Case Study : Survival Analysis on Lung Cancer

Raghotham R C - 2348141

Introduction :

Survival analysis is a branch of statistics for analysing the expected duration of time until one or more events occur. The method is also known as duration analysis or duration modelling, time-to-event analysis, reliability analysis and event history analysis.

The survival analysis is used to analyse following questions:

1. A proportion of population surviving up to a given time
2. Rate at which they are dying
3. Understanding the impact of covariates on survival

Cancer remains one of the leading causes of morbidity and mortality worldwide. The ability to predict patient survival, understand the impact of various treatments, and identify key prognostic factors is crucial in the management of cancer patients. Survival analysis is a statistical method specifically designed to handle time-to-event data, making it highly suitable for studying patient survival times. This study focuses on the survival analysis of cancer patients using a dataset that includes variables such as age, sex, performance status, and nutritional intake.

The objective of this study is to estimate survival probabilities, identify significant prognostic factors, and assess the impact of various treatments on patient survival. The analysis will provide insights that could inform clinical decisions and potentially improve patient outcomes.

Objectives :

- The aim is to understand the survival of lung cancer patents using time to event analysis.
- To estimate the survival probabilities of cancer patients over time.
- To identify significant prognostic factors affecting survival.
- To assess the impact of different clinical and nutritional factors on patient survival.
- To compare survival probabilities across different patient subgroups.

Data Description :

The data consist of 228 observations and 10 variables/columns. The variable description is presented as the following:

- inst: Institution code
- time (d1): Survival time in days
- status (d2): censoring status 1 = censored, 2 = dead
- age (i1): Age in years
- sex (i2): Male = 1 Female = 2
- ph.ecog (i3): ECOG performance score as rated by the physician. 0 = asymptomatic, 1 = symptomatic but completely ambulatory, 2 = in bed <50% of the day, 3 = in bed > 50% of the day but not bed bound, 4 = bed bound
- ph.karno (i4): Karnofsky performance score (bad = 0; good = 100) rated by physician
- pat.karno (i4): Karnofsky performance score as rated by patient
- meal.cal (i5): Calories consumed at meals
- wt.loss (i6): Weight loss in last six months

Where: 'i' indicates the independent variables (covariates), and 'd' indicates dependent variables. Actually, the dependent variable or response is the time until the occurrence of an event (i.e., the lung cancer patient dies).

To see the sourse of the lung cancer data that we are going to use for the analysis. The data is originally available in R programming language (in survival library) [1]. So, I have downloaded it from R to use it for learning purpose. The data source is mentioned below.

Data Source: Loprinzi CL. Laurie JA. Wieand HS. Krook JE. Novotny PJ. Kugler JW. Bartel J. Law M. Bateman M. Klatt NE. et al. Prospective evaluation of prognostic variables from patient-completed questionnaires. North Central Cancer Treatment Group. Journal of Clinical Oncology. 12(3):601–7, 1994.

Analysis :

In [ ]:
```
import numpy as np
import pandas as pd
from lifelines import KaplanMeierFitter
import matplotlib.pyplot as plt
```

In [ ]:
```
data = pd.read_csv("/Users/ragu/Documents/Codes/Survival/lung.csv", index_co
data.head()
```

Out[ ]:

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

In [ ]:
```
data.shape
```

Out[ ]:
```
(228, 10)
```

Base label fixing

The status and sex are categorical data. The status consists of 1: censored and 2: dead; while, sex consist of 1: Male and 2 :Female. We can keep it like that, but I usually prefer the base label for categorical variable as 0 (zero). Thus, to make the base label 0 we need to deduct 1 from the variables.

In [ ]:
```
data = data[['time', 'status', 'age', 'sex', 'ph.ecog', 'ph.karno','pat.karn
data["status"] = data["status"] - 1
data["sex"] = data["sex"] - 1
data.head()
```

Out[ ]:

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

In [ ]:
```
data.dtypes
```

Out[ ]:
```
time           int64
status         int64
age            int64
sex            int64
ph.ecog      float64
ph.karno     float64
pat.karno    float64
meal.cal     float64
wt.loss      float64
dtype: object
```

In [ ]:
```
data.isnull().sum()
```

Out[ ]:
```
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
```

Checking and imputing missing values

The next important step is to check whether the variables contain missing values. We can check that using .isnull( ) method and by applying the .sum( ) over it. The resulting table illustrates the following:

- ph.ecog: 1 missing value
- ph.karno: 1 missing value
- pat karno: 3 missing values
- meal.cal: 47 missing values
- wt.loss: 14 missing values

In [ ]:
```
data["ph.karno"].fillna(data["ph.karno"].mean(), inplace = True)
data["pat.karno"].fillna(data["pat.karno"].mean(), inplace = True)
data["meal.cal"].fillna(data["meal.cal"].mean(), inplace = True)
data["wt.loss"].fillna(data["wt.loss"].mean(), inplace = True)
data.dropna(inplace=True)
data["ph.ecog"] = data["ph.ecog"].astype("int64")
```

```
/var/folders/hs/zwl45n4s2l9c7m87lqrhshh40000gn/T/ipykernel_23456/409845501
8.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame
or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behav
es as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'd
f.method({col: value}, inplace=True)' or df[col] = df[col].method(value) in
stead, to perform the operation inplace on the original object.


  data["ph.karno"].fillna(data["ph.karno"].mean(), inplace = True)
/var/folders/hs/zwl45n4s2l9c7m87lqrhshh40000gn/T/ipykernel_23456/409845501
8.py:2: FutureWarning: A value is trying to be set on a copy of a DataFrame
or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behav
es as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'd
f.method({col: value}, inplace=True)' or df[col] = df[col].method(value) in
stead, to perform the operation inplace on the original object.


  data["pat.karno"].fillna(data["pat.karno"].mean(), inplace = True)
/var/folders/hs/zwl45n4s2l9c7m87lqrhshh40000gn/T/ipykernel_23456/409845501
8.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame
or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behav
es as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'd
f.method({col: value}, inplace=True)' or df[col] = df[col].method(value) in
stead, to perform the operation inplace on the original object.


  data["meal.cal"].fillna(data["meal.cal"].mean(), inplace = True)
/var/folders/hs/zwl45n4s2l9c7m87lqrhshh40000gn/T/ipykernel_23456/409845501
8.py:4: FutureWarning: A value is trying to be set on a copy of a DataFrame
or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behav
es as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'd
f.method({col: value}, inplace=True)' or df[col] = df[col].method(value) in
stead, to perform the operation inplace on the original object.


  data["wt.loss"].fillna(data["wt.loss"].mean(), inplace = True)
```

