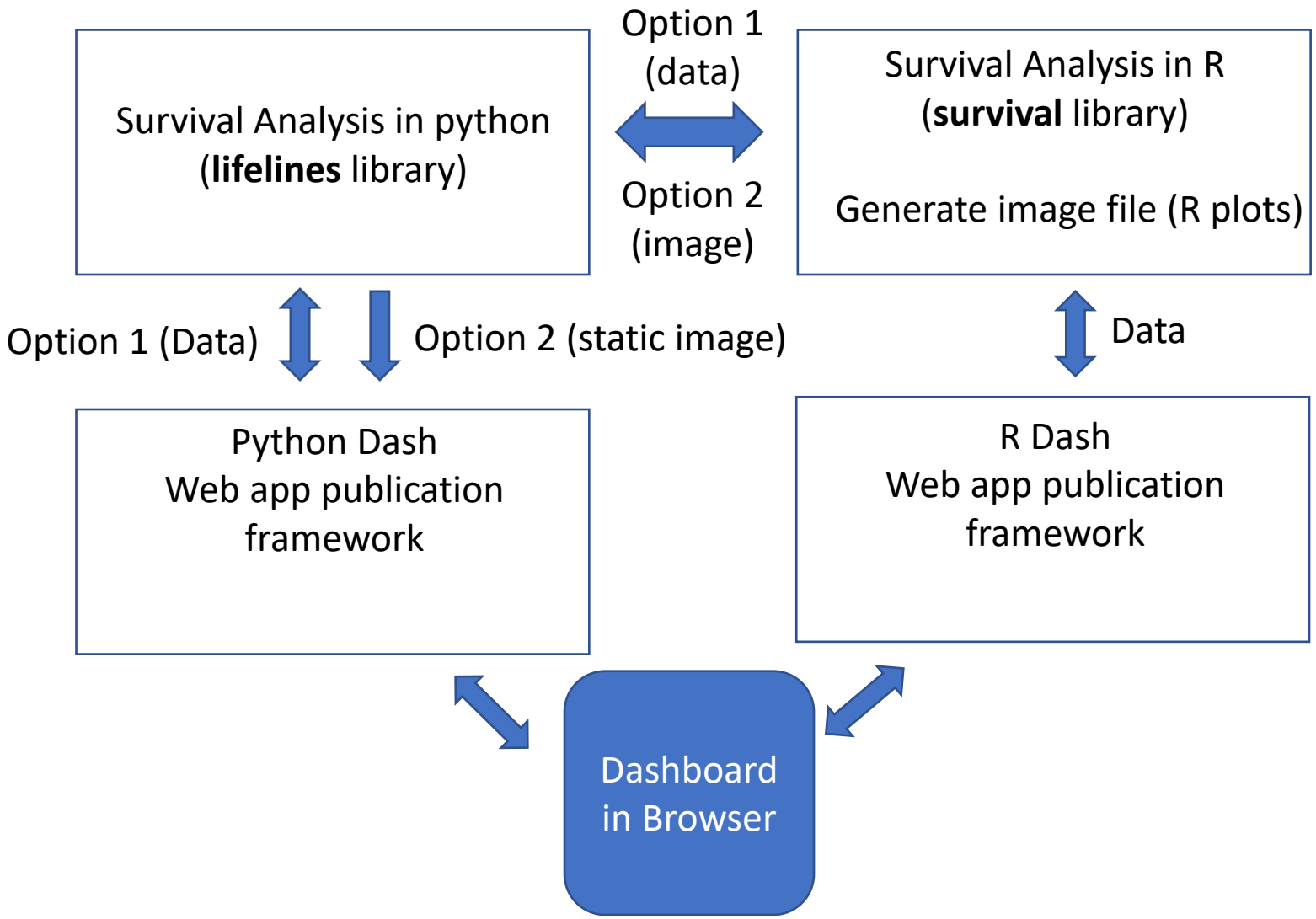
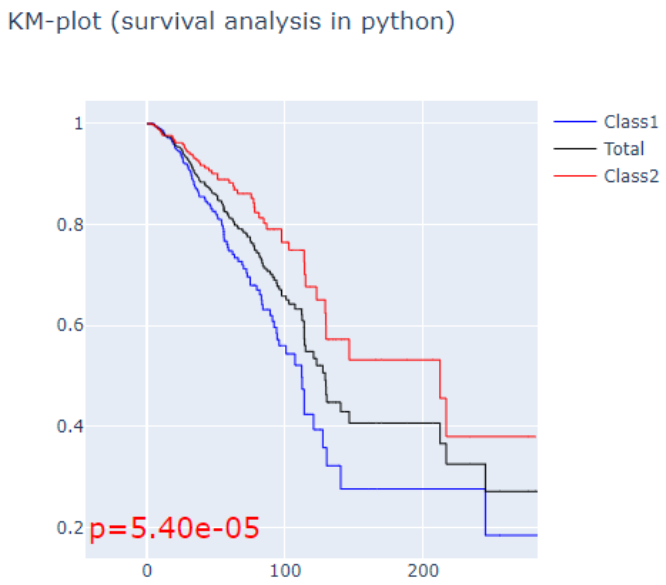
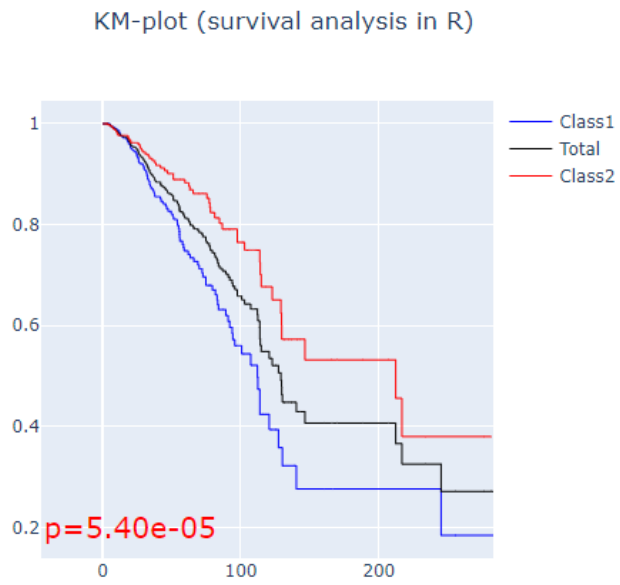
Survival Analysis in Python and R and display K-M plot in Dash

