EDA（探索性数据分析）——全球新冠肺炎...

EDA简介

EDA（Exploratory Data Analysis），即**探索性数据分析。**

区别于传统数据分析，EDA 不做任何前置的假设，而是直接通过对原始数据进行分析，用可视化技术和各种统计的方法来探寻数据隐含的规律和信息。**简单来说就是从数据中寻找规律，而不是基于人工假设。**

一般来说，EDA 分为以下几个步骤：

- 确定分析任务的目标
- 筛选、清洗数据
  - 检测异常值与缺失值
- 数据分析，可视化
  - 挖掘特征之间的相互关系
  - 挖掘特征与目标变量之间的关系
- 根据上一步的结果构建模型
- 得出最终结论

项目背景

在过去一年多的时间，对人类影响最大的事件就是全球新冠肺炎（COVID-19）的大流行。新冠肺炎除了对患者带来痛苦之外，对医疗系统也带来了巨大的挑战，许多患者陷入危险的本质原因就是医疗资源的紧张。

从数据科学角度来看，而如果可以根据现有确诊病例的数据来预测将来可能的确诊数，那么政府和医院就能够提前对医疗资源进行规划和准备，从而改善确诊患者的医疗环境。

新冠肺炎的蔓延趋势分析备受关注，数据分析竞赛平台 kaggle 也陆续放出一些数据集给数据分析的爱好者们分析。本次任务就基于其中的一份数据集，来尝试**预测不同国家确诊病例数随着时间的变化趋势。**

注：本次分析基于python语言完成，使用Anaconda+VS Code+Jupyter Notebook搭建开发环境，用到的工具包有pandas、numpy、matplotlib、plotly、xgboost等。

数据集描述

本次任务的数据集由两个文件组成，分别是train.csv 和 test.csv。顾名思义，train.csv是用来做数据分析及训练模型的，而test.csv是用来做预测的。

train.csv数据集描述：

字段名	含义
ID	记录id
Province_Stat	省份
Country_Region	国家
Date	日期
ConfirmedCases	确诊病例总数
Fatalities	死亡总数

test.csv数据集描述:

字段名	含义
ForcastId	预测id
Province_Stat	省份
Country_Region	国家
Date	日期

确定任务目标

为进一步明确任务目标，同时确保分析的可行性，首先来查看一下数据：

首先导入必要的工具包：

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random
from plotly import tools
import plotly.express as px
from plotly.offline import init_notebook_mode,iplot,plot
import plotly.figure_factory as ff
import plotly.graph_objs as go
```

导入两个数据文件，分别查看：

```
df_train=pd.read_csv("train.csv")
df_test=pd.read_csv("test.csv")
```

查看train.csv文件中的内容

```
df_train
```

	Id	Province_State	Country_Region	Date	ConfirmedCases	Fatalities
0	1	NaN	Afghanistan	2020-01-22	0.0	0.0
1	2	NaN	Afghanistan	2020-01-23	0.0	0.0
2	3	NaN	Afghanistan	2020-01-24	0.0	0.0
3	4	NaN	Afghanistan	2020-01-25	0.0	0.0
4	5	NaN	Afghanistan	2020-01-26	0.0	0.0
...	...	...	...	...	...	...
35990	35991	NaN	Zimbabwe	2020-05-11	36.0	4.0
35991	35992	NaN	Zimbabwe	2020-05-12	36.0	4.0
35992	35993	NaN	Zimbabwe	2020-05-13	37.0	4.0
35993	35994	NaN	Zimbabwe	2020-05-14	37.0	4.0
35994	35995	NaN	Zimbabwe	2020-05-15	42.0	4.0

35995 rows × 6 columns

查看test.csv文件中的内容

