You can use “lifelines” library to do some survival analysys:

```
from lifelines import KaplanMeierFitter
from lifelines import WeibullFitter
from lifelines.statistics import multivariate_logrank_test
from lifelines.statistics import logrank_test
from lifelines.statistics import pairwise_logrank_test
```

When your data is structured like this:

```
df.head()
```

	tiempo	status	sexo	replica	cepa	fecha
0	12	1	m	1.0	CEP085	2023-11-21
1	12	1	f	1.0	CEP085	2023-11-21
2	14	1	m	1.0	CEP085	2023-11-21
3	14	1	f	1.0	CEP085	2023-11-21
4	14	1	f	1.0	CEP085	2023-11-21

You can make your own color palette!

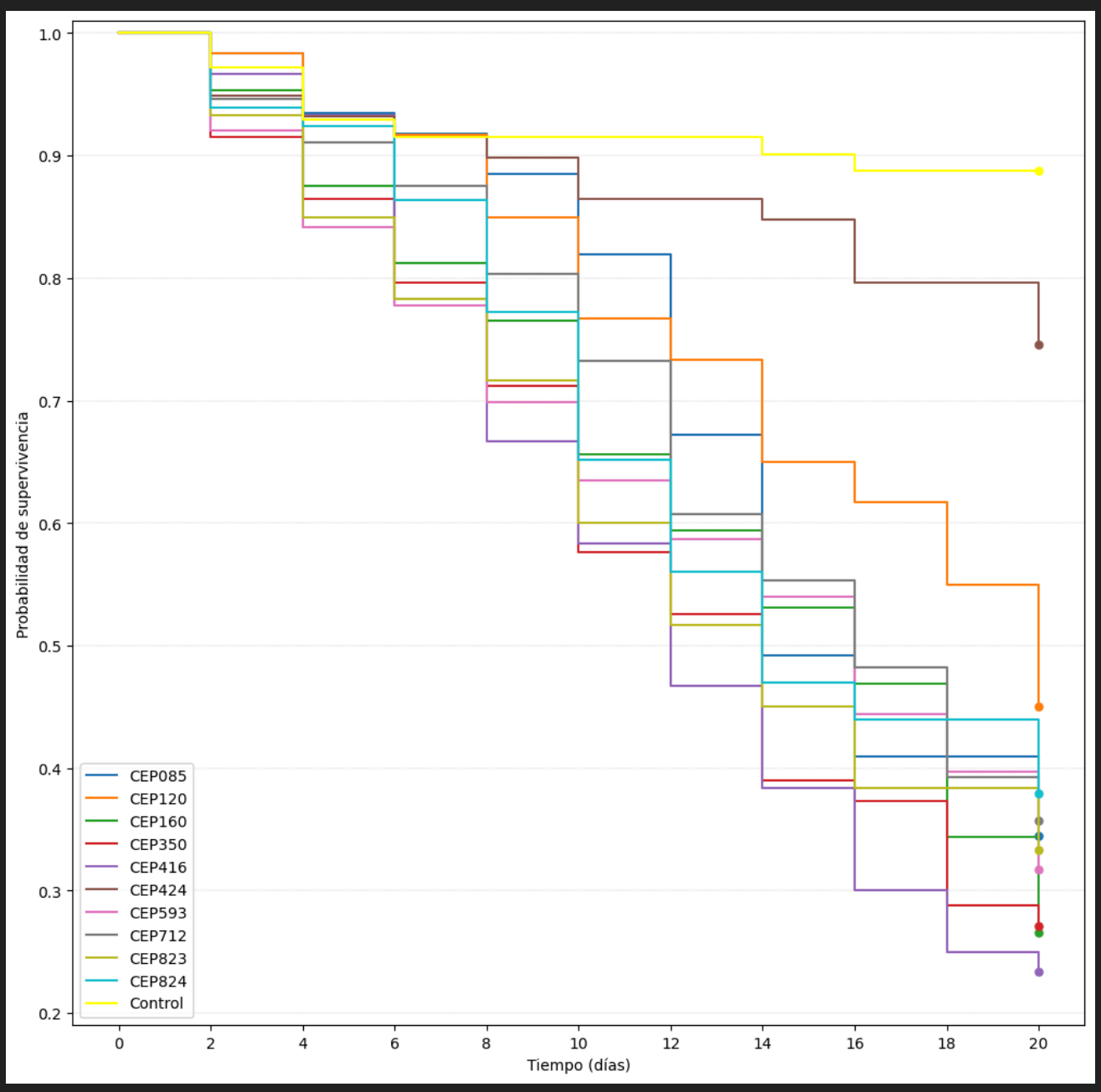
```
paleta = sns.color_palette( n_colors=11)
amarillo=(1,1,0)
paleta[-1] = amarillo
paleta
```

