In [1]:

```
import requests
import pandas as pd
from lxml import etree

html = 'https://ncov.dxy.cn/ncovh5/view/pneumonia'
html_data = requests.get(html)
html_data.encoding = 'utf-8'
html_data = etree.HTML(html_data.text, etree.HTMLParser())
html_data = html_data.xpath(
    '//*[@id="getListByCountryTypeService2true"]/text()')  # xpath方法选择疫情的数据集合
ncov_world = html_data[0][49:-12]
ncov_world = ncov_world.replace('true', 'True')
ncov_world = ncov_world.replace('false', 'False')
ncov_world = eval(ncov_world)

country = []
confirmed = []
lived = []
dead = []

for i in ncov_world:  # 分离国家名称，确诊人数，治愈人数和死亡人数并存入dataframe里备用
    country.append(i['provinceName'])
    confirmed.append(i['confirmedCount'])
    lived.append(i['curedCount'])
    dead.append(i['deadCount'])

data_world = pd.DataFrame()
data_world['国家名称'] = country
data_world['确诊人数'] = confirmed
data_world['治愈人数'] = lived
data_world['死亡人数'] = dead
data_world.head(5)
```

Out[1]:

	国家名称	确诊人数	治愈人数	死亡人数
0	法国	29583616	368023	149044
1	德国	26244107	4328400	138864
2	韩国	18086462	336548	24167
3	英国	22455392	6491069	178880
4	西班牙	12326264	150376	106341

In [2]:

```
data_economy = pd.read_csv(
    "https://labfile.oss.aliyuncs.com/courses/2791/gpd_2016_2020.csv", index_col=0)
time_index = pd.date_range(start='2016', periods=18, freq='Q')
data_economy.index = time_index
data_economy
```

Out[2]:

	国内生产总值	第一产业增加值	第二产业增加值	第三产业增加值	农林牧渔业增加值	工业增加值	制造业增加值	建筑业增加值	批发和零售业增加值	交运输、储和政业加
2016-03-31	162410.0	8312.7	61106.8	92990.5	8665.5	53666.4	45784.0	7763.0	16847.5	718
2016-06-30	181408.2	12555.9	73416.5	95435.8	13045.5	60839.2	52378.3	12943.8	17679.8	829
2016-09-30	191010.6	17542.4	75400.5	98067.8	18162.2	61902.5	52468.3	13870.6	18513.0	859
2016-12-31	211566.2	21728.2	85504.1	104334.0	22577.8	68998.4	58878.4	16921.5	20684.1	896
2017-03-31	181867.7	8205.9	69315.5	104346.3	8595.8	60909.3	51419.7	8725.3	18608.9	809
2017-06-30	201950.3	12644.9	82323.0	106982.4	13204.2	68099.8	58172.1	14574.4	19473.6	939
2017-09-30	212789.3	18255.8	84574.1	109959.5	18944.2	69327.2	58632.6	15590.1	20342.9	968
2017-12-31	235428.7	22992.9	95368.0	117067.8	23915.8	76782.9	65652.1	19015.8	22731.1	994
2018-03-31	202035.7	8575.7	76598.2	116861.8	9005.8	66905.6	56631.9	10073.8	20485.5	880
2018-06-30	223962.2	13003.8	91100.6	119857.8	13662.2	75122.1	64294.9	16404.3	21374.2	1017
2018-09-30	234474.3	18226.9	93112.5	123134.9	18961.8	76239.6	64348.2	17294.5	22334.1	1058
2018-12-31	258808.9	24938.7	104023.9	129846.2	25929.0	82822.1	70662.1	21720.4	24710.0	1077
2019-03-31	218062.8	8769.4	81806.5	127486.9	9249.4	71064.5	60357.1	11143.1	21959.2	938
2019-06-30	242573.8	14437.6	97315.6	130820.6	15108.7	79820.7	68041.8	17954.2	23097.0	1086
2019-09-30	252208.7	19798.0	97790.4	134620.4	20629.0	79501.8	66823.8	18734.6	23993.6	1131
2019-12-31	278019.7	27461.6	109252.8	141305.2	28579.9	86721.6	73952.4	23072.4	26795.9	1124
2020-03-31	206504.3	10186.2	73638.0	122680.1	10708.4	64642.0	53852.0	9377.8	18749.6	786
2020-06-30	250110.1	15866.8	99120.9	135122.3	16596.4	80402.4	69258.8	19156.8	23696.1	1065

In [3]:

```
data_area = pd.read_csv('https://labfile.oss.aliyuncs.com/courses/2791/DXYArea.csv')
data_news = pd.read_csv('https://labfile.oss.aliyuncs.com/courses/2791/DXYNews.csv')
```

In [4]:

```
data_area = data_area.loc[data_area['countryName'] == data_area['provinceName']]
data_area_times = data_area[['countryName', 'province_confirmedCount',
                             'province_curedCount', 'province_deadCount', 'updateTime']]

time = pd.DatetimeIndex(data_area_times['updateTime'])  # 根据疫情的更新时间来生成时间序列
data_area_times.index = time  # 生成索引
data_area_times = data_area_times.drop('updateTime', axis=1)
data_area_times.head(5)

data_area_times.isnull().any()  # 查询是否有空值
```

Out[4]:

```
countryName               False
province_confirmedCount   False
province_curedCount       False
province_deadCount        False
dtype: bool
```

In [5]:

```
data_news_times = data_news[['pubDate', 'title', 'summary']]
time = pd.DatetimeIndex(data_news_times['pubDate'])
data_news_times.index = time  # 生成新闻数据的时间索引
data_news_times = data_news_times.drop('pubDate', axis=1)
data_news_times.head(5)
```

Out[5]:

pubDate	title	summary
2020-07-17 05:40:08	美国新增71434例新冠肺炎确诊病例，累计确诊超354万例	据美国约翰斯·霍普金斯大学统计数据显示，截至美东时间7月16日17:33时（北京时间17日0...
2020-07-17 06:06:49	巴西新冠肺炎确诊病例破201万，近六成大城市确诊病例加速增长	截至当地时间7月16日18时，巴西新增新冠肺炎确诊病例45403例，累计确诊2012151例...
2020-07-16 22:31:00	阿塞拜疆新增493例新冠肺炎确诊病例 累计确诊26165例	当地时间7月16日，阿塞拜疆国家疫情防控指挥部发布消息，在过去24小时内，阿塞拜疆新增新冠肺...
2020-07-16 22:29:48	科威特新增791例新冠肺炎确诊病例 累计确诊57668例	科威特卫生部当地时间16日下午发布通告，确认过去24小时境内新增791例新冠肺炎确诊病例，同...
2020-07-16 21:26:54	罗马尼亚新增777例新冠肺炎确诊病例 累计确诊35003例	据罗马尼亚政府7月16日公布的数据，过去24小时对19097人进行新冠病毒检测，确诊777例...

