

Overview

This notebook will show you how to create and query a table or DataFrame that you uploaded to DBFS. DBFS (<https://docs.databricks.com/user-guide/dbfs-databricks-file-system.html>) is a Databricks File System that allows you to store data for querying inside of Databricks. This notebook assumes that you have a file already inside of DBFS that you would like to read from.

This notebook is written in **Python** so the default cell type is Python. However, you can use different languages by using the `%LANGUAGE` syntax. Python, Scala, SQL, and R are all supported.

Project Topic: Clustering Countries for Targeted Socio-Economic and Health Interventions

Business Problem: Many countries face significant challenges in meeting the socio-economic and health needs of their populations. Limited resources necessitate a targeted approach to development and funding. This project aims to categorize countries based on key indicators to identify specific needs and prioritize interventions in areas such as infrastructure, health, and education.

1. Import Libraries

Import the required libraries

```
import pandas as pd
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, when, mean, stddev
from pyspark.ml.feature import Imputer, VectorAssembler, StandardScaler
from pyspark.ml import Pipeline
import numpy as np

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report

import matplotlib.pyplot as plt
import seaborn as sns

import warnings

import plotly.express as px

#import tensorflow as tf
#from tensorflow.keras import Sequential
#from tensorflow.keras.layers import Dense, Dropout, InputLayer
#from tensorflow.keras.optimizers import Adam
from tabulate import tabulate
```


2. Load the Data

```
# File location and type
file_location = "/FileStore/tables/Country_data.csv"
file_type = "csv"

# CSV options
infer_schema = "false"
first_row_is_header = "true"
delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
    .option("inferSchema", infer_schema) \
    .option("header", first_row_is_header) \
    .option("sep", delimiter) \
    .load(file_location)

display(df.head(10))
```

▸  df: pyspark.sql.dataframe.DataFrame = [country: string, child_mort: string ... 8 more fields]

Table

	A ^B C country	A ^B C child_mort	A ^B C exports	A ^B C health	A ^B C imports	A ^B C income	A ^B C inflation	A ^B C life_exp
1	Afghanistan	90.2	10	7.58	44.9	1610	9.44	56.2
2	Albania	16.6	28	6.55	48.6	9930	4.49	76.3
3	Algeria	27.3	38.4	4.17	31.4	12900	16.1	76.5
4	Angola	119	62.3	2.85	42.9	5900	22.4	60.1
5	Antigua and Barbu...	10.3	45.5	6.03	58.9	19100	1.44	76.8
6	Argentina	14.5	18.9	8.1	16	18700	20.9	75.8
7	Armenia	18.1	20.8	4.4	45.3	6700	7.77	73.3
8	Australia	4.8	19.8	8.73	20.9	41400	1.16	82
9	Austria	4.3	51.3	11	47.8	43200	0.873	80.5

10 rows

8

```
# Assuming 'spark_df' is your PySpark DataFrame
pandas_df = df.toPandas()

# Now you can work with your pandas DataFrame (pandas_df)
(pandas_df.head(3))
```

	country	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp
0	Afghanistan	90.2	10	7.58	44.9	1610	9.44	56.2	5.82	553
1	Albania	16.6	28	6.55	48.6	9930	4.49	76.3	1.65	4090
2	Algeria	27.3	38.4	4.17	31.4	12900	16.1	76.5	2.89	4460

3. Data Exploration

Fing the total quantity of the data, missing data, na values, outliers, maximum, minimum

10

```
#Convert the data to appropriate datatype
pandas_df['country'] = pandas_df['country'].astype('category')
pandas_df['child_mort'] = pandas_df['child_mort'].astype(float)
pandas_df['exports'] = pandas_df['exports'].astype(float)
pandas_df['health'] = pandas_df['health'].astype(float)
pandas_df['imports'] = pandas_df['imports'].astype(float)
pandas_df['income'] = pandas_df['income'].astype(float)
pandas_df['inflation'] = pandas_df['inflation'].astype(float)
pandas_df['life_expec'] = pandas_df['life_expec'].astype(float)
pandas_df['total_fer'] = pandas_df['total_fer'].astype(float)
pandas_df['gdpp'] = pandas_df['gdpp'].astype(float)
```

