



المدرسة الوطنية للعلوم التطبيقية بتطوان  
ⵜⴰⵎⴰⵔⵜ ⵜⴰⵏⵓⵔⴰⵢⵜ ⵜⴰⵎⴰⵔⴰⵢⵜ ⵜⴰⵏⵓⵔⴰⵢⵜ  
ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES  
DE TÉTOUAN

## Guide détaillée : World Happiness Dashboard

REALISE PAR :

NOM/PRÉNOM	EMAIL INSTITUTIONNEL
KHOLTI HAMZA	HAMZA.KHOLTI@ETU.UAE.AC.MA
BOUZHAR ANOUAR	BOUZHAR.ANOUAR@ETU.UAE.AC.MA
SALIM AYA	SALIM.AYA@ETU.UAE.AC.MA
ELHAOUDAR AYA	ELHAOUDAR.AYA @ETU.UAE.AC.MA

## **I. Data pretreatment:**

### *1. Importation des bibliothèques:*

```
import streamlit as st
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.subplots as ps
from streamlit_extras.metric_cards import style_metric_cards
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import plotly.figure_factory as ff
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import KNNImputer
```

- streamlit : pour la création de l'application web
- Pandas et numpy : pour la manipulation , l'analyse des données et les calculs numériques
- plotly.express , plotly.graph\_objects, plotly.figure\_factory, plotly.subplots :pour la création de visualisations interactives respectivement pour la création de sous-graphiques avec Plotly.....
- streamlit\_extras.metric\_cards : pour l'ajout de cartes de métriques personnalisées dans Streamlit
- sklearn.preprocessing.MinMaxScaler : pour la mise à l'échelle les données de grande valeur , normalisation/ standarisatation pour les présenter facilement s'il est nécessaire
- sklearn.impute.KNNImputer : pou le remplissage des données manquantes en utilisant l'algorithme k-NN(K-Nearest Neighbor)

### *2. l'importation et le traitement des données:*

```
# importation de donnees
df=["df"+str(i) for i in range(5,24)]
filepath= [r"C:\Users\dell\Desktop\BDIA1\Visualisation des données\v.par
python\data\{}.csv".format(year) for year in range(2005, 2024)]
j=0
for file,d in zip(filepath,df) :
    d=pd.read_csv(file)
    j+=1
    if j==1:
        data=d
    else:
        data=pd.concat([data,d.iloc[0:,:]],axis=0)
```

- Notre data est sous forme d'un dossier contient des fichiers csv représentant les données de « world happiness » depuis 2005 jusqu'à 2023, alors pour optimiser le code et ne fait pas le travaille deux fois nous avons utilisé une boucle qui va importer toutes les données et par la suite nous allons faire une concaténation pour avoir une seul DataFrame .....

Note : n'oubliez pas de changer la direction vers les données dans votre code

Soit par- print(data) ou -data en jupyter vous pouvez voir une partie de votre données comme la suite :

[4]: data

	Country	Year	Happiness Score	GDP per Capita	Social Support	Life Expectancy	Freedom	Generosity	Corruption
0	Italy	2005	6.853784	10.697855	0.928001	70.599998	0.802195	NaN	0.943912
1	Germany	2005	6.619550	10.690792	0.963490	69.900002	0.846624	NaN	0.781007
2	Lebanon	2005	5.491245	9.572591	0.796278	65.099998	0.703206	NaN	0.945177
3	France	2005	7.093393	10.636769	0.940338	70.699997	0.894819	NaN	0.687851
4	Saudi Arabia	2005	7.079644	10.674833	0.867819	61.200001	NaN	NaN	0.505149
...	...	...	...	...	...	...	...	...	...
132	Israel	2023	7.472900	10.638705	0.943344	72.697205	0.808866	-0.023080	0.708094
133	Ireland	2023	6.910800	11.527362	0.905368	71.299957	0.874381	0.091660	0.357952
134	Iraq	2023	4.940900	9.098433	0.718453	63.414734	0.645771	-0.005400	0.876264
135	Iran	2023	4.876300	9.610323	0.777547	66.599945	0.592562	0.172933	0.747031
136	Zimbabwe	2023	3.203500	7.640998	0.689918	54.049889	0.654055	-0.046230	0.765582

2336 rows × 9 columns

- Gestions des valeurs manquantes :

```
print(data.isnull().sum())
```

```
data.isnull().sum()
```

```
Country      0
Year          0
Happiness Score  0
GDP per Capita  20
Social Support  13
Life Expectancy  55
Freedom       33
Generosity    73
Corruption    116
dtype: int64
```

à ce moment nous avons une idée sur les valeurs manquantes pour chaque variable, la première idée c'est d'ajouter une colonne qui représente le continent cela aide beaucoup par la suite :

- L'ajout d'une colonne continent :  
Dans la liste countries les éléments sont déjà des listes contenant les noms des états, par la suite nous détectons quelques nations ne sont pas des états mais des régions, donc on va éliminer ces nations car ils ont beaucoup de valeurs manquantes de plus l'étude sur des pays n'est pas des régions :

```
continents= ['Africa', 'Asia', 'Europe', 'North_America', 'South_America','Oceania']
countries=[Africa,Asia,Europe,North_America,South_America,Oceania]
conditions = [data['Country'].isin(country_list) for country_list in countries]
data['Continent'] = np.select(conditions, continents,default=None)
```

```
print(data[data['Continent'].isna()]['Country'].unique())
```

```
array(['Hong Kong S.A.R. of China', 'Israel', 'Taiwan Province of China',
      'Congo (Brazzaville)', 'Congo (Kinshasa)', 'Somaliland region'],
      dtype=object)
```

- La suppression de cette « Country » sauf les deux Congo :

Résultat : éliminations environ 62 Valeur null

```
data.reset_index(drop=True, inplace=True)
data=data.loc[(data["Country"] != "Hong Kong S.A.R. of China")]
data=data.loc[(data["Country"] != "Taiwan Province of China")]
data=data.loc[(data["Country"] != "Kosovo")]
data=data.loc[(data["Country"] != "Somaliland region")]
data=data.loc[(data["Country"] != "Israel")]#free palestine
```

Note : y'a des autres pays qui n'ont pas reçu un continent car ne sont pas dans la liste des « Countries », tout simplement nous les ajoutons à la liste par contre Congo on va réunir les deux Congo vue qu'ils ont la même situation (presque même valeurs des variable)

```
df = data[data['Country'].isin(['Congo (Brazzaville)', 'Congo (Kinshasa)'])].reset_index(drop = True)
df = df.drop(['Continent', 'Country'], axis = 1)
congo_combined = df.groupby(['Year']).mean().reset_index()
congo_combined['Continent'] = 'Africa'
congo_combined['Country'] = "R.D.Congo"
congo_combined = congo_combined[['Country', 'Year', 'Happiness Score', 'GDP per Capita', 'Social Support', 'Life Expectancy', 'Freedom', 'Generosity', 'Corruption', 'Continent']]
data.drop(data[data['Country'].isin(['Congo (Brazzaville)', 'Congo (Kinshasa)'])].index, inplace=True)
data = pd.concat([data.reset_index(drop=True), congo_combined.reset_index(drop=True)], axis=0)
data = data.drop(data[(data["Country"] == "R.D.Congo") & (data["Year"].isin([2008, 2009]))].index)
```

