```
import pandas as pd
import pyodbc
from PIL import ImageTk, Image
import matplotlib.pyplot as plt
import tkinter as tk
import os.path
import shutil

conn = pyodbc.connect("DSN=lakartxela") # À changer si autre DSN
cursor = conn.cursor()

# Création d'une vue temporaire
try:
    cursor.execute("""CREATE VIEW `view_all`  AS  select
`MAccident`.`accident_id` AS `id`,`MAccident`.`gravite` AS `gravite de l
accident`,`MLieu`.`commune` AS `code postal`,`MLieu`.`x` AS `coordonné en
x`,`MLieu`.`y` AS `coordonné en y`,`MDate`.`DateFormatStandard` AS
`date`,`MDate`.`Commentaire` AS `commentaire de la
date`,`MCause`.`libelle` AS `libelle de la cause`,`MLuminosite`.`libelle`
AS `libelle de la luminosité`,`MIntemperie`.`libelle` AS `libelle de l
intemperie`,`MEtatSurface`.`libelle_etat_surface` AS `libelle de la
surface`,`MTypeEtatSurface`.`libelle_type_code_etat_surface` AS `libelle
du type de la surface`,`MImplique`.`libelle` AS `libelle de l
implication`,`MTypeImplication`.`libelleType` AS `libelle du type de l
implication`,`MAccident`.`nb_indemnes` AS `nombre de personnes
indemnes`,`MAccident`.`nb_blesses_legers` AS `nombre de blesses
legers`,`MAccident`.`nb_blesses_graves` AS `nombre de blesses
graves`,`MAccident`.`nb_morts` AS `nombre de morts` from
(((((((((`MAccident` join `MDate` on((`MDate`.`date_id` =
`MAccident`.`date_id`))) join `MLieu` on((`MLieu`.`lieu_id` =
`MAccident`.`lieu_id`))) join `MCause` on((`MCause`.`Cause` =
`MAccident`.`cause_id`))) join `MLuminosite` on((`MLuminosite`.`code` =
`MAccident`.`lum_id`))) join `MIntemperie` on((`MIntemperie`.`code` =
`MAccident`.`intemp_id`))) join `MEtatSurface`
on((`MEtatSurface`.`code_etat_surface` = `MAccident`.`etat_surface_id`)))
join `MTypeEtatSurface` on((`MTypeEtatSurface`.`id_etat_surface` =
`MEtatSurface`.`type_code_etat_surface`))) join `MImplique`
on((`MImplique`.`code` = `MAccident`.`impliq_id`))) join
`MTypeImplication` on((`MTypeImplication`.`id` =
`MImplique`.`type_code_implique`))) ; """)
except:
    pass

# Chemin permettant de stocker temporairement les images
if os.path.exists('temp/')==False:
    os.mkdir('temp/')
for f in os.listdir('temp/'):
    os.remove(os.path.join('temp/', f))

fenetre = tk.Tk()
fenetre.title("Tableau de bord : Jours particuliers")
fenetre.geometry("1920x1080+0+0")
fenetre.state("zoomed")

def destroy_all():
    for widget in fenetre.winfo_children():
        widget.destroy()
```

```
def boutonRetour():
    PBretour = tk.Button(fenetre,text="Retour",command=
lambda:creerPremierAppel())
    PBretour.place(x=124,y=46)

def afficherPeriode(deb,fin, color, ymin, ymax, alpha=0.8):
    plt.hlines(y=ymin+(ymax-ymin)/2, xmin=deb, xmax=fin, colors=color,
lw=18, label='vline_single - partial height', alpha=alpha)

def DeltaAll():
    plt.clf() # Pour vider le plot actuel sans le supprimer (Ø
superposition)
    cursor.execute(f"""SELECT COUNT(*)/5202 FROM view_all""")

    for row in cursor.fetchall():
        moyenne = row[0]

    cursor.execute(f"""SELECT COUNT(*)/14 FROM `view_all` GROUP BY
MONTH(`view_all`.`date`),DAY(`view_all`.`date`)""")

    listabs=[]
    for row in cursor.fetchall():
        listabs.append(float(row[0]-moyenne))

    dataframe = pd.DataFrame({'Value':listabs})
    axis = dataframe.plot(legend=False)
    plt.title("Delta du nombre d'accident par jour et la moyenne\ndu
nombre d'accident sur les 14 années")
    plt.xlabel("Jours")
    plt.ylabel("Delta du nombre d'accident")

    plt.xticks(rotation = 45)

    # Affichage des blocs de saisons
    afficherPeriode(0,78,"#FF00FF",-5,-4)    #Hiver début
    afficherPeriode(79,171,"#FFFF00",-5,-4)  #Printemps
    afficherPeriode(172,265,"#FF0000",-5,-4) #Été
    afficherPeriode(266,354,"#00FFFF",-5,-4) #Automne
    afficherPeriode(355,366,"#FF00FF",-5,-4) #Hiver fin

    plt.vlines(x=209, ymin=-4, ymax=5, colors='#88DD88', lw=76,
label='vline_single - partial height', alpha=0.2) # Delta : y = 0