11

```

def explore_data(df):
    results = []
    for col in df.columns:
        # Count missing values (NaN)
        missing_count = df[col].isnull().sum()
        # Count NA values (as a string)
        na_count = df[col].astype(str).str.contains('NA').sum()
        # Count total values
        total_count = len(df[col])
        # Calculate percentage of missing and NA values
        missing_percent = (missing_count / total_count) * 100
        na_percent = (na_count / total_count) * 100
        # Determine data type
        data_type = df[col].dtype

    # --- Additional Checks ---

    # Unique values
    unique_count = df[col].nunique()

    # Detect outliers (using IQR method)
    if pd.api.types.is_numeric_dtype(df[col]): # Only for numeric
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        outlier_count = ((df[col] < lower_bound) | (df[col] > upper_bound)).sum()
    else:
        outlier_count = np.nan # Not applicable for non-numeric

    # Minimum and Maximum values
    if pd.api.types.is_numeric_dtype(df[col]):
        minimum = df[col].min()
        maximum = df[col].max()
        skewness = df[col].skew()
    else:
        minimum = np.nan
        maximum = np.nan
        skewness = np.nan

    results.append([
        col, total_count, missing_count, missing_percent,
        na_count, na_percent, data_type, unique_count,
        outlier_count, minimum, maximum, skewness
    ])

exploration_df = pd.DataFrame(results, columns=[
    "Column", "Total Count", "Missing Count", "Missing (%)",
    "NA Count", "NA (%)", "Data Type", "Unique Values",

```

```

        "Outlier Count", "Minimum", "Maximum", "Skewness"
    ])
    return exploration_df

# Perform data exploration
exploration_results = explore_data(pandas_df)
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
exploration_results.head(100)

```

	Column	Total Count	Missing Count	Missing (%)	NA Count	NA (%)	Data Type	Unique Values	Outlier Count	Minimum	Maximum	Skewness
0	country	167	0	0.0	0	0.0	category	167	NaN	NaN	NaN	NaN
1	child_mort	167	0	0.0	0	0.0	float64	139	4.0	2.6000	208.00	1.450774
2	exports	167	0	0.0	0	0.0	float64	147	5.0	0.1090	200.00	2.445824
3	health	167	0	0.0	0	0.0	float64	147	2.0	1.8100	17.90	0.705746
4	imports	167	0	0.0	0	0.0	float64	151	4.0	0.0659	174.00	1.905276
5	income	167	0	0.0	0	0.0	float64	156	8.0	609.0000	125000.00	2.231480
6	inflation	167	0	0.0	0	0.0	float64	156	5.0	-4.2100	104.00	5.154049
7	life_expec	167	0	0.0	0	0.0	float64	127	3.0	32.1000	82.80	-0.970996
8	total_fer	167	0	0.0	0	0.0	float64	138	1.0	1.1500	7.49	0.967092
9	gdpp	167	0	0.0	0	0.0	float64	157	25.0	231.0000	105000.00	2.218051

4. Data Preparation

Selecting below columns for anlysis, removing columns with high number of NAs:

13

```

select_columns = ['country', 'child_mort', 'exports', 'health', 'imports', 'income', 'inflation', 'life_expec', 'total_fer', 'gdpp']

data_df = pandas_df[select_columns]

data_df.describe()

```

	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp
count	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000
mean	38.270060	41.108976	6.815689	46.890215	17144.688623	7.781832	70.555689	2.947964	12964.155689
std	40.328931	27.412010	2.746837	24.209589	19278.067698	10.570704	8.893172	1.513848	18328.704809
min	2.600000	0.109000	1.810000	0.065900	609.000000	-4.210000	32.100000	1.150000	231.000000
25%	8.250000	23.800000	4.920000	30.200000	3355.000000	1.810000	65.300000	1.795000	1330.000000
50%	19.300000	35.000000	6.320000	43.300000	9960.000000	5.390000	73.100000	2.410000	4660.000000
75%	62.100000	51.350000	8.600000	58.750000	22800.000000	10.750000	76.800000	3.880000	14050.000000
max	208.000000	200.000000	17.900000	174.000000	125000.000000	104.000000	82.800000	7.490000	105000.000000

14

```
# Drop NA records
#data_df.replace(['NA', 'Na'], pd.NA, inplace=True)
#data_df['Thermal comfort'].replace('Na', pd.NA, inplace=True)
data_df = data_df.dropna(how='any')
```

15

```
#Check data after dropping NAs
exploration_results = explore_data(data_df)
exploration_results.head(100)
```