In [ ]:
```
print(data)
```

```
     time  status  age  sex  ph.ecog  ph.karno  pat.karno     meal.cal  \
1     306       1   74    0        1      90.0      100.0  1175.000000
2     455       1   68    0        0      90.0       90.0  1225.000000
3    1010       0   56    0        0      90.0       90.0   928.779006
4     210       1   57    0        1      90.0       60.0  1150.000000
5     883       1   60    0        0     100.0       90.0   928.779006
..    ...     ...  ...  ...      ...       ...        ...          ...
224   188       0   77    0        1      80.0       60.0   928.779006
225   191       0   39    0        0      90.0       90.0  2350.000000
226   105       0   75    1        2      60.0       70.0  1025.000000
227   174       0   66    0        1      90.0      100.0  1075.000000
228   177       0   58    1        1      80.0       90.0  1060.000000

       wt.loss
1     9.831776
2    15.000000
3    15.000000
4    11.000000
5     0.000000
..         ...
224   3.000000
225  -5.000000
226   5.000000
227   1.000000
228   0.000000

[226 rows x 9 columns]
```

In [ ]:
```
data.isnull().sum()
```

Out[ ]:
```
time         0
status       0
age          0
sex          0
ph.ecog      0
ph.karno     0
pat.karno    0
meal.cal     0
wt.loss      0
dtype: int64
```

In [ ]:
```
data.shape
```

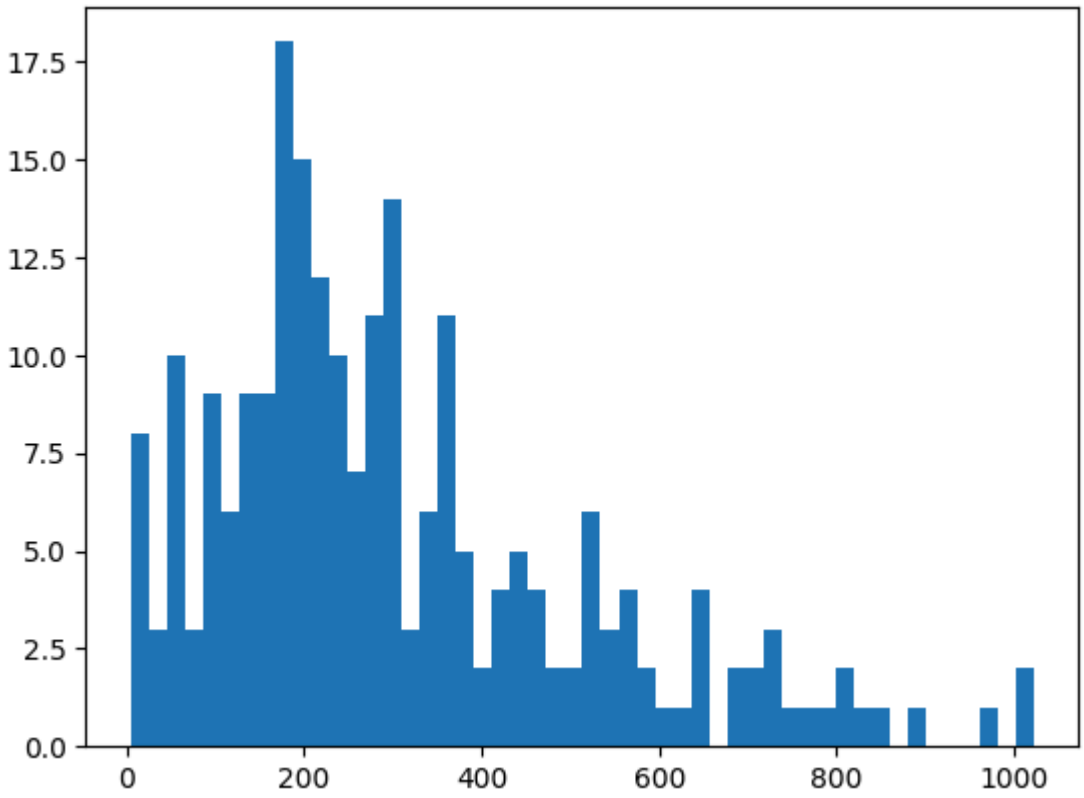
Out[ ]:
```
(227, 9)
```

Data distribution

First, save the time variable in T and event/status variable in E that we will use during the model fitting process.

Let's, plot a histogram of the time variable to get an overall idea of the distribution. The histogram shows that the time variable almost follows a Weibull or Log-normal distribution.

In [ ]:
```
T = data["time"]
E = data["status"]
plt.hist(T, bins = 50)
plt.show()
```



Kaplan-Maier Curve Estimation (Non-Parametric)

To start with survival analysis, the first step is to plot a survival curve of the overall data. It can be done by generating a Kaplan-Maier curve.

The Kaplan-Meier approach, also called the product-limit approach, is a popular approach which re-estimates the survival probability each time an event occurs. It is a non-parametric method, means it does not assume the distribution of the outcome variable (i.e., time).

