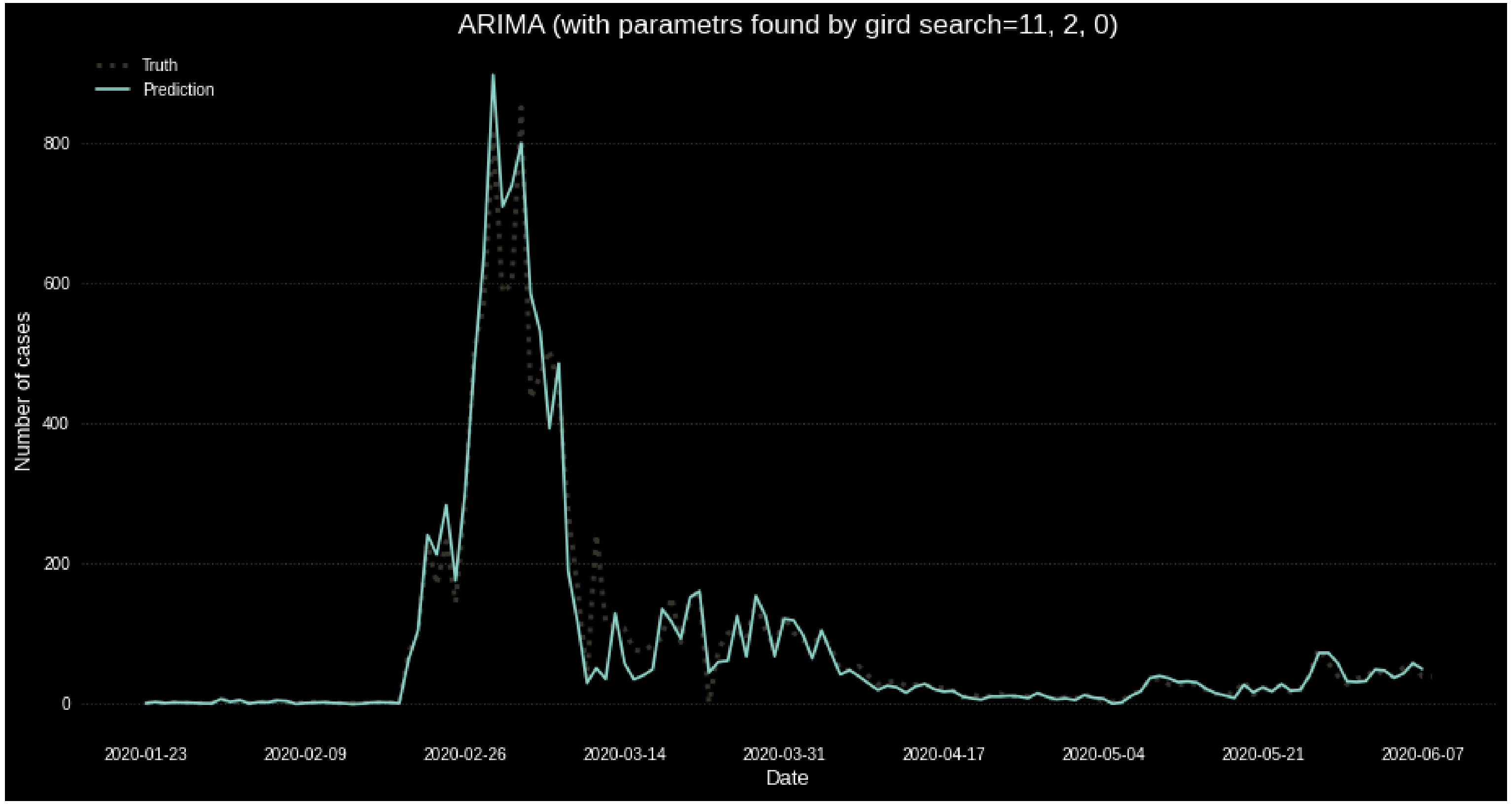
```
/home/spark/.local/lib/python3.6/site-packages/statsmodels/base/model.py:568: ConvergenceWarning: Maximum Likelihood optimization failed t
o converge. Check mle_retvals
```




```
Minium MAPE: 0.5464 by 11, 2, 0 (p, d, q)
```