In [6]:

```
print(data_world.isnull().any())
print(data_economy.isnull().any())
print(data_area_times.isnull().any())
print(data_news_times.isnull().any())  # 确认各个数据集是否空集
```

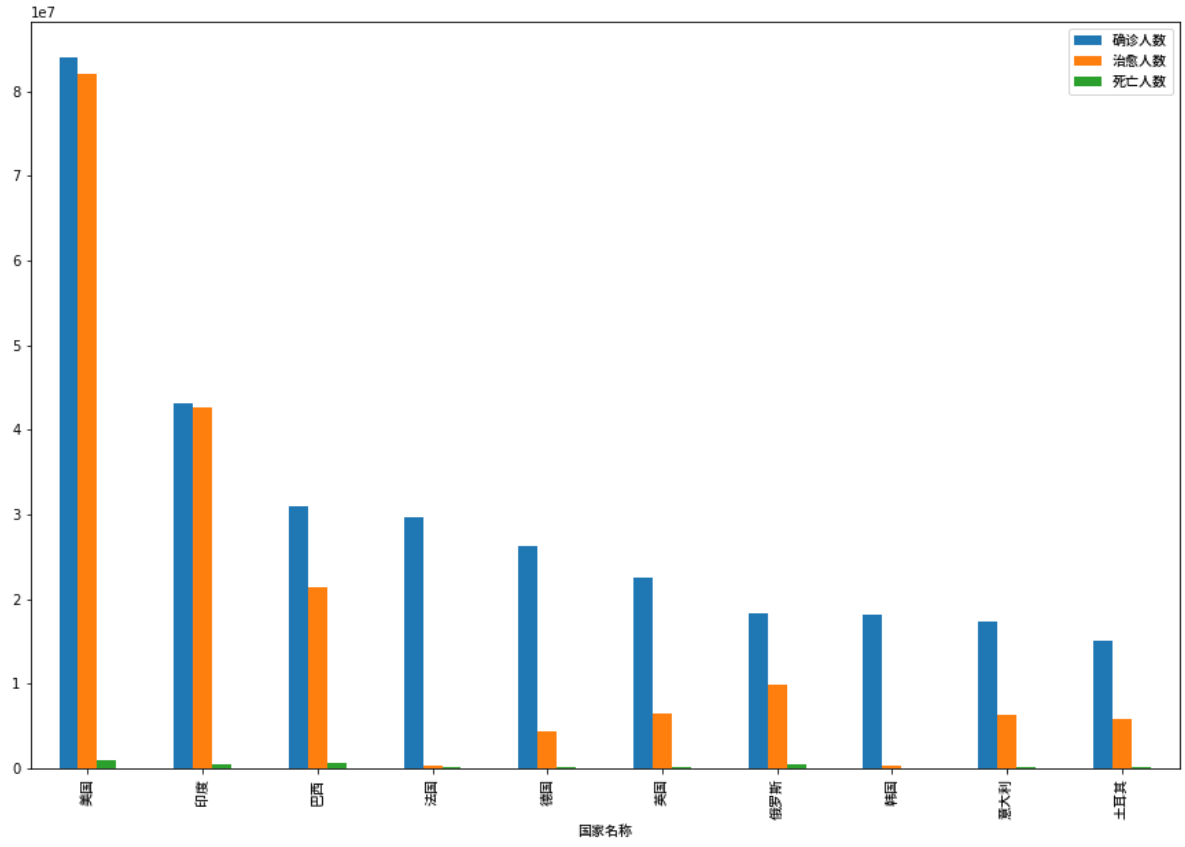
```
国家名称     False
确诊人数     False
治愈人数     False
死亡人数     False
dtype: bool
国内生产总值                 False
第一产业增加值                False
第二产业增加值                False
第三产业增加值                False
农林牧渔业增加值               False
工业增加值                  False
制造业增加值                 False
建筑业增加值                 False
批发和零售业增加值              False
交通运输、仓储和邮政业增加值         False
住宿和餐饮业增加值              False
金融业增加值                 False
房地产业增加值                False
信息传输、软件和信息技术服务业增加值    False
租赁和商务服务业增加值             False
其他行业增加值                False
dtype: bool
countryName                False
province_confirmedCount    False
province_curedCount        False
province_deadCount         False
dtype: bool
title      False
summary    False
dtype: bool
```

In [7]:

```
import matplotlib.pyplot as plt
import matplotlib
import os

%matplotlib inline
# 指定中文字体
fpath = os.path.join("./NotoSansCJK.otf")
myfont = matplotlib.font_manager.FontProperties(fname=fpath)
# 绘图
data_world = data_world.sort_values(by='确诊人数', ascending=False)  # 按确诊人数进行排序
data_world_set = data_world[['确诊人数', '治愈人数', '死亡人数']]
data_world_set.index = data_world['国家名称']
data_world_set.head(10).plot(kind='bar', figsize=(15, 10))  # 对排序前十的国家数据进行绘图
plt.xlabel('国家名称', fontproperties=myfont)
plt.xticks(fontproperties=myfont)
plt.legend(fontsize=30, prop=myfont)  # 设置图例
```

Out[7]:

<matplotlib.legend.Legend at 0x1bd59422940>



In [8]:

```
!pip install pyecharts==1.7.1
```

```
Requirement already satisfied: pyecharts==1.7.1 in e:\anaconda3\lib\site-packages
(1.7.1)
Requirement already satisfied: prettytable in c:\users\12131\appdata\roaming\python
\python39\site-packages (from pyecharts==1.7.1) (3.3.0)
Requirement already satisfied: jinja2 in c:\users\12131\appdata\roaming\python\pytho
n39\site-packages (from pyecharts==1.7.1) (3.1.2)
Requirement already satisfied: simplejson in c:\users\12131\appdata\roaming\python\p
ython39\site-packages (from pyecharts==1.7.1) (3.17.6)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\12131\appdata\roaming\pyt
hon\python39\site-packages (from jinja2->pyecharts==1.7.1) (2.1.1)
Requirement already satisfied: wcwidth in c:\users\12131\appdata\roaming\python\pyth
on39\site-packages (from prettytable->pyecharts==1.7.1) (0.2.5)

WARNING: There was an error checking the latest version of pip.
```

