June 12, 2023

```
[ ]: #Q1 ALK 40     60
     #Q2     PFS
     #Q3
     #Q4:       KM
```

```
[ ]: import numpy as np
     import pandas as pd
     import os
     from openpyxl import utils
     import warnings
     import statistics
     warnings.filterwarnings("ignore", category=UserWarning)
     from scipy.stats import fisher_exact

     from lifelines import KaplanMeierFitter
     import matplotlib.pyplot as plt
     from lifelines import CoxPHFitter
```

```
[ ]:   = pd.read_excel('C:/Users/NC-010/Dropbox/  /   /    .xlsx',␣
      ↪sheet_name='  100 ')
       = pd.read_excel('C:/Users/NC-010/Dropbox/  /   /    .xlsx',␣
      ↪sheet_name=' 100 ')
```

```
[ ]: ##  .
```

```
[ ]: # recoding
     from datetime import datetime
       =  ['   ']
     df = pd.DataFrame(  )

     # Remove rows with NA values
       = df.dropna()
     #print( )
      _ =np.mean( )
      max=np.max( )
      min=np.min( )
     print(  _ )
     print( min)
```

```
print( max)
#recode   greater than mean=1, lower than mean=0
# Assuming you have a DataFrame with an 'Age' column
# Sample age values with NA
ages =  ['  ']
 _  = pd.DataFrame({'Age': ages})
# Fill NA values with a specific value (e.g., 0)
 _ ['Age'] =  _ ['Age'].fillna(64)

#print( _ )
 recode = pd.DataFrame({'Age':  ['  ']})

# Calculate the mean age
mean_age = 64.1

# Create a new column for recoded categories
 recode['Age_Category'] = ''
 recode= recode.fillna(0)
# Recode ages based on mean
 recode.loc[ recode['Age'] > mean_age, 'Age_Category'] = '1'
 recode.loc[ recode['Age'] < mean_age, 'Age_Category'] = '0'

# Print the updated DataFrame
#print( recode)
ALK= ['ALK']
ALK= ['ALK'].fillna(1)
ALK_recoded = ALK.where(ALK != ' ', 0)
#print(ALK_recoded)
#calculate the fisher's exact test p-value
from scipy.stats import fisher_exact
import pandas as pd
# Assuming you have a DataFrame with exposure and outcome columns
  _  = pd.DataFrame({'Exposure':  recode['Age_Category'],
                   'Outcome': ALK_recoded})
# Use replace() method to recode the variable
# Extract the exposure and outcome columns as separate variables
exposure =   _ ['Exposure']
outcome =   _ ['Outcome']
# Perform Fisher's exact test
odds_ratio, p_value = fisher_exact(pd.crosstab(exposure, outcome))
# Print the results
#print("Odds Ratio:", odds_ratio)
print("p-value:", p_value)
```

```
64.11627906976744
38.0
81.0
p-value: 0.4135961199851336
```

[ ]:
```python
#
age= [' '].describe()
print(age)
```

```
count     100
unique      2
top
freq       98
Name:  , dtype: object
```

[ ]:
```python
#
 = ['  '].value_counts()
print( )
```

```

       85
       10
         4
Name: count, dtype: int64
```

[ ]:
```python
#TNM
TNM= ['TNM '].value_counts()
print(TNM)
```

```
TNM
IV      36
IIIB    32
IVA     17
IVB     10
IIIC     5
Name: count, dtype: int64
```

[ ]:
```python
#EGFR
EGFR= ['EGFR'].value_counts()
print(EGFR)
```

```
EGFR
     40
      1
Name: count, dtype: int64
```

[ ]:
```python
#ALK
ALK= ['ALK'].value_counts()
print(ALK)
```

```
ALK
     40
Name: count, dtype: int64
```

[ ]:
```
#
  = ['     '].value_counts()
print(  )
```

```
4    31
2    30
3    19
5    12
6     6
8     1
7     1
Name: count, dtype: int64
```

[ ]:
```
#
 = ['      mg '].describe()
print( )
```

```
count      98.000000
mean      794.418367
std       402.080618
min       180.000000
25%       510.000000
50%       705.000000
75%      1080.000000
max      1800.000000
Name:       mg , dtype: float64
```

