Import des librairies

pip install pandas

import pandas as pd

Import de la base de données

Base = pd.read_csv("EdStatsData.csv")

Afficher toute les colonnes et toutes les lignes

pd.options.display.max_columns = 10

pd.options.display.max_rows = 10

pd.options.display.max_rows = None

pd.set_option('display.max_columns', 30)

pd.set_option('display.max_rows', 30)

pd.set_option('display.min_rows', 100)

Base.loc[Base["Country Name"]=="Arab World",["Country Name","Country Code","Indicator Name","Indicator Code"]]

Base[["Indicator Code"]]

Base[["Indicator Code","Country Name","Country Code"]]

Base.loc[:,["Indicator Code","Country Name","Country Code"]]

Base.loc[Base.index==0,:]

ATTENTION SELECTION MULTIPLES CONDITIONS

Sélectionner des lignes avec des conditions multiples sur une colonne composée de strings (chaînes de caractères) grâce à la fonction isin():

La colonne "Country Code" de la base ne se compose que de strings: "ARB", "ZWE", etc.

Si on veut sélectionner les lignes pour lesquelles la valeur de la colonne "Country Code" est soit "FRA", "ZWE", "ARB" ou bien "EAS",

il faut faire:

```
Base.loc[Base["Country Code"].isin(['FRA','ZWE','ARB', 'EAS']),:]
```

REMARQUE: Cette méthode du isin() marche aussi avec des chiffres.

Graphiques et histogrammes

Histogrammes

```
sns.set_theme(style="whitegrid")
sns.histplot(data=Base.loc[Base["Country Code"].isin(['ZWE','ARB', 'EAS']),:], x="Country Code")
```

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(20,8))
sns.set_theme(style="whitegrid")
sns.histplot(data=Scoring_Personal_Computers.head(20),x="Country Name")
```

```
fichier1 = pd.read_csv('EdStatsData.csv')


# Sur le graphique ci dessus on voit que la colonne 'Unnamed: 69' ne contient aucune données. On va donc la supprimer.


fichier1 = fichier1.drop(columns=["Unnamed: 69"])


# Aussi les années allant de 1970 à 2000 ne nous intéressent pas car à cette époque les individus n'avaient pas accès à Internet.

# Nous allons donc garder les années allant de 2000 à 2040 car au-dessus d'ici 2040 la situation aura certainement évolué et les

# prévisions pour les années futures (jusqu'à 2100) auront changé.


# On supprime toutes les années (colonnes) allant de 1970 jusqu'à 2015


fichier1 = fichier1.drop(list(map(lambda x : str(x), list(range(1970,2000)))), axis=1)

fichier1 = fichier1.drop(list(map(lambda x : str(x), list(range(2045,2100,5)))), axis=1)

fichier1 = fichier1.drop(['2100'], axis=1)

fichier1 = fichier1.drop(['Country Code', 'Indicator Code'], axis=1)

fichier1


indexes_pays_supprimer = [……..]

fichier1 = fichier1.drop(indexes_pays_supprimer, axis = 0)

fichier1
```

Barplot

```
import matplotlib.pyplot as plt

fig=plt.figure(figsize=(8,8))

sns.set_theme(style="whitegrid")

ax = sns.barplot(x="Personal Computer (%)", y="Country Name",
data=Scoring_Personal_Computers.head(5))
```

```
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(8,8))

sns.set_theme(style="whitegrid")

ax = sns.barplot(y="Personal Computer (%)", x="Country Name",
data=Scoring_Personal_Computers.head(5))
```

Barplot mising values

```
import minssingno as msno

msno.bar(Site_Energy_Use)
```

Lineplot

```
sns.set_theme(style="whitegrid")

fig = plt.figure(figsize=(20,8))

sns.lineplot(x="Année", y="Upper Secondary 15-24 ans (Croissance en %)",
data=Upper_Secondary_15_24_df, hue = "Country Name")
```

```
fig = plt.figure(figsize=(20,8))

sns.lineplot(x="Année", y="Upper Secondary 15-24 ans (Croissance en %)",
data=Upper_Secondary_15_24_df)
```

Pieplot