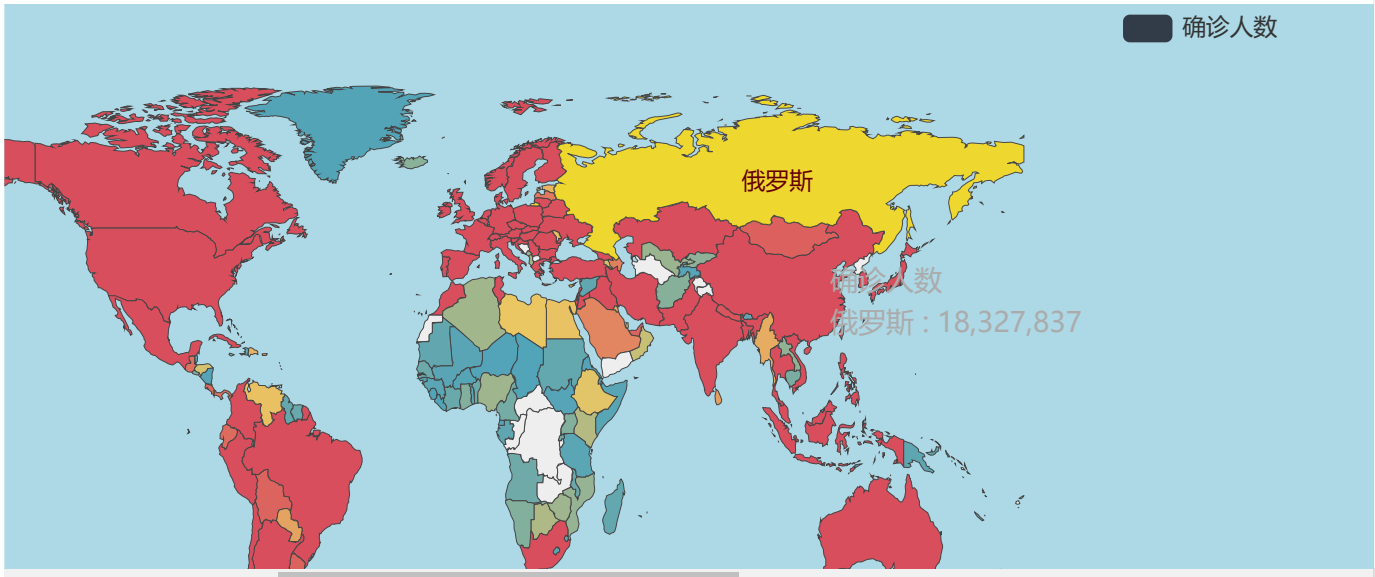
In [9]:

```
from pyecharts.charts import Map
from pyecharts import options as opts
from pyecharts.globals import CurrentConfig, NotebookType

CurrentConfig.NOTEBOOK_TYPE = NotebookType.JUPYTER_NOTEBOOK
name_map = {  # 世界各国数据的中英文对比
    'Singapore Rep.': '新加坡',
    'Dominican Rep.': '多米尼加',
    'Palestine': '巴勒斯坦',
    'Bahamas': '巴哈马',
    'Timor-Leste': '东帝汶',
    'Afghanistan': '阿富汗',
    'Guinea-Bissau': '几内亚比绍',
    "Côte d'Ivoire": '科特迪瓦',
    'Siachen Glacier': '锡亚琴冰川',
    "Br. Indian Ocean Ter.": '英属印度洋领土',
    'Angola': '安哥拉',
    'Albania': '阿尔巴尼亚',
    'United Arab Emirates': '阿联酋',
    'Argentina': '阿根廷',
    'Armenia': '亚美尼亚',
    'French Southern and Antarctic Lands': '法属南半球和南极领地',
    'Australia': '澳大利亚',
    'Austria': '奥地利',
    'Azerbaijan': '阿塞拜疆',
    'Burundi': '布隆迪',
    'Belgium': '比利时',
    'Benin': '贝宁',
    'Burkina Faso': '布基纳法索',
    'Bangladesh': '孟加拉国',
    'Bulgaria': '保加利亚',
    'The Bahamas': '巴哈马',
    'Bosnia and Herz.': '波斯尼亚和黑塞哥维那',
    'Belarus': '白俄罗斯',
    'Belize': '伯利兹',
    'Bermuda': '百慕大',
    'Bolivia': '玻利维亚',
    'Brazil': '巴西',
    'Brunei': '文莱',
    'Bhutan': '不丹',
    'Botswana': '博茨瓦纳',
    'Central African Rep.': '中非',
    'Canada': '加拿大',
    'Switzerland': '瑞士',
    'Chile': '智利',
    'China': '中国',
    'Ivory Coast': '象牙海岸',
    'Cameroon': '喀麦隆',
    'Dem. Rep. Congo': '刚果民主共和国',
    'Congo': '刚果',
    'Colombia': '哥伦比亚',
    'Costa Rica': '哥斯达黎加',
    'Cuba': '古巴',
    'N. Cyprus': '北塞浦路斯',
    'Cyprus': '塞浦路斯',
    'Czech Rep.': '捷克',
    'Germany': '德国',
    'Djibouti': '吉布提',
    'Denmark': '丹麦',
```

```
    'Algeria': '阿尔及利亚',
    'Ecuador': '厄瓜多尔',
    'Egypt': '埃及',
    'Eritrea': '厄立特里亚',
    'Spain': '西班牙',
    'Estonia': '爱沙尼亚',
    'Ethiopia': '埃塞俄比亚',
    'Finland': '芬兰',
    'Fiji': '斐',
    'Falkland Islands': '福克兰群岛',
    'France': '法国',
    'Gabon': '加蓬',
    'United Kingdom': '英国',
    'Georgia': '格鲁吉亚',
    'Ghana': '加纳',
    'Guinea': '几内亚',
    'Gambia': '冈比亚',
    'Guinea Bissau': '几内亚比绍',
    'Eq. Guinea': '赤道几内亚',
    'Greece': '希腊',
    'Greenland': '格陵兰',
    'Guatemala': '危地马拉',
    'French Guiana': '法属圭亚那',
    'Guyana': '圭亚那',
    'Honduras': '洪都拉斯',
    'Croatia': '克罗地亚',
    'Haiti': '海地',
    'Hungary': '匈牙利',
    'Indonesia': '印度尼西亚',
    'India': '印度',
    'Ireland': '爱尔兰',
    'Iran': '伊朗',
    'Iraq': '伊拉克',
    'Iceland': '冰岛',
    'Israel': '以色列',
    'Italy': '意大利',
    'Jamaica': '牙买加',
    'Jordan': '约旦',
    'Japan': '日本',
    'Kazakhstan': '哈萨克斯坦',
    'Kenya': '肯尼亚',
    'Kyrgyzstan': '吉尔吉斯斯坦',
    'Cambodia': '柬埔寨',
    'Korea': '韩国',
    'Kosovo': '科索沃',
    'Kuwait': '科威特',
    'Lao PDR': '老挝',
    'Lebanon': '黎巴嫩',
    'Liberia': '利比里亚',
    'Libya': '利比亚',
    'Sri Lanka': '斯里兰卡',
    'Lesotho': '莱索托',
    'Lithuania': '立陶宛',
    'Luxembourg': '卢森堡',
    'Latvia': '拉脱维亚',
    'Morocco': '摩洛哥',
    'Moldova': '摩尔多瓦',
    'Madagascar': '马达加斯加',
    'Mexico': '墨西哥',
    'Macedonia': '马其顿',
    'Mali': '马里',
```

```
    'Myanmar': '缅甸',
    'Montenegro': '黑山',
    'Mongolia': '蒙古',
    'Mozambique': '莫桑比克',
    'Mauritania': '毛里塔尼亚',
    'Malawi': '马拉维',
    'Malaysia': '马来西亚',
    'Namibia': '纳米比亚',
    'New Caledonia': '新喀里多尼亚',
    'Niger': '尼日尔',
    'Nigeria': '尼日利亚',
    'Nicaragua': '尼加拉瓜',
    'Netherlands': '荷兰',
    'Norway': '挪威',
    'Nepal': '尼泊尔',
    'New Zealand': '新西兰',
    'Oman': '阿曼',
    'Pakistan': '巴基斯坦',
    'Panama': '巴拿马',
    'Peru': '秘鲁',
    'Philippines': '菲律宾',
    'Papua New Guinea': '巴布亚新几内亚',
    'Poland': '波兰',
    'Puerto Rico': '波多黎各',
    'Dem. Rep. Korea': '朝鲜',
    'Portugal': '葡萄牙',
    'Paraguay': '巴拉圭',
    'Qatar': '卡塔尔',
    'Romania': '罗马尼亚',
    'Russia': '俄罗斯',
    'Rwanda': '卢旺达',
    'W. Sahara': '西撒哈拉',
    'Saudi Arabia': '沙特阿拉伯',
    'Sudan': '苏丹',
    'S. Sudan': '南苏丹',
    'Senegal': '塞内加尔',
    'Solomon Is.': '所罗门群岛',
    'Sierra Leone': '塞拉利昂',
    'El Salvador': '萨尔瓦多',
    'Somaliland': '索马里兰',
    'Somalia': '索马里',
    'Serbia': '塞尔维亚',
    'Suriname': '苏里南',
    'Slovakia': '斯洛伐克',
    'Slovenia': '斯洛文尼亚',
    'Sweden': '瑞典',
    'Swaziland': '斯威士兰',
    'Syria': '叙利亚',
    'Chad': '乍得',
    'Togo': '多哥',
    'Thailand': '泰国',
    'Tajikistan': '塔吉克斯坦',
    'Turkmenistan': '土库曼斯坦',
    'East Timor': '东帝汶',
    'Trinidad and Tobago': '特里尼达和多巴哥',
    'Tunisia': '突尼斯',
    'Turkey': '土耳其',
    'Tanzania': '坦桑尼亚',
    'Uganda': '乌干达',
    'Ukraine': '乌克兰',
    'Uruguay': '乌拉圭',
```

```
    'United States': '美国',
    'Uzbekistan': '乌兹别克斯坦',
    'Venezuela': '委内瑞拉',
    'Vietnam': '越南',
    'Vanuatu': '瓦努阿图',
    'West Bank': '西岸',
    'Yemen': '也门',
    'South Africa': '南非',
    'Zambia': '赞比亚',
    'Zimbabwe': '津巴布韦',
    'Comoros': '科摩罗'
}

map = Map(init_opts=opts.InitOpts(width="1900px", height="900px",
                                  bg_color="#ADD8E6", page_title="全球疫情确诊人数"))  # 获得世界地
map.add("确诊人数", [list(z) for z in zip(data_world['国家名称'], data_world['确诊人数'])],
        is_map_symbol_show=False,  # 添加确诊人数信息
        # 通过name_map来转化国家的中英文名称方便显示
        maptype="world", label_opts=opts.LabelOpts(is_show=False), name_map=name_map,
        itemstyle_opts=opts.ItemStyleOpts(color="rgb(49,60,72)"),
        ).set_global_opts(
    visualmap_opts=opts.VisualMapOpts(max_=1000000),  # 对视觉映射进行配置
)
map.render_notebook()  # 在notebook中显示
```