- Remplissage des valeur manquant (par la moyenne) :

```
l=data.loc[data["Social Support"].isnull(),"Country"]
l=list(l)
j=0
for index,row in data.iterrows():
    if pd.isnull(row["Social Support"]):
        data.at[index,"Social Support"]=data.loc[data["Country"]==l[j],"Social Support"].mean()
        j+=1
```

Remplir les valeur manquants de social support par la moyenne au fil des années vue que ce dernière ne dépend pas par les années pour chaque pays (l'objectif c'est d'utiliser cette colonne dans un modèle de machine learning pour prédire les valeurs manquant d'une autre colonnes ,vous trouvez par la suite le model mais à la fin du compte nous n'avons pas l'intégrer dans le code vu que le temp nécessaire pour l'entrainement de model -12 heures - )

Note : cette partie reste optionnelle, nous allons gérer les valeurs manquantes par une méthode plus rapide et efficace

```
251]: # in fact this mission not easy so we will use a model of machine Learning to predict the missing value
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV
data_na=data.dropna()
features=["Year","Happiness Score","Social Support","Life Expectancy","Freedom","Corruption"]
target=["GDP per Capita"]
x=data_na[features]
y=data_na[target]
model = KNeighborsRegressor()
param_grid = {
    'n_neighbors': [3, 5, 7],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}
# Perform grid search
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(x,y)
best_model = grid_search.best_estimator_
best_model.score(x,y)

251]: 1.0

252]: #alors nous avons obtenir une bonne precision de tel sort que les données manquantes similaire avec celle de données de l'entrainement
x_f=data[data["GDP per Capita"].isnull()].drop(columns=["Country","GDP per Capita","Generosity"])
y_p=best_model.predict(x_f)
y_prediction=[]
for i in range(len(y_p)):
    y_prediction.append(y_p[i,0])
```

## Deuxième model de machine learning

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
features=["Year","Happiness Score","GDP per Capita","Social Support","Life Expectancy"]
target=["Freedom"]
x,y=data_na2[features],data_na2[target]
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
rf_regressor = RandomForestRegressor(random_state=42)
param_grid = {
    'n_estimators': [i for i in range(50,301,10)],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
grid_search = GridSearchCV(rf_regressor, param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(x, y)
best_rf_model = grid_search.best_estimator_
best_rf_model.score(x_train,y_train)

0.947421314556803

#on va chercher les valeurs des freedom
X=data.loc[data["Freedom"].isnull()].drop(columns=["Country","Freedom","Generosity","Corruption"])
Y=best_rf_model.predict(X)
data.loc[data["Freedom"].isnull(),"Freedom"]=Y
```

Country	Year	Happiness Score	GDP per Capita	Social Support	Life Expectancy	Freedom	Generosity	Corruption
---------	------	-----------------	----------------	----------------	-----------------	---------	------------	------------

- Remplissage des valeur manquant (par K-NN imputer) :

```
selected_data = data.drop(['Country', 'Continent'], axis = 1)
imputer = KNNImputer(n_neighbors = 2)
imputed_data = imputer.fit_transform(selected_data)
to_dataframe = pd.DataFrame(imputed_data, columns = ['Year', 'Happiness Score', 'GDP
per Capita',
    'Social Support', 'Life Expectancy', 'Freedom', 'Generosity',
    'Corruption'])
data = pd.concat([to_dataframe.reset_index(drop=True), data[['Country',
'Continent']]].reset_index(drop=True)], axis=1)
col_oredre = ['Country', 'Year', 'Happiness Score', 'GDP per Capita',
    'Social Support', 'Life Expectancy', 'Freedom', 'Generosity',
    'Corruption', 'Continent']
data = data[col_oredre]
data["Year"] = data["Year"].astype(int)
```

K-NN ce n'est pas un modèle de machine learning en fait mais il'est un peu proche, il gère des données à partir des points plus proche (plus proche au niveau des coordonnées n'est pas dans la carte)

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

```
[536]: Country      0
      Year        0
      Happiness Score  0
      GDP per Capita  0
      Social Support  0
      Life Expectancy  0
      Freedom       0
      Generosity     0
      Corruption     0
      Continent     0
      Rank          0
      dtype: int64
```

Résultat : les données prétraité avec aucun valeurs manquant, en fin du compte nous exécutons data.describe() pour détecter les anomaly ,pour les corrigé d'une part et les comprendre d'une autre part

```
[538]: data.describe()
```

```
[538]:
```

	Year	Happiness Score	GDP per Capita	Social Support	Life Expectancy	Freedom	Generosity	Corruption
count	2255.000000	2255.000000	2255.000000	2255.000000	2255.000000	2255.000000	2255.000000	2255.000000
mean	2014.693126	5.466047	9.375683	0.809406	63.398762	0.750706	0.000087	0.745949
std	5.023266	1.125280	1.150382	0.122276	6.812232	0.139023	0.159604	0.181687
min	2005.000000	1.281271	5.526723	0.228217	6.720000	0.257534	-0.337527	0.035198
25%	2011.000000	4.634562	8.483966	0.742571	59.344999	0.659308	-0.108327	0.691169
50%	2015.000000	5.420331	9.494558	0.835509	65.120003	0.771577	-0.022595	0.798842
75%	2019.000000	6.286840	10.341864	0.905332	68.532501	0.861601	0.091004	0.865855
max	2023.000000	8.018934	11.663788	0.987343	74.474998	0.985178	0.702708	0.983276

## II. Web applications (page configurations+sidebare ):

### 1. Page configurations :

- Etablir la page par la commande suivant :

```
st.set_page_config(
    page_title='Happiness',
    page_icon='😊',
    layout='wide',
    initial_sidebar_state='expanded'
)
st.subheader("Happiness in The World ")
st.markdown(
    """
    <style>
    body {
        color: #333333; /* Remplacez #333333 par la couleur du texte de votre choix */
    }
    </style>
    """,
    unsafe_allow_html=True
)
```

Le dernier code est un code CSS pour les colleurs et la décoration de page

Démarrer le code et par la suite dans le terminal taper la commande :

Streamlit run.\script.py server.port {une port de votre choix plus souvent 8080}

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

PS C:\Users\dell\Desktop\BDIA1\Visualisation des données\v.par python\travail> & 'c:\Users\dell\AppData\Local\Programs\Python\Python311\python.exe' 'c:\
ode\extensions\ms-python.debugpy-2024.0.0-win32-x64\bundle\libs\debugpy\adapter\..\..\debugpy\launcher' '60594' '--' 'c:\Users\dell\Desktop\BDIA1\Visu
nées\v.par python\travail\bi_v4.py'
2024-02-28 00:36:19.108
Warning: to view this Streamlit app on a browser, run it with the following
command:

streamlit run c:\Users\dell\Desktop\BDIA1\Visualisation des données\v.par python\travail\bi_v4.py [ARGUMENTS]
PS C:\Users\dell\Desktop\BDIA1\Visualisation des données\v.par python\travail> streamlit run .\bi_v4.py server.port 8080]
```

Résultat : une page s'ouvre automatiquement, et un lien URL pour se connecter

You can now view your Streamlit app in your browser.