Steps for generating KM curve:

step 1: Instantiate KaplanMeierFitter( ) class object

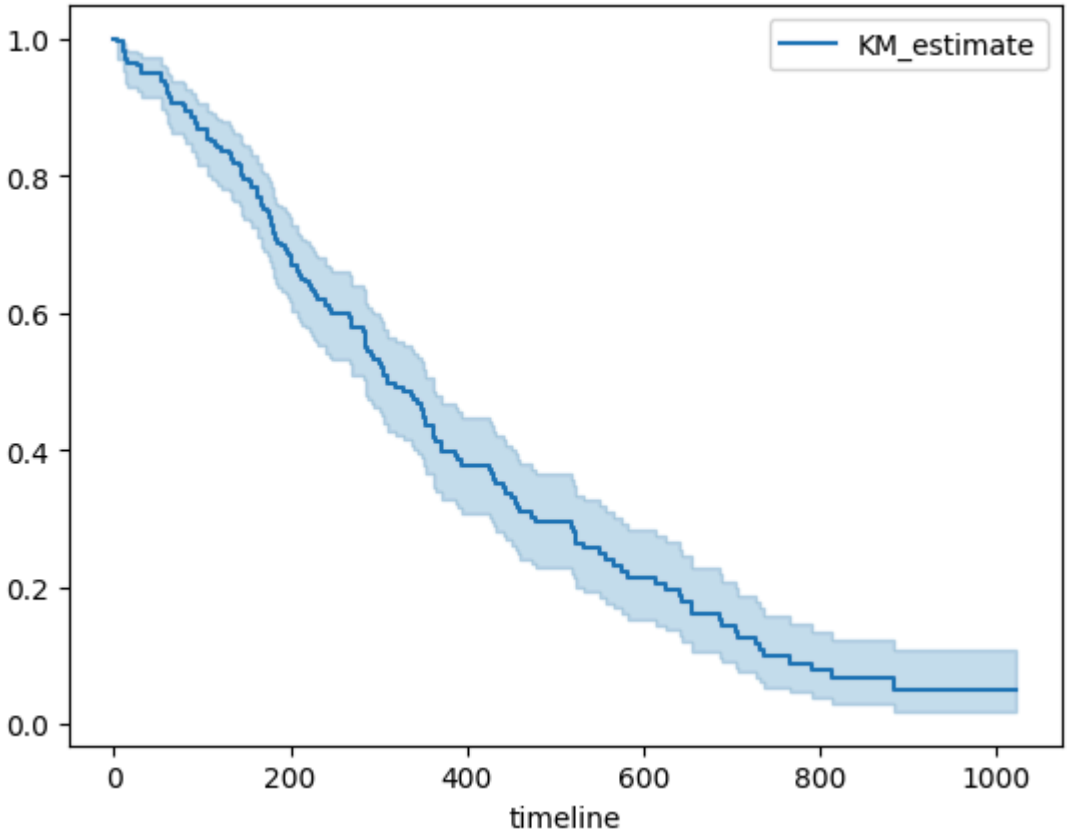
step 2: use .fit( ) method and supply duration = T and event_observed = E

step 3: use plot_survival_function( ) to generate a KM curve

The curve illustrates how the survival probabilities changes over the time horizon. As the time passes, the survival probabilities of lung cancer patents reduces.

In [ ]:
```
kmf = KaplanMeierFitter()
kmf.fit(durations = T, event_observed = E)
kmf.plot_survival_function()
```

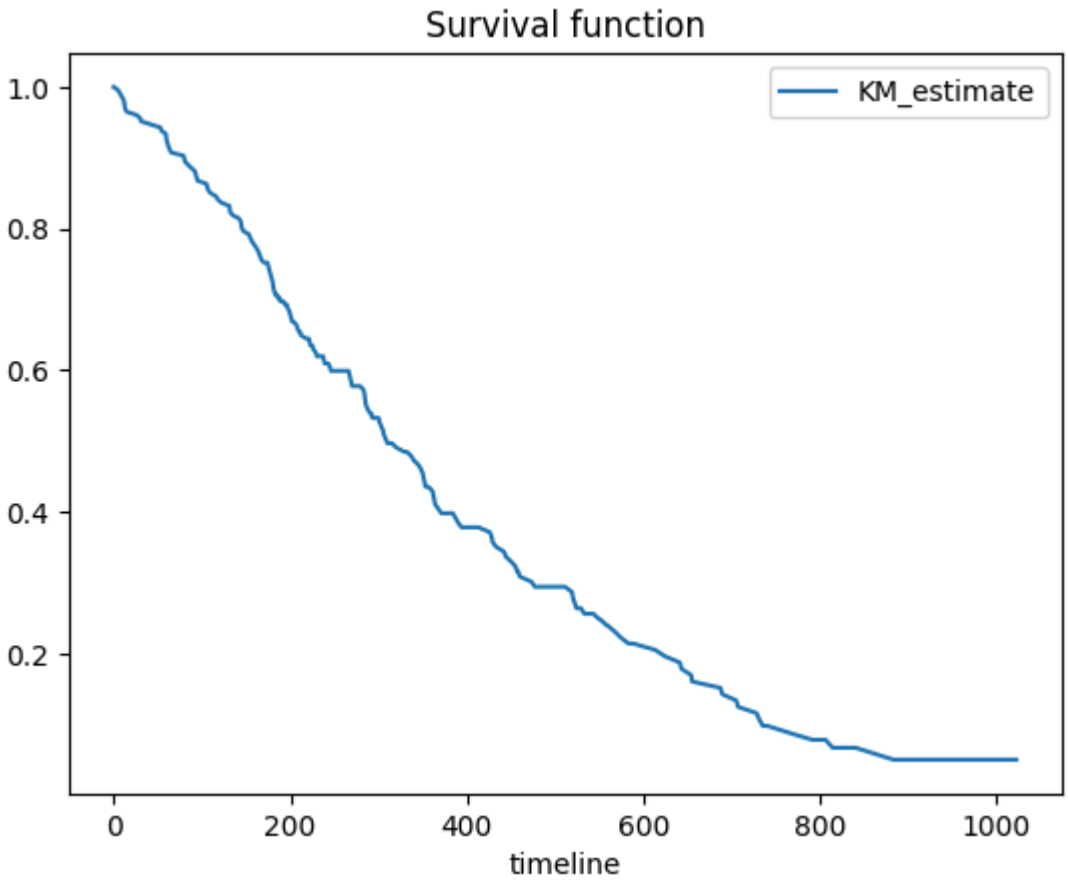
Out[ ]:
```
<Axes: xlabel='timeline'>
```



Generating the same plot without the 95% confidence interval using survival*function*.plot() method.