	Column	Total Count	Missing Count	Missing (%)	NA Count	NA (%)	Data Type	Unique Values	Outlier Count	Minimum	Maximum	Skewness
0	country	167	0	0.0	0	0.0	category	167	NaN	NaN	NaN	NaN
1	child_mort	167	0	0.0	0	0.0	float64	139	4.0	2.6000	208.00	1.450774
2	exports	167	0	0.0	0	0.0	float64	147	5.0	0.1090	200.00	2.445824
3	health	167	0	0.0	0	0.0	float64	147	2.0	1.8100	17.90	0.705746
4	imports	167	0	0.0	0	0.0	float64	151	4.0	0.0659	174.00	1.905276
5	income	167	0	0.0	0	0.0	float64	156	8.0	609.0000	125000.00	2.231480
6	inflation	167	0	0.0	0	0.0	float64	156	5.0	-4.2100	104.00	5.154049
7	life_expec	167	0	0.0	0	0.0	float64	127	3.0	32.1000	82.80	-0.970996
8	total_fer	167	0	0.0	0	0.0	float64	138	1.0	1.1500	7.49	0.967092
9	gdpp	167	0	0.0	0	0.0	float64	157	25.0	231.0000	105000.00	2.218051

16

```
#describe the data
data_df.describe()
```

	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp
count	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000
mean	38.270060	41.108976	6.815689	46.890215	17144.688623	7.781832	70.555689	2.947964	12964.155689
std	40.328931	27.412010	2.746837	24.209589	19278.067698	10.570704	8.893172	1.513848	18328.704809
min	2.600000	0.109000	1.810000	0.065900	609.000000	-4.210000	32.100000	1.150000	231.000000
25%	8.250000	23.800000	4.920000	30.200000	3355.000000	1.810000	65.300000	1.795000	1330.000000
50%	19.300000	35.000000	6.320000	43.300000	9960.000000	5.390000	73.100000	2.410000	4660.000000
75%	62.100000	51.350000	8.600000	58.750000	22800.000000	10.750000	76.800000	3.880000	14050.000000
max	208.000000	200.000000	17.900000	174.000000	125000.000000	104.000000	82.800000	7.490000	105000.000000

5. Visualize the Data

Correlation Matrix: Correlations between numerical features.

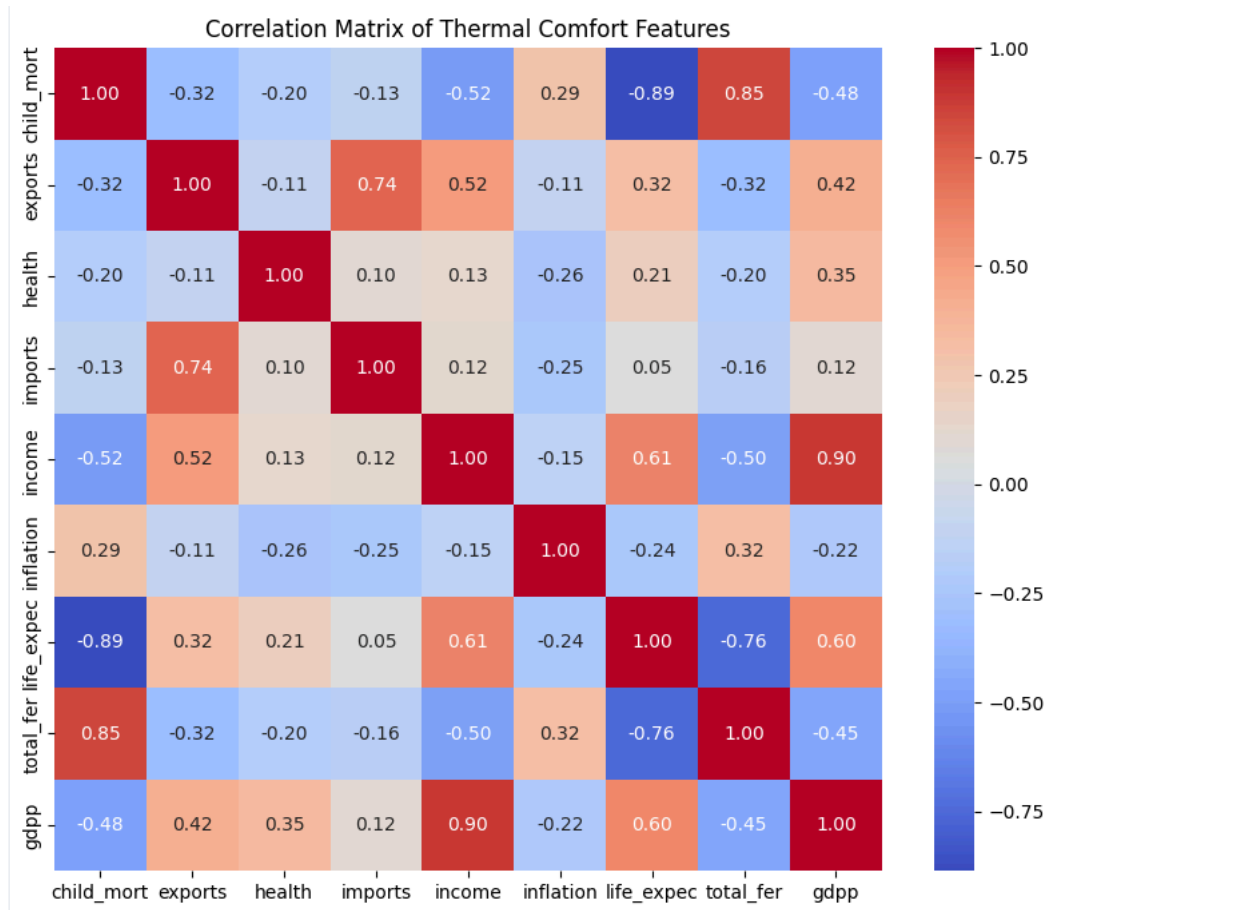
19

```
# Calculate the correlation matrix
corr_matrix = data_df.corr()

# Plotting the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Thermal Comfort Features')
plt.show()
```