And fit the data to Kaplan-Meier curves:

```
kmf = KaplanMeierFitter()
contador=0
#Creo un grafico de supervivencia para cada tratamiento con un ciclo for
for cepa in df['cepa'].unique():#Iterar sobre cada cepa única en el dataframe
    cepa_data = df[df['cepa'] == cepa]#Filtrar el dataframe por la cepa
    kmf.fit(cepa_data['tiempo'], event_observed=cepa_data['status'], label=f'{cepa}')#Ajustar el modelo Kaplan-Meier para la cepo filtrada
    kmf.plot_survival_function(color=paleta[contador],
        show_censors=True, censor_styles={'ms': 5, 'marker': 'o'},
        #at_risk_counts=True,
        ci_alpha=0)#Graficar la curva de supervivencia sin intervalo de confianza visible

    #Obtener TL50
    tl50_time = kmf.median_survival_time_# Obtener el TL50
    ci_1 = kmf.confidence_interval_survival_function_
    print(f'TL50 de Cep{cepa}: {tl50_time} días')# Imprimir el TL50
    print(kmf.survival_function_)
    print(ci_1)
    contador += 1

plt.xlabel('Tiempo (días)')
plt.ylabel('Probabilidad de supervivencia')
plt.legend(loc='best')
plt.ylim([0.19, 1.01])
plt.yticks([.2,.3,.4,.5,.6,.7,.8,.9,1])
plt.xticks(range(0, int(max(df['tiempo']))+1, 2))#Mostrar el tiempo en el eje x de 2 en 2
plt.grid(True, linestyle='dashed', linewidth=.2, ydata=None)
plt.show()
```



You can also compare these curves pairwise using the log-rank test to see if they are significantly different:

```
#LOGRANK DE A PARES
resultados = pairwise_logrank_test(df['tiempo'], df['cepa'], df['status'],t_0=20)
resultados
```

	t_0	20		
	null_distribution	chi squared		
	degrees_of_freedom	1		
	test_name	logrank_test		
		test_statistic	**p**	**-log2(p)**
CEP085	**CEP120**	1.47	0.22	2.15
	CEP160	0.96	0.33	1.61
	CEP350	2.29	0.13	2.94
	CEP416	4.06	0.04	4.51
	CEP424	18.30	<0.005	15.69
	CEP593	0.44	0.51	0.98
	CEP712	0.01	0.93	0.10
	CEP823	0.81	0.37	1.44
	CEP824	0.06	0.80	0.32
	Control	39.30	<0.005	31.36
CEP120	**CEP160**	5.05	0.02	5.34

And use the Benjamini-Hochberg method for false discovery rate:

```python
rejected, p_adjusted = fdrcorrection(p_values, alpha=0.05, method='indep', is_sorted=False)
#Convertir los resultados a un dataframe
adjusted_p_values_df = pd.DataFrame({
    "name" : names,
    'p_valor_original': p_values,
    'p_valor_ajustado': p_adjusted,
    'rechazar_hipotesis_nula': rejected
})
adjusted_p_values_df
```

	name	p_valor_original	p_valor_ajustado	rechazar_hipotesis_nula
0	(CEP085, CEP120)	2.248358e-01	3.747263e-01	False
1	(CEP085, CEP160)	3.265381e-01	4.988776e-01	False
2	(CEP085, CEP350)	1.300333e-01	2.554225e-01	False
3	(CEP085, CEP416)	4.400920e-02	1.052394e-01	False
4	(CEP085, CEP424)	1.888368e-05	6.924016e-05	True
5	(CEP085, CEP593)	5.066439e-01	6.333049e-01	False
6	(CEP085, CEP712)	9.312107e-01	9.549439e-01	False
7	(CEP085, CEP823)	3.692478e-01	5.192231e-01	False