In [10]:

```
country = data_area_times.sort_values('province_confirmedCount', ascending=False).drop_duplicates(
    subset='countryName', keep='first').head(6)['countryName']
country = list(country)  # 对于同一天采集的多个数据，只保留第一次出现的数据也就是最后一次更新的数据
country
```

Out[10]:

['美国', '巴西', '印度', '俄罗斯', '秘鲁', '智利']

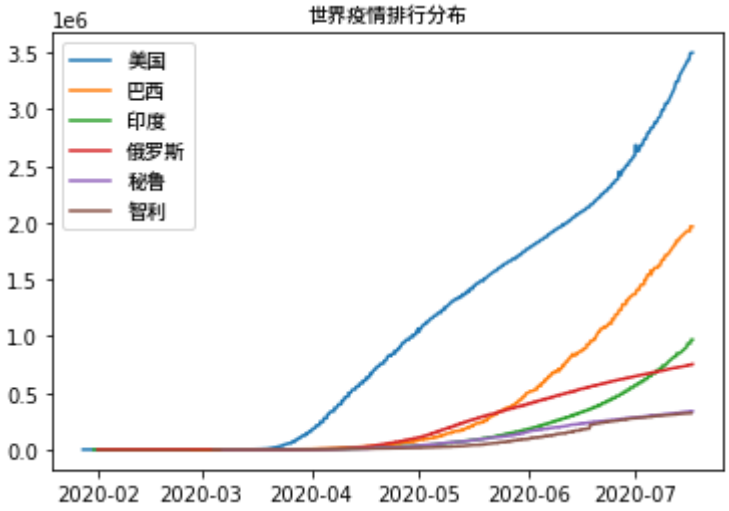
In [11]:

```
data_America = data_area_times[data_area_times['countryName'] == '美国']
data_Brazil = data_area_times[data_area_times['countryName'] == '巴西']
data_India = data_area_times[data_area_times['countryName'] == '印度']
data_Russia = data_area_times[data_area_times['countryName'] == '俄罗斯']
data_Peru = data_area_times[data_area_times['countryName'] == '秘鲁']
data_Chile = data_area_times[data_area_times['countryName'] == '智利']

timeindex = data_area_times.index
timeindex = timeindex.floor('D')  # 对于日期索引，只保留具体到哪一天
data_area_times.index = timeindex

timeseries = pd.DataFrame(data_America.index)
timeseries.index = data_America.index
data_America = pd.concat([timeseries, data_America], axis=1)
data_America.drop_duplicates(
    subset='updateTime', keep='first', inplace=True)  # 对美国数据进行处理，获得美国确诊人数的时间
data_America.drop('updateTime', axis=1, inplace=True)

timeseries = pd.DataFrame(data_Brazil.index)
timeseries.index = data_Brazil.index
data_Brazil = pd.concat([timeseries, data_Brazil], axis=1)
# 对巴西数据进行处理，获得巴西确诊人数的时间序列
data_Brazil.drop_duplicates(subset='updateTime', keep='first', inplace=True)
data_Brazil.drop('updateTime', axis=1, inplace=True)

timeseries = pd.DataFrame(data_India.index)
timeseries.index = data_India.index
data_India = pd.concat([timeseries, data_India], axis=1)
# 对印度数据进行处理，获得印度确诊人数的时间序列
data_India.drop_duplicates(subset='updateTime', keep='first', inplace=True)
data_India.drop('updateTime', axis=1, inplace=True)

timeseries = pd.DataFrame(data_Russia.index)
timeseries.index = data_Russia.index
data_Russia = pd.concat([timeseries, data_Russia], axis=1)
# 对俄罗斯数据进行处理，获得俄罗斯确诊人数的时间序列
data_Russia.drop_duplicates(subset='updateTime', keep='first', inplace=True)
data_Russia.drop('updateTime', axis=1, inplace=True)

timeseries = pd.DataFrame(data_Peru.index)
timeseries.index = data_Peru.index
data_Peru = pd.concat([timeseries, data_Peru], axis=1)
# 对秘鲁数据进行处理，获得秘鲁确诊人数的时间序列
data_Peru.drop_duplicates(subset='updateTime', keep='first', inplace=True)
data_Peru.drop('updateTime', axis=1, inplace=True)

timeseries = pd.DataFrame(data_Chile.index)
timeseries.index = data_Chile.index
data_Chile = pd.concat([timeseries, data_Chile], axis=1)
# 对智利数据进行处理，获得智利确诊人数的时间序列
data_Chile.drop_duplicates(subset='updateTime', keep='first', inplace=True)
data_Chile.drop('updateTime', axis=1, inplace=True)

plt.title("世界疫情排行分布", fontproperties=myfont)
plt.plot(data_America['province_confirmedCount'])
plt.plot(data_Brazil['province_confirmedCount'])
plt.plot(data_India['province_confirmedCount'])
plt.plot(data_Russia['province_confirmedCount'])
plt.plot(data_Peru['province_confirmedCount'])
```

```
plt.plot(data_Chile['province_confirmedCount'])
plt.legend(country, prop=myfont)
```