In [ ]:
```
kmf.survival_function_.plot()
plt.title('Survival function')
```

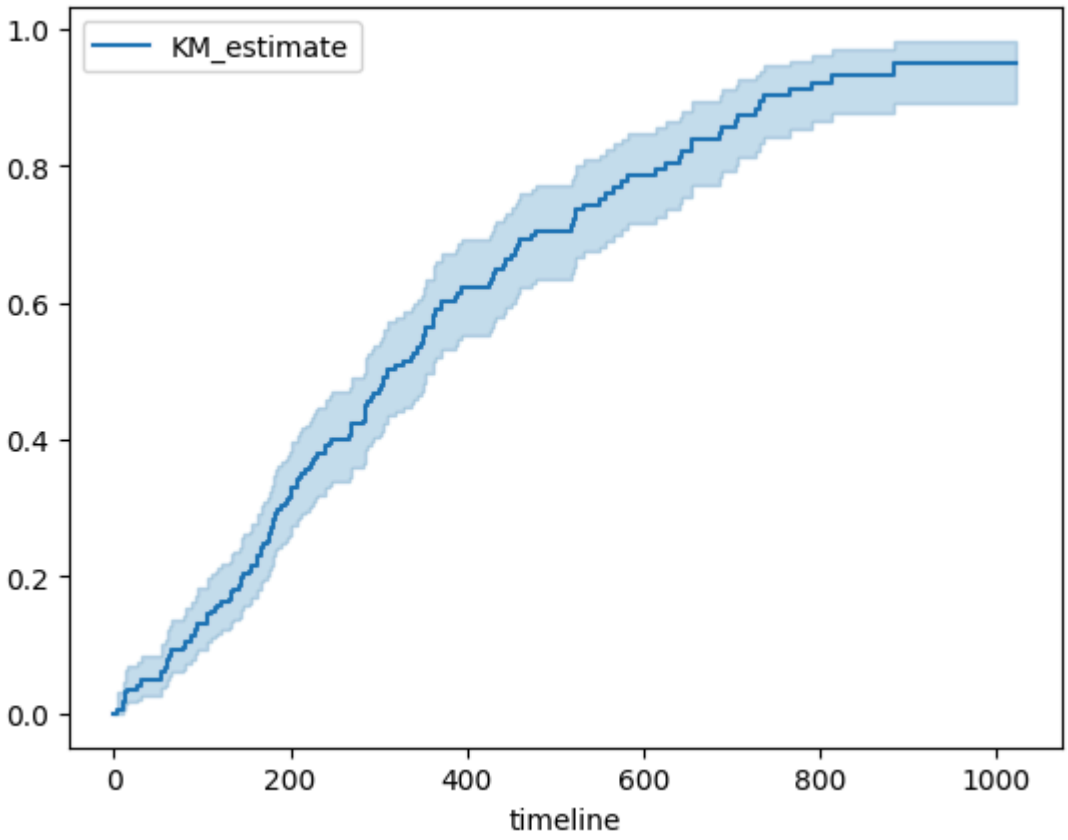
Out[ ]:
```
Text(0.5, 1.0, 'Survival function')
```



We can also plot a failure curve. It is just opposite of survival, i.e., Failure/death probabilities over time.

In [ ]:
```
kmf.plot_cumulative_density()
```

Out[ ]:
```
<Axes: xlabel='timeline'>
```



Median Survival Time and Confidence Intervals

The next step is to estimate the median survival time and 95% confidence intervals. This can be done using the .median_survival*time* and median_survival_times( ).

Here, the median survival time is 310 days, which indicates that 50% of the sample live 310 days and 50% dies within this time.

The 95% CI lower limit is 284 days, while the upper limit is 361 days.

In [ ]:
```
from lifelines.utils import median_survival_times
median_ = kmf.median_survival_time_
median_confidence_interval_ = median_survival_times(kmf.confidence_interval_
print(median_)
print(median_confidence_interval_)
```

```
310.0
     KM_estimate_lower_0.95  KM_estimate_upper_0.95
0.5                   284.0                   361.0
```

KM Plot for Gender/Sex Categories

Using the KM estimate, we can check the difference between categorical groups. Though, it is only viable when the variable has fewer categories. Here is an example where we are plotting the two survival curves, one for Male and another for Female.

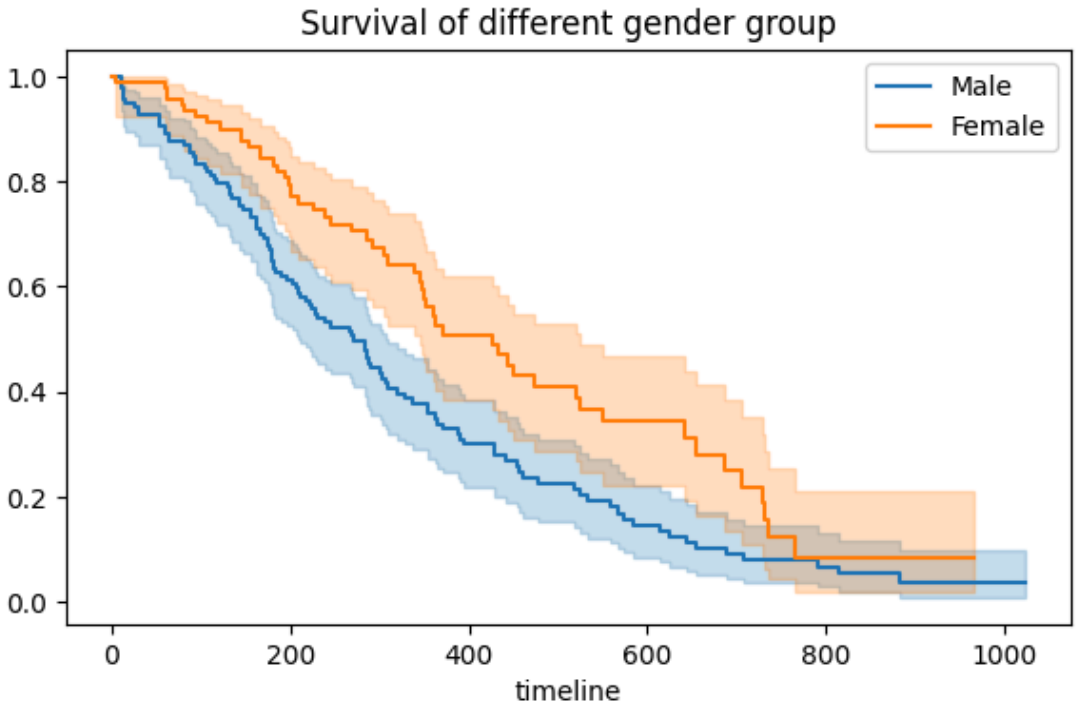
The plot generation steps are:

- create an axis object
- create a mask (filtering object) “m” where males are true
- Fit and plot curves for Male and Female observations

The curve illustrates that the survival probabilities of female patients are overall higher compared to male patients at any instance of time.