Local URL: <http://localhost:8506>  
Network URL: <http://192.168.1.103:8506>



## 2. Sidebar configurations :

- Généralement un side bare se fait par la syntaxe suivant :

with st.sidebar:

```
st.header("le titre de sidebar")
```

```
variable = st.sidebar.multiselect(
```

```
    label = "petite_titre",
```

```
    options = data_deselections
```

```
    default=les_valeur_par_default)
```

- Ci-dessous notre sidebar complet contient un filtre des continent, years , country ; noter bien qu'il y a la possibilité de multichoix ou bien une choix unique :

```
with st.sidebar:
    st.header("Filter dataset")
    Year = st.sidebar.multiselect(
        label = "select year/years",
        options = data["Year"].unique(),
        default=[2023],
    )
    st.header("Filter dataset")
    continent=st.sidebar.multiselect(
        label="select continents",
        options=continents,
        default=continents,
    )
    selected_country = st.selectbox('selectionner une etate : ',
    sorted(data['Country'].unique()), index=sorted(data['Country'].unique()).index('Morocco'))

    selected_unique_year = st.selectbox('selectionner une annees :
    ',sorted(data['Year'].unique(), reverse=True), index=sorted(data['Year'].unique(),
    reverse=True).index(2023))
```

\*Pour chaque sidebar nous devons intégrer et sélectionner data qui correspond

### Filter dataset

select year/years

2023 ×

### Filter dataset

select continents

Africa × Asia ×

Europe ×

North\_America ×

South\_America ×


Oceania ×

selectionner une etate :

Morocco

selectionner une annees :

2023

 **US happiness Dashboard**

Select a color theme

blues

Résultat : nous obtenons une sidebar bien stylé, y'a deux choix soit régler les couleurs dans panneau setting ou bien inclure un style de css .....

### Edit active theme

Changes made to the active theme will exist for the duration of a session. To discard changes and recover the original theme, refresh the page.

Primary color ?

#178E24

Background color ?

#02081A

Text color ?

#F8F8FD

Secondary background color ?

#09235E

Font family ?

Sans serif

To save your changes, copy your custom theme into the clipboard and paste it into the `[theme]` section of your `.streamlit/config.toml` file.

Copy theme to clipboard

### 3. La metric :

```
def compact_metric(title, value, delta):
    st.markdown(
        f"""
        <div style="box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1); padding: 10px; border-radius:
10px; text-align: center; width: 230px; height: 190px; background-color: #ffffff;">
            <h3 style="font-size: 18px; margin-bottom: 10px;">{title}</h3>
            <h2 style="font-size: 24px; color: #55FD55; margin-bottom: 15px;">{value}
            </h2>
            <p style="font-size: 14px; color: #555555;">{delta} 🌐</p>
        </div>
        """,
        unsafe_allow_html=True
    )

def metric():
    with st.container():
        def draw_mwtric(col, param, compact_metric):
            with col:
                if param == 'Corruption':
                    m = df_select.groupby(by="Country")[param].mean()
                    compact_metric(f"Min {param} Score", round(m.min(), 2), f"Country:
{m.idxmin()}")
                elif param == 'Happiness Score':
                    m = df_select.groupby(by="Country")[param].mean()
                    compact_metric(f"Max {param}", round(m.max(), 2), f"Country:
{m.idxmax()}")
                else:
                    m = df_select.groupby(by="Country")[param].mean()
                    compact_metric(f"Max {param} Score", round(m.max(), 2), f"Country:
{m.idxmax()}")
            paramaters = ["Happiness Score", "Freedom", 'Life Expectancy', "GDP per Capita",
'Happiness Score', 'Corruption']
            colones = [cont1, cont2, cont3, cont4, cont5]
            for i, j in zip(paramaters, colones):
                draw_mwtric(j, i, compact_metric)
        metric()
    st.markdown("<hr style='margin: 20px 0;'", unsafe_allow_html=True)
```

Explication : Pour la metric nous avons besoin d'importer le model metrics\_cards par la suite on peut utiliser la syntaxe suivant :

With Column\_i #entrer la colonne dans la quelle on va afficher notre contenu

Column.metric(title= 'grande titre',value= ,delta ='une petite titre ')

C'est la syntaxe générale mais nous avons défini une fonctions qui contient un peu plus des performances et pour le cas on a plus d'une carte alors la fonction ça vas aide beaucoup pour cette situation



Modification : il est très utile de ranger les colonnes dans une partie de code pour contrôler la structure facilement comme la suite :

```
# _____ordre de
plot_chart _____
_
cont1, cont2, cont3, cont4, cont5 = st.columns([0.22, 0.22, 0.22, 0.22, 0.22])
_, slider, _ = st.columns([0.1, 0.8, 0.1])
st.divider()
_, col4, _ = st.columns([0.01, 0.88, 0.01])
st.divider()
col5, col6 = st.columns([0.5, 0.5])
st.divider()
col9, col10 = st.columns([0.5, 0.5])
st.divider()
_, col8, _ = st.columns([0.005, 0.99, 0.005])
st.divider()
col11, _, col12 = st.columns([0.45, 0.1, 0.8])
st.divider()
col14, col15, col16 = st.columns([0.5, 0.5, 0.5])
```

## I. Les graphes

- On commence par ordonner les graphes que l'on va dessiner. L'ordre d'affichage des graphes suivra l'ordre d'exécution de ces colonnes.

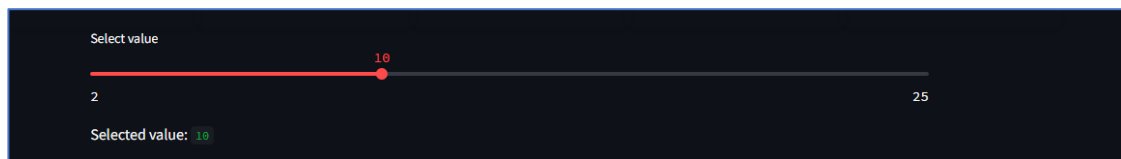
On sépare ces colonnes par des lignes en utilisant la fonction : **st.divider()**

```
# ordre de plot_chart
cont1, cont2, cont3, cont4, cont5 = st.columns([0.22, 0.22, 0.22, 0.22, 0.22])
_, slider, _ = st.columns([0.1, 0.8, 0.1])
st.divider()
_, col4, _ = st.columns([0.01, 0.88, 0.01])
st.divider()
col5, col6 = st.columns([0.5, 0.5])
st.divider()
col9, col10 = st.columns([0.5, 0.5])
st.divider()
_, col8, _ = st.columns([0.005, 0.99, 0.005])
st.divider()
col11, _, col12 = st.columns([0.45, 0.1, 0.8])
st.divider()
col14, col15, col16 = st.columns([0.5, 0.5, 0.5])
_, col13, _ = st.columns([0.2, 0.6, 0.35])
```

La fonction **st.columnn([])** permet de construire des colonnes dans lesquelles on pourra construire des graphiques. Par exemple, `col1, col2 = st.columnsn([0.7, 0.3])`, `[0.7, 0.3]` crée deux colonnes où la première occupe 70 % de la largeur disponible et la deuxième occupe 30 %. Ici, on commence par définir toutes les colonnes que l'on va utiliser par la suite. Cela va ordonner notre tableau de bord.