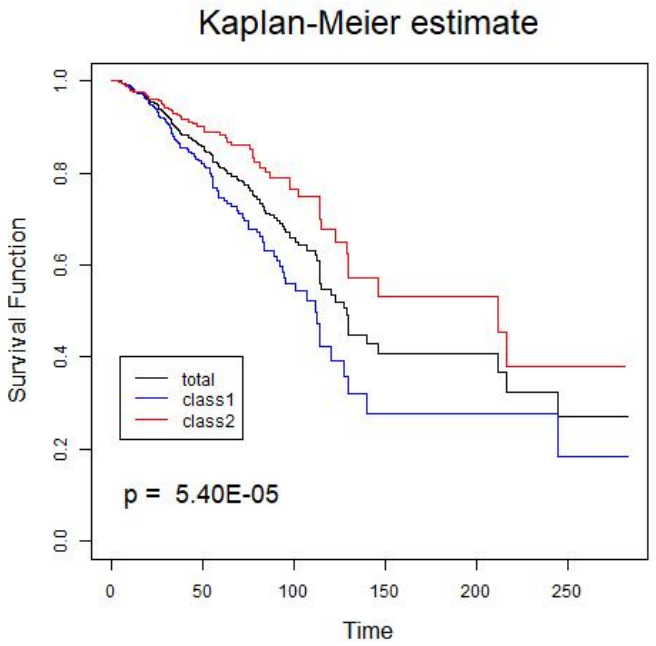




Survival Analysis in python
(**lifelines** library)
Displared in Dash using Plotly