Out[11]:

```
<matplotlib.legend.Legend at 0x1bd5d9b4ac0>
```



In [12]:

```
pip install wordcloud==1.8.0
```

```
Collecting wordcloud==1.8.0
  Using cached wordcloud-1.8.0.tar.gz (217 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Requirement already satisfied: numpy>=1.6.1 in e:\anaconda3\lib\site-packages (fr
om wordcloud==1.8.0) (1.20.3)
Requirement already satisfied: pillow in e:\anaconda3\lib\site-packages (from wor
dcloud==1.8.0) (8.4.0)
Requirement already satisfied: matplotlib in e:\anaconda3\lib\site-packages (from
wordcloud==1.8.0) (3.4.3)
Requirement already satisfied: pyparsing>=2.2.1 in e:\anaconda3\lib\site-packages
(from matplotlib->wordcloud==1.8.0) (3.0.4)
Requirement already satisfied: python-dateutil>=2.7 in e:\anaconda3\lib\site-pack
ages (from matplotlib->wordcloud==1.8.0) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in e:\anaconda3\lib\site-package
s (from matplotlib->wordcloud==1.8.0) (1.3.1)
Requirement already satisfied: cycler>=0.10 in e:\anaconda3\lib\site-packages (fr
om matplotlib->wordcloud==1.8.0) (0.10.0)
Requirement already satisfied: six in e:\anaconda3\lib\site-packages (from cycler
```

In [14]:

```
import jieba
import re
from wordcloud import WordCloud

def word_cut(x): return jieba.lcut(x)

news = []
reg = "[^\u4e00-\u9fa5]"
for i in data_news['title']:
    if re.sub(reg, '', i) != '':
        news.append(re.sub(reg, '', i))

words = []
counts = {}
for i in news:
    words.append(word_cut(i))
for word in words:
    for a_word in word:
        if len(a_word) == 1:
            continue
        else:
            counts[a_word] = counts.get(a_word, 0)+1
words_sort = list(counts.items())
words_sort.sort(key=lambda x:[1],reverse=True)

newcloud = WordCloud(font_path="./NotoSansCJK.otf",
                     background_color="white",width=600,height=300,max_words=50)
newcloud.generate_from_frequencies(counts)
image = newcloud.to_image()
image
```

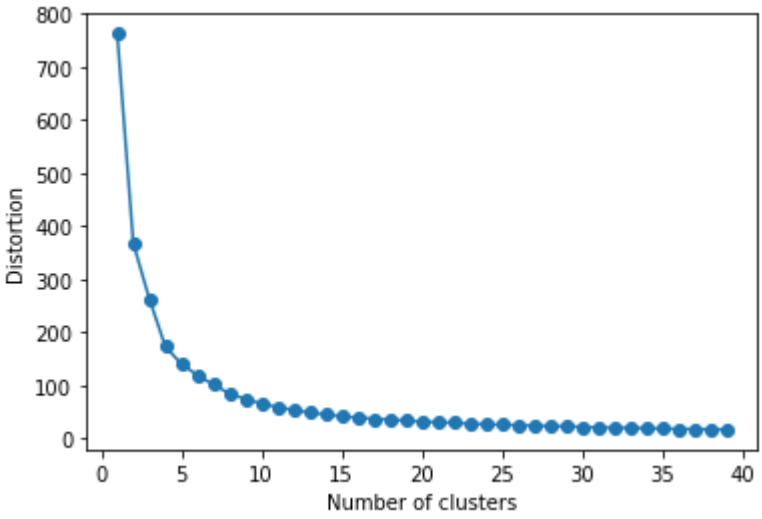
Out[14]:

In [15]:

```
from gensim.models import Word2Vec
from sklearn.cluster import KMeans
import warnings
warnings.filterwarnings('ignore')

words = []

for i in news:
    words.append(word_cut(i))
model = Word2Vec(words, sg=0, vector_size=300, window=5, min_count=5)  # 词向量进行训练
keys = model.wv.key_to_index.keys()  # 获取词汇列表
wordvector = []
for key in keys:
    wordvector.append(model.wv[key])  # 对词汇列表里的所有的词向量进行整合

distortions = []
for i in range(1, 40):
    word_kmeans = KMeans(n_clusters=i,
                         init='k-means++',
                         n_init=10,
                         max_iter=300,
                         random_state=0)  # 分别聚成1-40类
    word_kmeans.fit(wordvector)
    distortions.append(word_kmeans.inertia_)  # 算出样本距离最近的聚类中心的距离总和

plt.plot(range(1, 40), distortions, marker='o')  # 绘图
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
```

Out[15]:

```
Text(0, 0.5, 'Distortion')
```



In [16]:

```
word_kmeans = KMeans(n_clusters=10)  # 聚成10类
word_kmeans.fit(wordvector)

labels = word_kmeans.labels_

for num in range(0, 10):
    text = []
    for i in range(len(keys)):
        if labels[i] == num:
            text.append(list(keys)[i])  # 分别获得10类的聚类结果
    print(text)
```

```
['加拿大', '截至', '菲律宾', '重庆市', '土耳其', '山东省', '马来西亚', '湖北省',
'突破', '小时', '直播', '阴性', '岁', '第例', '来自', '紧急状态', '西藏', '年月日
时', '阿联酋', '机构', '纽约州', '解除', '相关', '医学观察', '最高', '严重', '达
到', '管控', '钻石', '钟南山', '强制', '首批', '时间', '再次', '疑似', '以外',
'康复', '确认', '含', '医生', '解封', '外交部', '支援', '重新', '兵团', '共有',
'逝世', '儿童', '结束', '以来', '服务', '复课', '逐步', '已经', '成', '推迟', '数
据', '赴', '集中', '会', '已有', '延期', '接受', '高风险', '一个', '到', '能力',
'进一步', '日起', '社会', '复产', '全省', '卫生部长', '至少', '开展', '反弹', '方
舱', '邮轮', '面临', '数', '安全', '医务人员', '建议', '一天', '疾控中心', '结
果', '共', '增长', '控制', '加强', '信息', '允许', '去世', '约', '亿元', '卫生
部', '广州', '并', '烈士', '禁止', '提醒', '曾', '加州', '密切接触', '出台', '回
应', '放宽', '高考', '需', '地', '以上', '推出', '高校', '留学生', '痊愈', '武汉
市', '口岸', '免费', '正式', '医护', '事件', '官员', '旅游', '学生', '正', '国务
院', '发放', '预计', '小区', '问题', '各地', '各国', '感谢', '冲击', '范围', '队
员', '两周', '场所', '最后', '上', '一线', '加速', '临时', '运抵', '旅行', '关
于', '我', '通过', '居民', '一律', '每日', '以', '积极', '公共卫生', '铁路', '中
小学', '媒体', '公司', '消费', '我国', '不会', '导致', '纽约', '蔬菜', '占',
'州', '未来', '中央', '第二', '决定', '必须', '突发', '成为', '大规模', '高峰',
'是否', '注意', '家庭', '深切', '下半旗', '表示', '游客', '通知', '未', '还', '医
```