In [ ]:
```
ax = plt.subplot(111)
m = (data["sex"] == 0)
kmf.fit(durations = T[m], event_observed = E[m], label = "Male")
kmf.plot_survival_function(ax = ax)
kmf.fit(T[~m], event_observed = E[~m], label = "Female")
kmf.plot_survival_function(ax = ax, at_risk_counts = True)
plt.title("Survival of different gender group")
```

Out[ ]:
```
Text(0.5, 1.0, 'Survival of different gender group')
```



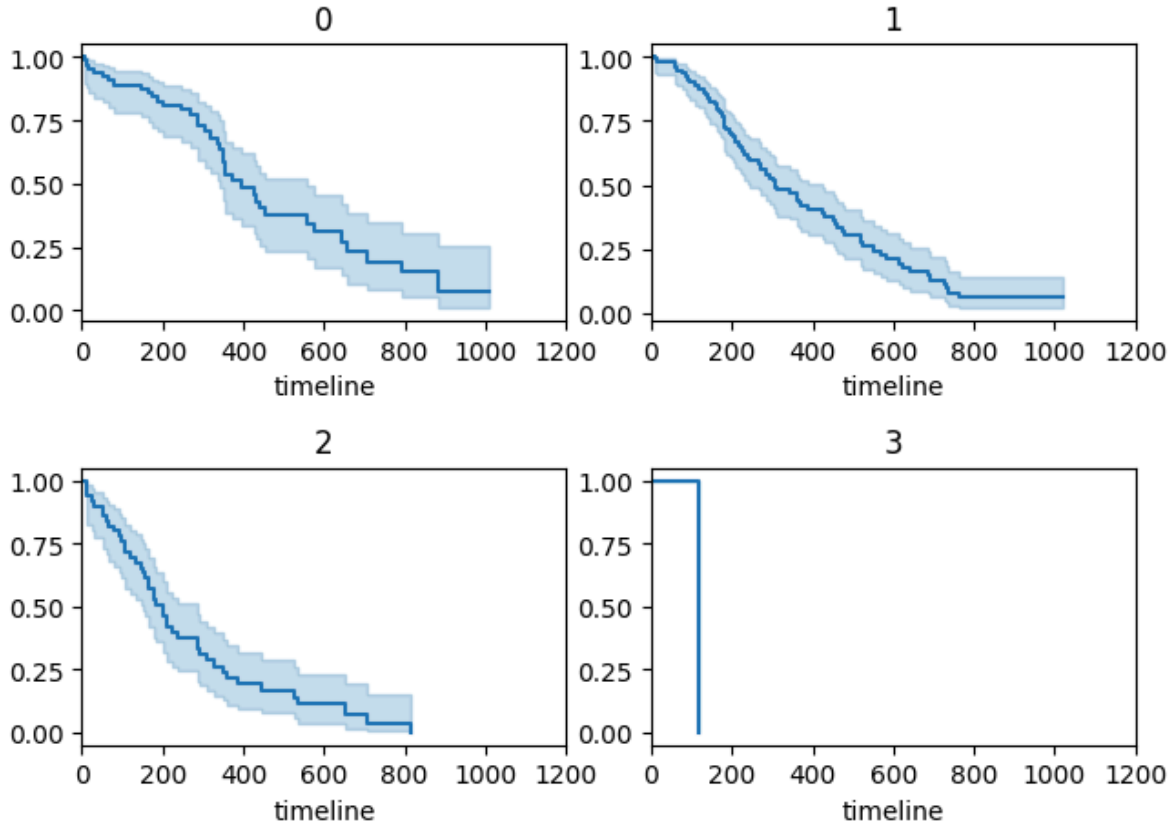
Female						
At risk	90	66	26	11	2	0
Censored	0	6	27	34	35	37
Events	0	18	37	45	53	53

KM Plot for ph.ecog Categories

Similar to Gender/Sex, we can also plot separate survival curves for other categorical variables. Here, I have used a for loop that iterate over all ph.ecog categories and plot their survival function over a single plot.

The fourth plot (row 2, column 2) where the ecog == 3, looks incomplete.

In [ ]:
```
ecog_types = data.sort_values(by = ['ph.ecog'])["ph.ecog"].unique()
for i, ecog_types in enumerate(ecog_types):
  ax = plt.subplot(2, 2, i + 1)
  ix = data['ph.ecog'] == ecog_types
  kmf.fit(T[ix], E[ix], label = ecog_types)
  kmf.plot_survival_function(ax = ax, legend = False)
  plt.title(ecog_types)
  plt.xlim(0, 1200)
plt.tight_layout()
```



We can further investigate the ph.ecog == 3 using value_counts( ) method. It shows that the category 3 has only one observation which does not contribute much if we fit a model.

In [ ]:
```
data['ph.ecog'].value_counts()
```

Out[ ]:
```
ph.ecog
1    113
0     63
2     50
3      1
Name: count, dtype: int64
```

Thus, we need to remove this observation from the data. We can achieve this using a filtering process as shown below. Now, the dataset contains 226 observations and 9 variables.

In [ ]:
```
data = data[data["ph.ecog"] != 3]
data.shape
```

Out[ ]:
```
(226, 9)
```

Let's check the value count to reconfirm it. Now, it looks good, the 3rd category has been removed.

In [ ]:
```
data['ph.ecog'].value_counts()
```

Out[ ]:
```
ph.ecog
1    113
0     63
2     50
Name: count, dtype: int64
```

Log- Rank test

The log-rank test is employed to compare the survival distributions between different groups. This test evaluates whether there is a statistically significant difference in survival between groups such as males vs. females or different ECOG status groups.