```
df_test
```

	ForecastId	Province_State	Country_Region	Date
0	1	NaN	Afghanistan	2020-04-02
1	2	NaN	Afghanistan	2020-04-03
2	3	NaN	Afghanistan	2020-04-04
3	4	NaN	Afghanistan	2020-04-05
4	5	NaN	Afghanistan	2020-04-06
...	...	...	...	...
13454	13455	NaN	Zimbabwe	2020-05-10
13455	13456	NaN	Zimbabwe	2020-05-11
13456	13457	NaN	Zimbabwe	2020-05-12
13457	13458	NaN	Zimbabwe	2020-05-13
13458	13459	NaN	Zimbabwe	2020-05-14

13459 rows × 4 columns

可以看到，两份数据的“省份”字段都存在缺失值，需要在数据清洗环节进行处理。

此外，两份数据的字段也类似，不过test.csv中不包含确诊病例数和死亡病例数。由此可以确定本次分析的任务目标：**找出影响确诊病例数的关键特征，然后根据关键特征从df_train数据集中**

训练出模型，最后预测 df_test 数据集中，不同的国家在不同的日期中的确诊病例数和死亡病例数。

筛选、清洗数据

1.处理缺失值

首先查看缺失值：

```
df_train.isnull().sum()
```

输出如下：

```
Id                   0
Province_State    20700
Country_Region        0
Date                  0
ConfirmedCases        0
Fatalities            0
dtype: int64
```

可以看到，除了省份，其他字段都没有缺失值，但省份的缺失值数量很大，有 2w 条，而我们的数据集一共才 3w+ 条数据，这代表我们后续分析不适合从省份入手，不然会有较大的偏差。

先简单用空字符串填充：

```
df_train=df_train.fillna("")
df_train
```

	Id	Province_State	Country_Region	Date	ConfirmedCases	Fatalities
0	1		Afghanistan	2020-01-22	0.0	0.0
1	2		Afghanistan	2020-01-23	0.0	0.0
2	3		Afghanistan	2020-01-24	0.0	0.0
3	4		Afghanistan	2020-01-25	0.0	0.0
4	5		Afghanistan	2020-01-26	0.0	0.0
...	...	...	...	...	...	...
35990	35991		Zimbabwe	2020-05-11	36.0	4.0
35991	35992		Zimbabwe	2020-05-12	36.0	4.0
35992	35993		Zimbabwe	2020-05-13	37.0	4.0
35993	35994		Zimbabwe	2020-05-14	37.0	4.0
35994	35995		Zimbabwe	2020-05-15	42.0	4.0

35995 rows × 6 columns

2.处理异常值

填充缺失值后，通过describe函数来看df_train中的统计分布信息：