[ ]:
```
#
   = ['     '].value_counts()
print(   )
```

```
       61
       19
        14
        3
        2
        1
Name: count, dtype: int64
```

[ ]:
```
#
    = ['     '].value_counts()
print(   )
```

```
2     32
3     25
1     13
```

```
4      10
5       7
7       3
10      2
12      2
9       2
6       2
8       2
Name: count, dtype: int64
```

[ ]:

```

```

[ ]:

```
##  .
```

[ ]:

```
#
 = ['  '].value_counts()
print( )
#
 = ['  '].value_counts()
print( )
# PFS
PFS= [' PFS '].value_counts()
print(PFS)
```

[ ]:

```
# Convert date columns to datetime data type
   ={'start': ['      '] ,'end':  ['      ']}
  =pd.DataFrame(    )
  ['start'] = pd.to_datetime(   ['start'] )
  ['end'] = pd.to_datetime(   ['end'])

# Subtract the columns and create a new column with the result
  ['date_difference'] =    ['end'] -    ['start']

# Print the resulting dataset
print(  ['date_difference'])
```

```
0      67 days
1     144 days
2     323 days
3     356 days
4     157 days
        …
95     51 days
96      0 days
97     26 days
98     27 days
99     26 days
Name: date_difference, Length: 100, dtype: timedelta64[ns]
```