    # Quadrillage
    for i in range(-4,6,2):
        plt.hlines(y=i, xmin=-5, xmax=365, colors='#DBDBDB', lw=1,
label='vline_single - partial height')
    plt.hlines(y=0, xmin=-5, xmax=365, colors='red', lw=1,
label='vline_single - partial height')
    for i in range (12):
        plt.vlines(x=i*30.5, ymin=-5, ymax=5, colors='#DBDBDB', lw=0.7,
label='vline_single - partial height')

    plt.text(188, 4, 'Été', fontsize = 18) # Vacances d'été : moins
d'accidents
    plt.savefig('temp/choix2.jpg')

def joursFeries(param):
```

```
    plt.clf() # Pour vider le plot actuel sans le supprimer (Ø
superposition)
    frame=[]
    for date in param:
        if date == "Normal":
            commande=f"""SELECT count(*)/(365-11)as nb ,hour(date) as
heure from `view_all` where `commentaire de la date` is null group by
hour(date)"""
            sql_query = pd.read_sql_query (commande,conn)
            frame.append(pd.DataFrame(sql_query))
        else:
            commande=f"""SELECT hour(date) as heure, COUNT(*)as nb from
`view_all` where `commentaire de la date` = "Jour férié : {date}" GROUP
BY HOUR(date)"""
            sql_query = pd.read_sql_query (commande,conn)
            frame.append(pd.DataFrame(sql_query))
    i=0
    for df in frame:
        plt.plot(df['heure'], df['nb'],label=param[i])
        i=i+1
    plt.xticks([x for x in range(0,24)])
    for i in range(24):
        plt.vlines(x=i, ymin=0, ymax=9, colors='#DBDBDB', lw=0.7)
    for i in range(0,9,2):
        plt.hlines(y=i, xmin=0, xmax=24, colors='#DBDBDB', lw=0.7)
    plt.legend(bbox_to_anchor=(1.012, 1.15))
    plt.tight_layout()
    plt.savefig('temp/choix4.jpg')

def validation(JF,CKListe,dictBoutonDK):
    plt.clf() # Pour vider le plot actuel sans le supprimer (Ø
superposition)
    destroy_all()
    PBretour = tk.Button(fenetre,text="Retour",command=
lambda:creerPremierAppel())
    PBretour.place(x=0,y=0)
    for JFNom in CKListe:
        if dictBoutonDK[JFNom][0].get() == True:
            JF.append(JFNom)
    joursFeries(JF)
    image1 = Image.open('temp/choix4.jpg')
    test = ImageTk.PhotoImage(image1)

    label1 = tk.Label(image=test)
    label1.image = test
    label1.place(x=0,y=26)



def optionCKList(CKListe):
    JF = []
    destroy_all()
    boutonRetour()
    dictBoutonDK = {}
    for JFNom in CKListe:
        dictBoutonDK[JFNom] = []
        dictBoutonDK[JFNom].append(tk.BooleanVar())
```

```
        dictBoutonDK[JFNom].append(tk.Checkbutton(fenetre, text=JFNom,
variable=dictBoutonDK[JFNom][0]))
        dictBoutonDK[JFNom][1].place(x=50,y=len(dictBoutonDK)*30 + 70)
    boutonValider = tk.Button(fenetre,text="Valider", command=
lambda:validation(JF,CKListe,dictBoutonDK))
    boutonValider.place(x=0,y=50)

def anneeSimple(annee):
    plt.clf() # Pour vider le plot actuel sans le supprimer (Ø
superposition)
    cursor.execute(f"""SELECT COUNT(*) FROM `view_all` WHERE
YEAR(`view_all`.`date`)={annee} GROUP BY MONTH(`view_all`.`date`)""")

    listabs=[]
    for row in cursor.fetchall():
        listabs.append(int(row[0]))

    dataframe = pd.DataFrame({'Value':listabs}, index=['Janvier',
'Février', 'Mars', 'Avril', 'Mai', 'Juin', 'Juillet', 'Août',
'Septembre', 'Octobre','Novembre', 'Décembre'])
    axis = dataframe.plot(legend=False)
    plt.title(f"Nombre d'accidents en fonction du mois dans l'année
{annee}")
    plt.xticks(rotation=45)

    for i in range (12):
        plt.vlines(x=i, ymin=0, ymax=max(listabs), colors='#DBDBDB',
lw=0.7, label='vline_single - partial height')

    plt.xlabel("Mois")
    plt.ylabel("Nombre d'accidents")
    plt.savefig('temp/choix1.jpg')

def choix1validation():
    destroy_all()
    boutonRetour()
    STRINGannee = tk.StringVar()
    Lannee = tk.Label(text="Veuillez saisir l'année :")
    Lannee.place(x=0,y=0)
    SELannee = tk.Entry(fenetre, textvariable=STRINGannee)
    SELannee.place(x=0,y=20)
    PBvalidation = tk.Button(fenetre,text="Valider",height=1,command=