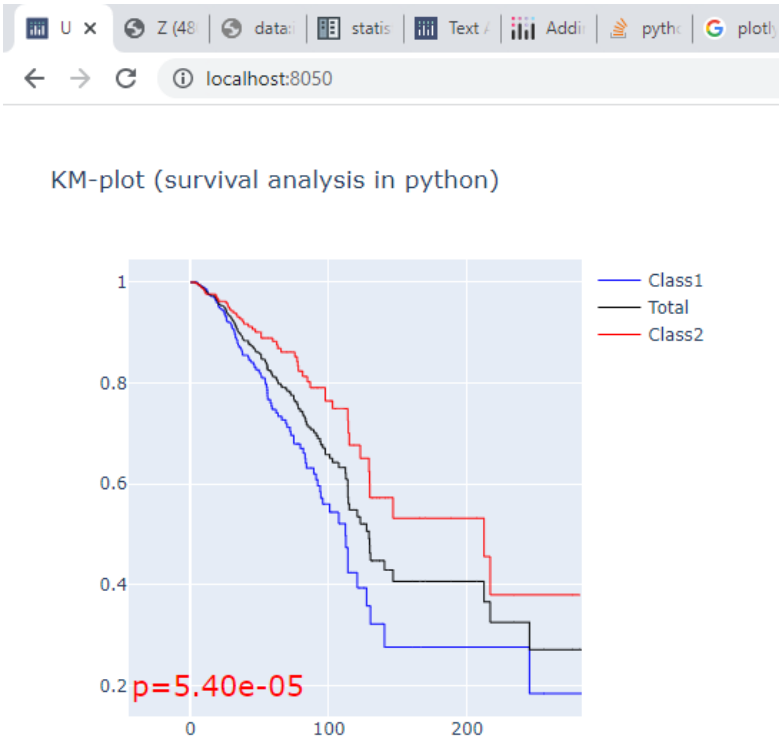
Survival Analysis in R
(**survival** library)
Displared in Dash using Plotly



Survival Analysis in R
(**survival** library)
Plot in R and save image as file. Image file is read by python and displayed in Dash

	A	B	C	D
1	Sample	Time	Status	Class
2	TCGA-E9-A1N3-01	34.82	0	class1
3	TCGA-AN-A04A-01	2.96	0	class1
4	TCGA-E9-A3X8-01	30.44	0	class2
5	TCGA-D8-A27N-01	17.06	0	class2
6	TCGA-A2-A0D3-01	61.58	0	class1
7	TCGA-B6-A0X4-01	28.27	1	class2
8	TCGA-GM-A2DM-01	106.06	0	class2