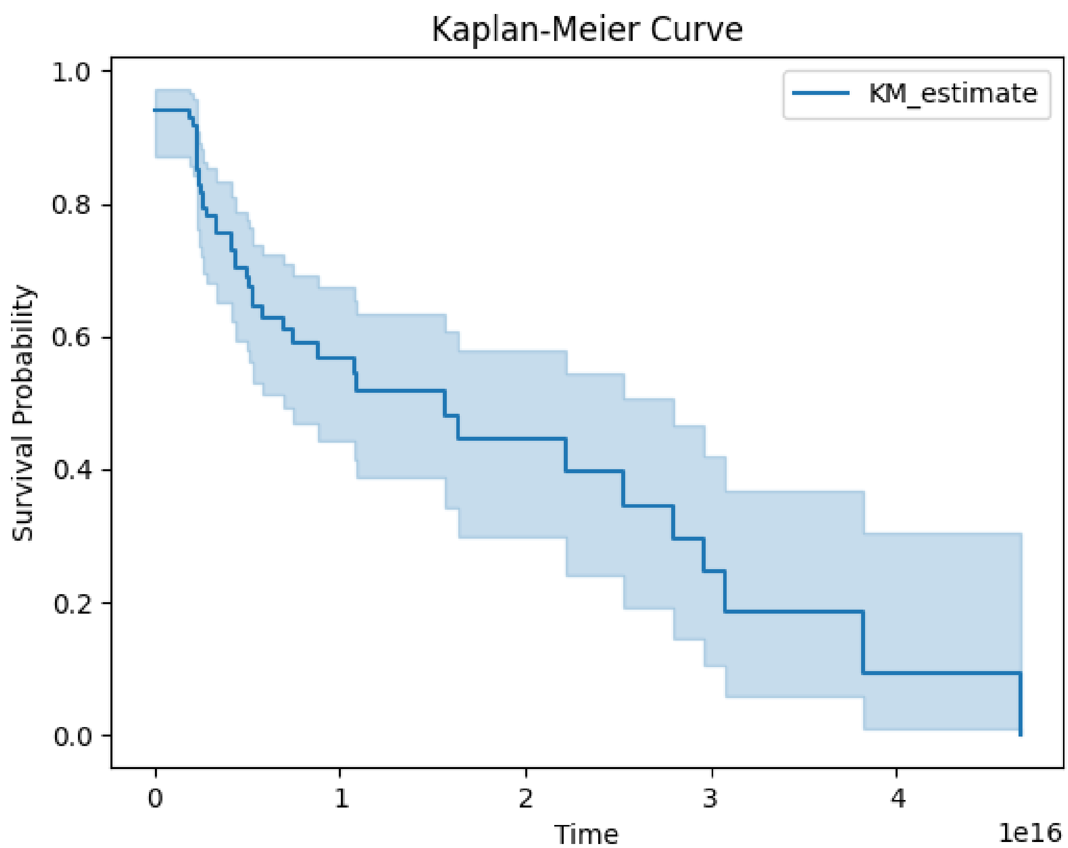
[ ]:

```

```

```
0    82
1    18
Name: count, dtype: int64

1    58
0    42
Name: count, dtype: int64
 PFS
0    55
1    45
Name: count, dtype: int64
```

[ ]:

```python
data = {
    'time':    ['date_difference'],
    'event':   [' PFS ']
}

df = pd.DataFrame(data)

# Fit Kaplan-Meier estimator
kmf = KaplanMeierFitter()
kmf.fit(df['time'], event_observed=df['event'])

# Generate KM curve
kmf.plot()
plt.xlabel('Time')
plt.ylabel('Survival Probability')
plt.title('Kaplan-Meier Curve')
plt.show()
```




```
[ ]: import pandas as pd
     from lifelines import KaplanMeierFitter
     from lifelines.statistics import logrank_test

     # Create two sample datasets for comparison
     group1 = {
         'time': [10, 15, 20, 25, 30],
         'event': [1, 0, 1, 0, 1]
     }

     group2 = {