/root/.ipykernel/884/command-3007437159044641-3861020466:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.

```
corr_matrix = data_df.corr()
```

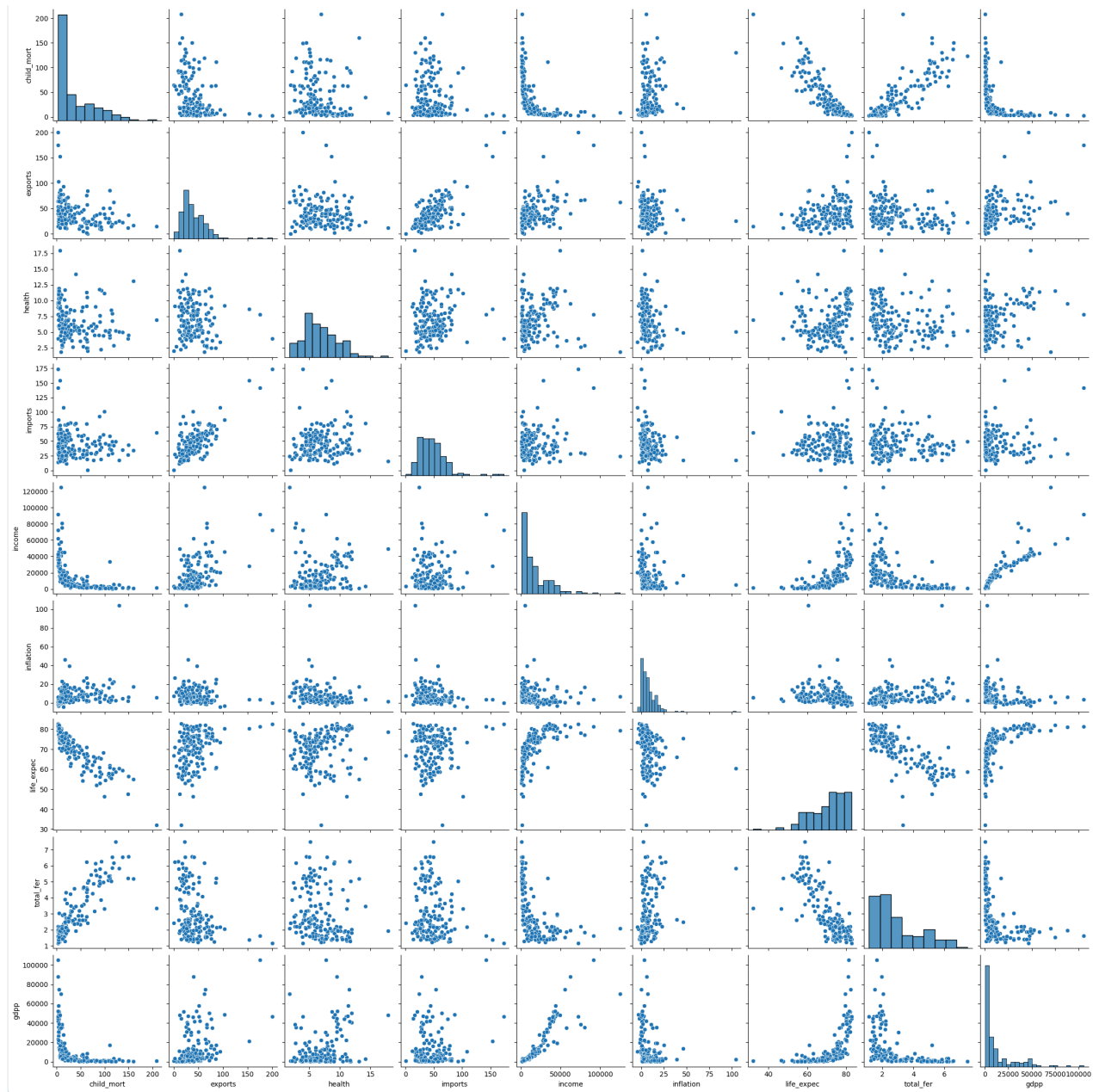



Pair Plot: To see both distributions of individual variables and their relationships

21

```
sns.pairplot(data_df)
plt.show()
```

/databricks/python/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight
self._figure.tight_layout(*args, **kwargs)



Box Plot Distribution of a numerical variable across different categories.

23

```
#plt.figure(figsize=(20, 10))
#sns.boxplot(x='country', y='total_fer', data=data_df)
#plt.xlabel('Season')
#plt.ylabel('Air Temperature (C)')
#plt.title('Air Temperature by Season')
#plt.show()
```

Violin Plots Density of the data distribution.

25

```
#plt.figure(figsize=(20, 10))
#sns.violinplot(x='Season', y='Outdoor monthly air temperature (C)', data=data_df)
#plt.xlabel('Season')
#plt.ylabel('Outdoor monthly air temperature (C)')
#plt.title('Outdoor monthly air temperature (C) by Season')
#plt.show()
```

26

```
#plt.figure(figsize=(20, 10))
#sns.countplot(x='Koppen climate classification', data=data_df)
#plt.xlabel('Koppen Climate Classification')
#plt.ylabel('Count')
#plt.title('Frequency of Koppen Climate Classifications')
#plt.xticks(rotation=45) # Rotate x-axis labels if needed
#plt.show()
```