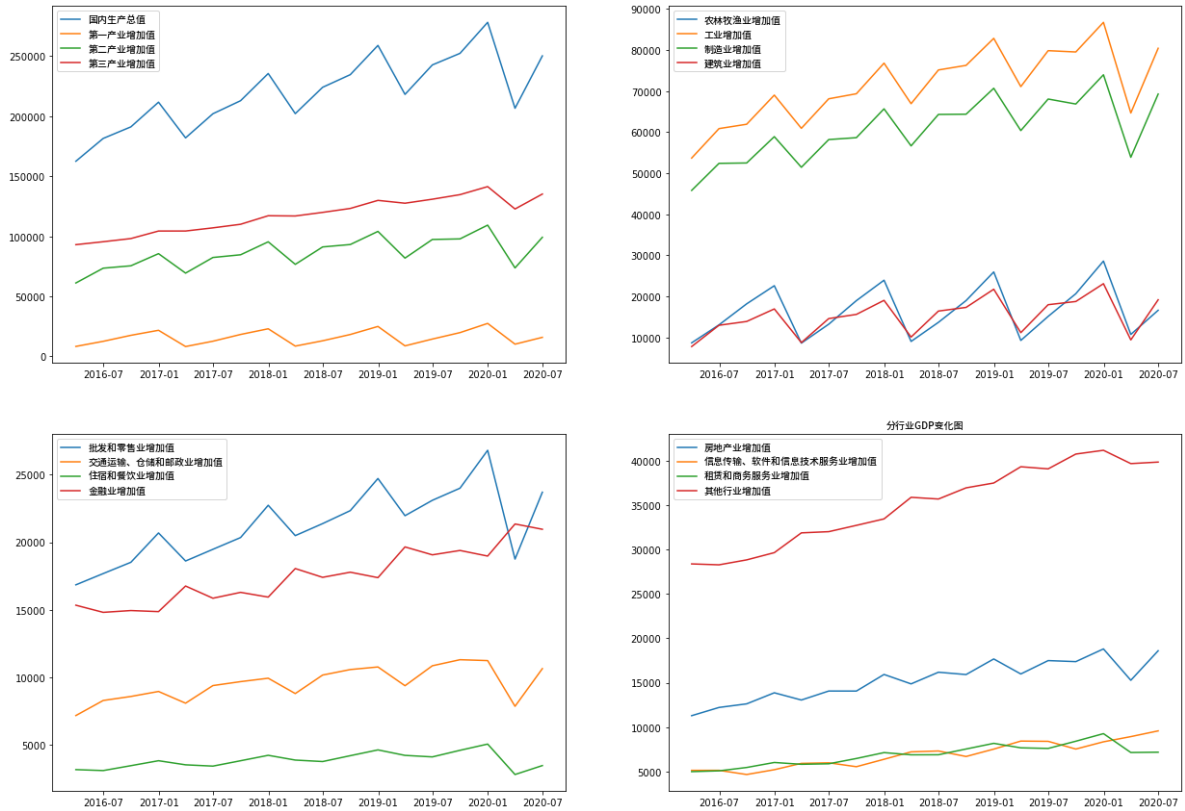
In [17]:

```
sum_GDP = ['国内生产总值', '第一产业增加值', '第二产业增加值', '第三产业增加值']
industry_GDP = ['农林牧渔业增加值', '工业增加值', '制造业增加值', '建筑业增加值']
industry2_GDP = ['批发和零售业增加值', '交通运输、仓储和邮政业增加值', '住宿和餐饮业增加值', '金融业
industry3_GDP = ['房地产业增加值', '信息传输、软件和信息技术服务业增加值',
                 '租赁和商务服务业增加值', '其他行业增加值']  # 对不同行业分四类来展现

fig = plt.figure()
fig, axes = plt.subplots(2, 2, figsize=(21, 15))  # 分别用四个子图来展现数据变化情况

axes[0][0].plot(data_economy[sum_GDP])
axes[0][0].legend(sum_GDP, prop=myfont)
axes[0][1].plot(data_economy[industry_GDP])
axes[0][1].legend(industry_GDP, prop=myfont)
axes[1][0].plot(data_economy[industry2_GDP])
axes[1][0].legend(industry2_GDP, prop=myfont)
axes[1][1].plot(data_economy[industry3_GDP])
axes[1][1].legend(industry3_GDP, prop=myfont)

plt.title('分行业GDP变化图', fontproperties=myfont)
```

Out[17]:

Text(0.5, 1.0, '分行业GDP变化图')

<Figure size 432x288 with 0 Axes>



In [18]:

```
from statsmodels.tsa.arima_model import ARMA
from statsmodels.tsa.stattools import arma_order_select_ic

warnings.filterwarnings('ignore')
data_arma = pd.DataFrame(data_economy['国内生产总值'][:-2])  # 选取疫情期前的16个季度进行建模
a, b = arma_order_select_ic(data_arma, ic='hqic')['hqic_min_order']
arma = ARMA(data_arma, order=(a, b)).fit()  # 使用ARMA建模
rate1 = list(data_economy['国内生产总值'][-2] /
             arma.forecast(steps=1)[0])  # 获得疫情期当季度的预测值
rate1  # 实际值与预测值的比率
```

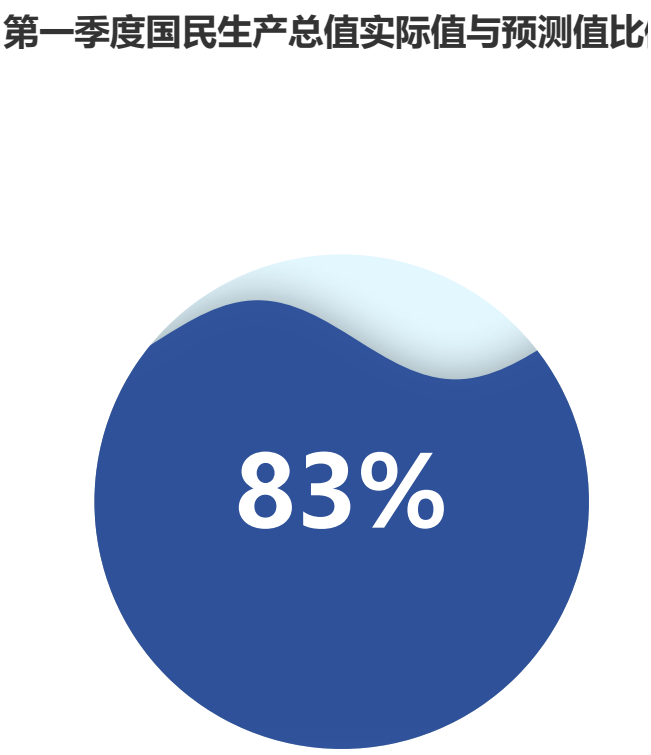
Out[18]:

```
[0.8273539514507257]
```

In [19]:

```
from pyecharts import options as opts
from pyecharts.charts import Liquid

c = (
    Liquid()
    .add("实际值/预测值", rate1, is_outline_show=False)
    .set_global_opts(title_opts=opts.TitleOpts(title="第一季度国民生产总值实际值与预测值比例",
                                               pos_left="center"))
)
c.render_notebook()
```