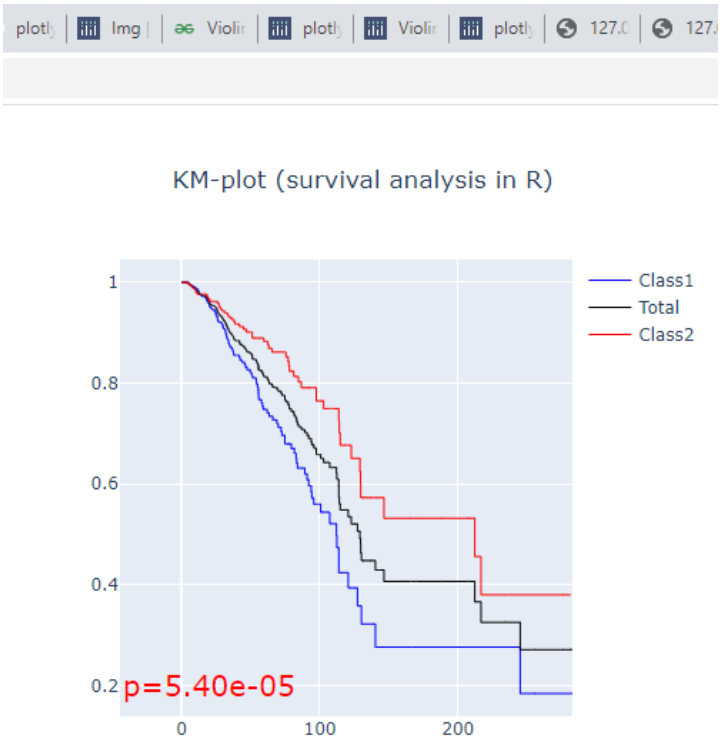
Sample Data



Survival Analysis in python
(lifelines library)
Plot in Python Plotly

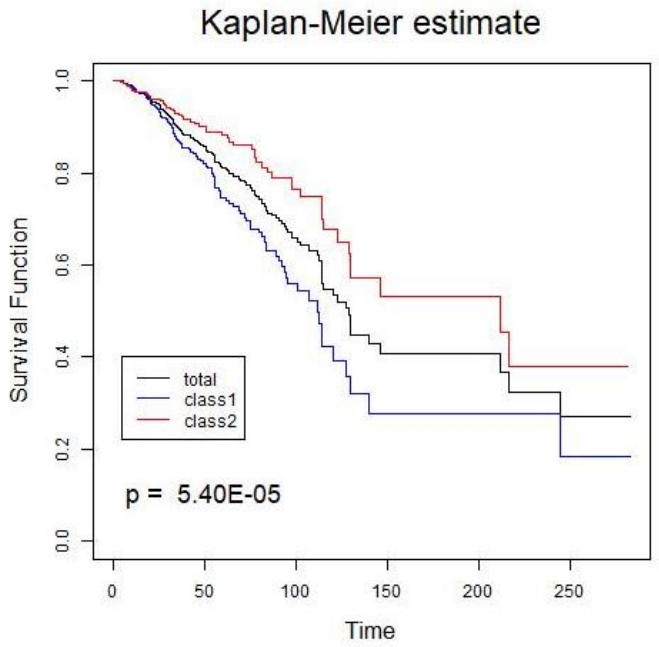
```
import dash
from dash import dcc, html
import plotly.graph_objects as go
import pandas as pd
import base64, random, subprocess, time
from lifelines import KaplanMeierFitter
from lifelines.statistics import logrank_test
```

```
def get_survival_plot():
    fig = go.Figure()
    i_survival_data = 'data/route_229.txt'
    df = pd.read_csv(i_survival_data, sep="\t", header=0)
    class1 = ((df.Class == 'class1'))
    class2 = ((df.Class == 'class2'))

    kmf = KaplanMeierFitter()

    kmf.fit(durations=df[class1].Time, event_observed=df[class1].Status,
           label='Class1')
    fig.add_trace(go.Scatter(x=kmf.survival_function_.index,
           y=kmf.survival_function_['Class1’], line=dict(shape='hv', width=1,
           color='rgb(0, 0, 255)’), showlegend=True, name='Class1'))

    kmf.fit(durations=df.Time, event_observed=df.Status, label='Total')
    fig.add_trace(go.Scatter(        x=kmf.survival_function_.index,
           y=kmf.survival_function_['Total’],         line=dict(shape='hv', width=1,
           color='rgb(0, 0, 0)’),         showlegend=True, name='Total’     ))

    kmf.fit(durations=df[class2].Time, event_observed=df[class2].Status,
           label='Class2')
    fig.add_trace(go.Scatter(        x=kmf.survival_function_.index,
           y=kmf.survival_function_['Class2’],         line=dict(shape='hv', width=1,
           color='rgb(255, 0, 0)'),        showlegend=True,  name='Class2'     ))

    output = logrank_test(durations_A=df[class1].Time, durations_B=df[class2].Time,
        event_observed_A=df[class1].Status, event_observed_B=df[class2].Status)
    p_value_scientific = "{:.2e}".format(output.p_value)

    fig.add_annotation(font=dict(color='red',size=20), x=20, y=0.2, xref="x", yref="y",
           text='p=' + p_value_scientific, xanchor='center', showarrow=False )

    fig.update_layout(height=500, width=500, title_text='KM-plot (survival analysis in
python)'    )

    return fig
```



Survival Analysis in R (**survival** library)
Displayed in Dash using Plotly

```
def get_survival_plotR():
    fig = go.Figure()
    i_survival_data = 'data\\route_229.txt'
    suffix = time.strftime("%Y%m%d%H%M%S") + "_" + str(random.randint(1, 100))
    o_survival_results = 'tmp\\' + suffix
    shell_cmd = ['C:\\Program Files\\R\\R-4.1.1\\bin\\Rscript.exe',
                 'survival_analysis.r', i_survival_data, o_survival_results]
    process = subprocess.call(shell_cmd, stdout=subprocess.PIPE)

    total_plot_data = o_survival_results + '_total'
    df = pd.read_csv(total_plot_data, sep="\t", header=0)

    total = ((df.Class == 'total'))
    class1 = ((df.Class == 'class1'))
    class2 = ((df.Class == 'class2'))

    fig.add_trace(go.Scatter(
        x=df[class1].time, y=df[class1].survival,
        line=dict(shape='hv', width=1, color='rgb(0, 0, 255)'),
        showlegend=True, name='Class1'
    ))

    pval_file = o_survival_results
    file = open(pval_file, mode='r')
    p_value = str(file.read().strip())
    file.close()

    p_value_scientific = "{:.2e}".format(float(p_value))
    fig.add_annotation(font=dict(color='red', size=20),
        x=20, y=0.2, xref="x", yref="y",
        text='p=' + p_value_scientific, xanchor='center', showarrow=False,
    )
    return fig
```