```
df_train.describe()
```

	Id	ConfirmedCases	Fatalities
count	35995.000000	35995.000000	35995.000000
mean	17998.000000	3683.508737	243.560217
std	10391.005806	18986.978708	1832.966999
min	1.000000	0.000000	0.000000
25%	8999.500000	0.000000	0.000000
50%	17998.000000	19.000000	0.000000
75%	26996.500000	543.000000	7.000000
max	35995.000000	345813.000000	33998.000000

从输出结果来看，数据基本正常，没有明显异常值。

数据分析、可视化

1.确诊病例数和国家之间的关系

首先从国家维度切入，查看不同国家的确诊病例数。

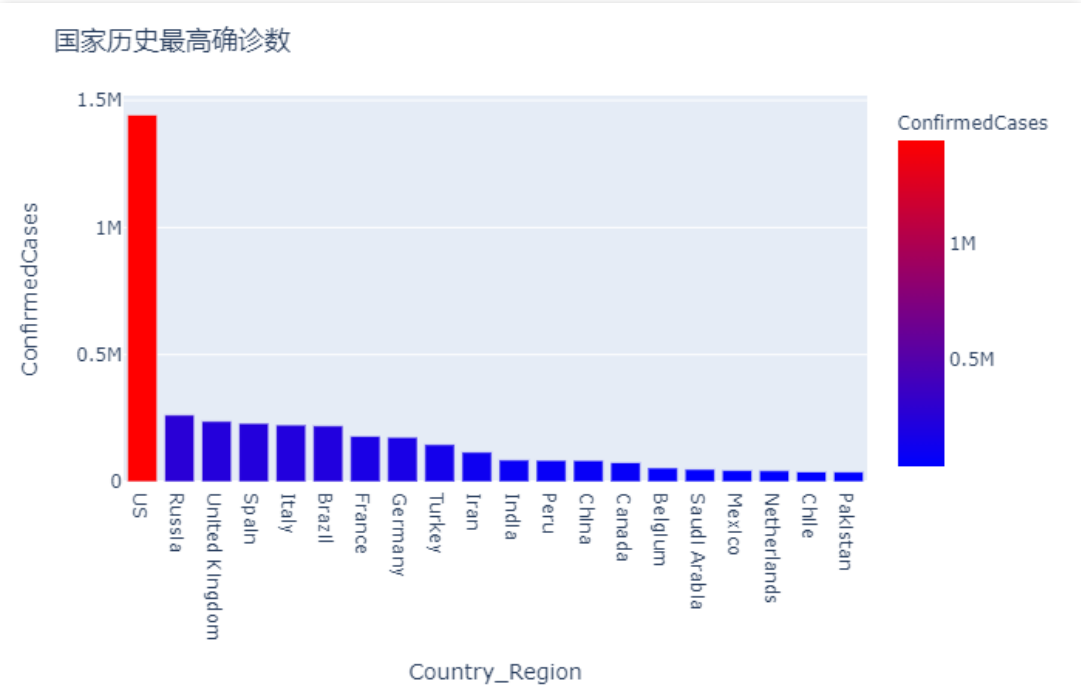
```
# 将数据按“国家”、“日期”维度聚合，并求和，得出各个国家在不同时期的总确诊病例数
df_countries=df_train.groupby(["Country_Region","Date"])
["ConfirmedCases"].sum()
# 再次按“国家”维度聚合，并取最大值，得出各个国家的历史最高确诊病例数
df_countries=df_countries.groupby(["Country_Region"]).max()
# 按最高确诊病例数降序排序后，取前20条数据备用
df_countries=df_countries.sort_values(ascending=False).head(20)
df_countries
```

```
Country_Region
US                1442653.0
Russia             262843.0
United Kingdom     238004.0
Spain              230183.0
Italy              223885.0
Brazil             220291.0
France             179630.0
Germany            175233.0
Turkey             146457.0
Iran               116635.0
India               85784.0
Peru                84495.0
China               84038.0
Canada              75945.0
Belgium             54644.0
Saudi Arabia        49176.0
Mexico              45032.0
Netherlands         43880.0
Chile               39542.0
Pakistan            38799.0
Name: ConfirmedCases, dtype: float64
```

将上一步的的结果可视化呈现：