lambda:choix1generation(STRINGannee.get()))
    PBvalidation.place(x=124,y=20)

def choix1generation(STRINGannee):
    destroy_all()
    PBretour = tk.Button(fenetre,text="Retour",command=
lambda:creerPremierAppel())
    PBretour.place(x=0,y=0)
    anneeSimple(STRINGannee)
    image1 = Image.open('temp/choix1.jpg')
    test = ImageTk.PhotoImage(image1)
    label1 = tk.Label(image=test)
    label1.image = test
    label1.place(x=0,y=26)

def choix2generation():
```

```
    destroy_all()
    PBretour = tk.Button(fenetre,text="Retour",command=
lambda:creerPremierAppel())
    PBretour.place(x=0,y=0)
    # Pour ne pas générer le même graphique plusieurs fois
    if os.path.exists('temp/choix2.jpg')==False:
        DeltaAll()

    image1 = Image.open('temp/choix2.jpg')
    test = ImageTk.PhotoImage(image1)

    label1 = tk.Label(image=test)
    label1.image = test

    label1.place(x=0,y=26)

def feriesVSnormal():
    plt.clf() # Pour vider le plot actuel sans le supprimer (Ø
superposition)
    commande=f'''SELECT count(*)/11 as nb,hour(date) as heure from
`view_all` where `commentaire de la date` is not null group by
hour(date)'''
    sql_query = pd.read_sql_query (commande,conn)
    frame=[]
    frame.append(pd.DataFrame(sql_query))
    commande=f"""SELECT count(*)/(365-11)as nb ,hour(date) as heure from
`view_all` where `commentaire de la date` is null group by hour(date)"""
    sql_query = pd.read_sql_query (commande,conn)
    frame.append(pd.DataFrame(sql_query))
    label=('Normal','Ferié')
    i=0
    for df in frame:
        plt.plot(df['heure'], df['nb'],label=label[i])
        i=i+1
    plt.xticks([x for x in range(0,24)])
    for i in range(24):
        plt.vlines(x=i, ymin=0, ymax=9, colors='#DBDBDB', lw=0.7)
    for i in range(0,9,2):
        plt.hlines(y=i, xmin=0, xmax=24, colors='#DBDBDB', lw=0.7)
    plt.legend(bbox_to_anchor=(1.012, 1.15))
    plt.savefig('temp/choix3.jpg')

def choix3generation():
    destroy_all()
    PBretour = tk.Button(fenetre,text="Retour",command=
lambda:creerPremierAppel())
    PBretour.place(x=0,y=0)

    # Pour ne pas générer le même graphique plusieurs fois
    if os.path.exists('temp/choix3.jpg')==False:
        feriesVSnormal()

    image1 = Image.open('temp/choix3.jpg')
    test = ImageTk.PhotoImage(image1)

    label1 = tk.Label(image=test)
    label1.image = test
```

```
    label1.place(x=0,y=26)

def choix4generation():
    optionCKList(["Normal","Armistice","Ascension","Assomption","Fête du
travail", "Fête Nationale","Jour de l'an","Lundi de Pâques", "Lundi de
Pantecôte","Noël","Toussaint","Victoire des alliés"])

def premierAppel(bouton):
    if bouton == "choix1":
        choix1validation()
    if bouton == "choix2":
        choix2generation()
    if bouton == "choix3":
        choix3generation()
    if bouton == "choix4":
        choix4generation()

def creerPremierAppel():
    destroy_all()
    Lselection = tk.Label(text="Veuillez sélectionner le type de
graphique que vous souhaitez afficher :")
    Lselection.place(x=0,y=0)
    choix1 = tk.Button(fenetre,text="L'évolution du nombre d'accidents
par mois pour une année en particulier",height=1,command=
lambda:premierAppel("choix1"))
    choix1.place(x=0,y=20)
    choix2 = tk.Button(fenetre,text="L'évolution du nombre d'accidents
sur une année en compilant les 14 années",height=1,command=
lambda:premierAppel("choix2"))
    choix2.place(x=0,y=46)
    choix3 = tk.Button(fenetre,text="Nombre d'accidents moyen en fonction
de l'heure de la journée et du type de jour",height=1,command=
lambda:premierAppel("choix3"))
    choix3.place(x=0,y=72)
    choix4 = tk.Button(fenetre,text="Affichage du nombre d'accident moyen
par jour en fonction du jour férié",height=1,command=
lambda:premierAppel("choix4"))
    choix4.place(x=0,y=98)

creerPremierAppel()

fenetre.mainloop()

cursor.execute("""DROP VIEW IF EXISTS `view_all`;""")
shutil.rmtree('temp/')
```