Survival Analysis in R (**survival** library)
Plot in R and image saved as file. Image file is read by python and displayed in Dash

```
def get_survival_plot_image_R():
    fig = go.Figure()

    i_survival_data = 'data\\route_229.txt'
    suffix = time.strftime("%Y%m%d%H%M%S") + "_" + str(random.randint(1, 100))
    # declare R output file location
    o_survival_results = 'tmp\\' + suffix
    # preparing Rscript command to run R program
    shell_cmd = ['C:\\Program Files\\R\\R-4.1.1\\bin\\Rscript.exe',
                 'survival_analysis.r', i_survival_data, o_survival_results]

    # executing shell command
    process = subprocess.call(shell_cmd, stdout=subprocess.PIPE)

    image_filename = o_survival_results + '_plot.jpg'
    encoded_image = base64.b64encode(open(image_filename, 'rb').read())

    return '{}'.format(encoded_image.decode())
```

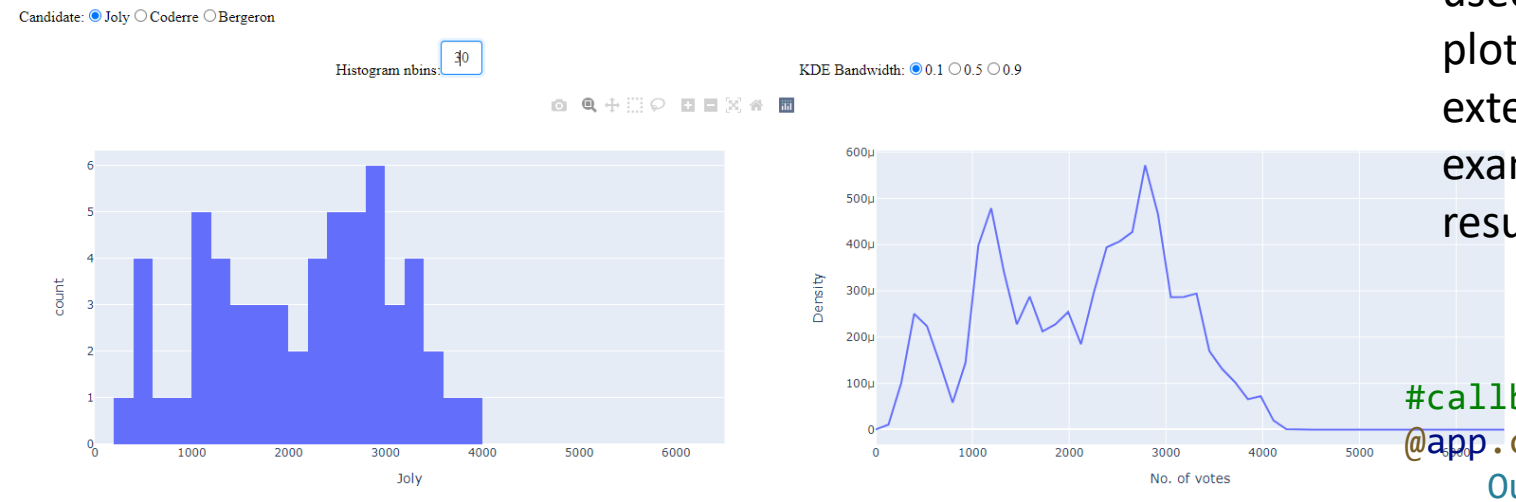
```
html.Img(
id="kmplotimageR", src=get_survival_plot_image_R(),
style={'height': '500', 'width': '500', 'display':
'inline-block'})
```

