Recharge de l'ensemble des données.

In [ ]:
```
data = pd.read_csv("owid-co2-data.csv", delimiter=",", header=0)
data
```

Out[ ]:

	country	year	iso_code	population	gdp	cement_co2	cement_co2
0	Afghanistan	1850	AFG	3752993.0	NaN	NaN	
1	Afghanistan	1851	AFG	3767956.0	NaN	NaN	
2	Afghanistan	1852	AFG	3783940.0	NaN	NaN	
3	Afghanistan	1853	AFG	3800954.0	NaN	NaN	
4	Afghanistan	1854	AFG	3818038.0	NaN	NaN	
...	...	...	...	...	...	...	
48053	Zimbabwe	2018	ZWE	15052191.0	2.271535e+10	0.558	
48054	Zimbabwe	2019	ZWE	15354606.0	NaN	0.473	
48055	Zimbabwe	2020	ZWE	15669663.0	NaN	0.496	
48056	Zimbabwe	2021	ZWE	15993525.0	NaN	0.531	
48057	Zimbabwe	2022	ZWE	16320539.0	NaN	0.531	

48058 rows × 79 columns

Séparation de la liste des colonnes catégoriels et numériques.

In [ ]:
```
columns_numerics = data.dtypes[data.dtypes != "object"].index.to_list()
columns_categoriels = data.dtypes[data.dtypes == 'object'].index.to_list()
print("Colonnes numériques:", columns_numerics)
print("Colonnes catégoriels:", columns_categoriels)
```

```
Colonnes numériques: ['year', 'population', 'gdp', 'cement_co2', 'cement_co2_per_
capita', 'co2', 'co2_growth_abs', 'co2_growth_prct', 'co2_including_luc', 'co2_in
cluding_luc_growth_abs', 'co2_including_luc_growth_prct', 'co2_including_luc_per_
capita', 'co2_including_luc_per_gdp', 'co2_including_luc_per_unit_energy', 'co2_p
er_capita', 'co2_per_gdp', 'co2_per_unit_energy', 'coal_co2', 'coal_co2_per_capit
a', 'consumption_co2', 'consumption_co2_per_capita', 'consumption_co2_per_gdp',
'cumulative_cement_co2', 'cumulative_co2', 'cumulative_co2_including_luc', 'cumul
ative_coal_co2', 'cumulative_flaring_co2', 'cumulative_gas_co2', 'cumulative_luc_
co2', 'cumulative_oil_co2', 'cumulative_other_co2', 'energy_per_capita', 'energy_
per_gdp', 'flaring_co2', 'flaring_co2_per_capita', 'gas_co2', 'gas_co2_per_capit
a', 'ghg_excluding_lucf_per_capita', 'ghg_per_capita', 'land_use_change_co2', 'la
nd_use_change_co2_per_capita', 'methane', 'methane_per_capita', 'nitrous_oxide',
'nitrous_oxide_per_capita', 'oil_co2', 'oil_co2_per_capita', 'other_co2_per_capit
a', 'other_industry_co2', 'primary_energy_consumption', 'share_global_cement_co
2', 'share_global_co2', 'share_global_co2_including_luc', 'share_global_coal_co
2', 'share_global_cumulative_cement_co2', 'share_global_cumulative_co2', 'share_g
lobal_cumulative_co2_including_luc', 'share_global_cumulative_coal_co2', 'share_g
lobal_cumulative_flaring_co2', 'share_global_cumulative_gas_co2', 'share_global_c
umulative_luc_co2', 'share_global_cumulative_oil_co2', 'share_global_cumulative_o
ther_co2', 'share_global_flaring_co2', 'share_global_gas_co2', 'share_global_luc_
co2', 'share_global_oil_co2', 'share_global_other_co2', 'share_of_temperature_cha
nge_from_ghg', 'temperature_change_from_ch4', 'temperature_change_from_co2', 'tem
perature_change_from_ghg', 'temperature_change_from_n2o', 'total_ghg', 'total_ghg
_excluding_lucf', 'trade_co2', 'trade_co2_share']
Colonnes catégoriels: ['country', 'iso_code']
```

