西北大学

2022-2023 学年第一学期
本科生课程作业

题目： COVID-19 患者死亡的危险因素分析及预测

课程名称	多元统计分析
姓　　名	杨浩然
学　　号	2020116130
院　　系	数学学院
专　　业	应用统计学

2023 年 1 月

目录：

一. 问题重述

冠状病毒病（COVID-19）是由一种新发现的冠状病毒引起的传染病。大多数感染 COVID-19 病毒的人会出现轻度至中度呼吸道疾病，无需特殊治疗即可康复。老年人以及患有心血管疾病、糖尿病、慢性呼吸道疾病和癌症等潜在健康问题的人更容易患上严重疾病。
在整个大流行病过程中，医疗保健提供者面临的主要问题之一是医疗资源短缺以及有效分配这些资源的适当计划。在这些困难时期，能够预测一个人在检测呈阳性时或在此之前可能需要什么样的资源，这将对政府有巨大帮助，因为他们将能够采购和安排必要的资源挽救那个病人的生命。
该项目的主要目标是建立一个机器学习模型，根据 Covid-19 患者的当前症状、状态和病史，预测患者是否处于高危状态。

二. 数据简介

该数据集由墨西哥政府提供。该数据集包含大量匿名的患者相关信息，包括先决条件。原始数据集包含 21 个独特的特征和 1,048,576 名独特的患者。在布尔特征中，1 表示“是”，2 表示“否”。值 97 和 99 是缺失数据。

- 性别：1 代表女性，2 代表男性。
- 年龄：患者。
- 分类：covid 测试结果。值 1-3 表示患者被诊断为不同程度的 covid。4 或更高意味着患者不是 covid 的携带者或测试没有定论。
- 患者类型：患者在单位接受的护理类型。1 人回家，2 人住院。
- 肺炎：患者是否已经有气囊炎症。
- 怀孕：患者是否怀孕。
- 糖尿病：患者是否患有糖尿病。
- copd：表示患者是否患有慢性阻塞性肺疾病。
- 哮喘：患者是否患有哮喘。
- inmsupr：患者是否免疫抑制。
- 高血压：患者是否患有高血压。
- 心血管：患者是否患有心脏或血管相关疾病。
- 肾慢性：患者是否患有慢性肾病。
- 其他疾病：患者是否患有其他疾病。
- 肥胖：患者是否肥胖。
- 烟草：患者是否是烟草使用者。
- usmr：表示患者是否接受过一级、二级或三级医疗单位的治疗。
- 医疗单位：提供医疗服务的国家卫生系统机构类型。
- 插管：患者是否连接到呼吸机。
- icu：表示患者是否已入住重症监护病房。
- 死亡日期：如果患者死亡，请注明死亡日期，否则为 9999-99-99。

三. 数据集概述

3.1 导入相关包

In [53]:

```
# 导入数据分析、科学计算及可视化的相关包
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings("ignore")   # 忽略警告
```

3.2 导入数据

In [54]:

```
# 数据集概述（概览）、预处理
df = pd.read_csv("Covid Data.csv")  # 读取数据，返回DataFrame
print("Shape of df :", df.shape)  # 查看数据的维数
df.head()  # 展示前5行数据
```

```
Shape of df : (1048575, 21)
```

Out[54]:

	USMER	MEDICAL_UNIT	SEX	PATIENT_TYPE	DATE_DIED	INTUBED	PNEUMONIA	AGE	PREGNANT	DIABETES	...	ASTHMA	INMSUPR	HIPERTENSION
0	2	1	1	1	03/05/2020	97	1	65	2	2	...	2	2	1
1	2	1	2	1	03/06/2020	97	1	72	97	2	...	2	2	1
2	2	1	2	2	09/06/2020	1	2	55	97	1	...	2	2	2
3	2	1	1	1	12/06/2020	97	2	53	2	2	...	2	2	2
4	2	1	2	1	21/06/2020	97	2	68	97	1	...	2	2	1

5 rows × 21 columns

3.3 查看数据基本信息

In [55]:

```
df.info()  # 查看数据的基本信息
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 21 columns):
 #   Column                Non-Null Count    Dtype
---  ------                --------------    -----
 0   USMER                 1048575 non-null  int64
 1   MEDICAL_UNIT          1048575 non-null  int64
 2   SEX                   1048575 non-null  int64
 3   PATIENT_TYPE          1048575 non-null  int64
 4   DATE_DIED             1048575 non-null  object
 5   INTUBED               1048575 non-null  int64
 6   PNEUMONIA             1048575 non-null  int64
 7   AGE                   1048575 non-null  int64
 8   PREGNANT              1048575 non-null  int64
 9   DIABETES              1048575 non-null  int64
 10  COPD                  1048575 non-null  int64
 11  ASTHMA                1048575 non-null  int64
 12  INMSUPR               1048575 non-null  int64
 13  HIPERTENSION          1048575 non-null  int64
 14  OTHER_DISEASE         1048575 non-null  int64
 15  CARDIOVASCULAR        1048575 non-null  int64
 16  OBESITY               1048575 non-null  int64
 17  RENAL_CHRONIC         1048575 non-null  int64
 18  TOBACCO               1048575 non-null  int64
 19  CLASIFFICATION_FINAL  1048575 non-null  int64
 20  ICU                   1048575 non-null  int64
dtypes: int64(20), object(1)
memory usage: 168.0+ MB
```

3.4 检查缺失值并打印每个特征的取值个数

```
In [56]:
1 df.isna().sum().sum()  # 检查是否有缺失值
```

```
Out[56]: 0
```

```
In [57]:
1 # 发现没有NA值。
2 # 打印每个特征的取值个数
3 for i in df.columns:
4     print(i, "=>\t", len(df[i].unique()))
```

```
USMER =>          2
MEDICAL_UNIT =>   13
SEX =>    2
PATIENT_TYPE =>   2
DATE_DIED =>      401
INTUBED =>        4
PNEUMONIA =>      3
AGE =>    121
PREGNANT =>       4
DIABETES =>       3
COPD =>   3
ASTHMA =>         3
INMSUPR =>        3
HIPERTENSION =>   3
OTHER_DISEASE =>          3
CARDIOVASCULAR =>         3
OBESITY =>        3
RENAL_CHRONIC =>          3
TOBACCO =>        3
CLASIFFICATION_FINAL =>   7
ICU =>    4
```

3.5 查看死亡日期的取值分布

```
In [58]:
1 df.DATE_DIED.value_counts()  # 查看死亡日期的取值分布（9999-99-99表示未死亡）
```

```
Out[58]: 9999-99-99    971633
06/07/2020      1000
07/07/2020       996
13/07/2020       990
16/06/2020       979
                 ...
24/11/2020         1
17/12/2020         1
08/12/2020         1
16/03/2021         1
22/04/2021         1
Name: DATE_DIED, Length: 401, dtype: int64
```

3.6 查看是否有肺炎的取值分布

```
In [59]:
1 df.PNEUMONIA.value_counts()  # 查看是否有肺炎的取值分布（1True、2False）
```

```
Out[59]: 2     892534
1     140038
99     16003
Name: PNEUMONIA, dtype: int64
```

四. 数据预处理

- 我们有一些功能，我们希望他们只有 2 个唯一的值。但是我们看到这些特征有 3 或 4 个唯一值。例如，"肺炎"功能有 3 个唯一值（1,2,99）99 表示 NaN 值。因此，我们将只取包含 1 和 2 值的行。
- 在“DATE_DIED"列中，我们有 971633 个“9999-99"值，代表活着的患者所以我会把这个功能作为一个“死亡"，包括病人是否死亡。

4.1 剔除除“INTUBED”、“怀孕”、“ICU”外的特征缺失值

In [9]:
```
# 数据预处理
# 剔除除“INTUBED”、“怀孕”、“ICU”外的特征缺失值
df = df[(df.PNEUMONIA == 1) | (df.PNEUMONIA == 2)]  # 剔除“是否有肺炎”缺失值
df = df[(df.DIABETES == 1) | (df.DIABETES == 2)]  # 剔除“是否有糖尿病”缺失值
df = df[(df.COPD == 1) | (df.COPD == 2)]  # 剔除“是否有慢性阻塞性肺疾病”缺失值
df = df[(df.ASTHMA == 1) | (df.ASTHMA == 2)]  # 剔除“是否有哮喘”缺失值
df = df[(df.INMSUPR == 1) | (df.INMSUPR == 2)]  # 剔除“是否有免疫抑制”缺失值
df = df[(df.HIPERTENSION == 1) | (df.HIPERTENSION == 2)]  # 剔除“是否有高血压”缺失值
df = df[(df.OTHER_DISEASE == 1) | (df.OTHER_DISEASE == 2)]  # 剔除“是否有其他疾病”缺失值
df = df[(df.CARDIOVASCULAR == 1) | (df.CARDIOVASCULAR == 2)]  # 剔除“是否有心血管疾病”缺失值
df = df[(df.OBESITY == 1) | (df.OBESITY == 2)]  # 剔除“是否有肥胖”缺失值
df = df[(df.RENAL_CHRONIC == 1) | (df.RENAL_CHRONIC == 2)]  # 剔除“是否有慢性肾病”缺失值
df = df[(df.TOBACCO == 1) | (df.TOBACCO == 2)]  # 剔除“是否吸烟”缺失值
```

4.2 预处理“日期死亡”列

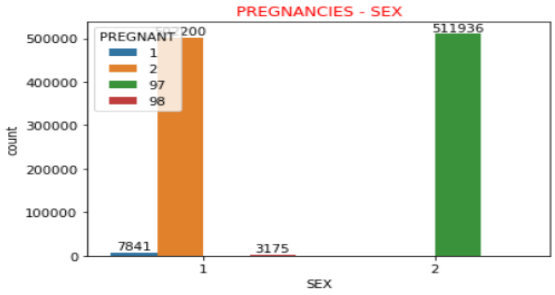
In [10]:
```
# 预处理“日期死亡”列：值“9999-99”意味着这个病人还活着。
# 遵循原始数据中的布尔特征表述：1表示死亡，2表示未死亡
df["DEATH"] = [2 if each == "9999-99-99" else 1 for each in df.DATE_DIED]
```

4.3 剔除“怀孕”缺失值

绘制怀孕-性关系图：

In [11]:
```
# 怀孕-性关系图
plt.figure()
# ax = sns.countplot(df.SEX, hue=df.PREGNANT)
ax = sns.countplot(x=df.SEX, hue=df.PREGNANT)  # hue参数指定分组变量
for bars in ax.containers:  # ax.containers是一个列表，包含了每个柱子的信息
    ax.bar_label(bars)  # 在每个柱子上添加标签
plt.title("PREGNANCIES - SEX", color="red")  # 添加标题
```

Out[11]: Text(0.5, 1.0, 'PREGNANCIES - SEX')



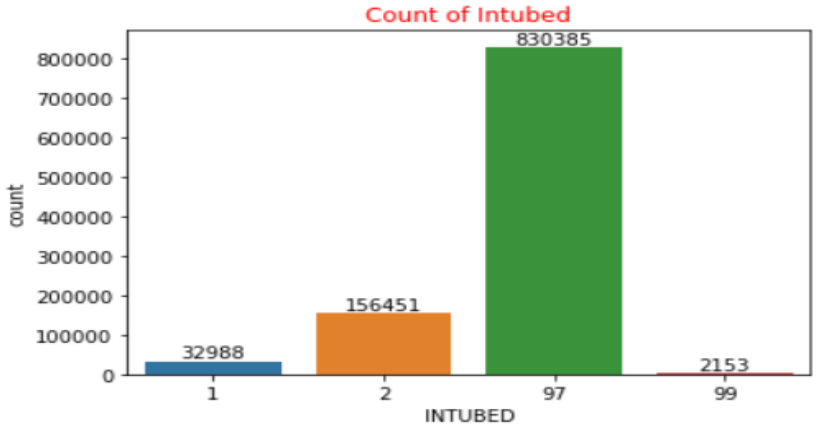
我们看到，所有的“97"值是男性和男性不能怀孕，所以我们将转换 97 为 2。根据以上推论转换过程：

In [12]:
```
# 我们看到，所有的“97”值是男性和男性不能怀孕，所以我们将转换97为2。

# 根据以上推论转换过程
df.PREGNANT = df.PREGNANT.replace(97, 2)

# 消除缺失值
# 剔除“怀孕”缺失值，剩余数据为：1True、2False
df = df[(df.PREGNANT == 1) | (df.PREGNANT == 2)]
```

4.4 INTUBED 特征缺失值分析

In [13]:
```
# INTUBED特征缺失值分析
ax = sns.countplot(x=df.INTUBED)  # 绘制柱状图
plt.bar_label(ax.containers[0])  # 显示柱状图的数值
plt.title("Count of Intubed", color="red")  # 设置标题
```

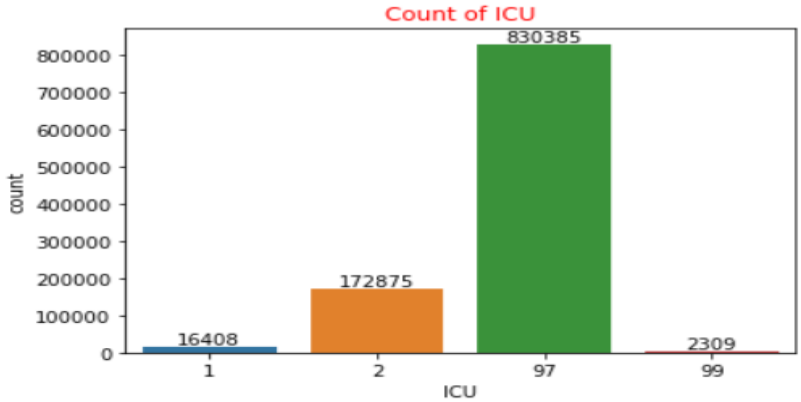
Out[13]: Text(0.5, 1.0, 'Count of Intubed')



4.5 ICU 特征缺失值分析

In [14]:
```
# “ICU”特征缺失值分析
ax = sns.countplot(x=df.ICU)  # 绘制柱状图
plt.bar_label(ax.containers[0])  # 显示柱状图的数值
plt.title("Count of ICU", color="red")  # 设置标题
```

Out[14]: Text(0.5, 1.0, 'Count of ICU')



- 在“INTUBED”和“ICU”中有太多缺少的值，下面将删除它们。
- 另外删除的还有“DATE_DEAD”列，将使用“Death”特征代替。

In [15]:
```
# 在“INTUBED”和“ICU”中有太多缺少的值，下面将删除它们。
# 另外删除的还有“DATE_DEAD”列，将使用“Death”特征代替。
df.drop(columns=["INTUBED", "ICU", "DATE_DIED"], inplace=True)
```

4.6 打印数据预处理后的每个特征的取值

至此，预处理基本结束，再次打印每个特征的取值个数

In [16]:
```
# 至此，预处理基本结束，再次打印每个特征的取值个数
for i in df.columns:
    print(i, "=>\t", len(df[i].unique()))
```

```
USMER =>         2
MEDICAL_UNIT =>  13
SEX =>   2
PATIENT_TYPE =>  2
PNEUMONIA =>     2
AGE =>   121
PREGNANT =>      2
DIABETES =>      2
COPD =>  2
ASTHMA =>        2
INMSUPR =>       2
HIPERTENSION =>  2
OTHER_DISEASE =>         2
CARDIOVASCULAR =>        2
OBESITY =>       2
RENAL_CHRONIC =>         2
TOBACCO =>       2
CLASIFFICATION_FINAL =>  7
DEATH =>         2
```

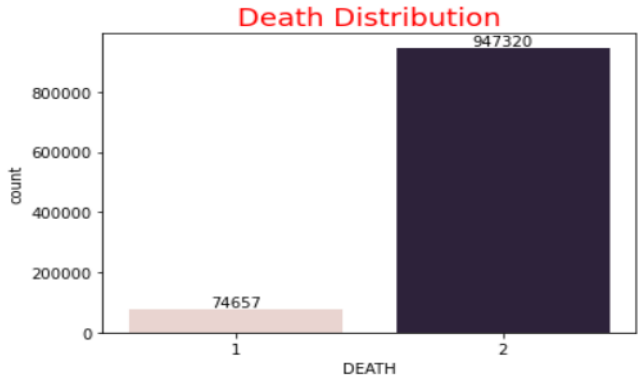
如上，我们只剩一个被称为“年龄”的数字特征，其余特征都是明确的。

五. 数据可视化

5.1 绘制死亡的直方图

In [17]:
```
# 数据可视化
ax = sns.countplot(x=df.DEATH, palette=sns.cubehelix_palette(2))  # 绘制柱状图
plt.bar_label(ax.containers[0])  # 显示柱状图的数值
plt.title("Death Distribution", fontsize=18, color="red")  # 设置标题
```

Out[17]: Text(0.5, 1.0, 'Death Distribution')

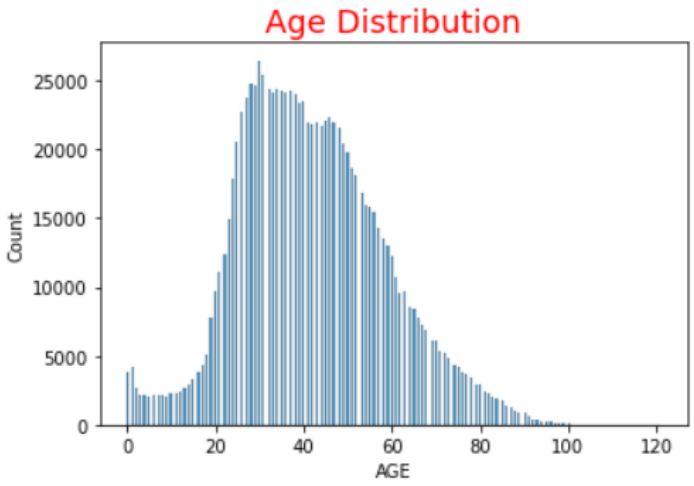


可以看到值在类别（目标）列中的分布并不平衡。这可能会导致模型在测试集合上出现不平衡的问题，这个留到后面再进行解决。

5.2 绘制年龄的直方图

In [18]:
```
# 绘制年龄的直方图
sns.histplot(x=df.AGE)
plt.title("Age Distribution", color="red", fontsize=18)  # 设置标题
```

Out[18]: Text(0.5, 1.0, 'Age Distribution')

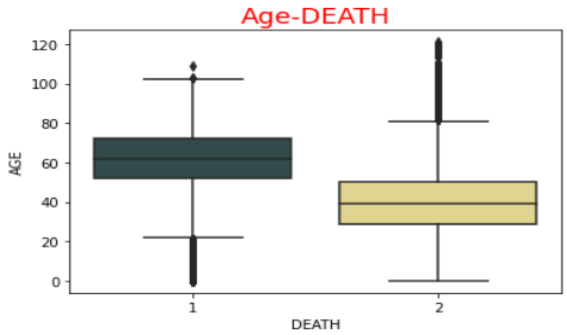


从直方图中可以看出患者大致集中在在 20~60 岁之间。

5.3 绘制年龄-死亡关系图

In [19]:
```
# 绘制年龄-死亡关系图
sns.boxplot(x="DEATH", y="AGE", data=df,
            palette=sns.color_palette(["#2f4f4f", "#eedd82"]))  # 绘制箱线图
plt.title("Age-DEATH", fontsize=18, color="red")  # 设置标题
```

Out[19]: Text(0.5, 1.0, 'Age-DEATH')

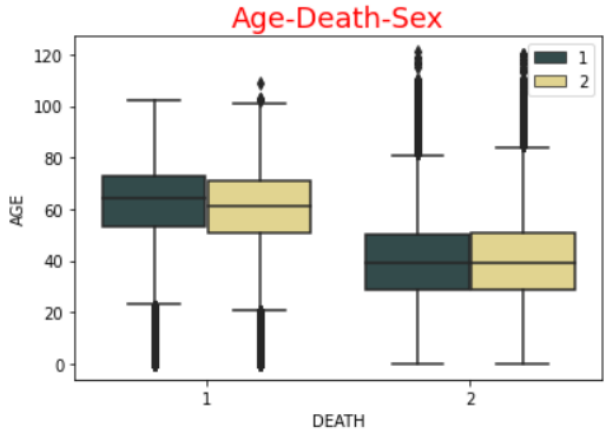


从上面的箱线图我们可以看出老年患者要比年轻患者更容易死亡

5.4 绘制年龄-死亡-性别关系图

In [20]:
```
# 绘制年龄-死亡-性别关系图
sns.boxplot(x="DEATH", y="AGE", hue="SEX", data=df,
            palette=sns.color_palette(["#2f4f4f", "#eedd82"]))  # 绘制箱线图
plt.title("Age-Death-Sex", fontsize=18, color="red")  # 设置标题
plt.legend(loc="best")  # 设置图例
```

Out[20]: <matplotlib.legend.Legend at 0x20321e3bd90>

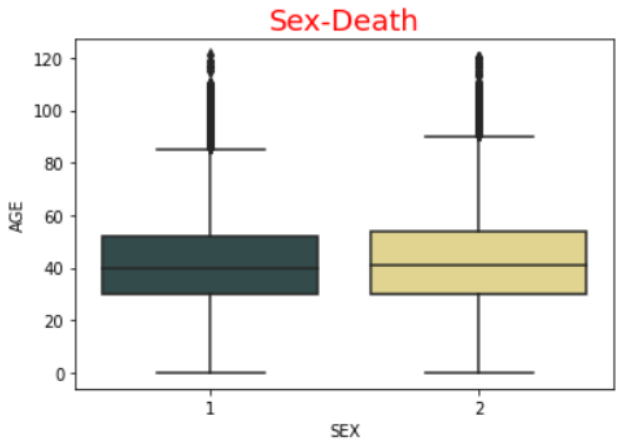


- 老年患者比年轻患者更容易死亡。（上面已经得出过了）
- 从新的箱线图中可以看出病人在平均发病率方面，男女之间没有很大的差别。（下面单独绘制进行校验）

5.5 绘制性别-死亡关系图

In [21]:
```
# 绘制性别-死亡关系图
# sns.countplot(df.SEX, hue=df.DEATH, palette=sns.cubehelix_palette(2))
sns.boxplot(x="SEX", y="AGE", data=df,
            palette=sns.color_palette(["#2f4f4f", "#eedd82"]))  # 绘制箱线图
plt.title("Sex-Death", fontsize=18, color="red")  # 设置标题
# plt.legend(loc="best")  # 设置图例
```

Out[21]: Text(0.5, 1.0, 'Sex-Death')

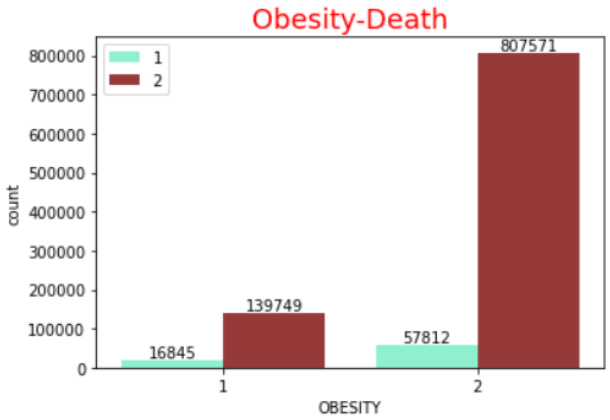


与女性相比，男性更容易死于 COVID。（不显著）

5.6 绘制肥胖-死亡关系图

In [22]:
```
# 绘制肥胖-死亡关系图
ax = sns.countplot(x=df.OBESITY, hue=df.DEATH,
                   palette=sns.color_palette(["#7fffd4", "#a52a2a"]))  # 绘制柱状图
plt.title("Obesity-Death", fontsize=18, color="red")  # 设置标题
plt.bar_label(ax.containers[0])  # 显示柱状图的数值
plt.bar_label(ax.containers[1])  # 显示柱状图的数值
plt.legend(loc="best")  # 设置图例
```

Out[22]: <matplotlib.legend.Legend at 0x20321ac3670>

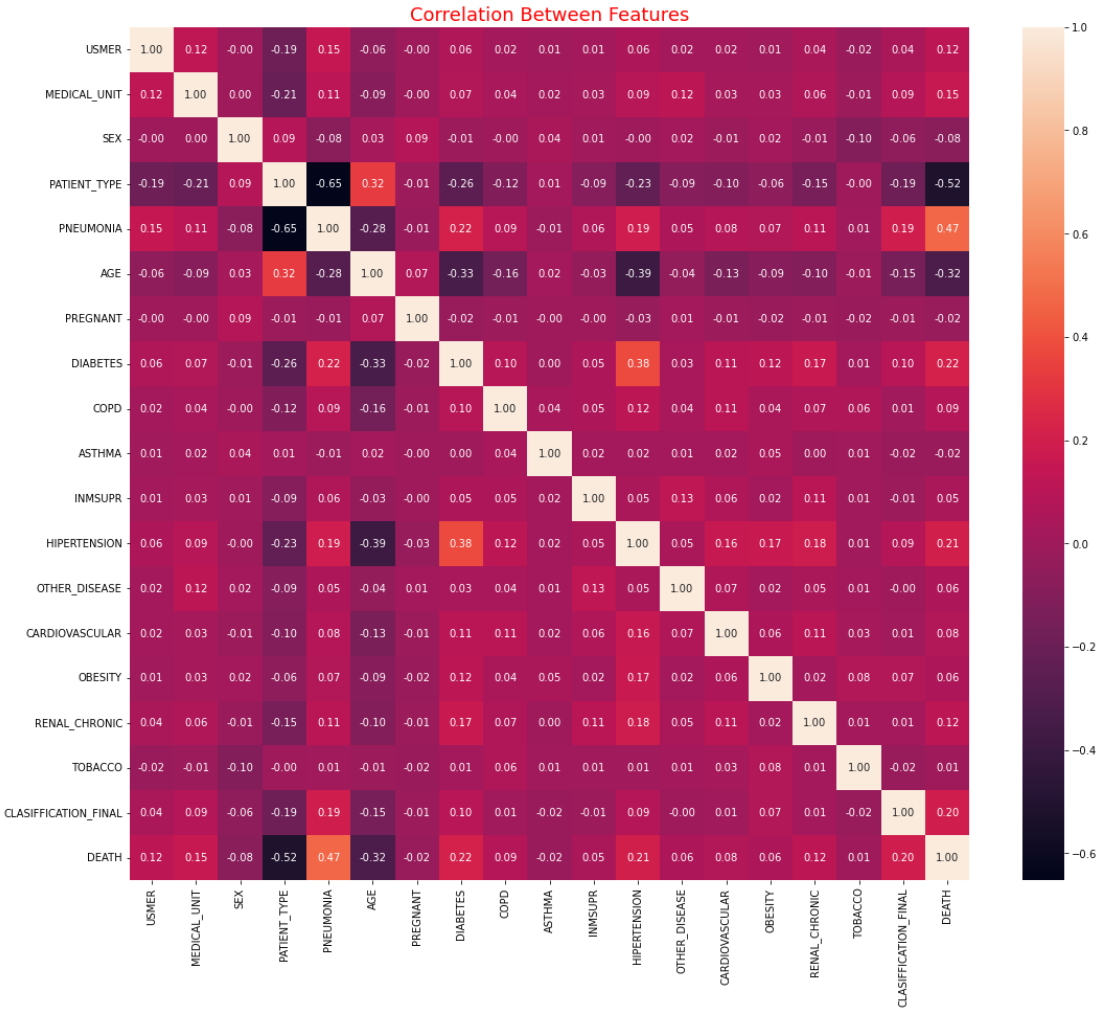


从肥胖-死亡关系柱状图中可以看出肥胖患者比非肥胖患者更容易死于 COVID。

5.7 使用热力图可视化特征之间的相关性

In [23]:
```
# 使用热力图可视化特征之间的相关性
plt.figure(figsize=(18, 15))  # 设置画布大小
sns.heatmap(df.corr(), annot=True, fmt=".2f")  # 绘制热力图
plt.title("Correlation Between Features", fontsize=18, color="red")  # 设置标题
```

Out[23]: Text(0.5, 1.0, 'Correlation Between Features')



六. 解决不平衡数据集问题

- 加载更多数据
- 更改性能指标
- 重采样（欠采样或过采样）
    - 欠采样是一种通过保持所有数据在少数类和减少大多数类的大小来平衡不均衡的数据集的技术。
    - 下面将使用欠采样的方式尝试解决数据集的不平衡问题，之所以选择欠采样是因为我们拥有很多的病人数据，如果使用过采样将增加太多数据。
    - 如果使用欠采样仍然没有解决问题再考虑其他。
- 改变算法
- 惩罚模型等。

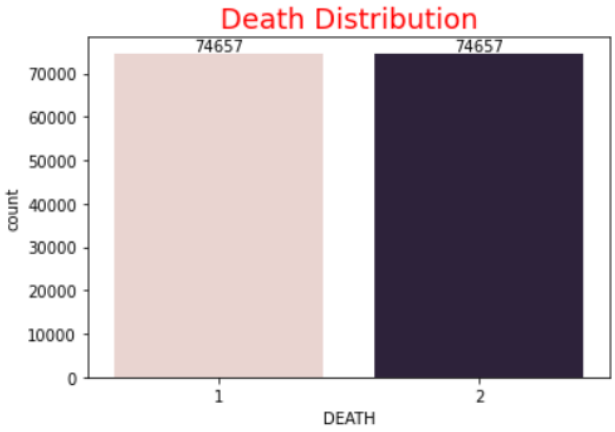
6.1 使用 RandomUnderSampler 进行欠采样

In [34]:
```
# 使用RandomUnderSampler进行欠采样
# 导入RandomUnderSampler
from imblearn.under_sampling import RandomUnderSampler

# 实例化RandomUnderSampler
rus = RandomUnderSampler(random_state=0)
x_resampled, y_resampled = rus.fit_resample(x, y)  # 施行欠采样
```

6.2 查看欠采样后的数据集

In [35]:
```
# 查看欠采样后的数据集
ax = sns.countplot(x=y_resampled, palette=sns.cubehelix_palette(2))  # 绘制柱状图
plt.bar_label(ax.containers[0])  # 显示柱状图的数值
plt.title("Death Distribution", fontsize=18, color="red")  # 设置标题
```

Out[35]: Text(0.5, 1.0, 'Death Distribution')



七. 建立逻辑回归模型

7.1 删除与"死亡"相关性较低的特征

In [24]:
```
# 根据热力图罗列出的与“DEATH”特征无关（相关性低）的特征
unrelevant_columns = ["SEX", "PREGNANT", "COPD", "ASTHMA", "INMSUPR", "OTHER_DISEASE", "CARDIOVASCULAR",
                      "OBESITY", "TOBACCO"]

# 删除无关特征
df.drop(columns=unrelevant_columns, inplace=True)
```

查看预处理后的数据集的前 5 行

In [25]:
```
# 查看预处理后的数据集的前5行
df.head()
```

Out[25]:

	USMER	MEDICAL_UNIT	PATIENT_TYPE	PNEUMONIA	AGE	DIABETES	HIPERTENSION	RENAL_CHRONIC	CLASIFFICATION_FINAL	DEATH
0	2	1	1	1	65	2	1	2	3	1
1	2	1	1	1	72	2	1	1	5	1
2	2	1	2	2	55	1	2	2	3	1
3	2	1	1	2	53	2	2	2	7	1
4	2	1	1	2	68	1	1	2	3	1

7.2 将分类特征转换为哑变量

In [26]:
```
# 将分类特征转换为哑变量
df = pd.get_dummies(
    df, columns=["MEDICAL_UNIT", "CLASIFFICATION_FINAL"], drop_first=True)
```

7.3 使用对离群值具有鲁棒性的统计量缩放功能

In [27]:
```
# 使用对离群值具有鲁棒性的统计量缩放功能。
from sklearn.preprocessing import RobustScaler  # 导入RobustScaler
scaler = RobustScaler()  # 实例化RobustScaler
df.AGE = scaler.fit_transform(df.AGE.values.reshape(-1, 1))  # 对AGE列进行缩放
```

7.4 将数据集分为特征集和标签集

In [28]:
```
# 将数据集分为特征集和标签集
x = df.drop(columns="DEATH")
y = df["DEATH"]
```

7.5 划分训练集和测试集

In [36]:
```
# 可以看到经过欠采样后，数据集中的“DEATH”特征的分布情况已经变得平衡了
# 接下来对欠采样后的数据集进行训练集和测试集的划分
train_x, test_x, train_y, test_y = train_test_split(
    x_resampled, y_resampled, test_size=0.2, random_state=42)

# 查看训练集和测试集的大小
print("Train_x :", train_x.shape)
print("Test_x :", test_x.shape)
print("Train_y :", train_y.shape)
print("Test_y :", test_y.shape)
```

```
Train_x : (119451, 25)
Test_x : (29863, 25)
Train_y : (119451,)
Test_y : (29863,)
```

7.6 建立逻辑回归模型

In [37]:
```
# 欠采样后的逻辑回归
# 重新初始化逻辑回归模型进行训练
logreg = LogisticRegression()
logreg.fit(train_x, train_y)
print("Logistic Regression Accuracy :",
      logreg.score(test_x, test_y))  # 计算模型的准确率并打印
```

```
Logistic Regression Accuracy : 0.9052673877373338
```

可以看出使用逻辑回归分析的准确性较好。但它可能会误导我们，所以接下来将检查其他指标。

7.7 使用 F1 Score 评估模型

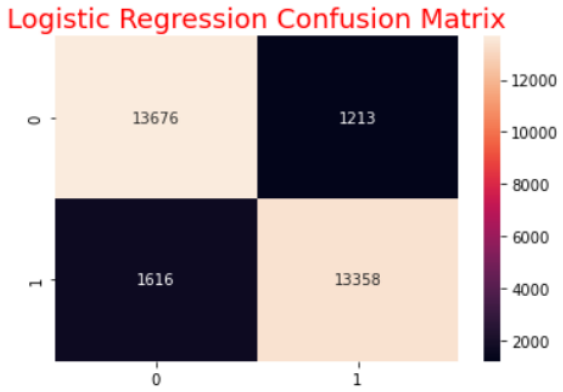
In [38]:
```
# 欠采样后的模型准确率反而下降了
# 推测：欠采样后的数据集中的“DEATH”特征的分布情况已经变得平衡了，但是数据集中的数据量变少了，所以模型的准确率反而下降了
# 使用F1 Score评估模型
print("Logistic Regression F1 Score :", f1_score(
    test_y, logreg.predict(test_x), average=None))
```

```
Logistic Regression F1 Score : [0.90626553 0.90424776]
```

7.8 使用混淆矩阵评估模型

In [39]:
```
# 我们可以发现使用欠采样后的数据进行逻辑回归模型的训练F1 Score的值得到的不错的结果。
# 使用混淆矩阵评估模型
sns.heatmap(confusion_matrix(test_y, logreg.predict(test_x)),
            annot=True, fmt=".0f")  # 绘制混淆矩阵
plt.title("Logistic Regression Confusion Matrix",
          fontsize=18, color="red")  # 设置标题
```

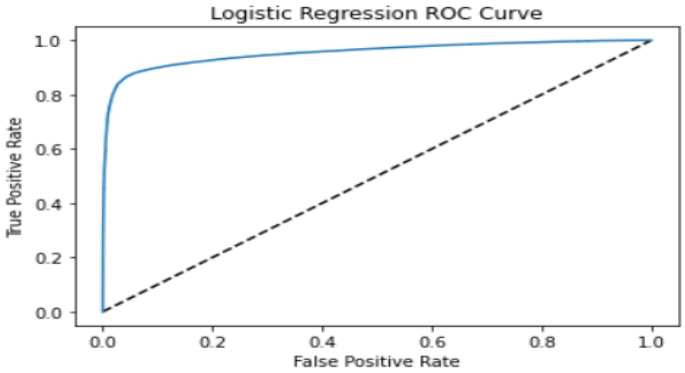
Out[39]: Text(0.5, 1.0, 'Logistic Regression Confusion Matrix')



通过 F1 和混淆矩阵发现使用欠采样可以很好解决前面提到过数据集分类特征的不均衡可能导致问题出现。

7.9 使用 ROC 曲线评估模型

In [40]:
```
# 绘制逻辑回归曲线
# 使用ROC曲线评估模型
from sklearn.metrics import roc_curve
# Replace 2 with 1 and 1 with 0
test_y = test_y.replace({2: 1, 1: 0})
logreg_pred_proba = logreg.predict_proba(test_x)  # 计算预测概率

fpr, tpr, thresholds = roc_curve(
    test_y, logreg_pred_proba[:, 1])  # 计算ROC曲线的FPR和TPR
plt.plot([0, 1], [0, 1], "k--")  # 绘制对角线
plt.plot(fpr, tpr, label="Logistic Regression")  # 绘制ROC曲线
plt.xlabel("False Positive Rate")  # 设置X轴标签
plt.ylabel("True Positive Rate")  # 设置Y轴标签
plt.title("Logistic Regression ROC Curve")  # 设置标题
plt.show()  # 显示图像
```



7.10 模型优点

- 本模型使用逻辑回归对于“死亡”的预测情况，准确率为 90.52%。
- 使用欠采样算法解决了不平衡的数据集问题。
- 本模型通过多项检验。

八. 建立随机森林模型

8.1 对欠采样后的数据集进行训练集和测试集的划分

In [49]:
```
# 可以看到经过欠采样后，数据集中的“DEATH”特征的分布情况已经变得平衡了
# 接下来对欠采样后的数据集进行训练集和测试集的划分
train_x, test_x, train_y, test_y = train_test_split(
    x_resampled, y_resampled, test_size=0.2, random_state=42)

# 查看训练集和测试集的大小
print("Train_x :", train_x.shape)
print("Test_x :", test_x.shape)
print("Train_y :", train_y.shape)
print("Test_y :", test_y.shape)
```

```
Train_x : (119451, 25)
Test_x : (29863, 25)
Train_y : (119451,)
Test_y : (29863,)
```

8.3 建立随机森林模型

In [43]:
```
from sklearn.ensemble import RandomForestClassifier

random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(train_x, train_y)
```

Out[43]:
```
RandomForestClassifier
RandomForestClassifier()
```

In [79]:
```
from sklearn.metrics import accuracy_score

print('Random Forest Accuracy Score :', accuracy_score(test_y,random_forest.predict(test_x)))
```

```
Random Forest Accuracy Score : 0.9014164685396645
```

8.4 使用 F1 Score 评估模型

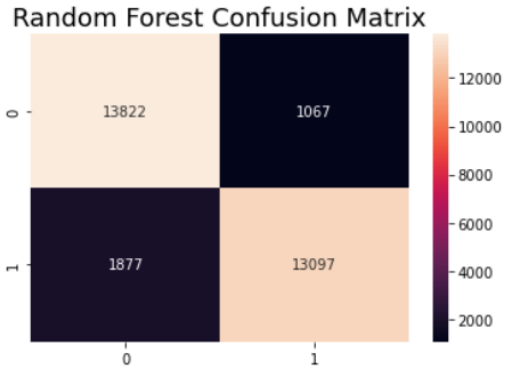
In [81]:
```
print("random_forest F1 Score :", f1_score(
    test_y, random_forest.predict(test_x), average=None))
```

```
random_forest F1 Score : [0.90375311 0.89896355]
```

8.5 使用混淆矩阵评估模型

In [80]:
```
sns.heatmap(confusion_matrix(test_y, random_forest.predict(test_x)), annot=True, fmt='.0f')
plt.title("Random Forest Confusion Matrix", fontsize=18)
```

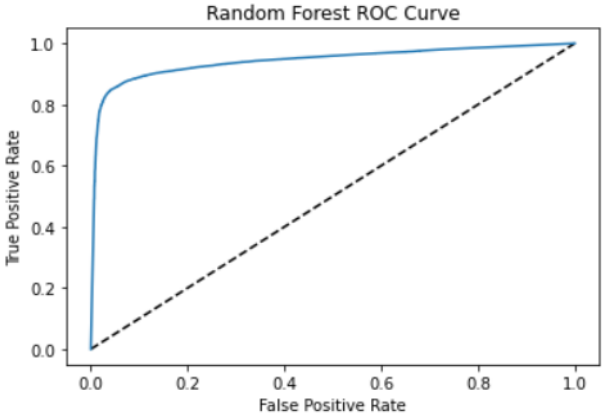
Out[80]:
```
Text(0.5, 1.0, 'Random Forest Confusion Matrix')
```



8.6 使用 ROC 曲线评估模型

In [82]:

```
# 使用ROC曲线评估模型
from sklearn.metrics import roc_curve
# Replace 2 with 1 and 1 with 0
test_y = test_y.replace({2: 1, 1: 0})
random_forest_pred_proba = random_forest.predict_proba(test_x)  # 计算预测概率

fpr, tpr, thresholds = roc_curve(
    test_y, random_forest_pred_proba[:, 1])  # 计算ROC曲线的FPR和TPR
plt.plot([0, 1], [0, 1], "k--")  # 绘制对角线
plt.plot(fpr, tpr, label="Random Forest")  # 绘制ROC曲线
plt.xlabel("False Positive Rate")  # 设置X轴标签
plt.ylabel("True Positive Rate")  # 设置Y轴标签
plt.title("Random Forest ROC Curve")  # 设置标题
plt.show()  # 显示图像
```



8.7 模型优点

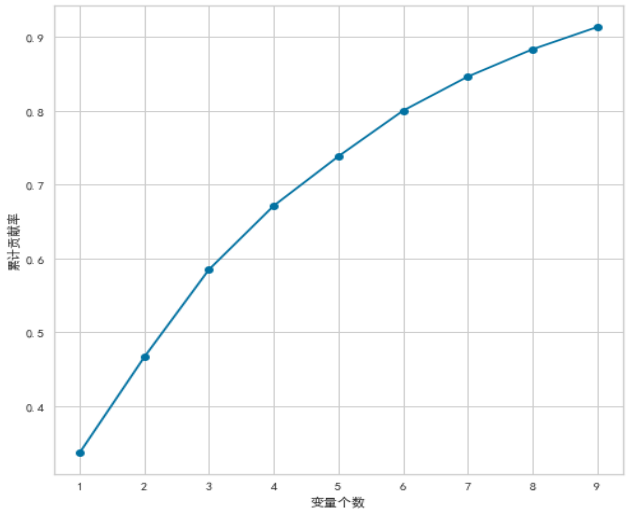
- 本模型使用随机森林模型对于“死亡”的预测情况，准确率为 90.14%。
- 使用欠采样算法解决了不平衡的数据集问题。
- 本模型通过多项检验

九. 建立 K-Means 聚类模型

9.1 利用 PCA 进行降维

In [166]:

```
from sklearn.decomposition import PCA
import matplotlib as mpl

mpl.rcParams['font.sans-serif']=['SimHei']
mpl.rcParams['axes.unicode_minus']=False

from sklearn.decomposition import PCA
X=train_x
pca=PCA(n_components=9)
pca=pca.fit(X)

plt.figure(figsize=(8,8))
# plt.style.use('default')
plt.plot(range(1,10),pca.explained_variance_ratio_.cumsum(),marker='o',linestyle='-')
plt.xlabel("变量个数")
plt.ylabel("累计贡献率")
```



由图知我们将对数据降维成 9 类数据，事实上，我们已经在前文根据相关性分析将数据降维成 9 类

In [167]:
```
pca = PCA(n_components=9)
pca.fit(X)
df_pca = pd.DataFrame(pca.transform(X),columns=['PA1','PA2','PA3','PA4','PA5','PA6','PA7','PA8','PA9'])
df_pca
```

Out[167]:

	PA1	PA2	PA3	PA4	PA5	PA6	PA7	PA8	PA9
0	-0.277297	-0.659165	-0.294841	-0.943657	-0.400686	-0.038213	-0.069070	0.064157	-0.077742
1	-0.891712	-0.438681	-0.689291	-0.682828	-0.100864	0.067827	-0.054470	0.013317	-0.079874
2	0.774203	0.430676	0.990031	0.142288	0.723534	-0.399989	-0.278389	-0.645350	-0.124941
3	-0.915793	-0.567765	0.772699	-0.328325	-0.088724	-0.202961	-0.156347	0.056548	-0.068173
4	0.244956	-0.846576	0.040442	-1.165361	-0.655534	-0.128347	-0.081479	0.107372	-0.075929
...	...	...	...	...	...	...	...	...	...
119446	-1.521295	-0.293590	0.177416	0.696733	-0.330827	0.201770	-0.221000	0.010262	0.007094
119447	0.164493	0.714543	0.795930	0.580811	-0.480233	-0.304461	-0.382517	0.106589	-0.213047
119448	-1.459854	-0.315638	0.216861	0.670650	-0.360809	0.191166	-0.222460	0.015346	0.007307
119449	-0.634902	-0.389981	0.110752	0.185242	-0.914490	0.208668	0.988736	-0.049169	-0.143175
119450	-0.391670	0.904297	-0.703473	-0.032721	-0.673823	0.287984	-0.302760	0.063033	0.097185

119451 rows × 9 columns

In [188]:
```
A=np.nan_to_num(df_pca)
B=np.nan_to_num(train_y)
print(B)
B=B[:,np.newaxis]
print(B.shape)
D=np.hstack((A,B))
newdata=pd.DataFrame(D)
newdata.columns=['PA1','PA2','PA3','PA4','PA5','PA6','PA7','PA8','PA9','death']
newdata.isnull().sum()
newdata
```

```
[2 2 1 ... 2 2 2]
(119451, 1)
```

Out[188]:

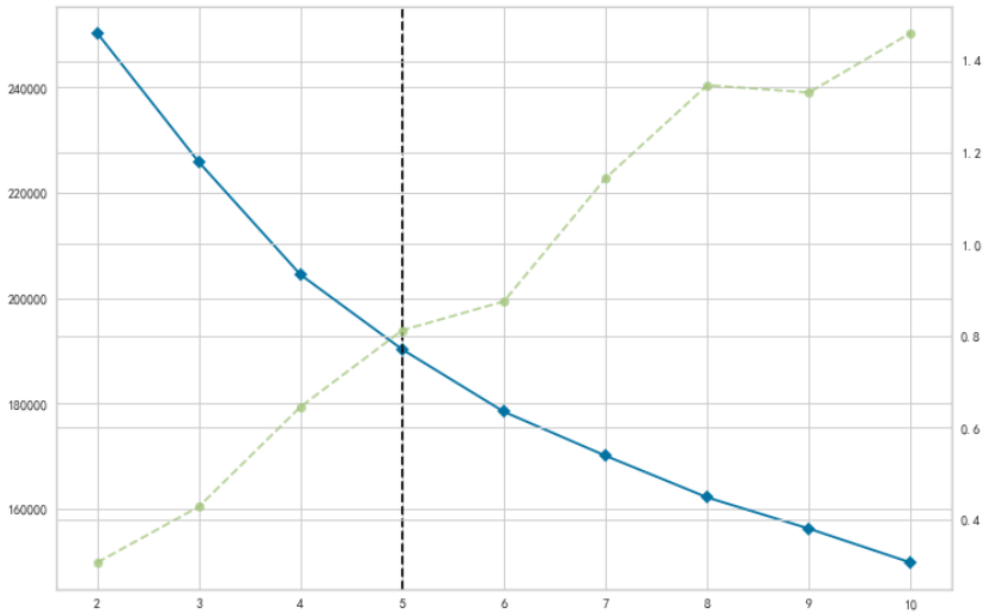
	PA1	PA2	PA3	PA4	PA5	PA6	PA7	PA8	PA9	death
0	-0.277297	-0.659165	-0.294841	-0.943657	-0.400686	-0.038213	-0.069070	0.064157	-0.077742	2.0
1	-0.891712	-0.438681	-0.689291	-0.682828	-0.100864	0.067827	-0.054470	0.013317	-0.079874	2.0
2	0.774203	0.430676	0.990031	0.142288	0.723534	-0.399989	-0.278389	-0.645350	-0.124941	1.0
3	-0.915793	-0.567765	0.772699	-0.328325	-0.088724	-0.202961	-0.156347	0.056548	-0.068173	2.0
4	0.244956	-0.846576	0.040442	-1.165361	-0.655534	-0.128347	-0.081479	0.107372	-0.075929	2.0
...	...	...	...	...	...	...	...	...	...	...
119446	-1.521295	-0.293590	0.177416	0.696733	-0.330827	0.201770	-0.221000	0.010262	0.007094	2.0
119447	0.164493	0.714543	0.795930	0.580811	-0.480233	-0.304461	-0.382517	0.106589	-0.213047	2.0
119448	-1.459854	-0.315638	0.216861	0.670650	-0.360809	0.191166	-0.222460	0.015346	0.007307	2.0
119449	-0.634902	-0.389981	0.110752	0.185242	-0.914490	0.208668	0.988736	-0.049169	-0.143175	2.0
119450	-0.391670	0.904297	-0.703473	-0.032721	-0.673823	0.287984	-0.302760	0.063033	0.097185	2.0

119451 rows × 10 columns

9.2 绘制肘图确定聚类数

In [178]:
```
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

plt.figure(figsize=(12,8))
elbow_graph = KElbowVisualizer(KMeans(random_state=123),k=10)
elbow_graph.fit(newdata)
elbow_graph.show
```

Out[178]:
```
<bound method Visualizer.show of KElbowVisualizer(ax=<AxesSubplot:>,
                      estimator=KMeans(n_clusters=10, random_state=123))>
```



由图可知，我们将聚为 5 类

9.3 建立 K-Means 模型

In [179]:
```
newdata1=newdata
kmeans = KMeans(n_clusters=5,random_state=1234).fit(newdata1)

newdata1['lable']=kmeans.labels_
newdata1
```

Out[179]:

	PA1	PA2	PA3	PA4	PA5	PA6	PA7	PA8	PA9	death	lable
0	-0.277297	-0.659165	-0.294841	-0.943657	-0.400686	-0.038213	-0.069070	0.064157	-0.077742	2.0	2
1	-0.891712	-0.438681	-0.689291	-0.682828	-0.100864	0.067827	-0.054470	0.013317	-0.079874	2.0	2
2	0.774203	0.430676	0.990031	0.142288	0.723534	-0.399989	-0.278389	-0.645350	-0.124941	1.0	3
3	-0.915793	-0.567765	0.772699	-0.328325	-0.088724	-0.202961	-0.156347	0.056548	-0.068173	2.0	4
4	0.244956	-0.846576	0.040442	-1.165361	-0.655534	-0.128347	-0.081479	0.107372	-0.075929	2.0	2
...	...	...	...	...	...	...	...	...	...	...	...
119446	-1.521295	-0.293590	0.177416	0.696733	-0.330827	0.201770	-0.221000	0.010262	0.007094	2.0	4
119447	0.164493	0.714543	0.795930	0.580811	-0.480233	-0.304461	-0.382517	0.106589	-0.213047	2.0	0
119448	-1.459854	-0.315638	0.216861	0.670650	-0.360809	0.191166	-0.222460	0.015346	0.007307	2.0	4
119449	-0.634902	-0.389981	0.110752	0.185242	-0.914490	0.208668	0.988736	-0.049169	-0.143175	2.0	2
119450	-0.391670	0.904297	-0.703473	-0.032721	-0.673823	0.287984	-0.302760	0.063033	0.097185	2.0	0

119451 rows × 11 columns

每个类别的个数：

```
In [180]:  1  newdata1.lable.value_counts()
```

```
Out[180]: 3    32468
          1    28008
          4    23147
          0    18306
          2    17522
          Name: lable, dtype: int64
```

聚类中心：

```
In [181]:  1  kmeans.cluster_centers_
```

```
Out[181]: array([[-0.62343452,  0.8951621 , -0.00403277, -0.05565545, -0.06945954,
                   0.02823474,  0.04223242, -0.0065898 ,  0.0158197 ,  1.86769769],
                 [ 0.71181435, -0.63652043, -0.17354644,  0.19658573,  0.12220537,
                  -0.10680041,  0.09161225, -0.0084925 ,  0.10672508,  1.08566633],
                 [-0.70778708, -0.36100184, -0.51805639, -0.35970069, -0.21760744,
                   0.18567414,  0.13407949, -0.01159299,  0.01375658,  1.95567346],
                 [ 0.96024444,  0.49215993,  0.08662406, -0.007424  ,  0.01759919,
                  -0.01229151, -0.06349369,  0.01224982, -0.13087091,  1.07141977],
                 [-1.17921976, -0.35456758,  0.48391366,  0.08896725,  0.04716386,
                  -0.01647611, -0.1566789 ,  0.00707903,  0.03153594,  1.96483649]])
```

9.4 对每个类别的解释

```
In [182]:  1  print(newdata1[newdata1['lable']==0].death.value_counts())
           2  print(newdata1[newdata1['lable']==1].death.value_counts())
           3  print(newdata1[newdata1['lable']==2].death.value_counts())
           4  print(newdata1[newdata1['lable']==3].death.value_counts())
           5  print(newdata1[newdata1['lable']==4].death.value_counts())
```

```
2.0    15883
1.0     2423
Name: death, dtype: int64
1.0    25607
2.0     2401
Name: death, dtype: int64
2.0    16746
1.0      776
Name: death, dtype: int64
1.0    30149
2.0     2319
Name: death, dtype: int64
2.0    22334
1.0      813
Name: death, dtype: int64
```

根据以上信息可知：

- 第一类人群的死亡率在 15%左右，为高危人群
- 第二类人群的死亡率在 9%左右，为次高危人群
- 第三类人群的死亡率在 4.6%左右，为较安全人群
- 第四类人群的死亡率在 7.6%左右，为较高危人群
- 第五类人群的死亡率在 3.6%左右，为安全人群

十. 总结

在通过同样的数据预处理方式，并通过欠采样方法解决了原始数据集不平衡的问题后，我们分别建立了三种机器学习模型。通过对比可知，其中逻辑回归模型的准确度最高，随机森林的准确度与逻辑回归模型相近，但是逻辑回归在 F1 检验的得分高于随机森林模型。而 K-means 聚类方法相较于前两种方法而言，优势不足。故逻辑回归模型是我们所建立的最优模型。