R script to run survival analysis

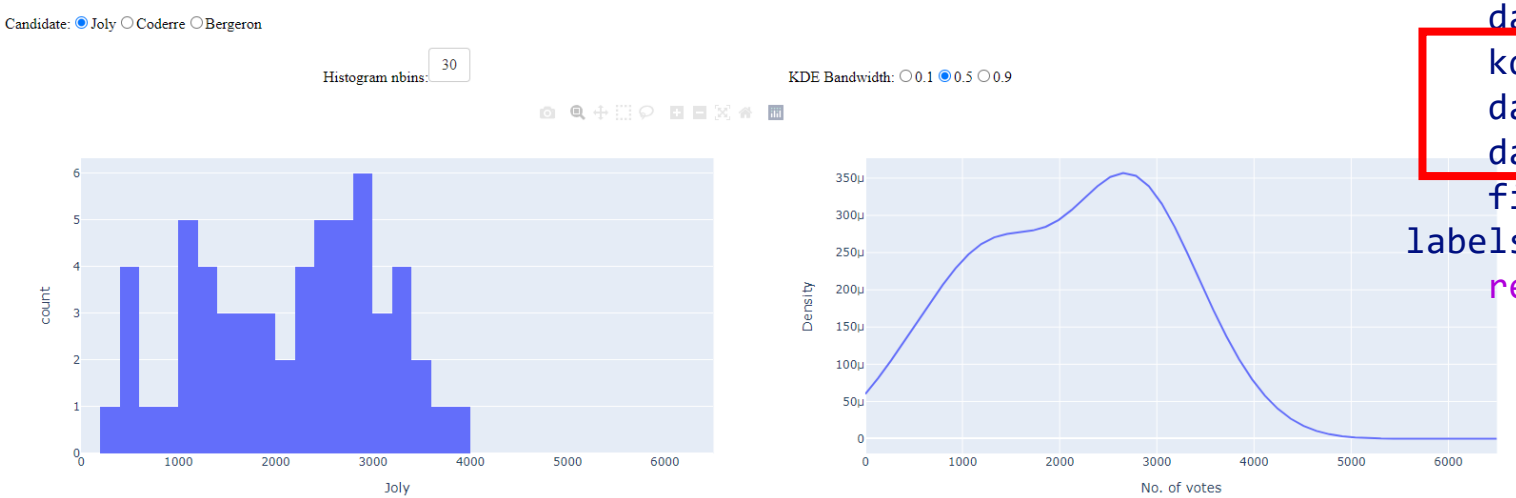
```
args <- commandArgs(trailingOnly = TRUE)
inputFile <- args[1]
outputFile <- args[2]
o_Total <- paste0(outputFile, "_total")
o_plot <- paste0(outputFile, "_plot.jpg")

library(survival)
data <- read.table(inputFile, header = TRUE)

status1 <- data[,3]  # 0:censored, 1: deceased
custom1 <- data[,4]  # 1:class1, 2: class2

class1 <- data[custom1=="class1",]
class2 <- data[custom1=="class2",]

time1 <- data[,2]
total_surv <- Surv(time1, status1)

time_class1 <- class1[,2]
status_class1 <- class1[,3]
class1_surv <- Surv(time_class1, status_class1)

time_class2 <- class2[,2]
status_class2 <- class2[,3]
class2_surv <- Surv(time_class2, status_class2)

fit_custom <- survdiff(total_surv ~ custom1)
p_custom <- 1-pchisq(fit_custom$chi,df=1)

total_fit <- survfit(total_surv ~ 1, conf.int=0)
class1_fit <- survfit(class1_surv ~ 1, conf.int=0)
class2_fit <- survfit(class2_surv ~ 1, conf.int=0)

jpeg(o_plot)
plot(total_fit, main="Kaplan-Meier estimate", xlab="Time", ylab="Survival Function", cex.main = 2, font.main = 1, cex.lab=1.4)
lines(class1_fit, col="blue")
lines(class2_fit,col="red")
legend(5, 0.4, c("total","class1","class2"), lty=c(1,1,1), col=c("black","blue","red"))
text(50, 0.1, paste("p = ",formatC(p_custom, format = "E", digits = 2)), cex=1.5)
dev.off()

total_data=cbind((total_fit$time), (total_fit$surv), (total_fit$n.censor), "total")
dimnames(total_data)[[2]]<-c("time","survival","#censor", "Class")
write.table(total_data, o_Total, quote=FALSE, row.names=FALSE, sep="\t")

class1_data=cbind((class1_fit$time), (class1_fit$surv), (class1_fit$n.censor), "class1")
#dimnames(class1_data)[[2]]<-c("time","survival","#censor", "Class")
write.table(class1_data, o_Total, quote=FALSE, row.names=FALSE, col.names = FALSE, sep="\t", append = TRUE)

class2_data=cbind((class2_fit$time), (class2_fit$surv), (class2_fit$n.censor), "class2")
#dimnames(class2_data)[[2]]<-c("time","survival","#censor")
write.table(class2_data, o_Total, quote=FALSE, row.names=FALSE, col.names = FALSE, sep="\t", append = TRUE)

sink(outputFile)
cat(format(p_custom,scientific=TRUE))
sink()
```