- Pour la construction d'un histogramme des pays les plus heureux du monde, les pays seront affichés en fonction des filtres de la barre latérale déjà implémentés, ainsi que du curseur. Le nombre de pays affichés dans l'histogramme sera stocké dans la variable `select_value`.



- Nous allons prendre les continents et les années sélectionnés par l'utilisateur. Ensuite, nous extrairons de notre base de données les lignes satisfaisant ces conditions. Étant donné que `st.multiselect()` permet de sélectionner plus d'une valeur, nous aurons besoin de calculer la moyenne du score de bonheur en commençant par regrouper les données selon les années.
- Ensuite, on utilise la colonne `col4` pour afficher le graphique. Avec cette dernière, on crée une figure `fig` dans laquelle on affecte un bar plot avec Plotly Express, de longueur 400 et de largeur 1000. On colore selon le Happiness Score avec une couleur continue `RdY11Bu` et un template Plotly. On met à jour la figure en arrondissant le Happiness Score et en l'affichant à l'intérieur du graphique, ainsi que les labels des axes. On termine par afficher le graphique dans notre page

```
with slider:
    select_value = st.slider(label="Select value", max_value=25, min_value=2, value=(10), step=1)
    st.write("Selected value:", select_value)

# Utilisation d'une colonne (col4) pour afficher un graphique basé sur la sélection
with col4:
    # Calcul des données en regroupant par pays et calculant la moyenne du score de bonheur
    data1 = df_select.groupby(by=["Country"])[["Happiness Score"]].mean().reset_index().sort_values(by=["Happiness Score"], ascending=False)

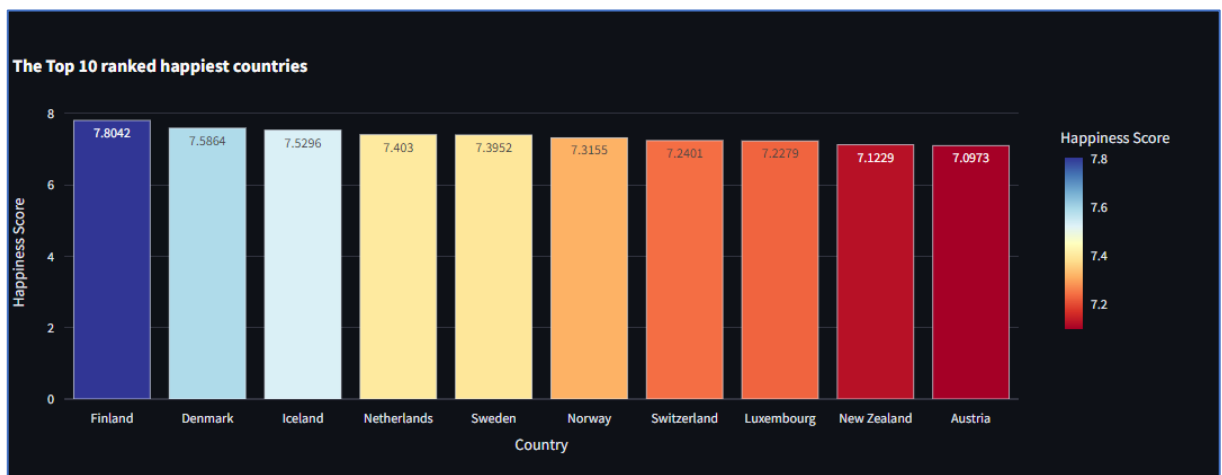
    # Sélection des premières lignes en fonction de la valeur choisie avec le curseur
    data1 = data1.head(select_value)

    # Création d'un graphique à barres avec Plotly express
    fig1 = px.bar(data1, x='Country', y='Happiness Score',
                  color='Happiness Score', color_continuous_scale='RdY11Bu',
                  labels={'pop': 'TOP 10'}, height=400, width=1000, template="plotly")

    # Mise à jour du texte à l'intérieur des barres avec les scores de bonheur arrondis
    fig1.update_traces(text=round(data1["Happiness Score"], 4), textposition='inside')

    # Mise à jour du layout du graphique avec les titres et les axes
    fig1.update_layout(xaxis_title='Country', yaxis_title='Happiness Score',
                      title=f'The Top {select_value} ranked happiest countries')

    # Affichage du graphique avec Streamlit
    st.plotly_chart(fig1, use_container_width=True)
```



Maintenant, on trace la variation du Happiness Score en fonction de la Freedom, du Social Support, du GDP et de la Life Expectancy. À partir d'une hypothèse selon laquelle le Happiness Score ne dépend pas du temps, nous allons regrouper par années et calculer la moyenne des autres facteurs. Dans ce cas, nous aurons un jeu de données contenant le Happiness Score et les autres facteurs, où la valeur de chaque variable d'un pays représente la moyenne sur les années.

```
# Création d'un nouveau DataFrame agrégé par pays et moyennant plusieurs variables
data2 = data.groupby(by=["Country"])[["Happiness Score", "Social Support", "Freedom", "Corruption"]].mean()
```

```
# Group data by 'Country' and compute mean for each column
grouped_data = data.drop(['Year', 'Continent'], axis=1).groupby('Country').mean()
```

Puisque l'ensemble de données contient, dans ce cas, plusieurs lignes, cela nous donnera une courbe avec plusieurs points, similaire au graphique de l'évolution du prix du bitcoin au fil des ans. C'est pourquoi nous allons diviser l'axe x en 10 intervalles. À chaque intervalle, nous allons calculer la moyenne du Happiness Score.

```
def points(variable, data):
    # Fonction pour calculer les points moyens en fonction d'une variable
    data1 = data.sort_values(by=variable)
    x_mean = []
    y_mean = []

    # Boucle pour créer des points moyens par intervalle de 10
    for i in range(0, 160, 10):
        data2 = data1.iloc[i:i+10, :]
        x_mean.append(data2[variable].mean())
        y_mean.append(data2['Happiness Score'].mean())

    x_mean = np.array(x_mean)
    y_mean = np.array(y_mean)
    return x_mean, y_mean
```

```
def mean_compute(param):
    # Initialize empty lists to store means
    x_mean = []
    y_mean = []

    # Iterate through data in intervals of 10
    for i in range(0, 160, 10):
        # Extract subset of data for each interval and compute means
        subset_data = sorted_data.iloc[i:i + 10]
        x_mean.append(subset_data['Happiness Score'].mean())
        y_mean.append(subset_data[param].mean())

    # Convert lists to NumPy arrays
    x_mean = np.array(x_mean)
    y_mean = np.array(y_mean)
    return x_mean, y_mean
```

Avec les colonnes col5, col6 et col10, nous traçons un graphique linéaire à l'aide de la bibliothèque `plotly.graph_objects`, représentant le Happiness Score en fonction du Social Support et de la Freedom avec col5, en fonction du GDP et de la Life Expectancy avec col6 et en fonction de Corruption en col10.