In [ ]:
```python
def valeurs_manquantes(data, drop_zero_missed_column=True):
    m,_ = data.shape
    missing_values = data.isnull().sum()
    if drop_zero_missed_column: missing_values = missing_values[missing_values >
    return ((missing_values / m) * 100).round(2).to_frame(name="% Pourcentages")

def plot_histogram(donnees_manquantes, x="Variables", y="% Pourcentages"):
    plt.figure()
    couleurs = ['skyblue', 'lightcoral', 'lightgreen', 'lightpink', 'lightsalmon
    donnees_manquantes.plot.barh(x=x, y=y, color=couleurs, legend=False)
    plt.title('Pourcentage des Valeurs Manquantes par Variable')
    plt.xlabel('Variables')
    plt.ylabel('Pourcentage de Valeurs Manquantes (%)')
    plt.show()

donnees_manquantes = valeurs_manquantes(data)
columns_missed = donnees_manquantes.index.to_list()
```

Prétraitement

Traitement des donnnées manquantes

Traitement de la colonne cible ***co2***

Comme mentionné dans la section sur la visualisation et les remarques concernant la colonne cible, avant d'entamer le prétraitement des valeurs manquantes, nous commencerons par éliminer les données manquantes de cette colonne et supprimer les doublons. Il convient de noter que cette colonne présente peu de valeurs manquantes et

que notre ensemble de données contient suffisamment d'échantillons. Par conséquent, cela n'aura qu'un impact mineur sur notre jeu de données.

In [ ]:
```
data = data.dropna(subset=["co2"])
data = data.drop_duplicates()
print("Ancien dimension = 48058 × 79")
print("Nouvelle dimension = {} × 79".format(data.shape[0]))
print("Pourcentage des lignes supprimées = {} %".format(round((1 - (data.shape[0
```

```
Ancien dimension = 48058 × 79
Nouvelle dimension = 30308 × 79
Pourcentage des lignes supprimées = 37 %
```

Ainsi, il est à noter que seules 37 % des lignes ont été supprimées, ce qui est négligeable compte tenu de la taille de l'ensemble de données

Eliminer les colonnes de plus de 60% de valeurs manquantes.

Avec une colonne présentant plus de 60 % de données manquantes, il serait très risqué d'essayer d'effectuer une imputation pour conserver ces colonnes. De plus, comme l'ont montré les observations de nos données, certaines colonnes ayant un faible nombre de valeurs manquantes sont calculées à partir d'autres colonnes. Par exemple, le cumul des émissions de CO2 des voitures depuis la première année peut remplacer les données manquantes concernant la production de CO2 par les voitures. Ainsi, en éliminant ces colonnes, la perte de données peut être considérée comme acceptable.

In [ ]:
```
missing_values = valeurs_manquantes(data)
missed_columns = missing_values[missing_values["% Pourcentages"] > 60].index.to_
data_cleared = data.drop(columns=missed_columns)
data_cleared = data_cleared.drop_duplicates()
print("Nous avons {} lignes et {} colonnes restantes, ce qui signifie que {} % d
    data_cleared.shape[0], data_cleared.shape[1], round((1 - (data_cleared.shape
```

```
Nous avons 30308 lignes et 56 colonnes restantes, ce qui signifie que 29.11 % des
colonnes ont été supprimées.
```

Ainsi, après l'élimination, nous constatons que seulement 29,11 % des colonnes ont été supprimées. De plus, nous notons qu'il n'y a pas eu de perte de données due aux doublons, ce qui est très positif pour la suite du prétraitement.

Traitement des valeurs manquantes dans des colonnes symétriques.

Cette section se concentrera sur les colonnes présentant des données manquantes qui suivent une distribution symétrique, comme nous l'avons observé lors de la visualisation des données.

In [ ]:
```
data_cleared_numerics = data_cleared.drop(columns=['country', 'iso_code', 'co2']
data_cleared_numerics.shape
```

Out[ ]:
```
(30308, 53)
```

Pour identifier les colonnes présentant des distributions symétriques et asymétriques ainsi que les taux de valeurs aberrantes, nous utilisons les méthodes skew et kurtosis de

pandas pour filtrer les colonnes. Cependant, avant d'appliquer ce filtre, nous retirons les données aberrantes des colonnes afin d'éviter toute introduction de biais dans le traitement de la distribution.