Common error in KDE plots using DASH (bandwidth parameter is unavailable in **plotly.figure_factory.create_distplot()**)

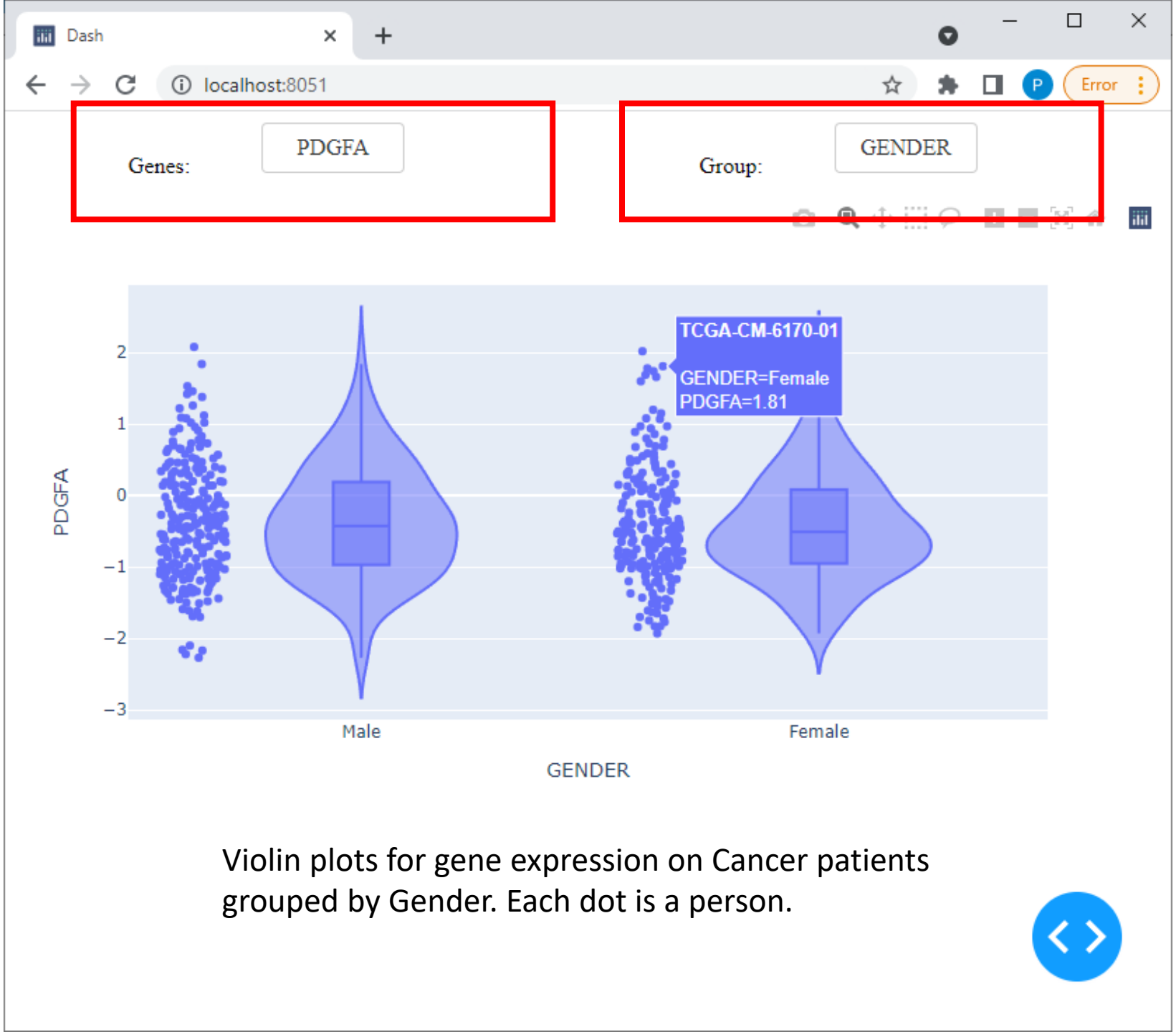
KDE computation doesn't need to be done using Plotly. We can use another library to compute the KDE and show the results using Plotly in DASH.