```
# 使用plotly将图表画出来
fig=px.bar(df_countries,x=df_countries.index,y='ConfirmedCases',labels={'x':'Country'},color='ConfirmedCases',color_continuous_scale=px.colors.sequential.Bluered)
fig.update_layout(title_text="国家历史最高确诊数")
fig.show()
```

输出结果如下：



由上图可以看出，各个国家的确诊病例总数差别很大，由此可初步判断：**确诊病例数和国家这一特征密切相关。**

2.确诊病例数随时间变化的趋势

接下来以美国为例，分析确诊病例数随着时间的变化趋势

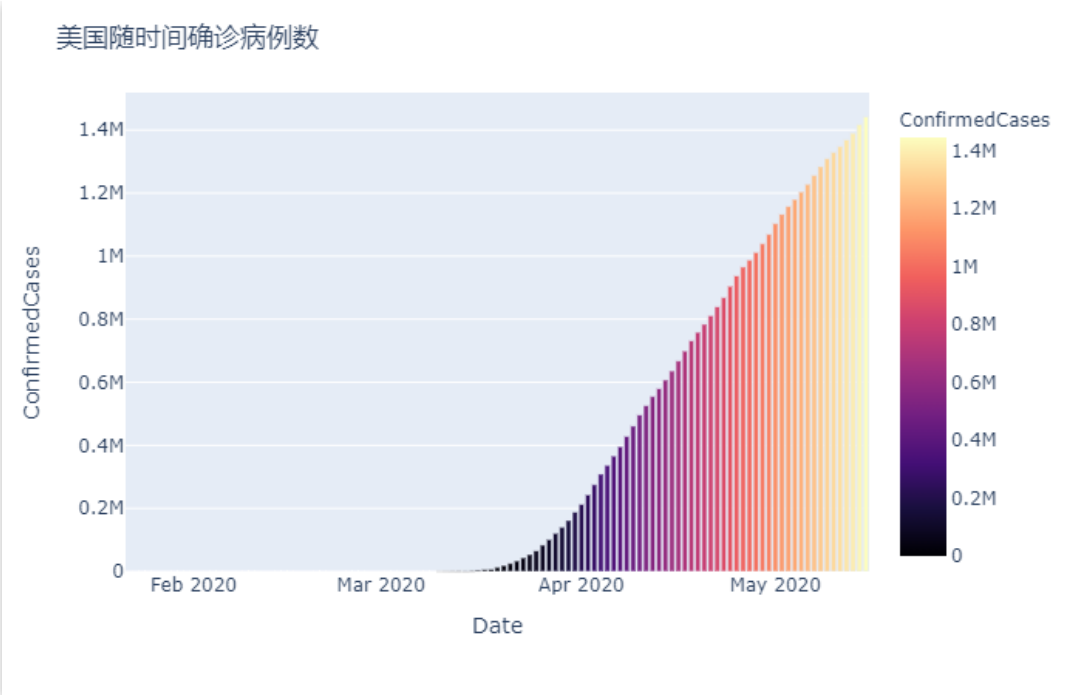
```
# 首先将数据按“国家”、“日期”维度聚合，得出所有国家确诊病例数随时间的变化趋势
df_country_records=df_train.groupby(["Country_Region","Date"]).sum()
# 过滤出美国的数据,并取确诊数和死亡数这两个关键字段
df_usa_records=df_country_records.loc["US",
["ConfirmedCases","Fatalities"]]
# 重置索引（自动添加序号索引），便于画图
df_usa_records=df_usa_records.reset_index()
df_usa_records