Conditional formatting can be applied to the p-values to display them clearly, and the same CLD method can be applied to show significant differences between survival functions:

	CEP085	CEP120	CEP160	CEP350	CEP416	CEP424	CEP593	CEP712	CEP823	CEP824	Control
CEP085		0,22	0,33	0,13	0,04	0,005	0,51	0,93	0,37	0,8	0,005
CEP120	0,22		0,02	0,01	0,005	0,005	0,06	0,21	0,06	0,22	0,005
CEP160	0,33	0,02		0,62	0,31	0,005	0,77	0,38	0,87	0,37	0,005
CEP350	0,13	0,01	0,62		0,66	0,005	0,48	0,19	0,58	0,21	0,005
CEP416	0,04	0,005	0,31	0,66		0,005	0,25	0,07	0,34	0,09	0,005
CEP424	0,005	0,005	0,005	0,005	0,005		0,005	0,005	0,005	0,005	0,04
CEP593	0,51	0,06	0,77	0,48	0,25	0,005		0,55	0,94	0,54	0,005
CEP712	0,93	0,21	0,38	0,19	0,07	0,005	0,55		0,47	0,96	0,005
CEP823	0,37	0,06	0,87	0,58	0,34	0,005	0,94	0,47		0,5	0,005
CEP824	0,8	0,22	0,37	0,21	0,09	0,005	0,54	0,96	0,5		0,005
Control	0,005	0,005	0,005	0,005	0,005	0,04	0,005	0,005	0,005	0,005	

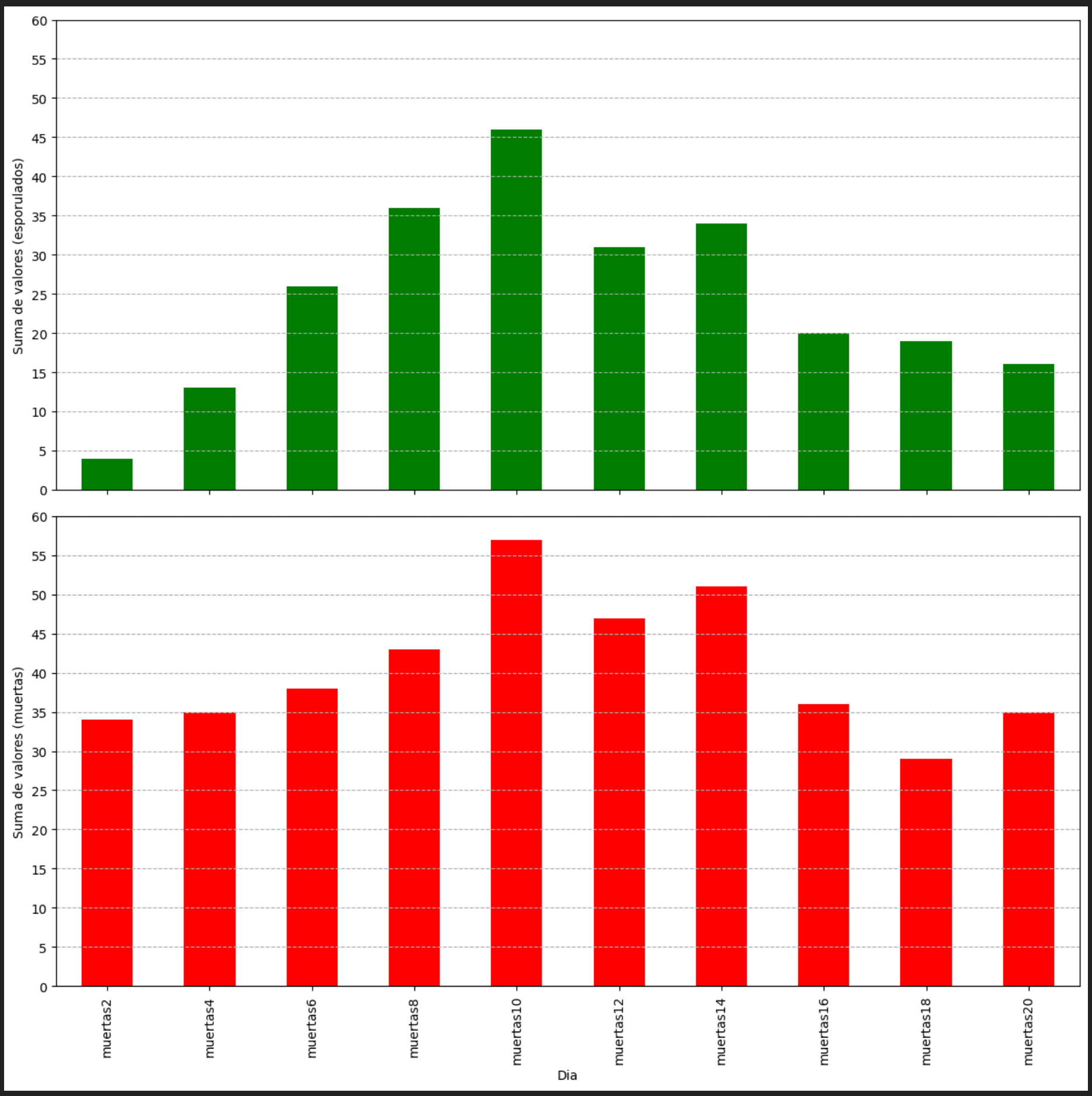
CTL	424	120	416	85	160	350	593	712	823	824
			a		a	a	a	a	a	a
		b		b			b	b	b	b
	c									
d										
d	c	b	a	b	a	a	ab	ab	ab	ab