In [ ]:
```
means_imputations_columns = []
hight_outlier_columns = []

# Afficher la distribution de chaque colonne
for i, column in enumerate(data_cleared_numerics.columns):
    # Filtrer les valeurs aberrantes
    serie = data_cleared_numerics[column].dropna()
    Q1 = serie.quantile(0.25)
    Q3 = serie.quantile(0.75)
    IQR = Q3 - Q1
    serie_filtre = serie[(serie >= Q1 - 1.5 * IQR) & (serie <= Q3 + 1.5 * IQR)]

    # Calculer le coefficient d'asymétrie (skewness) pour chaque colonne
    # Filtrer les colonnes avec des valeurs proches de zéro pour skewness et de
    # pour trouver les colonnes relativement symétriques.
    if (abs(serie_filtre.skew()) < 0.5) & (abs(serie_filtre.kurtosis()) - 3 < 0.
        means_imputations_columns.append((column, serie_filtre.mean()))
    elif (1 - (serie_filtre.shape[0] / serie.shape[0])) * 100 > 15:
        hight_outlier_columns.append(column)
means_imputations_columns
```

Out[ ]:
```
[('co2_growth_prct', 4.1566489021043),
 ('co2_including_luc_growth_prct', 0.7632530763994345),
 ('cumulative_flaring_co2', 0.0),
 ('flaring_co2', 0.0),
 ('flaring_co2_per_capita', 0.0),
 ('temperature_change_from_n2o', 0.0)]
```

Nous avons identifié 6 colonnes présentant une distribution symétrique. Pour ces colonnes, nous utiliserons l'imputation par la moyenne, conformément à ce qui a été expliqué dans la section sur les méthodes d'imputation de la visualisation.

In [ ]:
```
for column, _mean in means_imputations_columns:
    data_cleared[column] = data_cleared[column].fillna(_mean)
```

Nous vérifions que toutes les données de ces 6 colonnes ont été correctement traitées.

In [ ]:
```
columns_means_imputed = [column for column, _ in means_imputations_columns]
data_cleared[columns_means_imputed].isnull().sum()
```

Out[ ]:
```
co2_growth_prct                0
co2_including_luc_growth_prct  0
cumulative_flaring_co2         0
flaring_co2                    0
flaring_co2_per_capita         0
temperature_change_from_n2o    0
dtype: int64
```

Traitement des valeurs manquantes dans des colonnes avec des distributions asymétriques avec des valeurs aberrantes élevées.

Dans la section précédente, lors de la recherche des colonnes présentant des distributions asymétriques, nous avons dressé la liste des colonnes avec plus de 15 % de valeurs aberrantes. Dans cette section, nous allons appliquer une imputation en utilisant la méthode K-NN avec k = 2, comme discuté dans la section sur la visualisation.

In [ ]:
```
imputer = KNNImputer(n_neighbors=2)

# Application de l'imputation par k-nnn.
X_imputed = imputer.fit_transform(data_cleared_numerics[hight_outlier_columns])
X_imputed = pd.DataFrame(X_imputed, columns=data_cleared_numerics[hight_outlier_
X_imputed.to_csv('X_imputed_knn.csv', index=False)
X_imputed.isnull().sum()
```

Out[ ]:
```
population                                0
cement_co2                                0
co2_growth_abs                            0
co2_including_luc                         0
co2_including_luc_growth_abs              0
coal_co2                                  0
cumulative_cement_co2                     0
cumulative_co2                            0
cumulative_co2_including_luc              0
cumulative_coal_co2                       0
cumulative_gas_co2                        0
cumulative_luc_co2                        0
cumulative_oil_co2                        0
gas_co2                                   0
gas_co2_per_capita                        0
land_use_change_co2                       0
oil_co2                                   0
share_global_cement_co2                   0
share_global_co2                          0
share_global_co2_including_luc            0
share_global_coal_co2                     0
share_global_cumulative_cement_co2        0
share_global_cumulative_co2               0
share_global_cumulative_co2_including_luc 0
share_global_cumulative_coal_co2          0
share_global_cumulative_flaring_co2       0
share_global_cumulative_gas_co2           0
share_global_cumulative_luc_co2           0
share_global_cumulative_oil_co2           0
share_global_flaring_co2                  0
share_global_gas_co2                      0
share_global_luc_co2                      0
share_global_oil_co2                      0
share_of_temperature_change_from_ghg      0
temperature_change_from_ch4               0
temperature_change_from_co2               0
dtype: int64
```

Vérification de l'éffectivité de l'imputation par K-NN.

Dans la section suivante, nous aborderons le traitement des colonnes qui ne répondent pas aux deux critères précédents. Pour ces colonnes, nous opterons pour une approche d'imputation itérative. Cette méthode, comme mentionné dans la section dédiée à la visualisation, implique un processus itératif où les valeurs manquantes sont estimées à

l'aide des valeurs observées dans les autres colonnes. Nous explorerons en détail cette technique et son application spécifique à notre ensemble de données.