Pour plus d'informations sur les variables et les couleurs ..., vous pouvez toujours consulter la documentation. Nous allons mettre les liens de la documentation des bibliothèques utilisées à la fin du guide.

```
# Utilisation de col5 pour afficher le graphique
with col5:
    # Utilisation de la colonne col5

    # Calcul des points moyens pour différentes variables
    x1, y1 = points("Social Support", data2)
    # Calcul des points moyens pour la variable "Social Support" dans le dataframe data2

    x2, y2 = points("Freedom", data2)
    # Calcul des points moyens pour la variable "Freedom" dans le dataframe data2

    # Création d'une figure avec Plotly
    fig2 = go.Figure()
    # Initialisation d'une nouvelle figure Plotly

    # Ajout des traces pour Social Support et Freedom
    fig2.add_trace(go.Scatter(x=x1, y=y1, mode='lines', name="Social Support"))
    # Ajout d'une trace pour la variable "Social Support" à la figure

    fig2.add_trace(go.Scatter(x=x2, y=y2, mode='lines', name="Freedom"))
    # Ajout d'une trace pour la variable "Freedom" à la figure

    # Mise à jour du layout de la figure
    fig2.update_layout(
        xaxis_title='Social Support / Freedom', # Mise à jour du titre de l'axe x
        yaxis_title="Happiness Score", # Mise à jour du titre de l'axe y
        width=w, height=h, # Mise à jour de la largeur et de la hauteur de la figure
        plot_bgcolor='black', # Mise à jour de la couleur de fond du tracé
        paper_bgcolor='black', # Mise à jour de la couleur de fond du papier (de la figure)
        title='Variation du Happiness Score basée sur le Social Support et la Freedom' # Mise à jour du titre du graphique
    )

# Affichage du graphique dans l'application Streamlit
st.plotly_chart(fig2, use_container_width=True)
```



```
# Plot the data using Plotly
with col6:
    # Sort the data based on 'Happiness Score'
    sorted_data = grouped_data.sort_values(by='Happiness Score')

    # Compute means for 'Life Expectancy' and 'GDP per Capita'
    x_lif_exp, y_lif_exp = mean_compute('Life Expectancy')
    x_gdp, y_gdp = mean_compute('GDP per Capita')

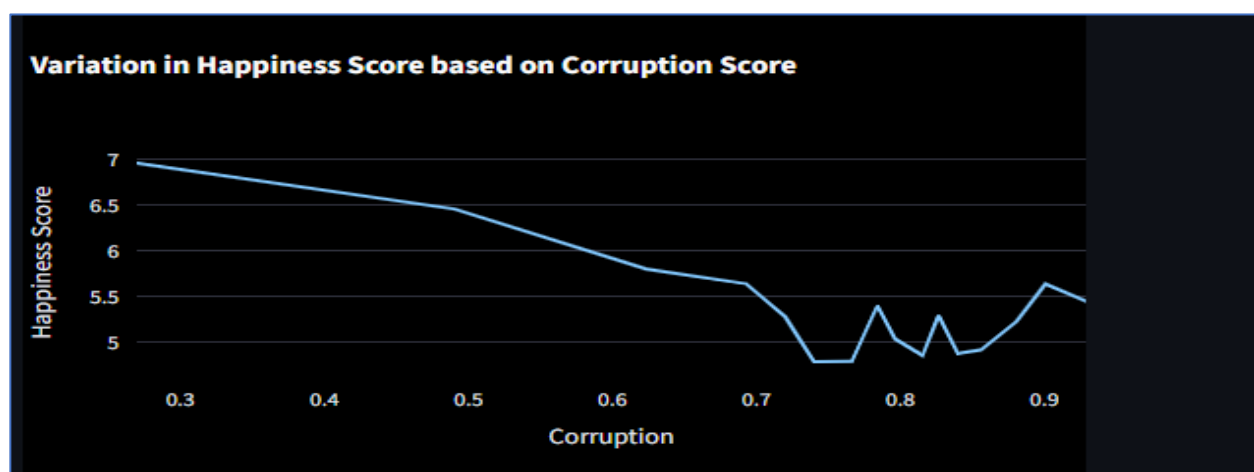
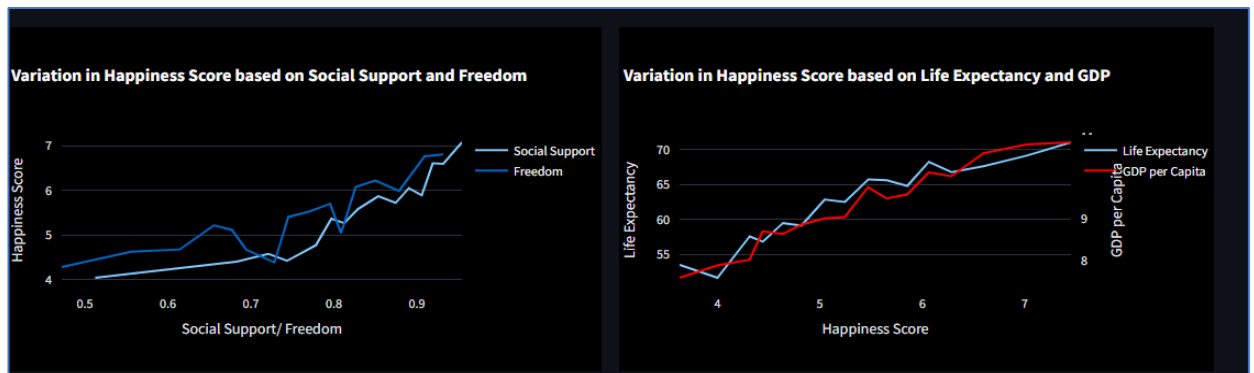
    # Create a Plotly figure
    fig = go.Figure()

    # Add traces for 'Life Expectancy' and 'GDP per Capita'
    fig.add_trace(go.Scatter(x=x_lif_exp, y=y_lif_exp, mode='lines', name='Life Expectancy', yaxis='y'))
    fig.add_trace(go.Scatter(x=x_gdp, y=y_gdp, mode='lines', name='GDP per Capita', line=dict(color='red'), yaxis='y2'))

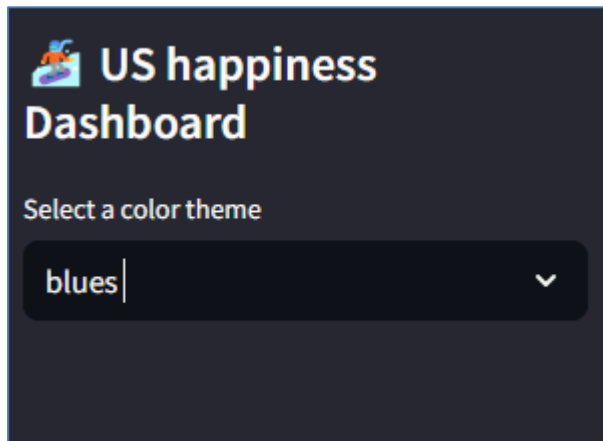
    # Update layout settings for the plot
    fig.update_layout(
        yaxis=dict(title='Life Expectancy'),
        yaxis2=dict(
            title='GDP per Capita',
            overlaying='y',
            side='right',
            showgrid=False,
        ),
        width=w, height=h, plot_bgcolor='black', paper_bgcolor='black',
        xaxis=dict(title='Happiness Score'),
        title='Variation in Happiness Score based on Life Expectancy and GDP'
    )

    # Display the plot using Streamlit
    st.plotly_chart(fig, use_container_width=True)
```

```
#line2-----Happiness-Score-VS corruption-----
with col10:
    x3,y3=points("Corruption",data2)
    fig2=go.Figure()
    fig2.add_trace(go.Scatter(x=x3, y=y3, mode='lines', name="Corruption"))
    fig2.update_layout(title='Variation in Happiness Score based on Corruption Score',
        xaxis_title='Corruption',
        yaxis_title='Happiness Score',
        width=w, height=h,
        plot_bgcolor='black',
        paper_bgcolor='black')
    st.plotly_chart(fig2, use_container_width=True)
```