         'time': [10, 15, 20, 25, 30],
         'event': [0, 0, 0, 0, 0]
     }

     df_group1 = pd.DataFrame(group1)
     df_group2 = pd.DataFrame(group2)

     # Fit Kaplan-Meier estimators for each group
```

```
kmf_group1 = KaplanMeierFitter()
kmf_group1.fit(df_group1['time'], event_observed=df_group1['event'])

kmf_group2 = KaplanMeierFitter()
kmf_group2.fit(df_group2['time'], event_observed=df_group2['event'])

# Perform log-rank test
results = logrank_test(df_group1['time'], df_group2['time'],
    df_group1['event'], df_group2['event'])

# Print the log-rank test statistic and p-value
print("Log-Rank Test Statistic: %.2f" % results.test_statistic)
print("Log-Rank Test p-value: %.4f" % results.p_value)
```

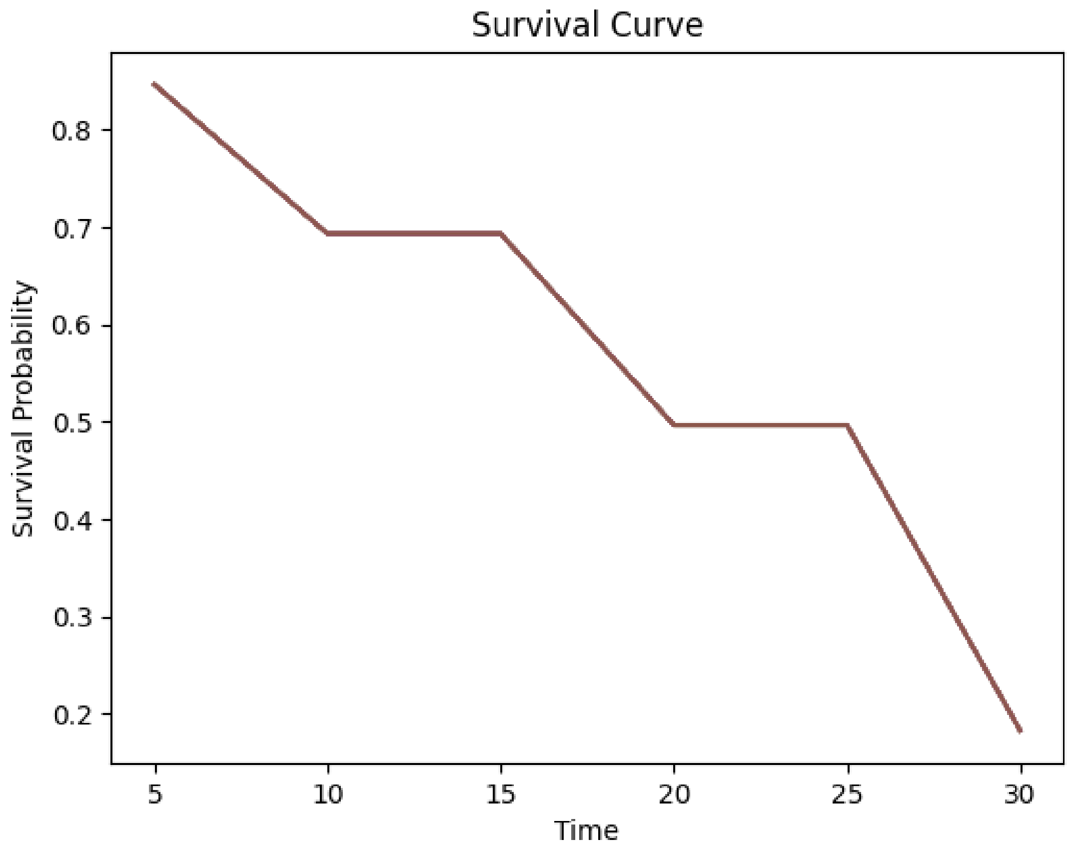
```
Log-Rank Test Statistic: 3.00
Log-Rank Test p-value: 0.0833
```