ARIMA

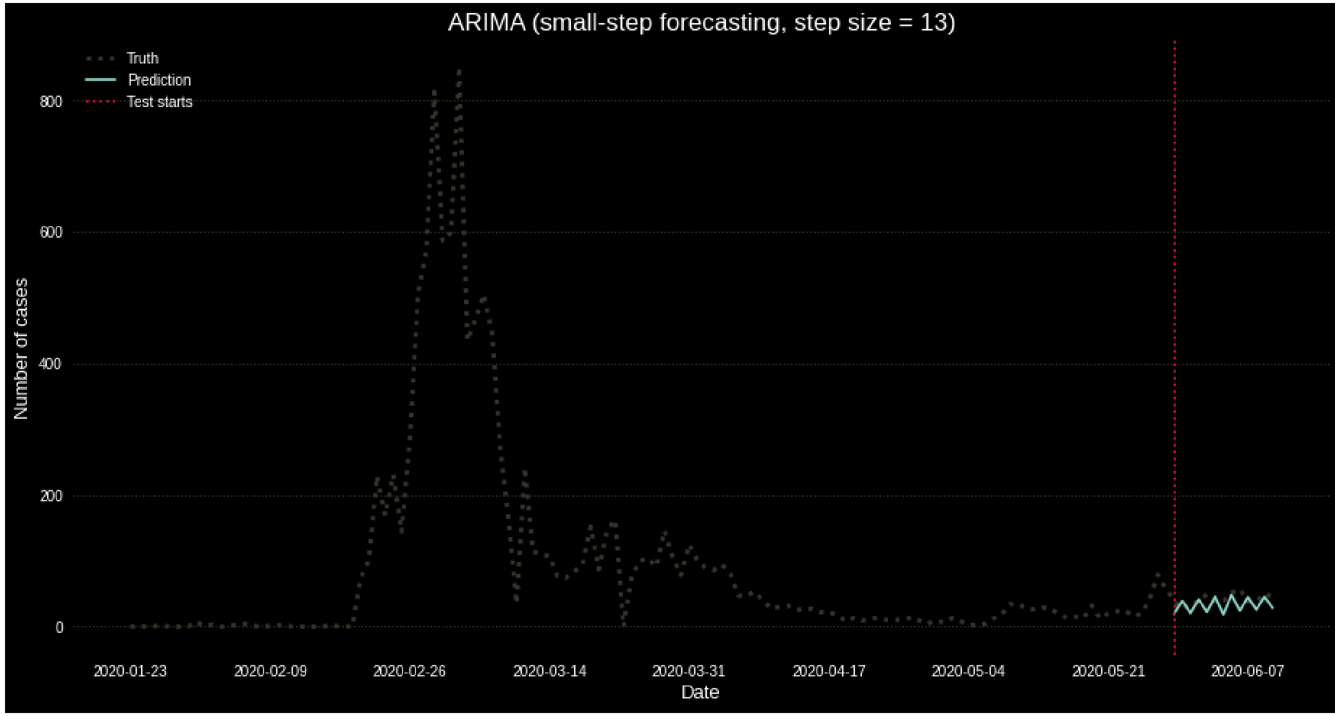
```python
## 1. Apply best parameter set
arima = ARIMA(daily_korea.TargetValue
              , dates=date_list
              , order=(11, 2, 0)
              , freq="D").fit()
arima_pred = arima.predict()

## 2. Overlap predictions(+1 step to the last observation) onto the truth
fig, ax = plt.subplots(figsize=(13, 7))

plt.plot(daily_korea.Date
         , daily_korea.TargetValue
         , color='#33322B', ls=':' , lw=3)
plt.plot(daily_korea.Date[:-1]
         , arima.predict()[1:])
plt.title('ARIMA (with parametrs found by gird search=11, 2, 0)', size=17)
plt.xlabel('Date', size=13)
plt.ylabel('Number of cases', size=13)
ax.set_xticks(ax.get_xticks()[::int(len(daily_korea.Date)/8)])
plt.legend(['Truth', 'Prediction'], loc='upper left')
plt.show()

## 3. Check scores
diff_grid, rmse_grid, mae_grid, mape_grid = diff_metrics(daily_korea.TargetValue[:-1], arima.predict()[1:])
scores = pd.DataFrame(
    {'rmse': rmse_grid
     , 'mae': mae_grid
     , 'mape': mape_grid}
    , index=['score']
)
scores
```