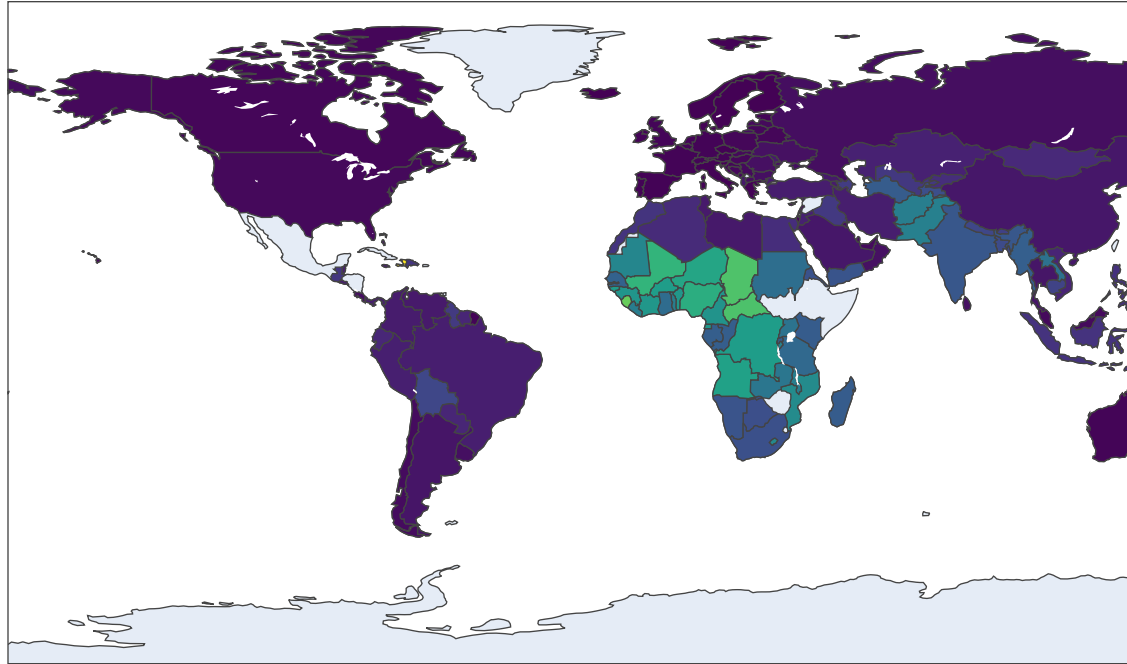
World map Country Vs Child mortality

28

```
plt.figure(figsize=(20, 10))
# Create the world map
fig = px.choropleth(data_df,
                    locations='country',
                    locationmode='country names',
                    color='child_mort', # Choose the column you want to visualize
                    hover_data=['gdp', 'life_expec', 'total_fer'],
                    color_continuous_scale='Viridis', # Choose a color scale
                    title='Child Mortality Rate Around the World',
                    width=1200,
                    height=800)

fig.show()
```

Child Mortality Rate Around the World



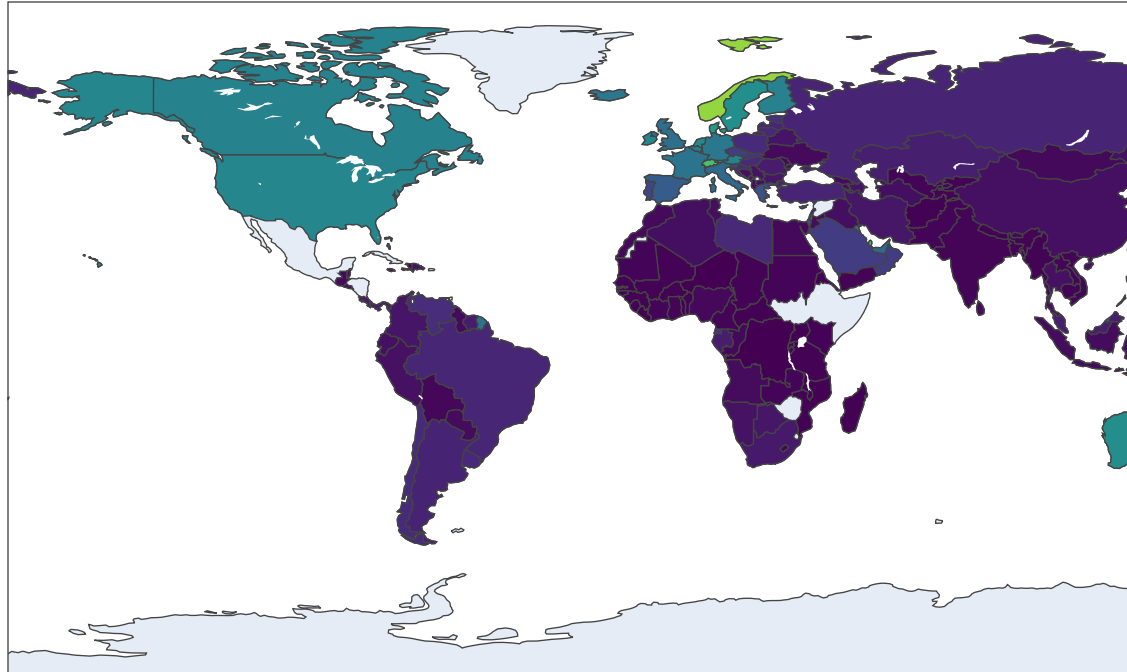
<Figure size 2000x1000 with 0 Axes>

World Map Country Vs GDP

```
# Create the world map
fig = px.choropleth(data_df,
                    locations='country',
                    locationmode='country names',
                    color='gdpp',
                    hover_data=['gdpp', 'life_expec', 'total_fer'],
                    color_continuous_scale='Viridis',
                    title='GDP around the World',
                    width=1200,
                    height=800)

fig.show()
```

GDP around the World



6. Model Training

```
datatrain_df= pandas_df
label = LabelEncoder()
datatrain_df['country'] = label.fit_transform(data_df['country'])
```

33

```
datatrain_df.columns
```

```
Index(['country', 'child_mort', 'exports', 'health', 'imports', 'income',
      'inflation', 'life_expec', 'total_fer', 'gdpp'],
      dtype='object')
```