On va maintenant tracer la carte : nous commençons par permettre aux utilisateurs de choisir la couleur de la carte dans la barre latérale.



```
# Configuration de la barre latérale
with st.sidebar:
    # Affiche le titre dans la barre latérale
    st.title('📊 Tableau de bord du bonheur aux États-Unis')

    # Définit une liste de thèmes de couleurs pour la carte choroplèthe
    color_theme_list = ['blues', 'cividis', 'greens', 'inferno', 'magma', 'plasma', 'reds', 'rainbow', 'turbo', 'viridis']

    # Permet à l'utilisateur de sélectionner un thème de couleur dans la liste
    selected_color_theme = st.selectbox('Sélectionnez un thème de couleur', color_theme_list)
```

Ensuite, on définit la fonction contenant les labels mis à jour :

```
# Fonction pour créer une carte choroplèthe
def make_choropleth(input_df, input_id, input_column, input_color_theme):
    # Crée une carte choroplèthe avec Plotly Express
    choropleth = px.choropleth(
        input_df,                # DataFrame contenant les données
        locations=input_id,      # Colonne spécifiant les emplacements
        color=input_column,      # Colonne spécifiant les valeurs de couleur
        locationmode="country names", # Définit le mode d'emplacement pour les noms de pays
        color_continuous_scale=input_color_theme, # Définit l'échelle de couleurs
        scope="world",          # Définit la portée de la carte au monde
        animation_frame="Year",  # Spécifie l'animation par année
        projection="natural earth", # Choix de la projection de la carte
        labels={'Happiness Score': 'Indice de bonheur'}, # Spécifie les étiquettes
        width=1000               # Définit la largeur de la carte
    )

    # Met à jour la mise en page pour une meilleure visualisation
    choropleth.update_layout(
        template='plotly_white', # Définit le modèle de la carte en fond blanc
        plot_bgcolor='rgba(0, 0, 0, 0)', # Définit la couleur de fond du tracé en transparent
        paper_bgcolor='rgba(0, 0, 0, 0)', # Définit la couleur de fond du papier en transparent
        margin=dict(l=0, r=0, t=0, b=0), # Définit la marge à 0 pour une mise en page propre
        height=400, # Définit la hauteur de la carte
    )

    return choropleth
```

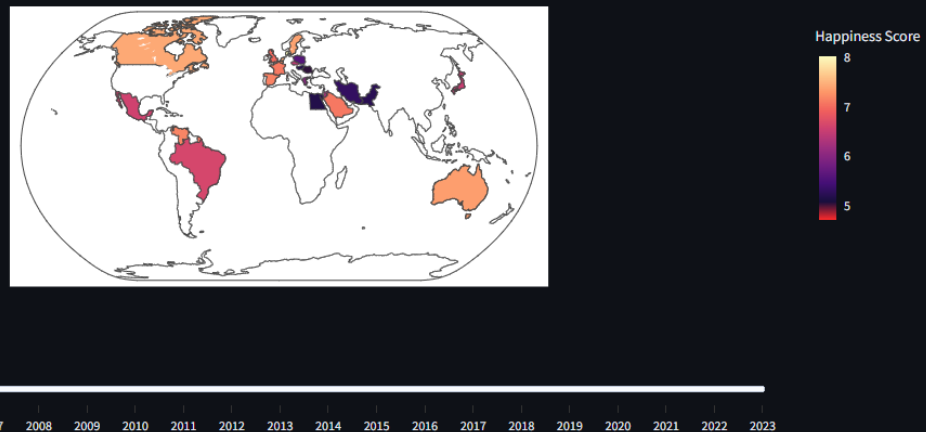
Avec la col8, on fait appel à cette fonction :

```
# Affiche une carte choroplèthe dans une mise en page en colonne
with col8:
    # Définit l'en-tête de la section de la carte
    st.markdown('## Scores de bonheur dans le monde')

    # Crée la carte choroplèthe en utilisant la fonction définie
    choropleth = make_choropleth(data, 'Country', 'Happiness Score', selected_color_theme)

    # Affiche la carte choroplèthe en utilisant Plotly
    st.plotly_chart(choropleth, use_container_width=True)
```

## Worldwide Happiness Scores



On passe maintenant au Matrice de Correlation, on commence par définir le jeu de données qu'on va utiliser `data3`. Après on calcule la matrice de corrélation et on ajoute l'annotation en utilisant la fonction `create_annotated_heatmap()` du module `plotly.figure_factory()` et mettant à jours les labels

```
# Utilisation de col9 pour sélectionner des colonnes spécifiques dans le dataframe 'data'
with col9:
    # Suppression des colonnes "Country" et "Continent" du dataframe 'data'
    data3 = data.drop(["Country", "Continent"], axis=1)

    # Calcul de la matrice de corrélation
    corr_matrix = data3.corr()

    # Conversion de la heatmap Matplotlib en heatmap Plotly
    fig4 = ff.create_annotated_heatmap(
        z=corr_matrix.values,
        x=corr_matrix.columns.tolist(),
        y=corr_matrix.index.tolist(),
        colorscale='blues', # Choix de la palette de couleurs
        annotation_text=corr_matrix.values.round(2), # Arrondi des valeurs pour les annotations
        showscale=True # Affichage de l'échelle des couleurs
    )

    # Mise à jour du layout de la figure
    fig4.update_layout(
        title='Correlation Heatmap', # Titre de la heatmap
        xaxis=dict(title='Features'), # Titre de l'axe x
        yaxis=dict(title='Features') # Titre de l'axe y
    )

    # Affichage de la heatmap avec Plotly dans Streamlit
    st.plotly_chart(fig4, use_container_width=True)
```