	rmse	mae	mape
score	34.954381	16.296871	0.54644

ARIMA

```python
## 1. Set test set(step) size as the difference between daily_korea and its raw version
test_size = 13

## 2. Copy date for updating the rolled predictions by ARIMA as the new truth
daily_korea_pred = copy.deepcopy(daily_korea[:-test_size])
daily_korea_pred.Date = daily_korea_pred.Date.apply(lambda x: datetime.strptime(x, '%Y-%m-%d').date())

## 3. Roll ARIMA test_size times to utilize the previous predictions as pseudo truths
for i in range(test_size):
    arima = ARIMA(daily_korea_pred.TargetValue
                  , dates=daily_korea_pred.Date
                  , order=(11, 2, 0)
                  , freq="D").fit()
    arima_pred = arima.predict()
    daily_korea_pred.loc[len(daily_korea_pred)] = (daily_korea_pred.Date.values[-1] + timedelta(1)
                                                   , arima_pred.values[-1])

## 4. Compare predictions with truths (from daily_korea_raw)
fig, ax = plt.subplots(figsize=(13, 7))
plt.plot(daily_korea_raw.Date
         , daily_korea_raw.TargetValue
         , color='#33322B', ls=':' , lw=3)
plt.plot(daily_korea_raw.Date[-test_size:],
         daily_korea_pred[-test_size:].TargetValue)
plt.title(f'ARIMA (small-step forecasting, step size = {test_size})', size=17)
plt.xlabel('Date', size=13)
plt.ylabel('Number of cases', size=13)
ax.set_xticks(ax.get_xticks()[::int(len(daily_korea.Date)/8)])
ax.axvline(daily_korea_raw.Date.values[-test_size], ls=':', color='crimson')
plt.legend(['Truth', 'Prediction', 'Test starts'], loc='upper left')
plt.show()
print('# All predictions before test are same as truth')
## 5. Check scores
diff_small, rmse_small, mae_small, mape_small = diff_metrics(daily_korea_raw.TargetValue[-test_size:]
                                                             , daily_korea_pred.TargetValue[-test_size:])
scores = pd.DataFrame(
    {'rmse': rmse_small
     , 'mae': mae_small
     , 'mape': mape_small}
    , index=['score']
)
display(scores)
```




```
# All predictions before test are same as truth
```

	rmse	mae	mape
score	16.416869	13.795402	0.321719

소감문

이재근

빅데이터 수업을 마치고 처음엔 데이터베이스와 빅데이터를 구분하는 방법도 몰랐지만 스파크와 파이썬, 머신러닝 등을 이용하면서 조금이나마 빅데이터분석에 한발짝 더 다가가고 낯선 코딩에 문을 여는 수업이 된 것 같습니다. 감사합니다!

홍문기

이번 기회를 통해 파이썬과 스파크를 이용해 계속해서 지속되는 코로나에 대해 어떤 데이터들이 연관성이 있는지 분석할 수 있는 기회를 가졌습니다. 그중 나이, 성별, 지역, 계절에 대한 데이터에 대해 코로나와 관련해 분석을 수행하였고 예측에 대해서는 시계열 분석을 이용하여 분석했는데 생각보다 어려웠고 분석 시 python보단 R을 많이 이용했는데 이번 기회를 통해 파이썬을 통해 분석 기술이 느는 좋은 기회를 가졌습니다.

최학준

빅데이터 분석 수업을 진행하면서 spark도 써보고 spark의 모듈인 machine learning도 써보고 마지막 통합 프로젝트까지 수행했는데 4차 산업 혁명 시대를 맞아 빅데이터라는 강력한 스킬을 배워 가는 것 같아 매우 보람찬 수업이 된 것 같습니다.