The results of the log-rank test are presented, showing p-values for each comparison.

In [ ]:
```
# Perform a log-rank test to compare the survival distributions between male
results = logrank_test(
    data[data['sex'] == 0]['time'],
    data[data['sex'] == 1]['time'],
    event_observed_A=data[data['sex'] == 0]['status'],
    event_observed_B=data[data['sex'] == 1]['status']
)

print(results)

# Print the results
print('Log-Rank Test p-value:', results.p_value)
```

```
<lifelines.StatisticalResult: logrank_test>
               t_0 = -1
 null_distribution = chi squared
degrees_of_freedom = 1
         test_name = logrank_test

---
 test_statistic      p  -log2(p)
           9.72 <0.005      9.10
Log-Rank Test p-value: 0.001826743756744226
```

The log-rank test yielded a test statistic of 9.72 and a p-value of 0.0018, indicating a statistically significant difference in survival between males and females. Since the p-value is much lower than 0.05, we reject the null hypothesis, meaning that the survival distributions are significantly different. This suggests that the covariate (e.g., gender) has a meaningful impact on survival outcomes. The result is important for clinical decision-making, indicating that survival probabilities vary based on this factor.

Cox proportional Hazard model (Semi - parameteric)

Cox-PH model is a semi-parametric model which solves the problem of incorporating covariates. In Cox's proportional hazard model, the log-hazard is a linear function of the covariates and a population-level baseline hazard

Hazard = h(t|x) = Baseline Hazard + Partial Hazard

where, Baseline hazard = b_0 t

Partial Hazard = exp($\sum_{i=1}^{n} b_i(x_i - \bar{x})$)

In the above equation, the first term is the baseline hazard and the second term known as partial hazard. The partial hazard inflates or deflates the baseline hazard based on covariates.

Cox-PH Model Assumptions

Cox proportional hazards regression model assumptions includes:

- Independence of survival times between distinct individuals in the sample
- A multiplicative relationship between the predictors and the hazard
- A constant hazard ratio over time

Definition of Hazard and Hazard Ratio

Hazard is defined as the slope of the survival curve. It is a measure of how rapidly subjects are dying. The hazard ratio compares two groups. If the hazard ratio is 2.0, then the rate of deaths in one group is twice the rate in the other group.

In [ ]:
```
dummies_ecog = pd.get_dummies(data["ph.ecog"], prefix = 'ecog')
dummies_ecog.head(4)
```

Out[ ]:

	ecog_0	ecog_1	ecog_2
1	False	True	False
2	True	False	False
3	True	False	False
4	False	True	False

We can dummy code the ph.ecog variable using pandas pd.get_dummies( ) method. I have added the prefix identifier for the generated columns. Here, we will consider the ecog_0 as the base category, so in next step we will remove it.

In [ ]:
```
dummies_ecog = dummies_ecog[["ecog_1", "ecog_2"]]
data = pd.concat([data, dummies_ecog], axis = 1)
data = data.drop("ph.ecog", axis = 1)
data.head()
```

Out[ ]:

	time	status	age	sex	ph.karno	pat.karno	meal.cal	wt.loss	ecog_1	ecog_2
1	306	1	74	0	90.0	100.0	1175.000000	9.831776	True	False
2	455	1	68	0	90.0	90.0	1225.000000	15.000000	False	False
3	1010	0	56	0	90.0	90.0	928.779006	15.000000	False	False
4	210	1	57	0	90.0	60.0	1150.000000	11.000000	True	False
5	883	1	60	0	100.0	90.0	928.779006	0.000000	False	False

Fitting Cox-PH Model

The next step is to fit the Cox-PH model.

The model fitting steps are:

- Instantiate CoxPHFitter( ) class object and save it in cph
- Call .fit( ) method and supply data, duration column and event column
- Print model estimate summary table

The summary table provides coefficients, exp(coef): also known as Hazard Ratio, confidence intervals, z and p-values.

In [ ]:

```
from lifelines import CoxPHFitter
cph = CoxPHFitter()
cph.fit(data, duration_col = 'time', event_col = 'status')
cph.print_summary()
```

model	lifelines.CoxPHFitter
duration col	'time'
event col	'status'
baseline estimation	breslow
number of observations	226
number of events observed	163
partial log-likelihood	-721.02
time fit was run	2024-09-03 07:30:55 UTC

	coef	exp(coef)	se(coef)	coef lower 95%	coef upper 95%	exp(coef) lower 95%	exp(coef) upper 95%	cmp to	z	
age	0.01	1.01	0.01	-0.01	0.03	0.99	1.03	0.00	1.33	0.1
sex	-0.58	0.56	0.17	-0.92	-0.25	0.40	0.78	0.00	-3.41	<0.00
ph.karno	0.02	1.02	0.01	-0.00	0.03	1.00	1.03	0.00	1.64	0.1
pat.karno	-0.01	0.99	0.01	-0.03	0.00	0.97	1.00	0.00	-1.77	0.0
meal.cal	0.00	1.00	0.00	-0.00	0.00	1.00	1.00	0.00	0.06	0.9
wt.loss	-0.01	0.99	0.01	-0.02	0.00	0.98	1.00	0.00	-1.73	0.0
ecog_1	0.62	1.86	0.24	0.15	1.09	1.17	2.97	0.00	2.60	0.0
ecog_2	1.20	3.31	0.37	0.46	1.93	1.59	6.89	0.00	3.19	<0.00

Concordance	0.65
Partial AIC	1458.05
log-likelihood ratio test	36.08 on 8 df
-log2(p) of ll-ratio test	15.85

Interpretation of Cox-PH Model Results/Estimates

The interpretation of the model estimates will be like this:

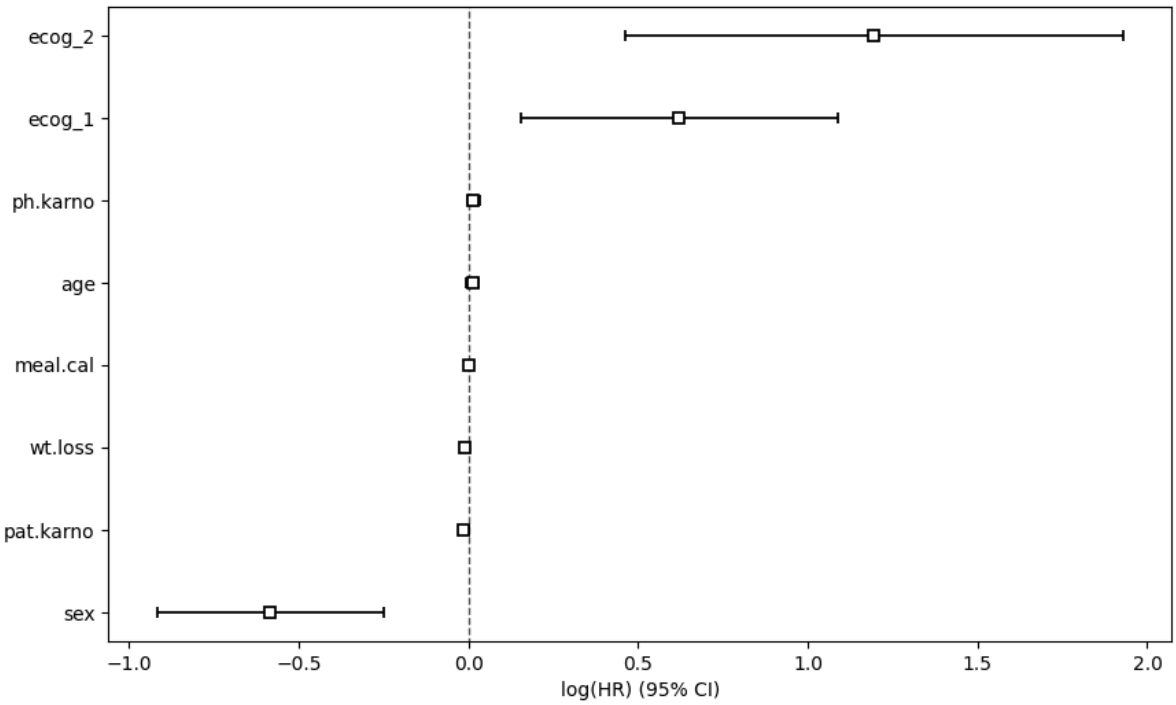
- Wt.loss has a coefficient of about -0.01. We can recall that in the Cox proportional hazard model, a higher hazard means more at risk of the event occurring. Here, the value of exp(-0.01) is called the hazard ratio.
- It shows that a one unit increase in wt loss means the baseline hazard will increase by a factor of exp(-0.01) = 0.99 → about a 1% decrease.
- Similarly, the values in the ph.ecog column are: [0 = asymptomatic, 1 = symptomatic but completely ambulatory and 2 = in bed <50% of the day]. The value of the coefficient associated with ecog2, exp(1.20), is the value of ratio of hazards (Hazard Ratio) associated with being “in bed <50% of the day (coded as 2)” compared to asymptomatic (coded as 0, base category). This indicates the risk (rate) of dying is 3.31 times for patients who are “in bed <50% of the day” compared to asymptomatic patients.

Plot Coefficients

Next we can plot the ranking of variables in terms of their log(HR) using the .plot( ) method.

In [ ]:
```
plt.subplots(figsize = (10, 6))
cph.plot()
```

Out[ ]:
```
<Axes: xlabel='log(HR) (95% CI)'>
```

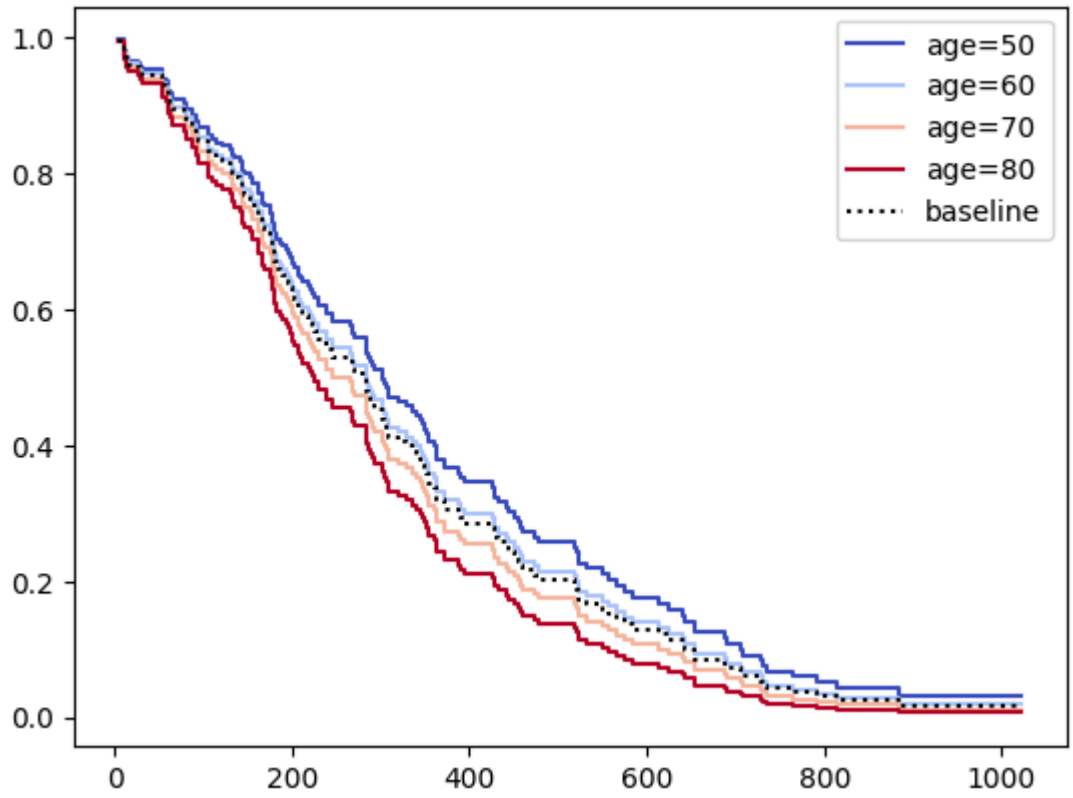


Plot Partial Effects on Outcome (Cox-PH Regression)

We can use our fitted model to see how the survival changes as we change the covariate values. Here, I have used the plot_partial_effects_on_outcome( ) method to see how the survival varies for age group of 50, 60, 70 and 80 years old patents compared to their baseline function. It clearly highlights that young patents has higher survival probabilities at any given instance of time compared to old patients.