- Dans cette partie de notre tableau de bord, nous créons des graphiques en fonction d'une seule année et d'un seul État. Nous allons créer un graphique "spyder plot" pour visualiser les variables de l'État choisi pour une année donnée, ainsi que l'évolution du Happiness Score au fil des années pour l'État choisi. C'est pourquoi nous définissons deux `selectbox()` dans la barre latérale pour permettre aux utilisateurs de choisir l'année ainsi que l'État.

sélectionner une état :

Morocco ▼

sélectionner une année :

2023 ▼



Les données ne sont pas à la même échelle, ce qui posera des problèmes pour le graphique "spyder plot" car il ne sera pas bien visible. C'est pourquoi nous avons normalisé les colonnes de l'espérance de vie (Life Expectancy) ainsi que celles du PIB (GDP) pour obtenir des valeurs comprises entre 0 et 1.

```
# Filtrer les données en fonction du pays sélectionné et de l'année unique sélectionnée
second_filter_data = data[(data['Country'] == selected_country) & (data['Year'] == selected_unique_year)]

# Sélectionner les colonnes spécifiées pour l'analyse
columns = ['Happiness Score', 'GDP per Capita', 'Social Support', 'Life Expectancy', 'Freedom', 'Generosity', 'Corruption']
data_selected = data[columns]

# Normaliser les données à l'aide de MinMaxScaler
scaler = MinMaxScaler()
data_normalized = scaler.fit_transform(data_selected)
data_normalized_df = pd.DataFrame(data_normalized, columns=columns)

# Concaténer les données normalisées avec les colonnes 'Country' et 'Year' de la table originale
data_normalized_df = pd.concat([data[['Country', 'Year']], data_normalized_df], axis=1)

# Refiltrer les données normalisées en fonction du pays et de l'année sélectionnés
second_filter_data = data_normalized_df[(data_normalized_df['Country'] == selected_country) & (data_normalized_df['Year'] == selected_unique_year)]
```

Nous traçons les deux graphiques, le "spyder plot" sur la colonne 11 et l'autre sur la colonne 12, en utilisant le jeu de données filtré selon le choix de l'année et de l'État.

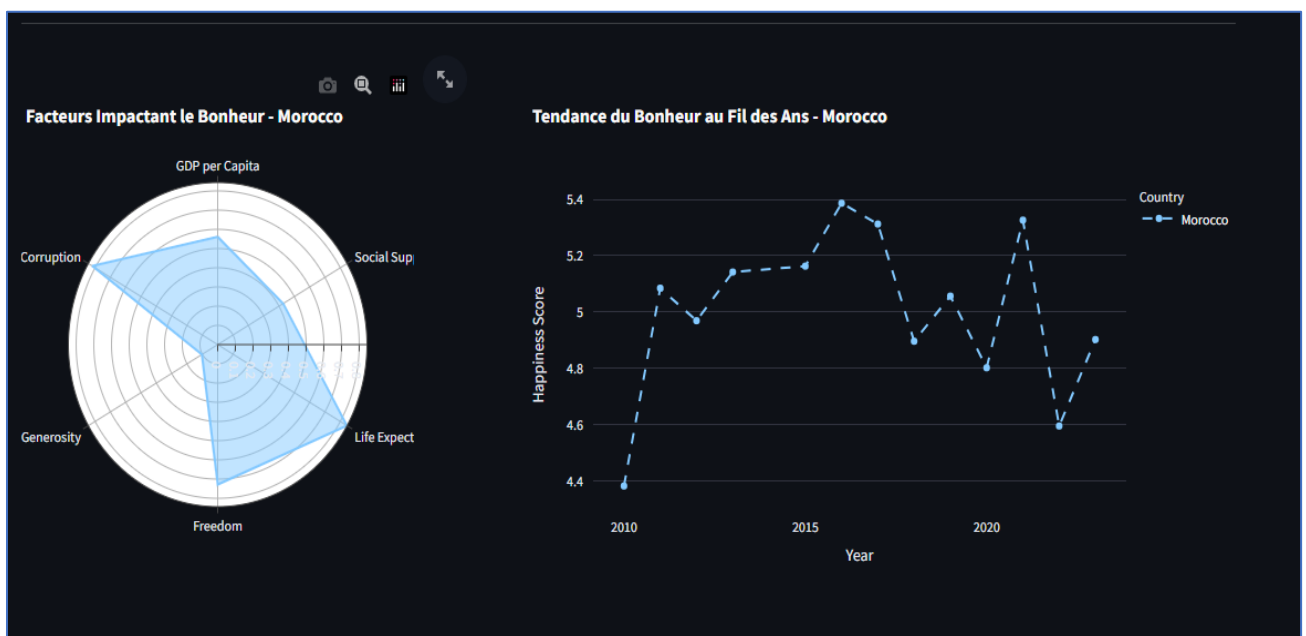
```
# Visualiser les facteurs impactant le bonheur pour le pays sélectionné
with col11:
    selected_melted_data = pd.melt(
        second_filter_data.drop(['Year', 'Happiness Score'], axis=1),
        id_vars='Country',
        var_name='Factors',
        value_name='value'
    )
    fig = px.line_polar(selected_melted_data, r='value', theta='Factors', line_close=True, title=f'Facteurs Impactant le Bonheur - {selected_country}')
    fig.update_traces(fill='toself')
    st.plotly_chart(fig, use_container_width=True)

# Visualiser la tendance du bonheur au fil des ans pour le pays sélectionné
with col12:
    fig = px.line(data[(data['Country'] == selected_country)], x='Year', y='Happiness Score', color='Country', markers=True, title=f'Tendance du Bonheur au Fil des Ans - {selected_country}')
    fig.update_traces(line_shape='linear', line=dict(dash='dash'), selector=dict(type='scatter'))
    st.plotly_chart(fig, use_container_width=True)
```

La fonction melt() de pandas permet de faire passer un DataFrame d'un format large à un format long :

Country	Factor1	Factor2	Factor3
0	A	10	25
1	B	15	30
2	C	20	35

Country Factors value			
0	A	Factor1	10
1	B	Factor1	15
2	C	Factor1	20
3	A	Factor2	25
4	B	Factor2	30
5	C	Factor2	35
6	A	Factor3	40
7	B	Factor3	45
8	C	Factor3	50