```
d = {'Nom':
['Guillaume','Guillaume','Guillaume','Nicolas','Guillaume','Guillaume','Nicolas','Baptiste','Baptiste','Ba
ptiste','Baptiste','Baptiste','Baptiste','Baptiste','Baptiste','Baptiste'],'Compte': 1}

df = pd.DataFrame(data=d)


df = df.groupby('Nom')['Compte'].sum()


import matplotlib.pyplot as plt

sns.set_theme(style="whitegrid")

plt.figure(figsize=(10, 10))

plt.pie(df,labels = df.index,autopct='%1.1f%%')

plt.show()
```

heatmap correlation

```
d = {'Numero 1': [1,2,3,5], 'Numero 2': [1,2,3,6], 'Numero 3': [8,6,9,5], 'Numero 4': [8,6,9,7]}

df = pd.DataFrame(data = d)


sns.heatmap(df.corr(), cmap="Blues",annot=True)
```

scatterplot

```
sns.scatterplot(x='total_bill',y='tip',hue='sex',data=tips)
```

Pairplot

```
sns.pairplot(data_pca, hue = 'Species')
```

Barplot des valeurs manquantes

```
import pandas as pd
import seaborn as sns
import numpy as np
import missingno as msno

msno.bar(Site_Energy_Use)
```

Probplot

```
from scipy import stats
import matplotlib.pyplot as plt
# ax4 = plt.subplot(224)
stats.probplot(Site_Energy_Use['SiteEnergyUse_kBtu'],plot=plt)
plt.show()
```

Seaborn countplot

```
import seaborn as sns
sns.countplot(Data['Survived'])
plt.show()
```

PCA Analysis

```
import pandas as pd
data = pd.read_csv("iris.csv")
# selection des colonnes à prendre en compte dans l'ACP
# selection des colonnes à prendre en compte dans l'ACP
```

```
data_pca = data[['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
      'Species']]
data_pca = data_pca.fillna(data_pca.mean())
```

Remplacer des chaines de caractères par des chiffres

```
def fonction(nom):
   if nom == 'Iris-setosa':
      return 0
   elif nom == 'Iris-versicolor':
      return 1
   else:
      return 2

data_pca['Species_nb']=list(map(fonction,data_pca['Species']))

data_pca = data_pca.drop(['Id'],axis = 1)
```

Afficher la matrice de corrélations sous forme de heat map :

```
sns.heatmap(data_pca.corr(),annot=True,cmap='Blues')
```

Standard Scaler

```
# Extraire les valeurs du data frame
X = data_pca.values

from sklearn.preprocessing import StandardScaler
# Standardizing the features
X = StandardScaler().fit_transform(X)
```

Autres commandes utiles pour l'analyse de données :

```
Base.info()
```

```
Base.describe()
```

PCA sklearn and PCA Graph

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)

principalComponents = pca.fit_transform(X)

principalDf = pd.DataFrame(data = principalComponents

        , columns = ['principal component 1', 'principal component 2'])

principalDf['Species'] = Y



import seaborn as sns

import matplotlib.pyplot as plt


sns.scatterplot(x='principal component 1', y='principal component 2', hue='Species', data =
principalDf)
```

Pycaret

```
pip install pycaret


import pandas as pd


pd.options.display.max_columns = None

pd.options.display.max_rows = None


Site_Energy_Use = pd.read_csv('Dataset_SiteEnergyUse.csv')

Total_GHG_Emissions = pd.read_csv('Dataset_TotalGHGEmissions.csv')

Total_GHG_Emissions_Sans_ENERGYSTARScore = pd.read_csv('Dataset_TotalGHGEmissions.csv')
```

```
Site_Energy_Use = Site_Energy_Use.drop('Unnamed: 0', axis=1)

Total_GHG_Emissions = Total_GHG_Emissions.drop('Unnamed: 0', axis=1)

Total_GHG_Emissions_Sans_ENERGYSTARScore =
Total_GHG_Emissions_Sans_ENERGYSTARScore.drop('Unnamed: 0', axis=1)

Total_GHG_Emissions_Sans_ENERGYSTARScore =
Total_GHG_Emissions_Sans_ENERGYSTARScore.drop('ENERGYSTARScore', axis=1)
```

```
from pycaret.regression import *

exp_name = setup(data = Site_Energy_Use,  target = 'SiteEnergyUse_kBtu', train_size = 0.8)
```

```
compare_models()
```

```
gbr = create_model('gbr')
```

```
# generate predictions on unseen data

y_pred = predict_model(gbr)
```