Just like in KM plot, different library (lifelines) was used to compute the survival values, which was plotted using Plotly Scatter plot. We can also use external programs (R program in previous example) to do the computation and get the results back into python.



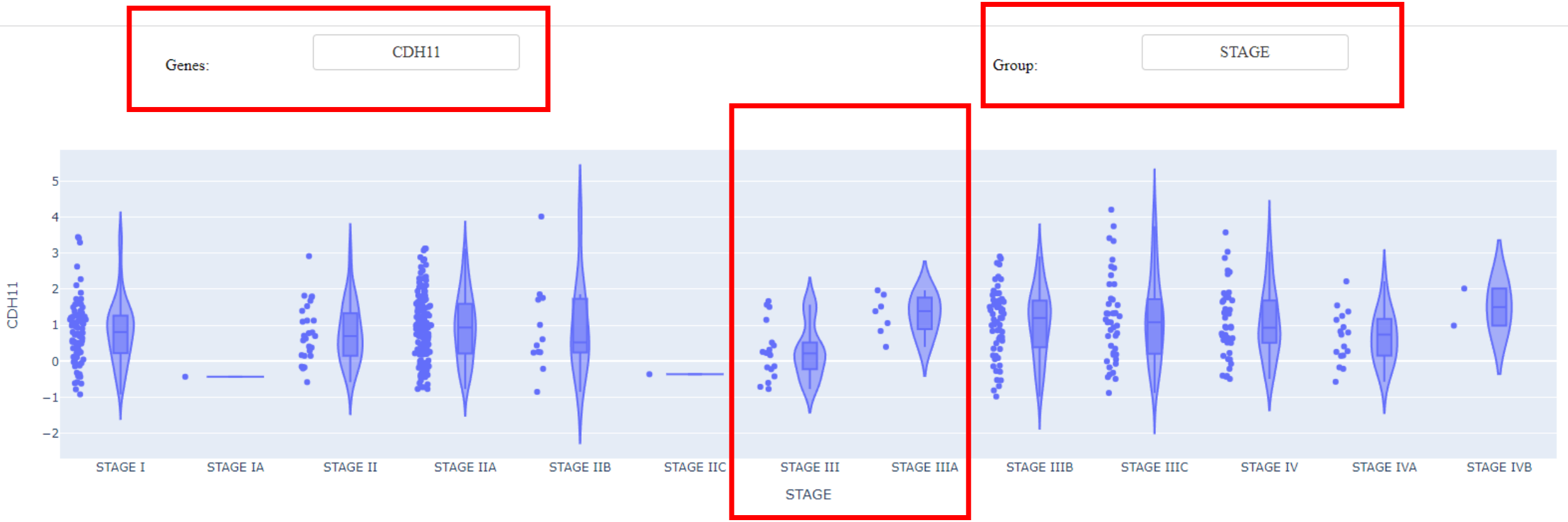

```
#callback for kde
@app.callback(
    Output("kdeplot", "figure"),
    [Input("candidate", "value"),
     Input("bw", "value")])
def generate_kde(candidate, bw):
    data = df[candidate]
    kde = gaussian_kde(data, bw_method=bw)
    data_x = np.linspace(0, 6500)
    data_y = kde.evaluate(data_x)
    fig = px.line(x = data_x, y = data_y,
labels={'x': 'No. of votes', 'y': 'Density'})
    return fig
```



Violin plots for gene expression on Cancer patients grouped by Gender. Each dot is a person.

```
@app.callback(
    Output("violin", "figure"),
    [Input("gene", "value"),
    Input("group", "value")])
def generate_violin(gene, group):
    global df
    fig = px.violin(df, y=gene
, x=group
, box=True
, hover_name='GENE'
, points="all")
    return fig
```

This makes individual dots visible, and we can hover over the dots to see their information.



Clear differences in distribution between Stage III and Stage IIIA