- On souhaite maintenant voir le classement d'un pays choisi au fil des années. On commence par créer une colonne "Rank" contenant le classement de chaque pays pour chaque année à l'aide de Pandas.

```
# _____the evolution of rank over years_____
# Boucle for parcourant les années de 2005 à 2023
for i in range(2005, 2024):
    # Filtrer les données pour l'année actuelle et les trier par le score de bonheur en ordre décroissant
    dr = data[data["Year"] == i].sort_values(by=["Happiness Score"], ascending=False).reset_index()

    # Créer une liste de classement pour chaque ligne dans le DataFrame trié
    rank = [j for j in range(1, dr.shape[0] + 1)]

    # Ajouter une colonne "Rank" au DataFrame contenant le classement pour chaque pays
    dr["Rank"] = rank

    # Concaténer le DataFrame actuel avec le DataFrame consolidé
    if i == 2005:
        df = dr
    else:
        df = pd.concat([df, dr], axis=0)
```

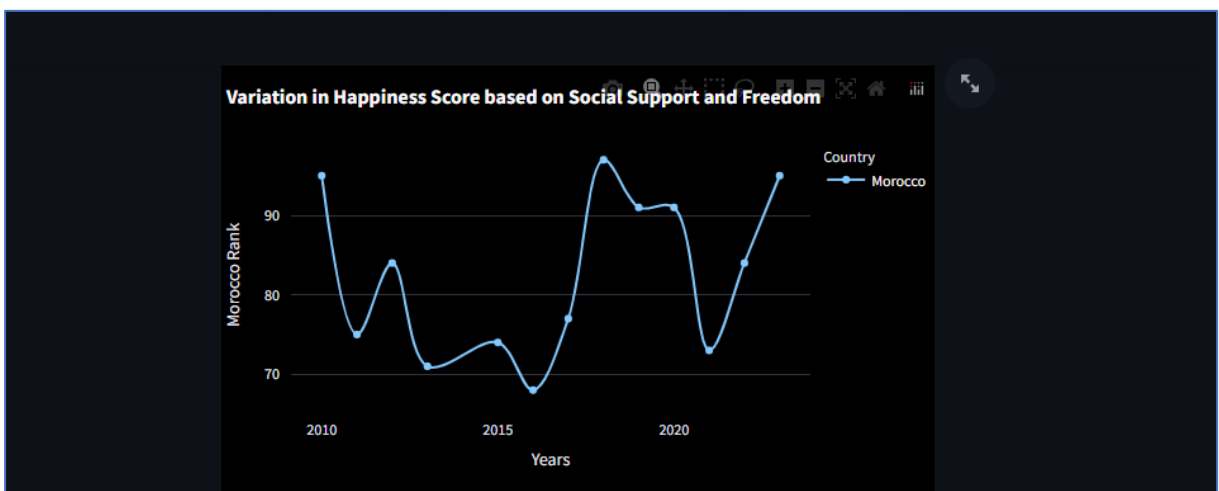
On trace maintenant le graphique en suivant les mêmes étapes qu'on vient d'expliquer :

```
# Sélectionner les données spécifiques au pays choisi dans le DataFrame consolidé
with coll3:
    datam = df[df["Country"] == selected_country]

    # Créer un graphique interactif avec Plotly pour visualiser l'évolution du classement du pays au fil des années
    figm = px.line(datam, x='Year', y='Rank', color='Country', symbol="Country", line_shape='spline')

    # Personnaliser la mise en page du graphique
    figm.update_layout(
        xaxis_title='Years', # Titre de l'axe des x (années)
        yaxis_title=f"{selected_country} Rank", # Titre de l'axe des y (classement du pays choisi)
        width=600, # Largeur du graphique
        height=350, # Hauteur du graphique
        plot_bgcolor='black', # Couleur de fond du graphique
        paper_bgcolor='black', # Couleur de fond du papier (zone autour du graphique)
        title='Variation in Happiness Score based on Social Support and Freedom' # Titre du graphique
    )

    # Afficher le graphique interactif dans l'application Streamlit
    st.plotly_chart(figm, use_container_width=True)
```



Ensuite, on crée trois métriques contenant le rang maximum et minimum du pays choisi au fil des années.

Country: Morocco

best Rank: 68 (In 2016) ↑

worst Rank: 97 (In 2018) ↓

On commence par définir trois variables : une pour le Country, le meilleur Rank, et la troisième pour le mauvais Rank. Ces trois fonctions utilisent une markdown qui va contrôler la couleur, la taille ainsi que la position. Puisque Streamlit ne permet pas de fournir cela, on aura besoin d'utiliser quelques codes HTML, mais attention au côté sécurité ! Si vous travaillez sur votre site web avec Streamlit et vous incluez des codes HTML ou CSS.

```
# -----Metric Metric-----
# Définition d'une fonction pour afficher une métrique compacte sans indicateur de tendance
def compact_metric1(param):
    st.markdown(
        f"""
        <div style="box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1); padding: 10px; border-radius: 10px; text-align: center; width: 230px; height: 190px; background-color: #000000;">
        | <h3 style="font-size: 18px; margin-bottom: 10px;">{param} </h3>
        </div>
        """, unsafe_allow_html=True)

# Définition d'une fonction pour afficher une métrique compacte avec une tendance positive
def compact_metric2(param):
    st.markdown(
        f"""
        <div style="box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1); padding: 10px; border-radius: 10px; text-align: center; width: 230px; height: 190px; background-color: #000000;">
        | <p style="font-size: 18px; color: #42FF33;">{param[0:5]} <span style="color: #ffffff;">{param[5:]</span> </p>
        </div>
        """, unsafe_allow_html=True)

# Définition d'une fonction pour afficher une métrique compacte avec une tendance négative
def compact_metric3(param):
    st.markdown(
        f"""
        <div style="box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1); padding: 10px; border-radius: 10px; text-align: center; width: 230px; height: 190px; background-color: #000000;">
        | <p style="font-size: 18px; color: #FC0303;">{param[0:5]} <span style="color: #ffffff;">{param[5:]</span> </p>
        </div>
        """, unsafe_allow_html=True)
```

On crée une fonction metric() qui va se charger de l'affichage des métriques en se basant sur les fonctions qu'on a définies précédemment.

```
# Fonction principale pour afficher les métriques
def metric():
    # Récupération des données pour le pays sélectionné
    m = df[df['Country'] == selected_country][['Year', 'Rank']]

    # Affichage de la première métrique (informations de base sur le pays)
    with col14:
        compact_metric1(f"Country: {selected_country}")

    # Affichage de la deuxième métrique (meilleur classement avec indication de tendance positive)
    with col15:
        compact_metric2(f'best Rank: {m["Rank"].min()} (In {m.loc[m["Rank"].idxmin(), "Year"]})')

    # Affichage de la troisième métrique (pire classement avec indication de tendance négative)
    with col16:
        compact_metric3(f'worst Rank: {m["Rank"].max()} (In {m.loc[m["Rank"].idxmax(), "Year"]})')

# Appel de la fonction pour afficher les métriques
metric()

# Ajout d'une ligne de séparation entre les métriques
st.markdown("<hr style='margin: 20px 0;'>", unsafe_allow_html=True)
```