Now, if we go deeper with the analysis, we will find this:

```python
#Filtro para exluir controles
df_filtered = df[df['Cepa'] != 'CTRL']
#Lista de columnas a sumar, y sumo
columns_to_sum = [col for col in df.columns if col.startswith('esporulados') or col.startswith('muertas')]
total_sum = df_filtered[columns_to_sum].sum()

#Graficos
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 12), sharex=True, gridspec_kw={'height_ratios': [1, 1]})
#Esporulados
total_sum[total_sum.index.str.startswith('esporulados')].plot(kind='bar', ax=ax1, color='green')
ax1.set_ylabel('Suma de valores (esporulados)')
ax1.set_ylim(0, 50)
ax1.set_yticks(range(0, 61, 5))
ax1.yaxis.grid(True, which='both', linestyle='--')
#Muertos
total_sum[total_sum.index.str.startswith('muertas')].plot(kind='bar', ax=ax2, color='red')
ax2.set_title('Suma de muertas')
ax2.set_xlabel('Dia')
ax2.set_ylabel('Suma de valores (muertas)')
ax2.set_ylim(0, 60)
ax2.set_yticks(range(0, 61, 5))
ax2.yaxis.grid(True, which='both', linestyle='--')

plt.tight_layout()
plt.show()
```



There is no need to run any analysis to see that mortality and sporulation (an irrefutable sign that mortality is linked to treatment) respond to two different distributions: while “sporulation” seems to respond to a normal distribution, mortality seems to be more linked to a constant.
In light of this, and according to the theory of survival analysis, the lifelines library gives us the possibility of “assigning different weights” to mortality at each time:

Weights & robust errors

Observations can come with weights, as well. These weights may be integer values representing some commonly occurring observation, or they may be float values representing some sampling weights (ex: inverse probability weights). In the `fit()` method, an kwarg is present for specifying which column in the DataFrame should be used as weights, ex: `CoxPHFitter(df, 'T', 'E', weights_col='weights')`.

When using sampling weights, it's correct to also change the standard error calculations. That is done by turning on the `robust` flag in `fit()`. Internally, `CoxPHFitter` will use the sandwich estimator to compute the errors.

but this is work for another day…