```

	Date	ConfirmedCases	Fatalities
0	2020-01-22	0.0	0.0
1	2020-01-23	0.0	0.0
2	2020-01-24	0.0	0.0
3	2020-01-25	0.0	0.0
4	2020-01-26	0.0	0.0
...	...	...	...
110	2020-05-11	1347710.0	80677.0
111	2020-05-12	1369403.0	82371.0
112	2020-05-13	1390235.0	84114.0
113	2020-05-14	1417603.0	85893.0
114	2020-05-15	1442653.0	87525.0

将上一步的结果可视化呈现：

```
# 使用plotly画图
fig=px.bar(df_usa_records,x="Date",y="ConfirmedCases",color="ConfirmedCase
s",color_continuous_scale=px.colors.sequential.Magma)
fig.update_layout(title_text='美国随时间确诊病例数')
fig.show()
```

输出结果如下：

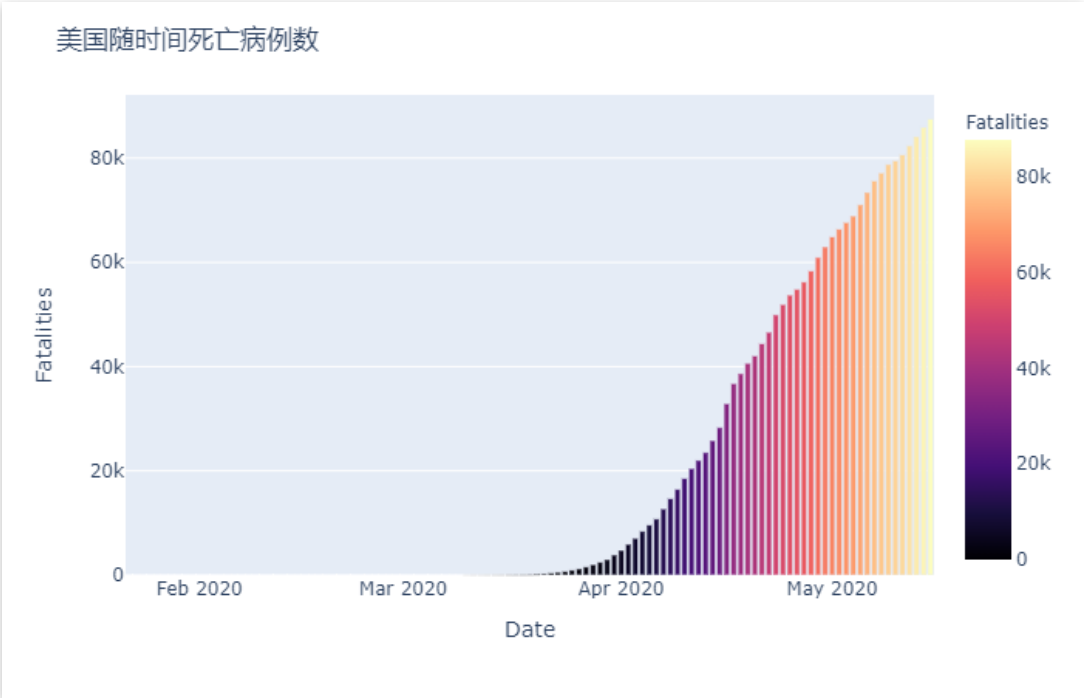


从上图可以看出，从3月中旬开始，美国的确诊病例数随着时间维度开始不断地攀升。由此可以得出，**时间也是影响核心确诊病例总数的核心特征之一。**

再来看一下美国的死亡病例数随时间的变化趋势：