In [ ]:
```
cph.plot_partial_effects_on_outcome(covariates = 'age', values = [50, 60, 70
```

Out[ ]:
```
<Axes: >
```



Check Proportional Hazard Assumption

Once we fit the model, the next step is to verify the proportional hazard assumption. We can directly use the check_assumptions( ) method that return a log rank test statistics.

The null (H0) hypothesis assumed that the proportional hazard criteria satisfied, while alternative hypothesis (H1) infer that the proportional hazard assumption criteria not met (violated).

In [ ]:
```
cph.check_assumptions(data, p_value_threshold = 0.05)
```

```
The ``p_value_threshold`` is set at 0.05. Even under the null hypothesis of
no violations, some
covariates will be below the threshold by chance. This is compounded when t
here are many covariates.
Similarly, when there are lots of observations, even minor deviances from t
he proportional hazard
assumption will be flagged.

With that in mind, it's best to use a combination of statistical tests and
visual tests to determine
the most serious violations. Produce visual plots using ``check_assumptions
(..., show_plots=True)``
and looking for non-constant lines. See link [A] below for a full example.
```

null_distribution	chi squared
degrees_of_freedom	1
model	<lifelines.CoxPHFitter: fitted with 226 total ...
test_name	proportional_hazard_test

		test_statistic	p	-log2(p)
age	**km**	0.42	0.52	0.95
	rank	0.18	0.67	0.58
ecog_1	**km**	1.57	0.21	2.25
	rank	1.41	0.23	2.09
ecog_2	**km**	1.20	0.27	1.87
	rank	0.86	0.35	1.50
meal.cal	**km**	5.32	0.02	5.57
	rank	4.73	0.03	5.08
pat.karno	**km**	0.21	0.64	0.64
	rank	0.17	0.68	0.55
ph.karno	**km**	3.62	0.06	4.13
	rank	3.07	0.08	3.65
sex	**km**	2.62	0.11	3.25
	rank	2.50	0.11	3.14
wt.loss	**km**	0.02	0.89	0.17
	rank	0.06	0.81	0.31

```
1. Variable 'meal.cal' failed the non-proportional test: p-value is 0.0210.

   Advice 1: the functional form of the variable 'meal.cal' might be incorr
ect. That is, there may
be non-linear terms missing. The proportional hazard test used is very sens
itive to incorrect
functional forms. See documentation in link [D] below on how to specify a f
unctional form.

   Advice 2: try binning the variable 'meal.cal' using pd.cut, and then spe
cify it in
`strata=['meal.cal', ...]` in the call in `.fit`. See documentation in link
[B] below.

   Advice 3: try adding an interaction term with your time variable. See do
cumentation in link [C]
below.


---
[A]  https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportio
nal%20hazard%20assumption.html
[B]  https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportio
nal%20hazard%20assumption.html#Bin-variable-and-stratify-on-it
[C]  https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportio
nal%20hazard%20assumption.html#Introduce-time-varying-covariates
[D]  https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportio
nal%20hazard%20assumption.html#Modify-the-functional-form
[E]  https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportio
nal%20hazard%20assumption.html#Stratification
```

Out[ ]:
```
[]
```

We can also use the proportional_hazard_test( ) method to perform the same.

The result revealed that at 5% significance level, only meal.cal violated the assumption. There are various approach to deal with it, for example we can convert it to a binned category, or we can use a parametric Cox-PH model.

In [ ]:
```
from lifelines.statistics import proportional_hazard_test
results = proportional_hazard_test(cph, data, time_transform='rank')
results.print_summary(decimals=3, model="untransformed variables")
```

time_transform	rank
null_distribution	chi squared
degrees_of_freedom	1
model	<lifelines.CoxPHFitter: fitted with 226 total ...
test_name	proportional_hazard_test

	test_statistic	p	-log2(p)
age	0.18	0.67	0.58
ecog_1	1.41	0.23	2.09
ecog_2	0.86	0.35	1.50
meal.cal	4.73	0.03	5.08
pat.karno	0.17	0.68	0.55
ph.karno	3.07	0.08	3.65
sex	2.50	0.11	3.14
wt.loss	0.06	0.81	0.31

The proportional hazards test results show that most covariates meet the proportional hazards assumption, with p-values above 0.05, indicating consistent effects over time. However, meal.cal (p = 0.03) shows a potential violation, suggesting its impact on survival may change over time. ph.karno (p = 0.08) is close to the threshold, indicating a marginal concern. These findings suggest that while most covariates are stable, meal calorie intake might require further investigation.

Conclusion :

In conclusion, the findings from the survival analysis of lung cancer patients, utilizing the Kaplan-Meier curve, log-rank test, and Cox proportional hazards model, provide a comprehensive understanding of survival dynamics in this patient population.

The Kaplan-Meier curve effectively illustrated the survival probabilities over time, revealing critical insights into the overall survival experience of lung cancer patients. The analysis indicated a decline in survival probabilities as time progressed, emphasizing the need for timely interventions and monitoring.

The log-rank test further enhanced our understanding by comparing survival distributions between different groups, such as males and females or varying ECOG performance statuses. The results demonstrated statistically significant differences in survival, highlighting the impact of demographic and clinical factors on patient outcomes.

Additionally, the Cox proportional hazards model allowed for the assessment of the influence of multiple covariates on survival simultaneously. This model identified significant prognostic factors, such as age, sex, and performance status, which can be crucial for risk stratification and personalized treatment planning.

Overall, these analytical methods collectively underscore the complexity of survival in lung cancer patients and the importance of considering various factors when predicting outcomes. The insights gained from this study can inform clinical practices, guiding healthcare professionals in making evidence-based decisions to improve patient care and survival rates. Future research should continue to refine these models and explore additional variables to enhance the predictive accuracy and applicability of survival analyses in lung cancer and other malignancies.