[ ]:

```
#logrank for 2 groups
import pandas as pd
from lifelines import KaplanMeierFitter
from lifelines.statistics import logrank_test

# Create two sample datasets for comparison
group1 = {
    'time': [10, 15, 20, 25, 30],
    'event': [1, 0, 1, 0, 1]
}

group2 = {
    'time': [5, 10, 15, 20, 25],
    'event': [1, 1, 0, 1, 0]
}

df_group1 = pd.DataFrame(group1)
df_group2 = pd.DataFrame(group2)

# Fit Kaplan-Meier estimators for each group
kmf_group1 = KaplanMeierFitter()
kmf_group1.fit(df_group1['time'], event_observed=df_group1['event'])

kmf_group2 = KaplanMeierFitter()
kmf_group2.fit(df_group2['time'], event_observed=df_group2['event'])

# Perform log-rank test
results = logrank_test(df_group1['time'], df_group2['time'],
    df_group1['event'], df_group2['event'])
```

```python
# Print the log-rank test statistic and p-value
print("Log-Rank Test Statistic: %.2f" % results.test_statistic)
print("Log-Rank Test p-value: %.4f" % results.p_value)
```

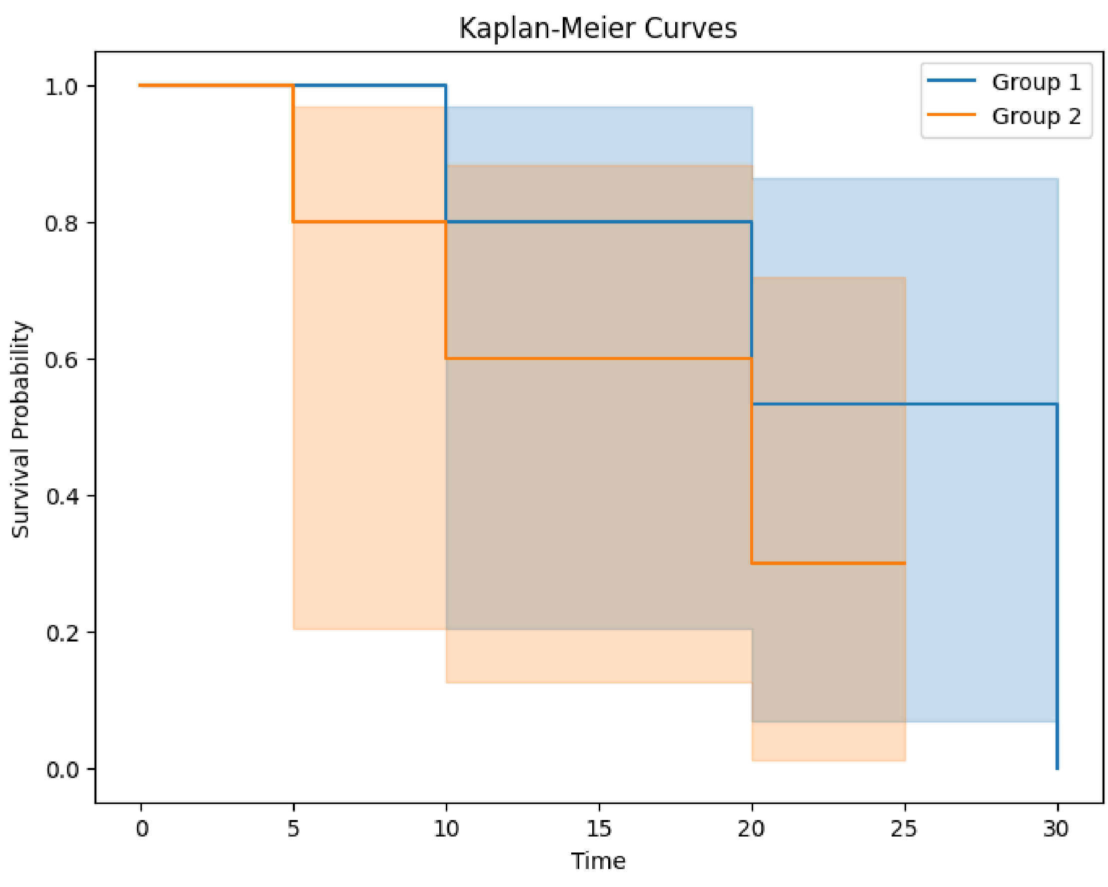
[ ]:
```python
import pandas as pd
from lifelines import CoxPHFitter
import matplotlib.pyplot as plt

# Create a sample dataset
data = {
    'time': [5, 10, 15, 20, 25, 30],
    'event': [1, 1, 0, 1, 0, 1]
}

df = pd.DataFrame(data)

# Fit Cox proportional hazards model
cph = CoxPHFitter()
cph.fit(df, 'time', event_col='event')

# Generate survival curve
survival_prob = cph.predict_survival_function(df)

# Plot the survival curve
plt.plot(survival_prob.index, survival_prob.values)
plt.xlabel('Time')
plt.ylabel('Survival Probability')
plt.title('Survival Curve')
plt.show()

# Get hazard ratios
hr = cph.hazard_ratios_
print(hr)
```




```
Series([], Name: exp(coef), dtype: float64)
```

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

# Create two sample datasets for comparison
group1 = {
    'time': [10, 15, 20, 25, 30],
    'event': [1, 0, 1, 0, 1]
}

group2 = {
    'time': [5, 10, 15, 20, 25],
    'event': [1, 1, 0, 1, 0]
}

df_group1 = pd.DataFrame(group1)
df_group2 = pd.DataFrame(group2)
```

```python
# Fit Kaplan-Meier estimators for each group
kmf_group1 = KaplanMeierFitter()
kmf_group1.fit(df_group1['time'], event_observed=df_group1['event'])

kmf_group2 = KaplanMeierFitter()
kmf_group2.fit(df_group2['time'], event_observed=df_group2['event'])

# Plot KM curves
plt.figure(figsize=(8, 6))
kmf_group1.plot(label='Group 1')
kmf_group2.plot(label='Group 2')
plt.xlabel('Time')
plt.ylabel('Survival Probability')
plt.title('Kaplan-Meier Curves')
plt.legend()
plt.show()
```



[ ]:
```python
import pandas as pd
from lifelines import CoxPHFitter
```

```python
# Create a sample dataset
data = {
    'time': [5, 10, 15, 20, 25, 30],
    'event': [1, 1, 0, 1, 0, 1],
    'group': [0, 0, 1, 1, 0, 1],
    'censored': [0, 1, 0, 1, 0, 1],
    'sex': [1, 0, 1, 0, 0, 1],
    'age': [20, 25, 30, 35, 40, 45]
}

df = pd.DataFrame(data)