```
fig=px.bar(df_usa_records,x="Date",y="Fatalities",color="Fatalities",color_continuous_scale=px.colors.sequential.Magma)
fig.update_layout(title_text='美国随时间死亡病例数')
fig.show()
```

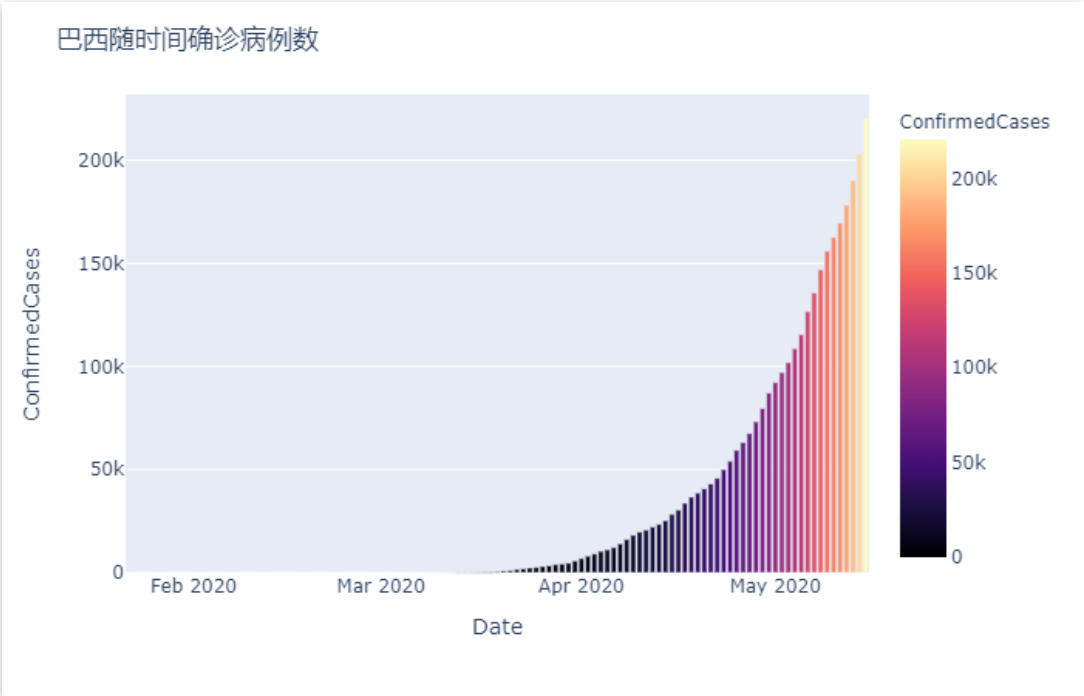
输出结果如下：



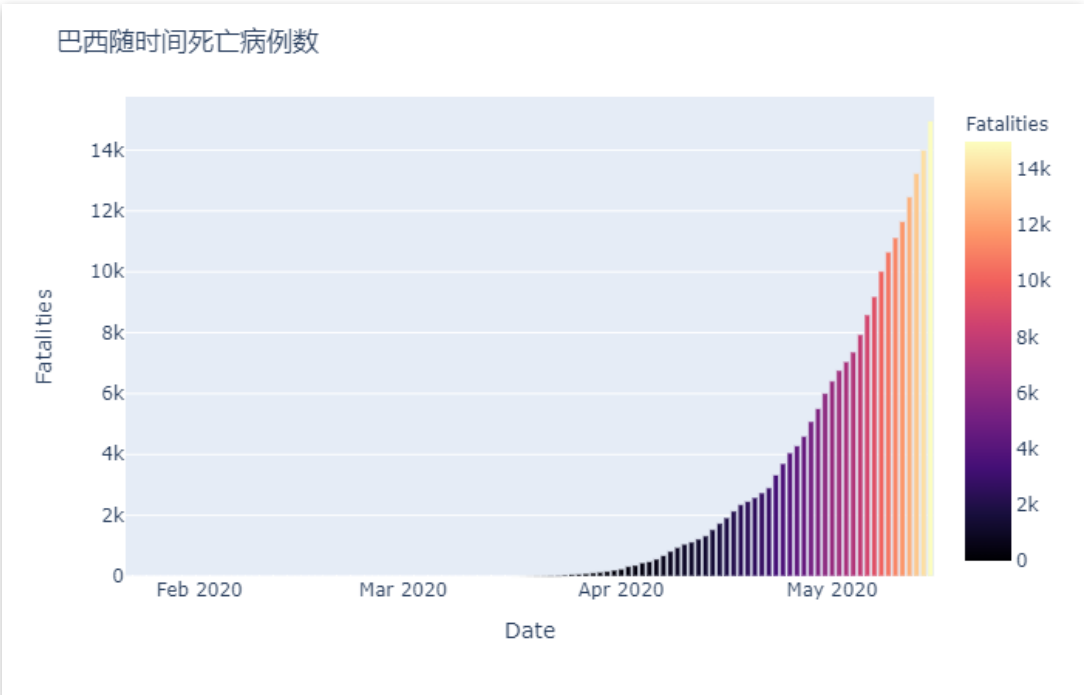
再随机抽样另一个国家的数据，以巴西为例，代码如下：