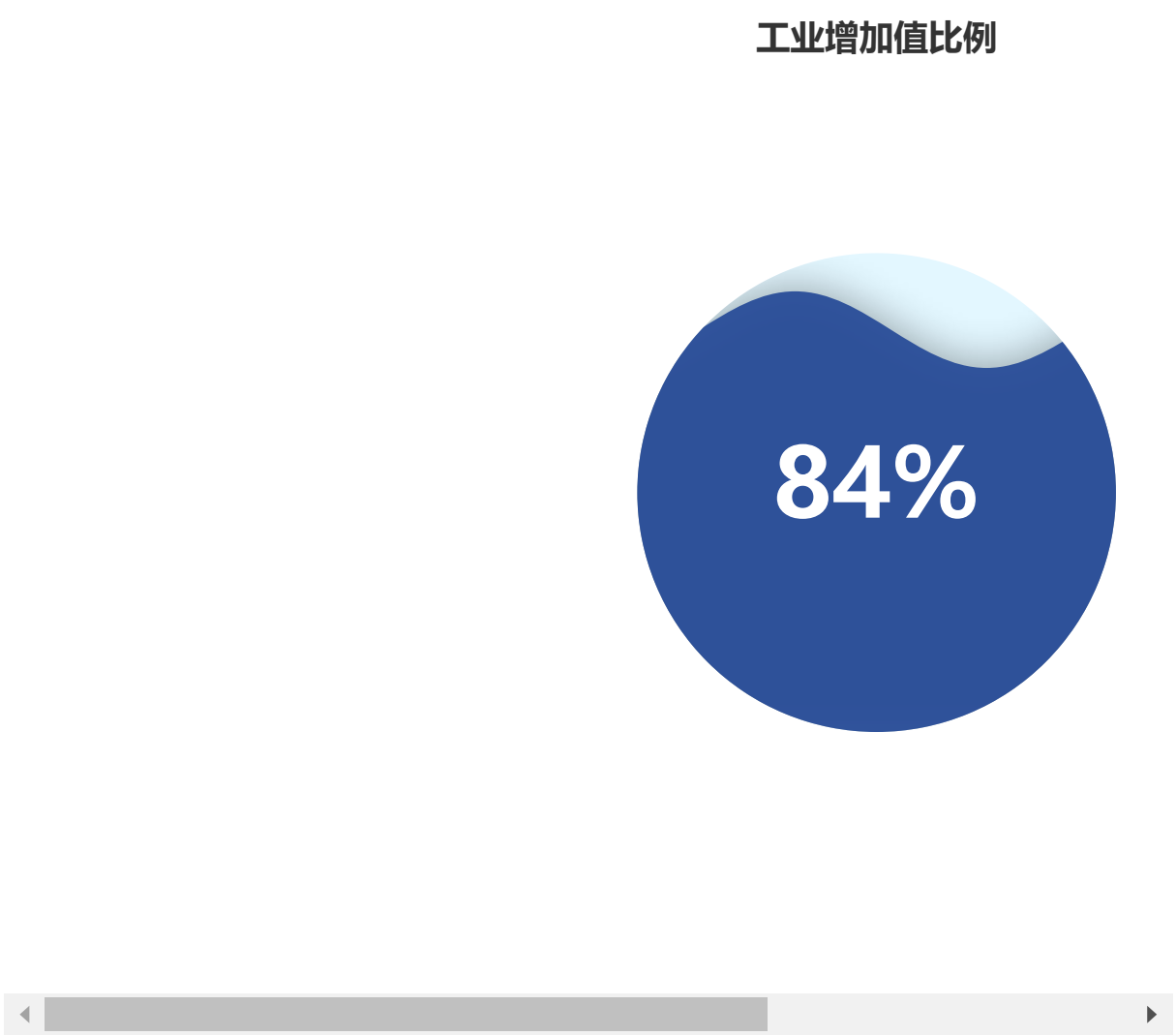
Out[19]:



In [20]:

```
warnings.filterwarnings('ignore')
data_arma = pd.DataFrame(data_economy['工业增加值'][:-2])
a, b = arma_order_select_ic(data_arma, ic='hqic')['hqic_min_order']
arma = ARMA(data_arma, order=(a, b)).fit()
rate2 = list(data_economy['工业增加值'][-2]/arma.forecast(steps=1)[0])
c = (
    Liquid()
    .add("实际值/预测值", rate2, is_outline_show=False)
    .set_global_opts(title_opts=opts.TitleOpts(title="工业增加值比例", pos_left="center"))
)
c.render_notebook()
```