# Fit Cox proportional hazards model
cph = CoxPHFitter()
cph.fit(df, 'time', event_col='event', show_progress=False)

# Get the hazard ratios
hr = cph.hazard_ratios_

# Print the hazard ratios
print(hr)
```

```
covariate
group          0.036118
censored     284.109120
sex         1174.127206
age            0.152692
Name: exp(coef), dtype: float64
```

```
c:\Users\NC-010\AppData\Local\Programs\Python\Python310\lib\site-
packages\lifelines\fitters\coxph_fitter.py:1586: ConvergenceWarning: The log-
likelihood is getting suspiciously close to 0 and the delta is still large.
There may be complete separation in the dataset. This may result in incorrect
inference of coefficients. See https://stats.stackexchange.com/q/11109/11867 for
more.

  warnings.warn(
c:\Users\NC-010\AppData\Local\Programs\Python\Python310\lib\site-
packages\lifelines\utils\__init__.py:1122: ConvergenceWarning: Column censored
have very low variance when conditioned on death event present or not. This may
harm convergence. This could be a form of 'complete separation'. For example,
try the following code:

>>> events = df['event'].astype(bool)
>>> print(df.loc[events, 'censored'].var())
>>> print(df.loc[~events, 'censored'].var())
```

```
A very low variance means that the column censored completely determines whether
a subject dies or not. See https://stats.stackexchange.com/questions/11109/how-
to-deal-with-perfect-separation-in-logistic-regression.

  warnings.warn(dedent(warning_text), ConvergenceWarning)
c:\Users\NC-010\AppData\Local\Programs\Python\Python310\lib\site-
packages\lifelines\utils\__init__.py:1165: ConvergenceWarning: Column age has
high sample correlation with the duration column. This may harm convergence.
This could be a form of 'complete separation'.     See
https://stats.stackexchange.com/questions/11109/how-to-deal-with-perfect-
separation-in-logistic-regression

  warnings.warn(dedent(warning_text), ConvergenceWarning)
c:\Users\NC-010\AppData\Local\Programs\Python\Python310\lib\site-
packages\lifelines\fitters\coxph_fitter.py:1611: ConvergenceWarning: Newton-
Rhaphson failed to converge sufficiently. Please see the following tips in the
lifelines documentation:
https://lifelines.readthedocs.io/en/latest/Examples.html#problems-with-
convergence-in-the-cox-proportional-hazard-model
  warnings.warn(
```

[ ]:
```
## .
```

[ ]:
```
 = ['  ']

# number of PR
# Count the occurrences of a specific string in a column
PR = 'PR'
count = ( ['  '] == PR).sum()
# Print the count
print("Count of '{}' is {}".format(PR, count))

# number of SD
# Count the occurrences of a specific string in a column
SD = 'SD'
count = ( ['  '] == SD).sum()
# Print the count
print("Count of '{}' is {}".format(SD, count))

# number of PD
# Count the occurrences of a specific string in a column
PD = 'PD'
count = ( ['  '] == PD).sum()
# Print the count
print("Count of '{}' is {}".format(PD, count))
# number of CR
# Count the occurrences of a specific string in a column
```

```
CR = 'CR'
count = ( ['   '] == CR).sum()
# Print the count
print("Count of '{}' is {}".format(CR, count))

#PR SD PD CR
PR=51/(51+50+18+13)
print("PR='{}'".format(PR))
SD=50/(51+50+18+13)
print("SD='{}'".format(SD))
PD=18/(51+50+18+13)
print("PD='{}'".format(PD))
CR=13/(51+50+18+13)
print("CR='{}'".format(CR))

#ORR DCR
ORR=CR+PR
print("ORR='{}'".format(ORR))
DCR=ORR+SD
print("DCR='{}'".format(DCR))
```

```
Count of 'PR' is 43
Count of 'SD' is 29
Count of 'PD' is 4
Count of 'CR' is 4
PR='0.38636363636363635'
SD='0.3787878787878788'
PD='0.13636363636363635'
CR='0.09848484848484848'
ORR='0.48484848484848486'
DCR='0.8636363636363636'
```