In [ ]:
```
columns_imputed = means_imputations_columns + hight_outlier_columns
other_columns = [column for column in data_cleared_numerics.columns if column no
print("Il reste {} colonnes à imputer par itération.".format(len(other_columns))
```

```
Il reste 17 colonnes à imputer par itération
```

Imputation des colonnes restantes par la méthode itérative comme vue dans la section visualisation.

Dans cette section, nous appliquerons l'imputation itérative en utilisant la méthode IterativeImputer de sklearn sur les 17 colonnes numériques restantes à traiter.

L'imputation itérative permet de remplacer les valeurs manquantes dans un ensemble de données en utilisant un processus itératif. Il commence par remplacer les valeurs manquantes par des estimations initiales, puis ajuste un modèle prédictif aux données complètes pour prédire les valeurs manquantes. Ce processus est répété plusieurs fois jusqu'à ce que les valeurs imputées convergent vers des estimations finales. C'est utile lorsque les données sont complexes et que les valeurs manquantes sont présentes dans plusieurs colonnes.

In [ ]:
```
imputer = IterativeImputer()

# Inputation par la méthode itérative.
result = imputer.fit_transform(data_cleared_numerics[other_columns])
data_iterative_imputed = pd.DataFrame(result, columns = other_columns)
data_iterative_imputed.isnull().sum()
```

Out[ ]:
```
year                              0
gdp                               0
cement_co2_per_capita             0
co2_growth_prct                   0
co2_including_luc_growth_prct     0
co2_including_luc_per_capita      0
co2_including_luc_per_gdp         0
co2_per_capita                    0
co2_per_gdp                       0
coal_co2_per_capita               0
cumulative_flaring_co2            0
flaring_co2                       0
flaring_co2_per_capita            0
land_use_change_co2_per_capita    0
oil_co2_per_capita                0
temperature_change_from_ghg       0
temperature_change_from_n2o       0
dtype: int64
```

Toutes les 17 colonnes ont été imputées avec succès.

Après l'application de la méthode itérative, toutes les valeurs manquantes ont été correctement imputées dans notre ensemble de données. Cette méthode a permis de remplacer les valeurs manquantes de manière efficace en utilisant un processus itératif qui prend en compte les relations entre les variables. Cette approche, nous avons pu

maximiser l'utilisation des données disponibles tout en minimisant les biais introduits par les valeurs manquantes, ce qui renforce la robustesse de notre analyse.

In [ ]:
```
X_imputed_columns = X_imputed.columns.tolist()
data_cleared_numerics[X_imputed_columns] = X_imputed.to_numpy()
data_cleared_numerics[other_columns] = data_iterative_imputed.to_numpy()

data_cleared[X_imputed_columns + other_columns] = data_cleared_numerics[X_impute
data_cleared.shape
```

Out[ ]:
```
(30308, 56)
```

In [ ]:
```
data_cleared.head(3)
```

Out[ ]:

	country	year	iso_code	population	gdp	cement_co2	cement_co2_
99	Afghanistan	1949.0	AFG	7356890.0	2.874382e+11	0.0	
100	Afghanistan	1950.0	AFG	7480464.0	9.421400e+09	0.0	
101	Afghanistan	1951.0	AFG	7571542.0	9.692280e+09	0.0	

3 rows × 56 columns

Traitement des valeurs manquantes dans les colonnes catégorielles.

Parmi les colonnes catégorielles, seule la colonne iso_code, qui contient les codes des pays, présente des valeurs manquantes. Nous pouvons envisager de supprimer cette colonne car le code d'un pays n'est pas directement lié à la variable cible (co2) qui est l'émission de CO2. De plus, cette colonne fournit des informations redondantes par rapport à la colonne country, qui représente déjà les pays.

In [ ]:
```
data_cleared = data_cleared.drop(columns=["iso_code"])
data_cleared.head(3)
```

Out[ ]:

	country	year	population	gdp	cement_co2	cement_co2_per_capita
99	Afghanistan	1949.0	7356890.0	2.874382e+11	0.0	0.0
100	Afghanistan	1950.0	7480464.0	9.421400e+09	0.0	0.0
101	Afghanistan	1951.0	7571542.0	9.692280e+09	0.0	0.0

3 rows × 55 columns