```
# 巴西确诊病例数情况
```

```
df_brz_records=df_country_records.loc["Brazil",
["ConfirmedCases","Fatalities"]]
df_brz_records=df_brz_records.reset_index()
fig=px.bar(df_brz_records,x="Date",y="ConfirmedCases",color="ConfirmedCase
s",color_continuous_scale=px.colors.sequential.Magma)
fig.update_layout(title_text='巴西随时间确诊病例数')
fig.show()
```




```
# 巴西死亡病例情况
fig=px.bar(df_brz_records,x="Date",y="Fatalities",color="Fatalities",color
_continuous_scale=px.colors.sequential.Magma)
fig.update_layout(title_text='巴西随时间死亡病例数')
fig.show()
```



从上图中可以看出，巴西的确诊数和死亡数与美国类似，都是随着时间不断攀升；同时，这两个国家的确诊病例增速也有所不同，**说明国家和日期这两个维度都是重要参考指标。**

特征工程

从上面的分析中可以得知，国家和日期维度都是重要的参考指标，但国家是字符串类型，日期是特殊类型，都需要转换成数字，从而方便建立模型。

1.处理日期数据

从日期中得出年、月、日，并分别新建字段，代码如下：

```
# 首先将Date字段转换为datetime类型
df_train.Date=pd.to_datetime(df_train.Date)
# apply与lambda配合使用，分别取出日期中的年、月、日，并新建字段
df_train["Year"]=df_train.Date.apply(lambda l:l.year)
df_train["Month"]=df_train.Date.apply(lambda l:l.month)
df_train["Day"]=df_train.Date.apply(lambda l:l.day)
df_train
```

	Id	Province_State	Country_Region	Date	ConfirmedCases	Fatalities	Year	Month	Day
0	1		Afghanistan	2020-01-22	0.0	0.0	2020	1	22
1	2		Afghanistan	2020-01-23	0.0	0.0	2020	1	23
2	3		Afghanistan	2020-01-24	0.0	0.0	2020	1	24
3	4		Afghanistan	2020-01-25	0.0	0.0	2020	1	25
4	5		Afghanistan	2020-01-26	0.0	0.0	2020	1	26
...	...	...	...	...	...	...	...	...	...
35990	35991		Zimbabwe	2020-05-11	36.0	4.0	2020	5	11
35991	35992		Zimbabwe	2020-05-12	36.0	4.0	2020	5	12
35992	35993		Zimbabwe	2020-05-13	37.0	4.0	2020	5	13
35993	35994		Zimbabwe	2020-05-14	37.0	4.0	2020	5	14
35994	35995		Zimbabwe	2020-05-15	42.0	4.0	2020	5	15

35995 rows × 9 columns

2.处理国家特征

在前面的环节中，我们知道省份这一字段存在较多的缺失值，不能直接作为一个特征，但也有1/3的记录是有值的，不能直接抛弃。

更好的方法是将国家+省份拼接在一起，作为一个特征，这样既能尽可能地使用省份这一字段，又能避免太多空值给模型造成影响。

代码如下：

```
# 将“国家”和“省份”拼接在一起
df_train["Country_Region"]=df_train["Country_Region"]+df_train["Province_State"]
# 查看拼接后的数据分布
```

```
df_train["Country_Region"].value_counts()
```

```
Belarus                      115
ChinaShanghai                115
AustraliaSouth Australia     115
Bangladesh                   115
Croatia                      115
                            ...
USGuam                       115
Spain                        115
Qatar                        115
Finland                      115
Guinea                       115
Name: Country_Region, Length: 313, dtype: int64
```

从输出结果中可以看到，有省份数据的记录，已经拼接到了国家这一字段。

接下来，使用sklearn工具包中的LabelEncoder对象，将国家这一字段转换为数字类型。

```
from sklearn.preprocessing import LabelEncoder
encoder=LabelEncoder()
df_train["Country_Region"]=encoder.fit_transform(df_train["Country_Region"
])
df_train
```

	Id	Province_State	Country_Region	Date	ConfirmedCases	Fatalities	Year	Month	Day
0	1		0	2020-01-22	0.0	0.0	2020	1	22
1	2		0	2020-01-23	0.0	0.0	2020	1	23
2	3		0	2020-01-24	0.0	0.0	2020	1	24
3	4		0	2020-01-25	0.0	0.0	2020	1	25
4	5		0	2020-01-26	0.0	0.0	2020	1	26
...	...	...	...	...	...	...	...	...	...
35990	35991		312	2020-05-11	36.0	4.0	2020	5	11
35991	35992		312	2020-05-12	36.0	4.0	2020	5	12
35992	35993		312	2020-05-13	37.0	4.0	2020	5	13
35993	35994		312	2020-05-14	37.0	4.0	2020	5	14
35994	35995		312	2020-05-15	42.0	4.0	2020	5	15

35995 rows x 9 columns

3.抽取训练特征和目标特征

数据处理完毕，接下里从表格中抽取需要的特征。

```
# 选取Country_Region、Year、Month、Day三个字段作为训练特征
df_train_final=df_train[["Country_Region","Year","Month","Day"]]
# 选取ConfirmedCases(确诊病例数)作为预测目标特征
labels=df_train.ConfirmedCases
```

模型训练

准备好特征后，开始着手建立模型。由于国家和确诊病例数之间是非线性关系，所以使用xgboost来建立非线性的模型。

代码如下：