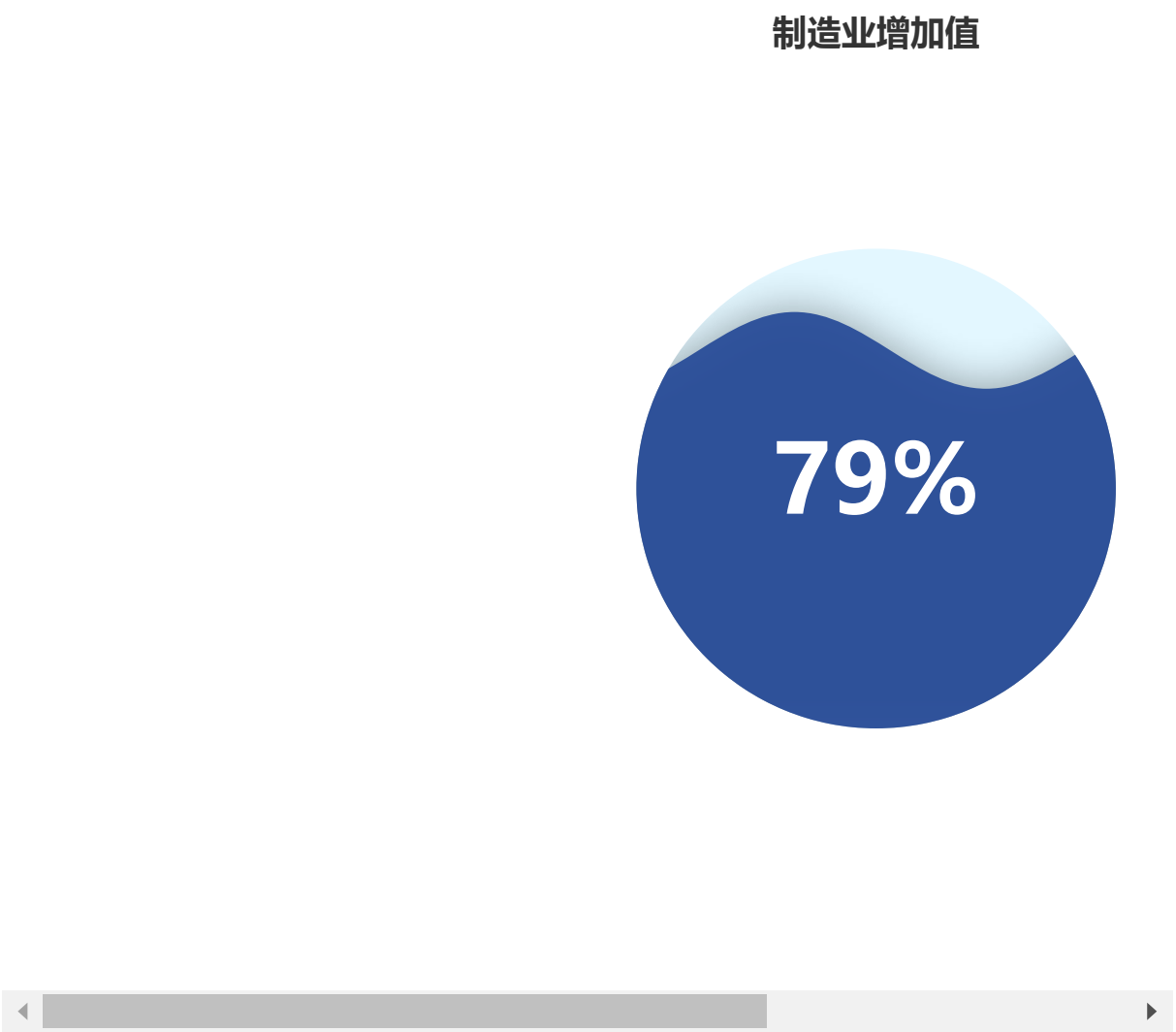
Out[20]:



In [21]:

```
warnings.filterwarnings('ignore')
data_arma = pd.DataFrame(data_economy['制造业增加值'][:-2])
a, b = arma_order_select_ic(data_arma, ic='hqic')['hqic_min_order']
arma = ARMA(data_arma, order=(a, b)).fit()
rate3 = list(data_economy['制造业增加值'][-2]/arma.forecast(steps=1)[0])
c = (
    Liquid()
    .add("实际值/预测值", rate3, is_outline_show=False)
    .set_global_opts(title_opts=opts.TitleOpts(title="制造业增加值", pos_left="center"))
)
c.render_notebook()
```

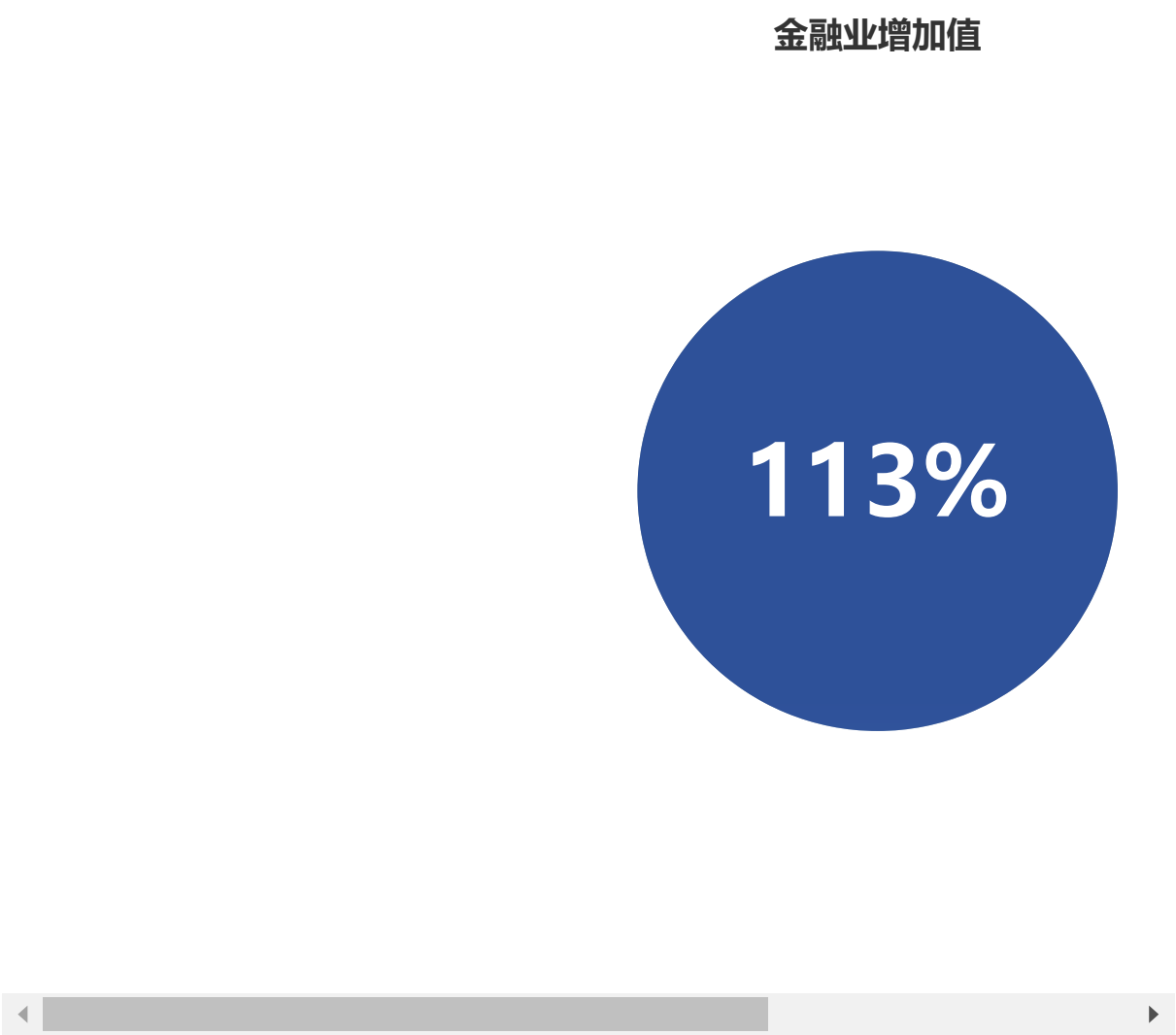
Out[21]:



In [22]:

```
data_arma = pd.DataFrame(data_economy['金融业增加值'][:-2])
a, b = arma_order_select_ic(data_arma, ic='hqic')['hqic_min_order']
arma = ARMA(data_arma, order=(a, b)).fit()
rate = list(data_economy['金融业增加值'][-2]/arma.forecast(steps=1)[0])
c = (
    Liquid()
    .add("实际值/预测值", rate, is_outline_show=False)
    .set_global_opts(title_opts=opts.TitleOpts(title="金融业增加值", pos_left="center"))
)
c.render_notebook()
```

Out[22]:



In [ ]: