

# 一、摘要

## 1. 选题背景

生存分析是一种统计方法，用于研究事件发生时间与相关因素之间的关系。

在临床研究中，在临床实践中，研究患者的生存情况和预测其生存时间是重要的研究方向。生存分析通过分析患者的生存数据，例如生存时间和事件发生情况，来评估患者的生存状况。这种分析有助于了解治疗方法的效果、确定与疾病相关危险因素以及预测患者的生存时间，为医生和患者做出更明智的决策提供依据。

生存分析依赖于大量的生存数据，而这些数据通常存储在主流的临床研究数据库中。其中包括但不限于以下数据库：

- The Cancer Genome Atlas (TCGA)：包含多种癌症类型的临床信息和基因组数据的综合性癌症基因组学数据库。
- Surveillance, Epidemiology, and End Results (SEER)：美国国家级的长期癌症监测、流行病学和结果数据库。
- North Central Cancer Treatment Group (NCCTG)：包含了多个癌症类型的临床研究数据库。

生存分析的常用工具较为多样，较为成熟和主流的分析工具包括但不限于：

- lifelines：一个Python库，提供了多种生存分析方法的实现，包括Kaplan-Meier估计、Cox比例风险模型等。
- R语言生存分析包：如survival、survminer等，提供了丰富的生存分析功能和绘图工具。
- 在线网站：如KMplot、GEPIA、cBioPortal等，通常整合了数据检索相关功能，具有操作门槛低、可交互性强等优点，但研究的自由度相对受限。

## 2. 研究内容

本项目的研究内容主要分为：

- 生存分析的基本概念、研究流程及意义
- 结合研究流程，基于NCCTG数据库的测试数据，分析lifelines库如何在生存分析中实践与应用
- 基于TCGA数据库的生存分析研究
- 使用python语言设计一款轻量化、可交互、自动化的生存分析工具

# 二、生存分析的基本操作流程与概念分析

## 1. 数据的概览与初步处理

### 1.1 数据删失

根据设定的终点事件是否在研究过程中发生，我们可以把生存分析数据分为完全数据（Complete data）和删失数据（Censoring data）两类。完全数据意味着在研究过程中明确地观察到了终点事件的发生，且数据中记录到了研究对象的生存时间（或发生终点事件的时间）。

删失数据则与完全数据相反，意味着在研究过程中研究对象发生了研究之外的其他事件或其他结局，数据中没有记录到研究对象的生存时间（或发生终点事件的时间），其类型可分为以下三种：

- 右删失

右删失是生存分析中最常见的数据删失类型，意味着在研究过程中，由于研究对象中途退出研究、研究结束时终点事件仍未发生等因素，导致无法获取具体的生存时间。

- 左删失

左删失意味着在研究人员对研究对象开展研究之前，其感兴趣的某种事件就已经发生，但发生事件的具体时间无法确定。在以患者的死亡为终点事件的生存分析研究中，我们通常不考虑左删失，因为死亡事件不可能多次发生。

- 区间删失

区间删失意味着研究人员未能获取研究对象终点事件发生的准确时间，只知道终点事件是在两次随访之间的时间区间内发生，导致未能准确地记录研究对象生存时间。

在TCGA数据库的clinical数据中，考虑days\_to\_birth和days\_to\_death两列，一般存在有表格中所示三种数据类型。数据的缺失以 '-' 标记。

case_id	days_to_birth	days_to_death	删失类型
1	-25654	840	无删失
2	-18353	'--'	右删失
3	'--'	'--'	左/右删失

## 1.2 数据导入

我们采用常用测试数据集——肺癌数据库 NCCTG 的Lung Cancer Data 作为输入数据，该数据为csv格式。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from lifelines import KaplanMeierFitter
# 导入基本库
```

```
df = pd.read_csv('cancer.csv')
# pandas库内置csv文件读取函数

pd.set_option('display.max_columns', None)
print(df.head(5))
# 查看前5行数据的所有列
```

	Unnamed: 0	inst	time	status	age	sex	ph.ecog	ph.karno	pat.karno	meal.cal	wt.loss
0	1	3.0	306	2	74	1	1.0	90.0	100.0	1175.0	NaN
1	2	3.0	455	2	68	1	0.0	90.0	90.0	1225.0	15.0
2	3	3.0	1010	1	56	1	0.0	90.0	90.0	NaN	15.0
3	4	5.0	210	2	57	1	1.0	90.0	60.0	1150.0	11.0
4	5	1.0	883	2	60	1	0.0	100.0	90.0	NaN	0.0

根据参考文件，各列含义如下：

- inst: 研究机构代码
- time: 生存时间 (单位为天)
- status: 患者状态, 1=患者仍存活, 2=患者已死亡
- age、sex: 患者年龄、性别 (1=男性, 2=女性)
- ph.ecog、ph.karno、pat.karno: 对患者的三种评分
- meal.cal: 患者用餐时消耗的卡路里
- wt.loss: 患者最近六个月减轻的体重

该数据设定的终点事件是患者死亡, 终点事件是否发生通过status列判断, status=1代表患者仍然存活, 对应右删失; status=2代表患者已经死亡, 为完全数据。此外, 生存时间已经明确给出, 这也与TCGA中的clinical数据有所不同, 其计算公式为:

$$time = \begin{cases} \text{研究结束时间} - \text{研究起始时间} & status = 1 \\ \text{患者死亡时间} - \text{研究起始时间} & status = 2 \end{cases}$$

`df.info()`

# 查看数据的更多信息, 包括样本数量、列数、不同列的数据类型、内存情况等  
# Non-Null Count是对有效数据 (Non-NA) 的计数

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 228 entries, 0 to 227

Data columns (total 11 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Unnamed: 0   228 non-null    int64
1   inst         227 non-null    float64
2   time         228 non-null    int64
3   status       228 non-null    int64
4   age          228 non-null    int64
5   sex          228 non-null    int64
6   ph.ecog      227 non-null    float64
7   ph.karno     227 non-null    float64
8   pat.karno    225 non-null    float64
9   meal.cal     181 non-null    float64
10  wt.loss      214 non-null    float64
dtypes: float64(6), int64(5)
memory usage: 19.7 KB
```

## 1.3 缺失值处理

在生存分析中, 标记为NA (缺失值) 的数据可能会对结果产生影响, 甚至导致错误的结论, 对它们的检查和处理是非常重要的。

`df.isnull().sum()`

# 计算每列中标记为NA数据的个数

```
Unnamed: 0      0
inst            1
time            0
status          0
age             0
sex             0
ph.ecog         1
ph.karno        1
pat.karno       3
meal.cal        47
wt.loss         14
dtype: int64
```

处理缺失值的方法取决于数据的性质和产生缺失值的原因，一般有以下几种：

- 删除：适用于缺失值比例较小且分布随机。如果缺失值的比例较大或不随机分布时，剔除包含缺失值的样本可能会导致样本偏差。
- 插补：适用于缺失值的比例较小且与其他变量相关。常见的插补方法包括均值插补、回归插补和多重插补。
- 创建指示变量：创建额外的指示变量来指示观测值是否缺失，并在后续建模过程中加以考虑。
- 使用专门模型：如利用多重（multiple）模型或者完全条件分析方法处理缺失值。

在该数据中，属性变量（如性别、年龄）的缺失分布相对随机且在所有样本中占比较少，因此在分析特定属性对患者生存的影响时可以将该属性为缺失的样本剔除。

有些时候，我们并不想要研究某些属性对患者生存的影响，或者某些属性中包含了大量的缺失数据以至于无法研究（在TCGA数据库中较为突出），此时我们可以将这些属性对应的列删除，以便于观察和节省内存。

```
df.drop(['Unnamed: 0'], axis=1, inplace=True)
# 将Unnamed: 0一列删除（在原始对象上进行修改）
```

为分析方便起见，也可引入新的列，比如新建dead列，dead=1代表患者死亡，dead=2代表患者存活。

```
df['dead'] = (df['status'] == 2).astype(int)
df['dead'] = df['dead'].replace({False: 0, True: 1})
# 新建dead列，用0和1替换False和True
```

## 2. 生存分析的基本参数

生存概率（survival probability）是指某个时段开始时存活的个体，到该时段结束时仍存活的概率，其计算，用p表示。而死亡概率与生存概率概念上相对，用q表示， $p+q=1$ 。

生存函数（survival function）又称为累计生存率，简称生存率（survival rate），指观察对象的生存时间T大于某个时间的概率，是多个单位时间生存概率的累计结果，常用 $S(t)$ 表示：

$$S(t) = P(T > t)$$

生存函数是一个随时间下降的函数， $S(t=0) = 1$ ，代表研究开始时所有患者均存活； $S(t \rightarrow \infty) = 0$ ，代表经过足够长的时间，所有患者均会死亡。

当 $S(t)=0.5$ 时，若对应的时间为t，则说明有50%的患者可以存活到比t更长的时间，这个时间一般称为中位生存期或半数生存期：

$$T_{50} = t(S = 0.5)$$

HAZ (Hazard) , 指患者在时间 $t$ 时存活, 但在接下来的时间 $\delta$ 内死亡的概率:

$$HAZ = P(T < t + \delta | T > t)$$

风险率比 (Hazard Ratio, HR) 是两组生存数据风险率的比值, 是根据整个实验数据得到的结果, 与时间无关, 在各时间点保持恒定, 是衡量两组生存数据区别/两组治疗方案效果的重要指标。

$$HR = \frac{HAZ_1}{HAZ_0}$$

可以看出, HR是相对的, 在默认实验组风险率/对照组风险率的情况下, 当 $HR < 1$ 时, 说明试验组相比对照组风险更低, 如果两组差异仅体现在疗法上, 则说明试验组疗法更有效。

### 3. Kaplan-Meier 模型在lifelines中的实现与基本应用

Kaplan-Meier 模型是一种非参数统计模型, 用于根据生存数据估计生存函数。

$$S(t_i) = S(t_{i-1}) * (1 - \frac{d_i}{n_i})$$

其中,  $t_i$ 代表第 $i$ 个时间点,  $n_i$ 代表在 $t_i$ 之前的生存人数,  $d_i$ 代表在第 $i$ 个时间点死亡的人数。

```

durations = df['time']
event_observed = df['dead']

kmf = KaplanMeierFitter()
kmf.fit(durations, event_observed)
# 使用fit方法拟合KM生存曲线

```

#### 3.1 查看事件表: event\_table方法

在KM生存分析中, 事件表通常用于记录时间数据和事件发生的状态, 提供了事件发生情况的总览, 一般包括以下列:

- event\_at: 时间轴上表示事件发生或观察到事件发生的所有时间点, 升序排列。
- removed: 在每个时间点离开观察的个体数量, 可能是由于终点事件发生, 也可能是由于删失。

$$removed(t) = observed(t) + censored(t)$$

- observed: 在每个时间点观察到发生终点事件的数量。
- censored: 在每个时间点发生删失的个体数量, 可能由于观察期结束或失去对个体的随访。
- entrance: 在每个时间点新进入观察的个体数量。
- at\_risk: 在每个时间点仍处于观察 (仍存在风险) 的个体数量。

$$at\_risk(t) = at\_risk(t - 1) + entrance(t) - removed(t)$$

```

event_tab = kmf.event_table
print(event_tab.head(5))
# 查看通过kmf为该数据生成的event_tab的前5行

```

event_at	removed	observed	censored	entrance	at_risk
0.0	0	0	0	228	228 # 初始时刻所有患者均生存
5.0	1	1	0	0	228 # 5.0时刻, 首个事件发生
11.0	3	3	0	0	227
12.0	1	1	0	0	224
13.0	2	2	0	0	223

## 3.2 计算生存概率：predict方法

在 $t=n$ 时刻的生存概率 $S(t)$ 可通过以下公式计算：

$$S(t = n) = \frac{at\_risk(t = n) - observed(t = n)}{at\_risk(t = n)}$$

为便于理解，展示使用event\_tab计算 $t = 11$ 时刻生存概率的示例代码：

```
# 原理示意：
third_row = event_tab.iloc[2,:]
# 对第三行进行索引
survival_at_11 = (third_row.at_risk - third_row.observed) / third_row.at_risk
print("t = 11: S =", round(survival_at_11, 3))
```

```
t = 11: S = 0.987
```

估计给定时间点的生存概率的功能在kmf类中可以通过predict方法实现：

```
# 利用kmf.predict()方法简易实现该功能
print("t = 11: S =", kmf.predict(11))
```

```
t = 11: S = 0.9824561403508766
```

## 3.3 获取完整生存函数：survival\_function\_方法

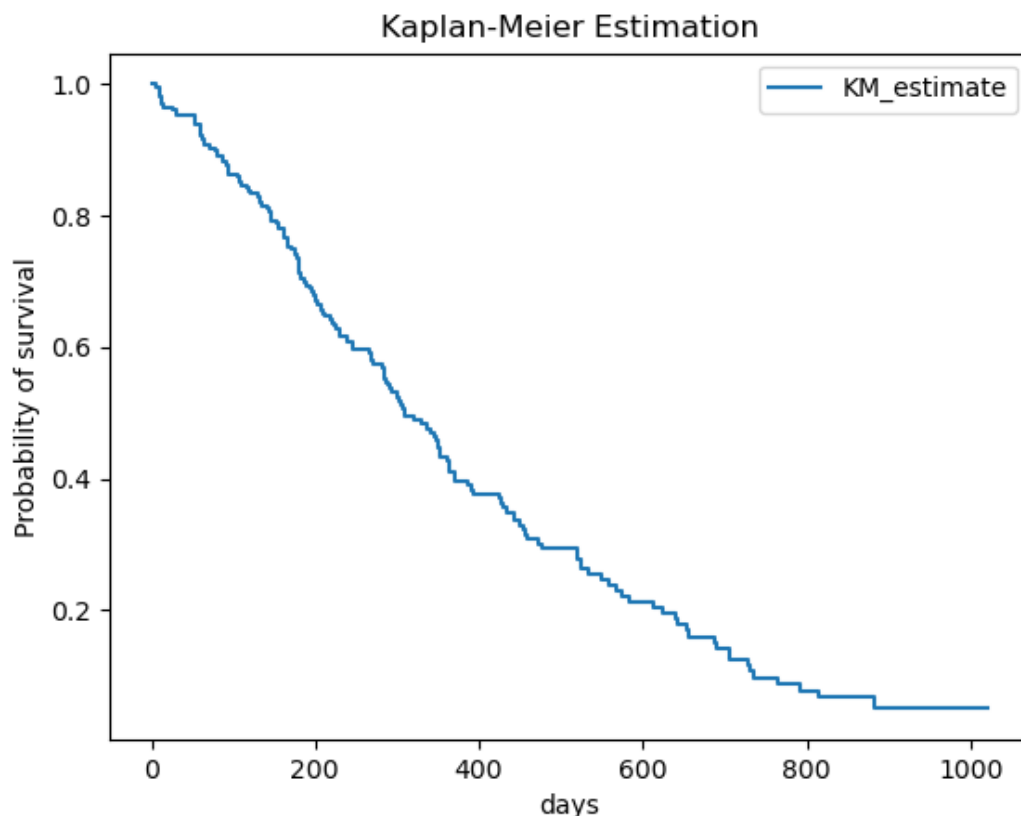
```
# 通过访问kmf类的survival_function_属性获取所有时间点的生存概率
# 即获取已拟合KM生存曲线的生存函数
ls = kmf.survival_function_
print(ls.head(5))
```

timeline	KM_estimate
0.0	1.000000
5.0	0.995614
11.0	0.982456
12.0	0.978070
13.0	0.969298

### 3.4 绘制KM生存曲线图：plot方法

使用 `kmf` 的 `plot` 方法绘制KM生存曲线图（单支）：

```
kmf.plot(ci_show=False)
# 禁用置信区间的显示
plt.title("Kaplan-Meier Estimation")
plt.xlabel("days")
plt.ylabel("Probability of survival")
plt.show()
```



### 3.5 计算中位生存时间：median\_survival\_time\_方法

```
print("Median survival time =", kmf.median_survival_time_, "days.")
# 利用kmf的median_survival_time_属性计算中位生存时间
```

```
Median survival time = 310.0 days.
# 患者平均存活310.0天
```

### 3.6 单独绘制置信区间曲线： confidence\_interval\_survival\_function\_方法

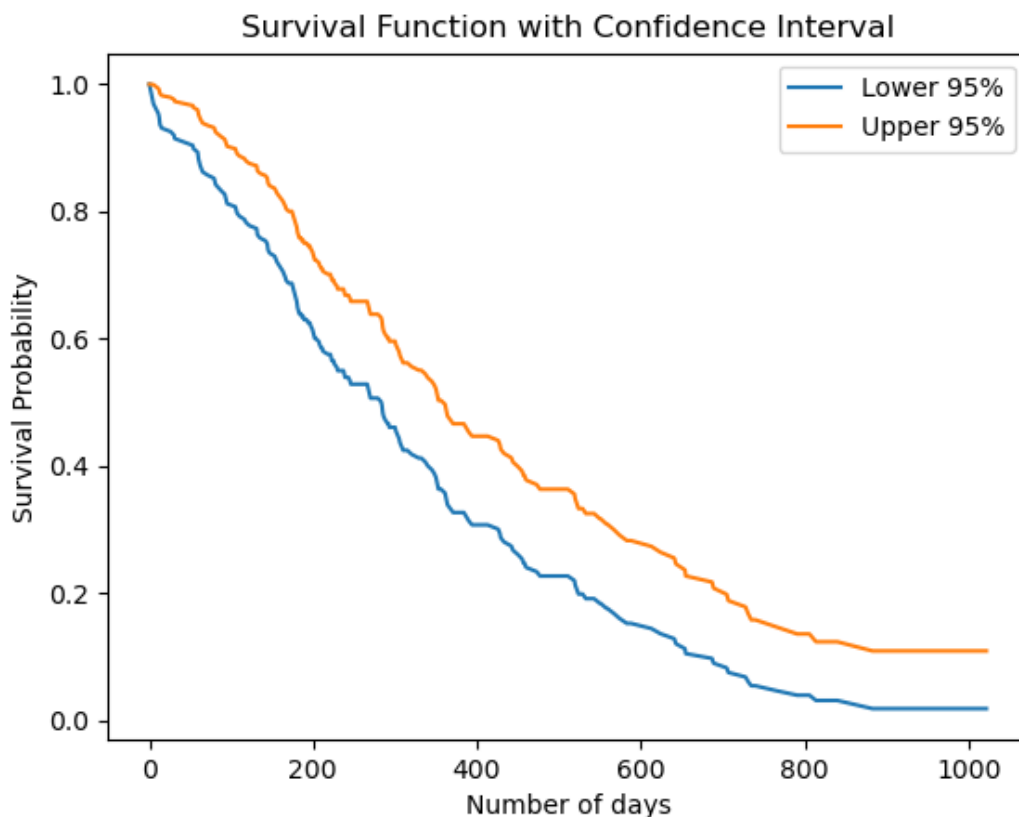
`kmf` 的 `confidence_interval_survival_function_` 属性提供了不同置信水平下的置信区间可供选择，如常用的95% 置信区间和90% 置信区间，获取的结果是一个 `DataFrame`，其中包含不同时间点的下置信区间和上置信区间，可以根据需要来提取。

```

confidence_data = kmf.confidence_interval_survival_function_

plt.plot(confidence_data["KM_estimate_lower_0.95"], label = "Lower 95%")
# 绘制下95%置信区间的生存曲线
plt.plot(confidence_data["KM_estimate_upper_0.95"], label = "Upper 95%")
# 绘制上95%置信区间的生存曲线
plt.title("Survival Function with Confidence Interval")
plt.xlabel("Number of days")
plt.ylabel("Survival Probability")
plt.legend()
# 创建图例
plt.show()

```



两条置信曲线中间的区域即为95%置信区间，表示在给定置信水平为 95% 的情况下，真实生存函数落在置信区间内的概率为 95%。

### 3.7 直接绘制置信区间范围

在通过 `ci_show=False` 开启置信区间显示功能时，`kmf.plot` 方法默认绘制95%置信区间范围。

```

kmf.plot(ci_show=False, ci_alpha=0.3)
# 开启置信区间的显示
# 置信区间透明度设置为0.3

```

可以为 `confidence_intervals` 设定参数，来指定所需的置信区间。

```

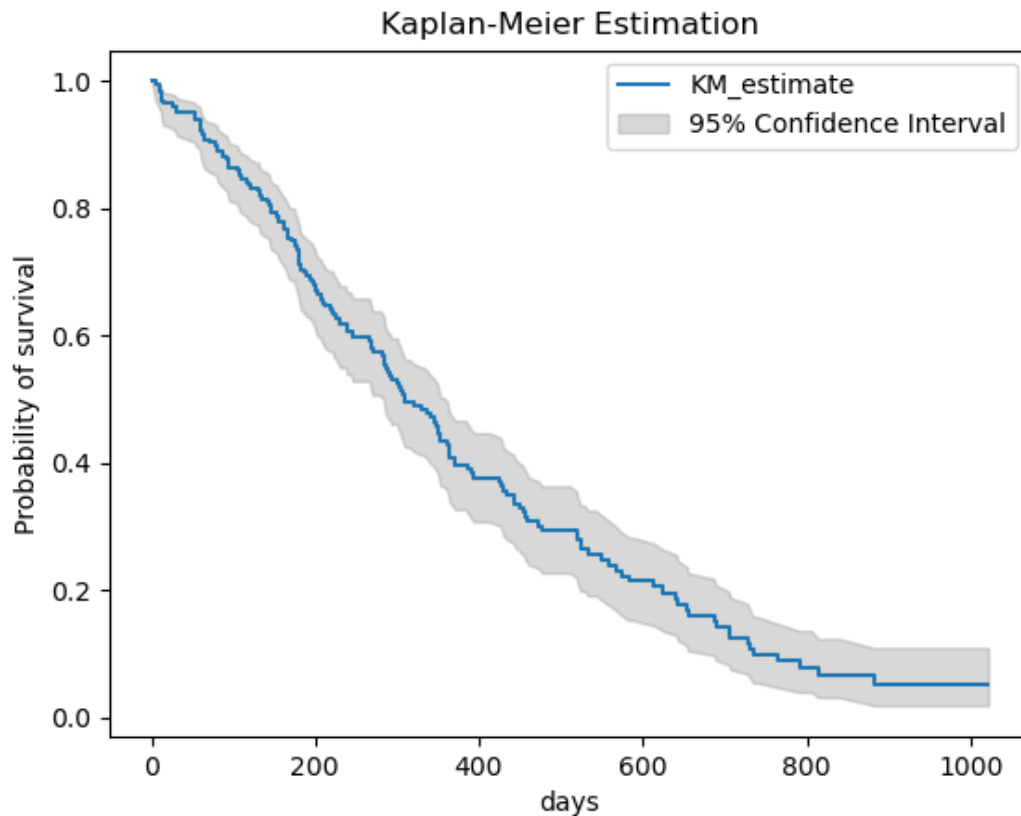
kmf.plot(ci_show=False, ci_alpha=0.3, ci_legend=True, confidence_intervals=
[(0.05, 0.95)])
plt.legend()

```



可以使用 `plt.fill_between` 自定义绘制方式：

```
plt.fill_between(kmf.survival_function_.index,  
kmf.confidence_interval_['KM_estimate_lower_0.95'],  
kmf.confidence_interval_['KM_estimate_upper_0.95'],  
color='gray', # 灰色  
alpha=0.3, # 透明度设置为0.3  
label='95% Confidence Interval') # 图例  
  
plt.legend()
```



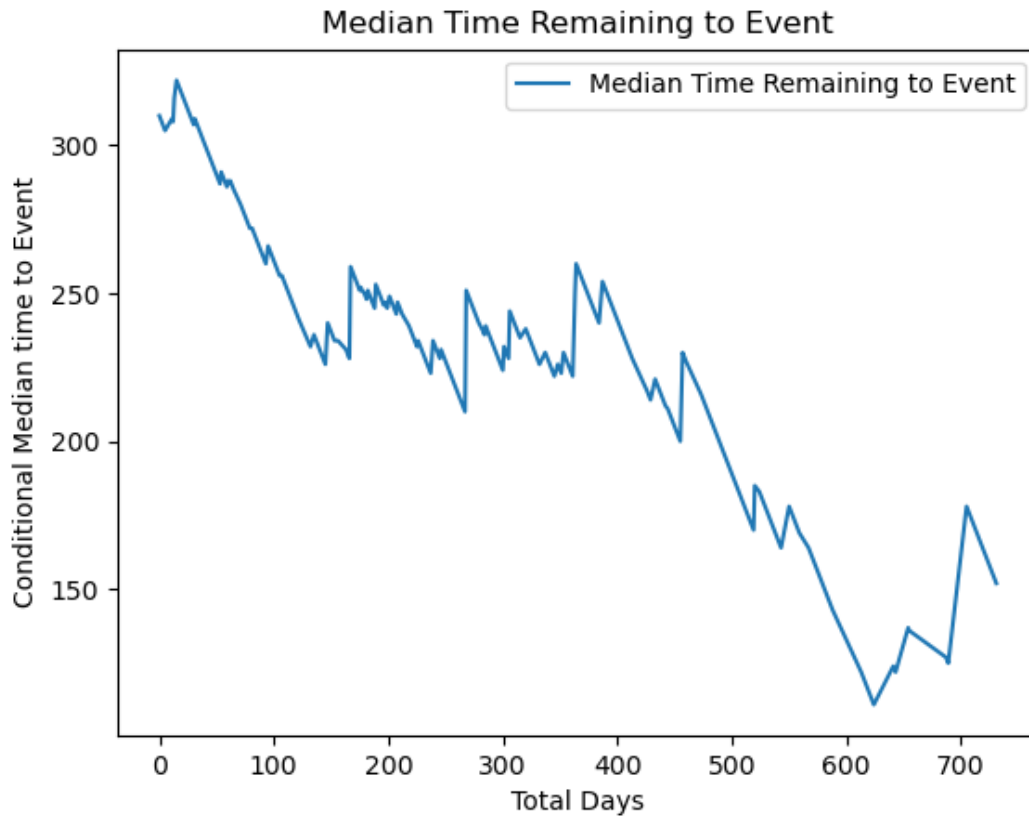
### 3.8 估计终点事件发生剩余时间：conditional\_time\_to\_event\_方法

`kmf.conditional_time_to_event_` 方法是 `Kaplan-Meier` 模型中的一个扩展，它允许根据指定的条件来估计事件发生剩余的时间。

```
median_time_to_event = kmf.conditional_time_to_event_  
print(median_time_to_event.head(5))
```

```
      KM_estimate - Conditional median duration remaining to event  
timeline  
0.0      310.0  
5.0      305.0  
11.0     309.0  
12.0     308.0  
13.0     316.0
```

```
plt.plot(kmf.conditional_time_to_event_, label = "Median Time Remaining to Event")
plt.title("Median Time Remaining to Event")
plt.xlabel("Total Days")
plt.ylabel("Conditional Median time to Event")
plt.legend()
plt.show()
```



## 4. 使用Nelson-Aalen方法估计风险函数

风险函数 $h(t)$ 表示某时刻 $t$ 有事件发生（病人死亡）的概率密度。

定义累计风险函数 $H(t)$ 为：

$$H(t) = \sum_{t_i \leq t} \frac{d_i}{n_i}$$

其中 $d_i$ 表示 $t_i$ 时刻发生事件的数量， $n_i$ 表示 $t_i$ 时刻还存活的个体数

```
from lifelines import NelsonAalenFitter

NAFitter=NelsonAalenFitter()
```

```
NAFitter.fit(df['time'],event_observed=df['dead']) #拟合数据
```

```
<lifelines.NelsonAalenFitter:"NA_estimate", fitted with 228 total observations,  
63 right-censored observations>
```

```
NAFitter.cumulative_hazard_ #累计风险函数的计算
```

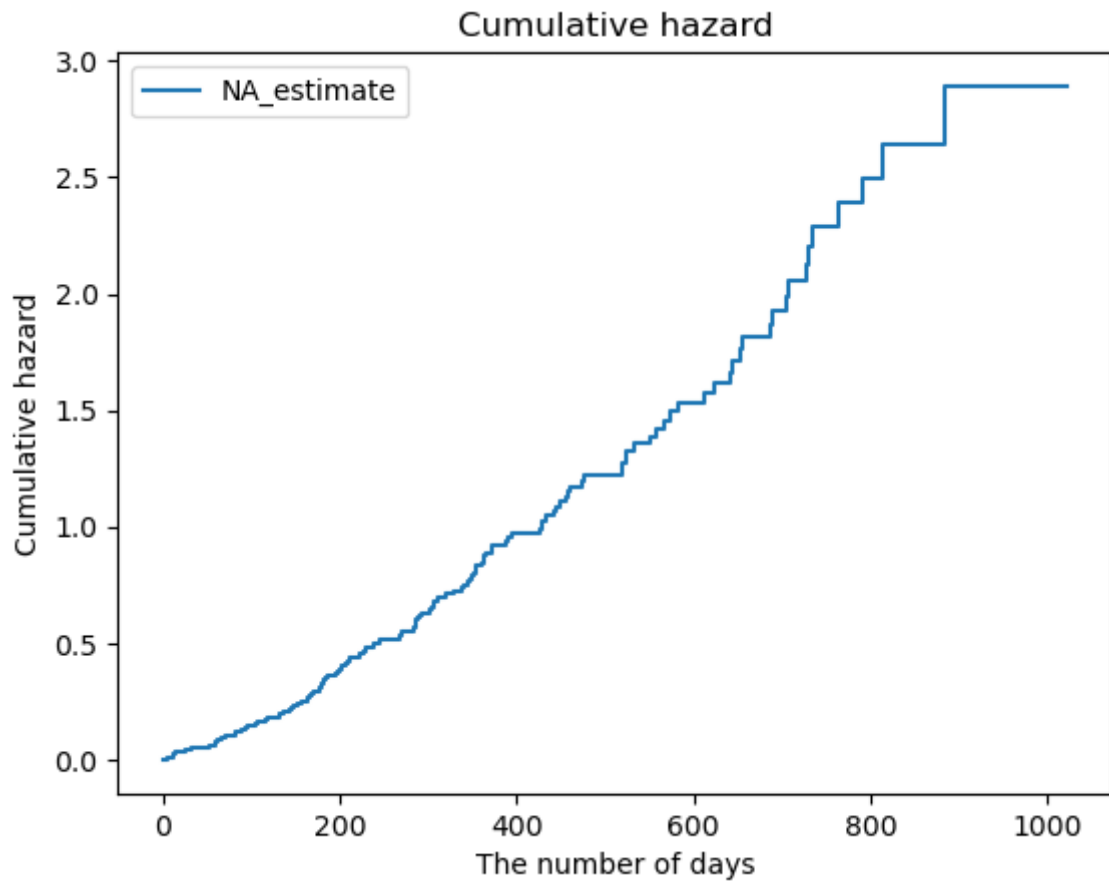
timeline	NA_estimate
0.0	0.000000
5.0	0.004386
11.0	0.017660
12.0	0.022125
13.0	0.031114
...	...
840.0	2.641565
883.0	2.891565
965.0	2.891565
1010.0	2.891565
1022.0	2.891565

187 rows × 1 columns

```
# 画出累计风险函数图
```

```
NAFitter.plot_cumulative_hazard(ci_show=False)  
plt.title("Cumulative hazard")  
plt.xlabel("The number of days")  
plt.ylabel("Cumulative hazard")
```

```
Text(0, 0.5, 'Cumulative hazard')
```



## 5. 使用Kaplan Meier曲线对不同因素造成的影响进行比较

### 5.1 按照年龄将患者分成两组，检查年龄对生存是否有影响

#按照患者年龄大小打上标记：1表示年龄大于等于65岁，0表示年龄小于65岁

```
df_age=df.copy()
df_age.loc[df_age['age']>=65,'age_tag']=1
df_age.loc[df_age['age']<65,'age_tag']=0
```

```
df_age.head(6)
```

	time	status	age	sex	ph.ecog	ph.karno	pat.karno	meal.cal	wt.loss	dead	age_tag
0	306	2	74	1	1.0	90.0	100.0	1175.0	NaN	1.0	1.0
1	455	2	68	1	0.0	90.0	90.0	1225.0	15.0	1.0	1.0
2	1010	1	56	1	0.0	90.0	90.0	NaN	15.0	0.0	0.0
3	210	2	57	1	1.0	90.0	60.0	1150.0	11.0	1.0	0.0
4	883	2	60	1	0.0	100.0	90.0	NaN	0.0	1.0	0.0
5	1022	1	74	1	1.0	50.0	80.0	513.0	0.0	0.0	1.0

#将所有患者分成2组

```
age_old=df_age.query("age_tag==1")
age_young=df_age.query("age_tag==0")
```

```
#对2组数据分别使用Kaplan Meier曲线进行拟合
```

```
KMFitter_old=KaplanMeierFitter()  
KMFitter_young=KaplanMeierFitter()
```

```
KMFitter_old.fit(durations=age_old['time'],event_observed=age_old['dead'],label=  
"More than 65 years old")  
KMFitter_young.fit(durations=age_young['time'],event_observed=age_young['dead'],  
label= "Less than 65 years old")
```

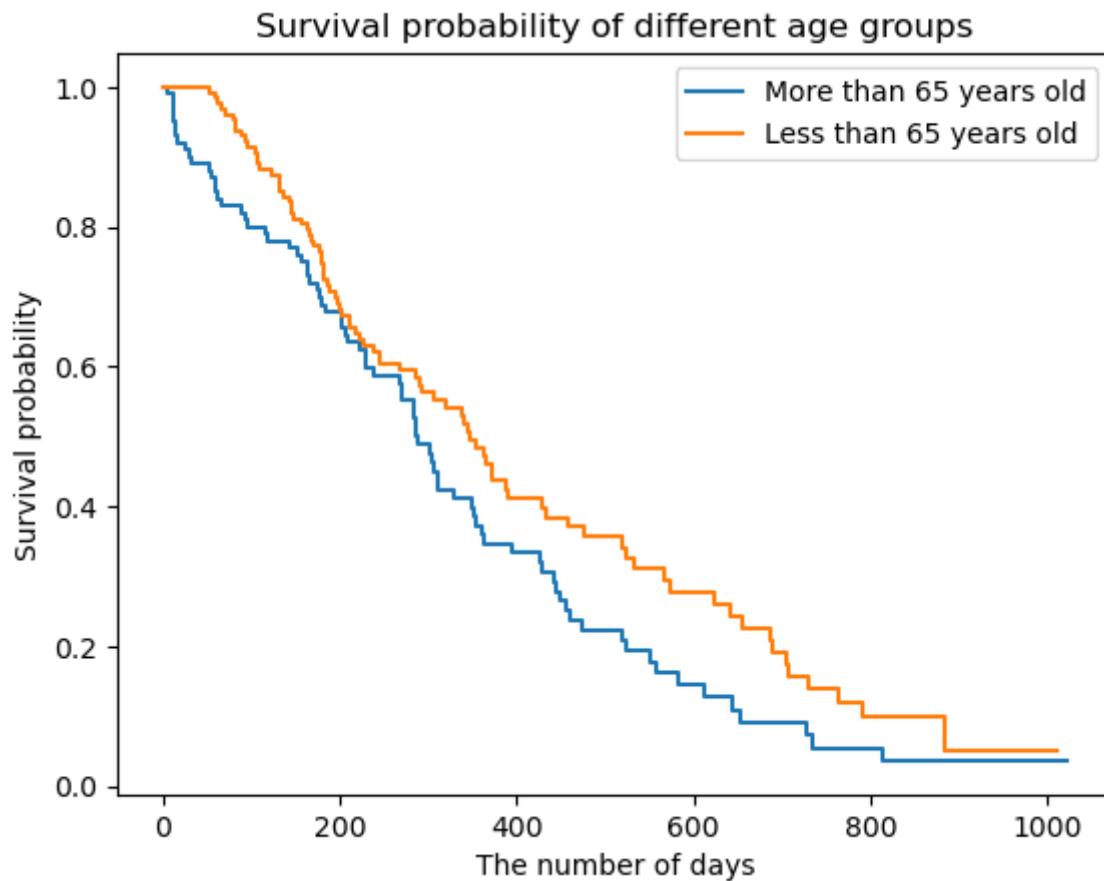
```
<lifelines.kaplanmeierfitter:"Less than 65 years old", fitted with 128 total  
observations, 42 right-censored observations>
```

```
#作出2组患者的生存率曲线
```

```
KMFitter_old.plot(ci_show=False)  
KMFitter_young.plot(ci_show=False)
```

```
plt.xlabel("The number of days")  
plt.ylabel("Survival probability")  
plt.title("Survival probability of different age groups")
```

```
Text(0.5, 1.0, 'Survival probability of different age groups')
```



结论：年龄高于65岁的患者生存率比年龄低于65岁的患者低

## 5.2 按照饮食摄入的热量(meal.cal)将患者分成两组，检查meal.cal对生存是否有影响

```
#按照患者摄入饮食的热量打上标记：1表示热量大于等于1000 cal，0表示热量小于1000 cal
df_cal=df.copy()
df_cal.loc[df_cal['meal.cal']>=1000,'cal_tag']=1
df_cal.loc[df_cal['meal.cal']<1000,'cal_tag']=0
```

```
df_cal.head(6)
```

	time	status	age	sex	ph.ecog	ph.karno	pat.karno	meal.cal	wt.loss	dead	cal_tag
0	306	2	74	1	1.0	90.0	100.0	1175.0	NaN	1.0	1.0
1	455	2	68	1	0.0	90.0	90.0	1225.0	15.0	1.0	1.0
2	1010	1	56	1	0.0	90.0	90.0	NaN	15.0	0.0	NaN
3	210	2	57	1	1.0	90.0	60.0	1150.0	11.0	1.0	1.0
4	883	2	60	1	0.0	100.0	90.0	NaN	0.0	1.0	NaN
5	1022	1	74	1	1.0	50.0	80.0	513.0	0.0	0.0	0.0

```
#将所有患者分成2组
cal_more=df_cal.query("cal_tag==1")
cal_less=df_cal.query("cal_tag==0")
```

```
#对2组数据分别使用Kaplan Meier曲线进行拟合
```

```
KMFitter_more=KaplanMeierFitter()
KMFitter_less=KaplanMeierFitter()

KMFitter_more.fit(durations=cal_more['time'],event_observed=cal_more['dead'],label="More than 1000 calories")
KMFitter_less.fit(durations=cal_less['time'],event_observed=cal_less['dead'],label="Less than 1000 calories")
```

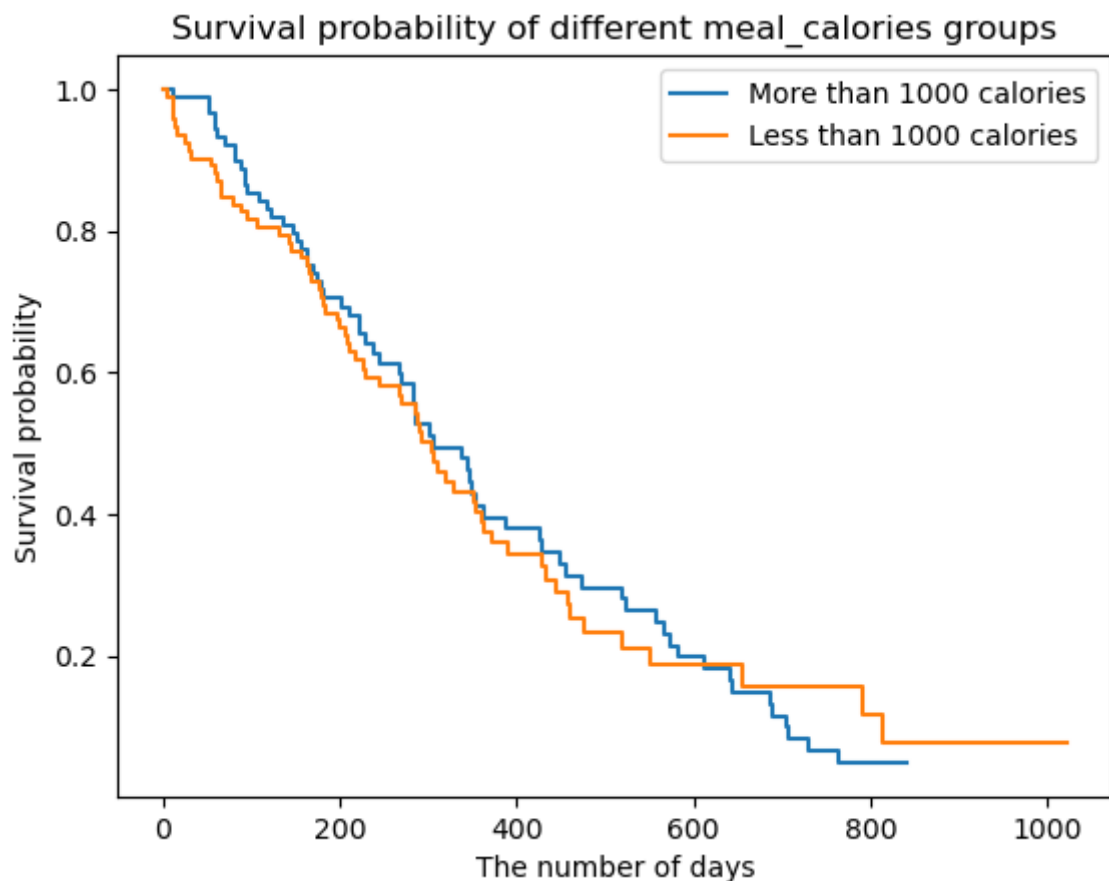
```
<lifelines.KaplanMeierFitter:"Less than 1000 calories", fitted with 92 total
observations, 26 right-censored observations>
```

```
#作出2组患者的生存率曲线
```

```
KMFitter_more.plot(ci_show=False)
KMFitter_less.plot(ci_show=False)

plt.xlabel("The number of days")
plt.ylabel("Survival probability")
plt.title("Survival probability of different meal_calories groups")
```

```
Text(0.5, 1.0, 'Survival probability of different meal_calories groups')
```



结论：饮食摄入能量的高低 ( $\geq 1000\text{cal}$  或  $< 1000\text{cal}$ ) 对生存率没有显著影响。

对于其他风险因素（如体重变化量、性别等），都可以采取类似的方法进行分析。

### 三、基于TCGA数据库的生存分析研究

我们可以通过<https://portal.gdc.cancer.gov/>访问TCGA数据库，在其中检索我们感兴趣的癌症数据。其中对于生存分析最为关键的数据是临床数据（Clinical），其中记录了患者的生存信息和一些基本信息。除此之外，TCGA还包括DNA测序数据（DNA Sequencing Data）、RNA表达数据（RNA Expression Data）、DNA甲基化数据（DNA Methylation Data）等，可以用于分析不同因素对患者生存的影响。

为了利用先前的研究基础，我们采取将TCGA数据转化为前篇使用的NCCTG-Lung Cancer Data测试数据的格式，它们的关键区别为：

- 文件格式不同，前者为tsv格式，后者为csv格式。
- 对患者生存信息的记录方式不同，前者在days\_to\_death中记录，后者在time和status中共同记录。
- 每个样本有两条记录，对应不同的治疗方式（本次不加以研究）。

```
import pandas as pd
import os

# 输入文件路径
file_path = input("请输入文件路径：")

# 获取文件扩展名
file_extension = os.path.splitext(file_path)[1]

# 判断文件格式
```

```

if file_extension.lower() == '.csv':
    # 对于CSV格式，直接读入数据
    df = pd.read_csv(file_path)
    print("成功读入CSV格式数据！")
elif file_extension.lower() == '.tsv':
    # 对于TSV格式，先转换为CSV格式再读入数据
    csv_path = os.path.splitext(file_path)[0] + '.csv'
    df = pd.read_csv(file_path, sep='\t')
    df.to_csv(csv_path, index=False)
    print("已将TSV格式转换为CSV格式，并成功读入数据！")
else:
    print("不支持的文件格式！")

```

```

# 判断是否经过TSV转换处理
if file_extension.lower() == '.tsv':
    # 删除第3、5、7、9...行，并同时移位其他行
    df = df.iloc[:,2].reset_index(drop=True)

    # 新建status列，表示患者状态
    df['status'] = df['days_to_death'].apply(lambda x: 2 if x == '--' else 1)

    # 新建dead列，表示患者是否死亡
    df['dead'] = df['status'].apply(lambda x: 1 if x == 2 else 0)

    # 新建time列，表示时间
    df['time'] = df['days_to_death'].apply(lambda x: -1 if x == '--' else
int(x))

    df = df[df['time'] != -1]

```

days\_to\_death为--代表右删失数据，将其time设置为了正常情况下不会出现的-1，可以将这些行直接删除：

```

df = df[df['time'] != -1]
df = df.reset_index(drop=True)
# 重置索引

```

也可以令time=days\_to\_last\_follow\_up，后续通过KM模型以删失数据类型加以处理：

```

df.loc[df['time'] == -1, 'time'] = df.loc[df['time'] == -1,
'days_to_last_follow_up']

```

## 四、具有简单GUI的生存分析工具的开发

### 1. 整体思路

为了方便生存分析工具的使用，我们利用Tkinter库开发了一套可以通过简单操作处理csv文件数据以绘制生存分析函数图像的小程序。Tkinter提供了一组用于创建窗口、标签、按钮、文本框等GUI组件的工具和方法，以及处理用户交互的事件处理机制。我们利用了下拉菜单，手动输入等方式完成了生存分析工具的简单化、集成化。



## 2. 基本思路

### 1. 手动输入文件名，各行表头名

```
def data_import(event):  
    # 获取文件路径  
    file_path = file_entry.get()  
    data = pd.read_csv(file_path)  
    data = data.dropna()  
  
    # 获取持续时间和事件列、性别列、年龄列和分组方式  
    duration_column = duration_entry.get()  
    event_column = event_entry.get()  
    gender_colomn = gender_entry.get()  
    age_colomn = age_entry.get()  
    group_by = group_by_var.get()
```

与此对应的文本框：

```
# 创建持续时间和事件列输入框  
duration_label = tk.Label(root, text='Duration Column:')  
duration_label.pack()  
duration_entry = tk.Entry(root)  
duration_entry.pack()  
  
event_label = tk.Label(root, text='Event Column:')  
event_label.pack()  
event_entry = tk.Entry(root)  
event_entry.pack()  
  
#创建性别和年龄列输入框  
gender_label = tk.Label(root, text='Gender Column:')  
gender_label.pack()  
gender_entry = tk.Entry(root)  
gender_entry.pack()  
  
age_label = tk.Label(root, text='Age Column:')  
age_label.pack()  
age_entry = tk.Entry(root)  
age_entry.pack()
```

### 2. 选择使用的方法

### 3. 分析以得到表格或数据

## 3. Kapler-Meier方法和 Nelson-Aaren 方法的功能实现

这个GUI提供了Kapler-Meier方法和 Nelson-Aaren 方法的绘图功能。通过输入表格信息和选择分组方式（以性别或年龄为例），该程序可以按照选择的分组方式绘制对照曲线。对于年龄，我们可以通过手动设置分类截断值将其分为两类或更多类，通过绘制不同年龄类别的生存曲线并加以对比，我们不难看出年龄因素对病人存活率造成的影响。

```

if group_by == 'Gender':
    data1 = data[data[gender_column] == 1]
    data2 = data[data[gender_column] == 2]
    label1 = 'MALE'
    label2 = 'FEMALE'
# 按性别分组，仅存在男女两组
elif group_by == 'Age':
    data1 = data[data[age_column] > 65]
    data2 = data[data[age_column] <= 65]
    label1 = 'ELDER'
    label2 = 'YOUNG'
# 按年龄分组，将截断值设置为65，小于等于65为YOUNG组，大于65为ELDER组

```

除了曲线绘制，该程序还通过Cox比例风险模型计算了数据的p值：

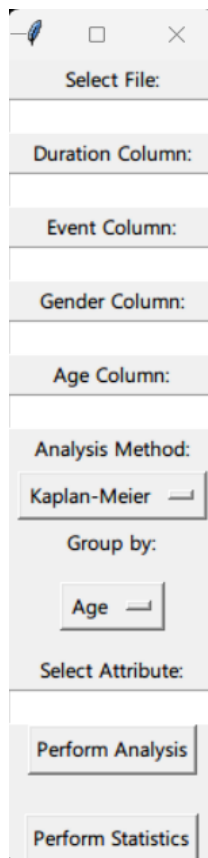
```

# Perform log-rank test to calculate p-value
results = logrank_test(data1[duration_column], data2[duration_column],
                        data1[event_column], data2[event_column])
p_value = results.p_value

plt.figtext(0.2, 0.02, f"p-value: {p_value:.3f}", ha='left')
plt.legend()
plt.show()

```

由此，通过简单的选择和输入即可获得生存分析曲线和p值：



Select File:

Duration Column:

Event Column:

Gender Column:

Age Column:

Analysis Method:

Kaplan-Meier

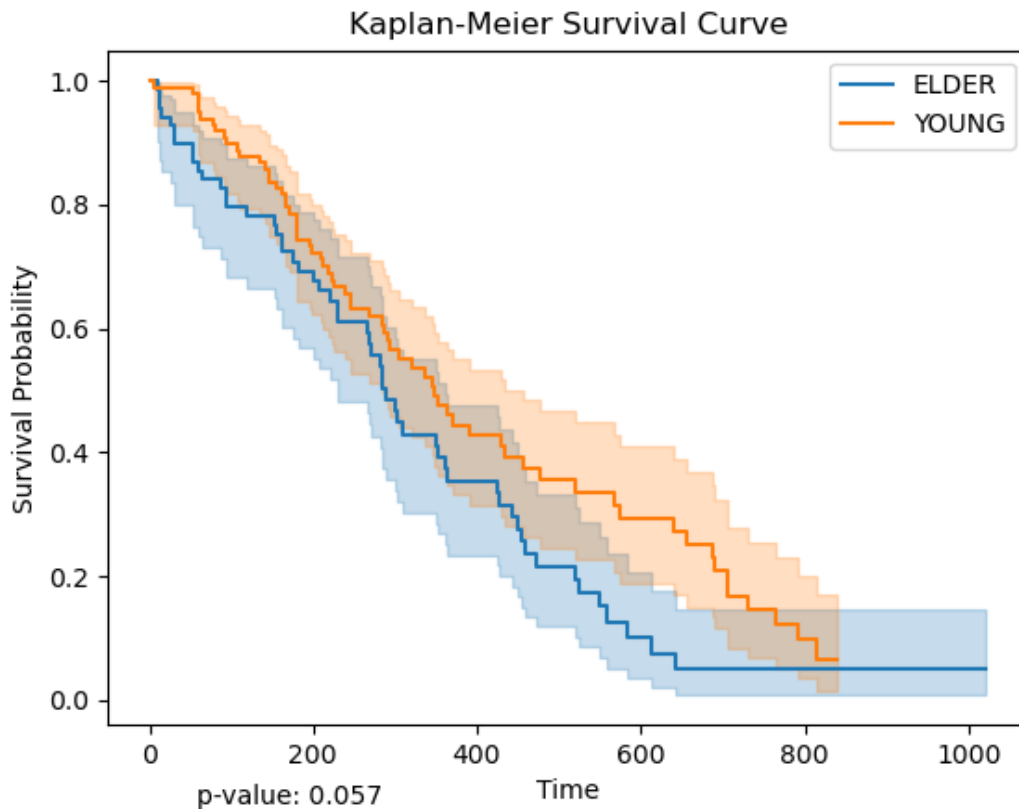
Group by:

Age

Select Attribute:

Perform Analysis

Perform Statistics



此外，这个GUI还包括了对数据某一列的统计，包括三分位、四分位、平均值、中位数等，通过一个对话框予以展示，这些统计数据可以在设置分类的截断值（cutoff）时供我们参考。

```
def select_attribute(df):
    attribute = attribute_entry.get()
    selected_column = None
    for column in df.columns:
        if attribute.lower() in column.lower():
            selected_column = column
            break
    return selected_column
# 根据用户输入寻找属性（寻找列）

def filter_na_values(df, selected_column):
    new_df = df.dropna(subset=[selected_column]).copy()
    return new_df
# 过滤掉在指定列中具有缺失值（NaN）的行

def calculate_statistics(new_df, selected_column):
    column_data = new_df[selected_column]
    mean = column_data.mean()
    median = column_data.median()
    upper_quartile = column_data.quantile(0.75)
    lower_quartile = column_data.quantile(0.25)
    upper_third = column_data.quantile(0.67)
    lower_third = column_data.quantile(0.33)

    return mean, median, upper_quartile, lower_quartile, upper_third, lower_third
# 计算并返回指定列的平均数、中位数、上四分位数、下四分位数、上三分位数、下三分位数
```

```
def perform_statistics():
    file_path = file_entry.get()
    data = pd.read_csv(file_path, error_bad_lines=False)
    data = data.dropna()
    df = data
    if df is not None:
        selected_column = select_attribute(df)
        if selected_column is not None:
            new_df = filter_na_values(df, selected_column)
            mean, median, upper_quartile, lower_quartile, upper_third,
lower_third = calculate_statistics(new_df, selected_column)
            messagebox.showinfo("Statistics", f"Mean: {mean}\nMedian:
{median}\nUpper Quartile: {upper_quartile}\nLower Quartile:
{lower_quartile}\nUpper Third: {upper_third}\nLower Third: {lower_third}")
# 展示平均数、中位数、上四分位数、下四分位数、上三分位数、下三分位数，便于我们设置截断值时参考
```