One Hot Encodding

```
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(handle_unknown='ignore')

#perform one-hot encoding on 'team' column

encoder_df = pd.DataFrame(encoder.fit_transform(Data[['genre']]).toarray())

Data = Data.join(encoder_df)
```

```
#rename columns
Data.columns = ['age', 'gender', 'genre', 'Acoustic', 'Classical','Dance','HipHop','Jazz']

Data.head(100)
```

Rééquilibrage (Undersampling) La classe 0 est la classe majoritaire

```
minority_class_len = len(Data.loc[Data['Survived']==1,:])

majority_class_indices = Data.loc[Data['Survived']==0,:].index

###### fonction np.random.choice (pour choisir des indices au hasard selon une certaine taille)
random_majority_indices = np.random.choice(majority_class_indices,
                                minority_class_len,
                                replace=False)
print(len(random_majority_indices))


minority_class_indices = Data.loc[Data['Survived']==1,:].index
minority_class_indices

under_sample_indices = np.concatenate([minority_class_indices,random_majority_indices])

under_sample = Data.loc[under_sample_indices,:]
```

Modèle

```
X = Data.drop(['Survived'],axis = 1)

y = Data.loc[:,['Survived']]


from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3)


from sklearn.model_selection import GridSearchCV

param_grid={'n_neighbors': [1, 20], 'metric': ['euclidean','manhattan']}


from sklearn.neighbors import KNeighborsClassifier

grid = GridSearchCV(KNeighborsClassifier(), param_grid, cv = 5)

grid.fit(X_train,y_train)


grid.best_score_

grid.best_params_

model = grid.best_estimator_

model.score(X_test,y_test)
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix

pd.DataFrame(confusion_matrix(y_test, model.predict(X_test)),columns = ['Death','Survived'],index = ['Death','Survived'])
```

######## Plot the confusion Matrix with sns.heatmmap()

```
sns.heatmap(pd.DataFrame(confusion_matrix(y_test, model.predict(X_test)),columns = ['Death','Survived'],index = ['Death','Survived']),cmap='Blues',annot=True)

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()
```

Evaluation des metrics (Accuracy, Precision and Recall)

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, model.predict(X_test))
```

```
from sklearn.metrics import precision_score
precision_score(y_test, model.predict(X_test))
```

```
from sklearn.metrics import recall_score
recall_score(y_test, model.predict(X_test))
```

Metriques de regression

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import *

y = np.array([3, -0.5, 2, 7])
y_pred = np.array([2.5, 0.0, 2, 8])

print('MAE:', mean_absolute_error(y,y_pred))
print('MSE:', mean_squared_error(y,y_pred))
print('RMSE:',np.sqrt(mean_squared_error(y,y_pred)))

# Coefficient de détermination
print('R2:', r2_score(y,y_pred))
```

Text Embedding TensorFlow Hub

pip install tensorflow

pip install tensorflow_hub

IL EST IMPORTANT D'INSTALLER TENSORFLOW EN PLUS DE TENSORFLOW_HUB SINON çA NE MARCHE PAS

```
import tensorflow_hub as hub

embed = hub.load("https://tfhub.dev/google/universal-sentence-encoder/4")

embeddings = embed([
    "The quick brown fox jumps over the lazy dog.",
    "I am a sentence for which I would like to get its embedding"])

print(embeddings)
```

```
phrase = "Et c'était qui ? Et c'était qui ?"
embeddings2=embed([phrase])
embeddings2
```