```
# 导入xgboost
from xgboost import XGBRegressor
# 创建xgboost,并配置参数 n_estimators:迭代次数
xgb=XGBRegressor(n_estimators=1000,random_state=0,max_depth=27)
xgb.fit(df_train_final,labels)
```

输出如下，代表模型训练成功。

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=27,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=2000, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

获取结论

模型训练完成后，开始对 test 数据集中的数据进行预测。在此之前，我们需要将 test 数据集进行和之前 train 数据集一样的操作，包括拆分日期，合并国家省份等，因为需要确保预测的特征和训练的特征一致，才能用刚才训练的模型进行预测。

```
# 首先整理test数据集的基本特征
df_test=df_test.fillna("")
df_test.Date=pd.to_datetime(df_test.Date)
df_test["Year"]=df_test.Date.apply(lambda l:l.year)
df_test["Month"]=df_test.Date.apply(lambda l:l.month)
df_test["Day"]=df_test.Date.apply(lambda l:l.day)
df_test["Country_Region"]=df_test["Country_Region"]+df_test["Province_Stat
e"]
df_test["Country_Region"]=encoder.fit_transform(df_test["Country_Region"])
df_test
```

	ForecastId	Province_State	Country_Region	Date	Year	Month	Day
0	1		0	2020-04-02	2020	4	2
1	2		0	2020-04-03	2020	4	3
2	3		0	2020-04-04	2020	4	4
3	4		0	2020-04-05	2020	4	5
4	5		0	2020-04-06	2020	4	6
...	...	...	...	...	...	...	...
13454	13455		312	2020-05-10	2020	5	10
13455	13456		312	2020-05-11	2020	5	11
13456	13457		312	2020-05-12	2020	5	12
13457	13458		312	2020-05-13	2020	5	13
13458	13459		312	2020-05-14	2020	5	14

13459 rows × 7 columns

接下来就是进行预测，并且将预测结果添加到 test 数据表中。

```
df_test_final=df_test[["Country_Region","Year,""Month","Day"]]
df_test["Predict_confirm"]=xgb.predict(df_test_final)
df_test
```

	ForecastId	Province_State	Country_Region	Date	Year	Month	Day	Predict_confirm
0	1		0	2020-04-02	2020	4	2	272.999054
1	2		0	2020-04-03	2020	4	3	281.000397
2	3		0	2020-04-04	2020	4	4	299.000580
3	4		0	2020-04-05	2020	4	5	349.000519
4	5		0	2020-04-06	2020	4	6	367.000305
...	...	...	...	...	...	...	...	...
13454	13455		312	2020-05-10	2020	5	10	36.000092
13455	13456		312	2020-05-11	2020	5	11	35.999012
13456	13457		312	2020-05-12	2020	5	12	36.000759
13457	13458		312	2020-05-13	2020	5	13	36.999180
13458	13459		312	2020-05-14	2020	5	14	37.001465

13459 rows × 8 columns

测试数据是从 4 月 2 号开始的，而训练数据也包含这个日期的数据，我们可以查看一下训练数据中这部分数据的取值。

```
# 查看训练数据，验证预测
df_train[df_train.Date>="2020-04-02"]
```

	Id	Province_State	Country_Region	Date	ConfirmedCases	Fatalities	Year	Month	Day
71	72		0	2020-04-02	273.0	6.0	2020	4	2
72	73		0	2020-04-03	281.0	6.0	2020	4	3
73	74		0	2020-04-04	299.0	7.0	2020	4	4
74	75		0	2020-04-05	349.0	7.0	2020	4	5
75	76		0	2020-04-06	367.0	11.0	2020	4	6
...	...	...	...	...	...	...	...	...	...
35990	35991		312	2020-05-11	36.0	4.0	2020	5	11
35991	35992		312	2020-05-12	36.0	4.0	2020	5	12
35992	35993		312	2020-05-13	37.0	4.0	2020	5	13
35993	35994		312	2020-05-14	37.0	4.0	2020	5	14
35994	35995		312	2020-05-15	42.0	4.0	2020	5	15

13772 rows x 9 columns

通过对比两张表，可以发现本次预测比较准确。

Tips：严格来说，训练数据不应包含测试数据，这样会导致模型对测试数据效果过于好，进而不足够说明模型的效果。 但本次实验以练习为主，且样本数据量较少，不宜拆分，所以没有进行额外的处理。