34

```
column_to_scale = ['country', 'child_mort', 'exports', 'health', 'imports', 'income',
                  'inflation', 'life_expec', 'total_fer', 'gdpp']
scaler = StandardScaler()
for colm in column_to_scale:
    datatrain_df[colm] = scaler.fit_transform(datatrain_df[colm].values.reshape(-1,1))
```

35

```
datatrain_df.describe()
```

	country	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp
count	167.000000	1.670000e+02	1.670000e+02	1.670000e+02	1.670000e+02	1.670000e+02	1.670000e+02	1.670000e+02	1.670000e+02	1.670000e+02
mean	0.000000	-3.722904e-17	2.127373e-16	5.504579e-16	2.765585e-16	-7.977650e-17	-1.063687e-17	3.696311e-16	3.044803e-16	5.850277e-17
std	1.003008	1.003008e+00	1.003008e+00	1.003008e+00	1.003008e+00	1.003008e+00	1.003008e+00	1.003008e+00	1.003008e+00	1.003008e+00
min	-1.721710	-8.871383e-01	-1.500192e+00	-1.827827e+00	-1.939940e+00	-8.603259e-01	-1.137852e+00	-4.337186e+00	-1.191250e+00	-6.968005e-01
25%	-0.860855	-7.466190e-01	-6.333367e-01	-6.922106e-01	-6.914785e-01	-7.174558e-01	-5.666409e-01	-5.927576e-01	-7.639023e-01	-6.366596e-01
50%	0.000000	-4.717981e-01	-2.235279e-01	-1.810007e-01	-1.487432e-01	-3.738080e-01	-2.269504e-01	2.869576e-01	-3.564309e-01	-4.544308e-01
75%	0.860855	7.466190e-01	6.333367e-01	6.922106e-01	6.914785e-01	7.174558e-01	5.666409e-01	5.927576e-01	7.639023e-01	6.366596e-01
max	1.721710	8.871383e-01	1.500192e+00	1.827827e+00	1.939940e+00	8.603259e-01	1.137852e+00	4.337186e+00	1.191250e+00	6.968005e-01

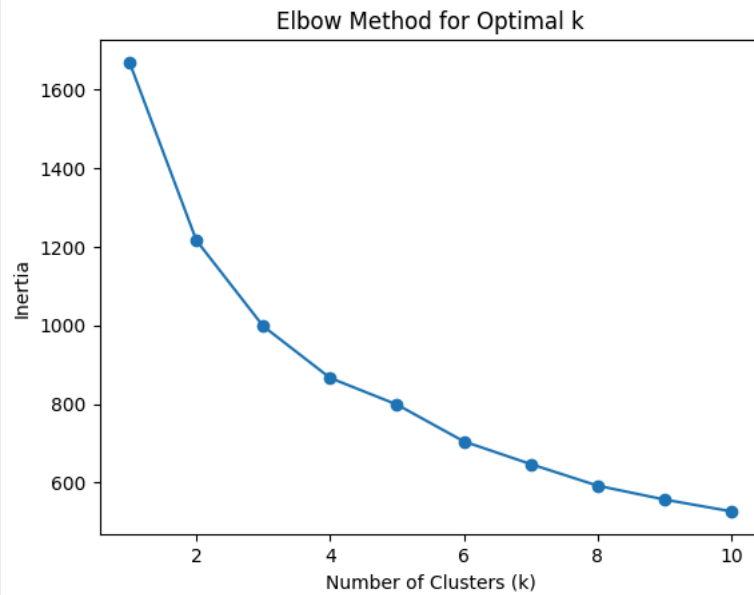
36

```
inertia = []
k_range = range(1, 11)
```

37


```
#k_range = range(1, len(inertia) + 1)

plt.plot(k_range, inertia, marker='o')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()
```



40

```
datatrain_df['cluster'] = kmeans.fit_predict(datatrain_df[column_to_scale])
datatrain_df['cluster'].head(10)
```

```

AttributeError: 'NoneType' object has no attribute 'split'
Exception ignored on calling ctypes callback function: <function _ThreadPoolInfo._find_modules_with_dl_iterate_phdr.<locals>.match_module_callback at 0x7f815af1a7a0>
Traceback (most recent call last):
  File "/databricks/python/lib/python3.11/site-packages/threadpoolctl.py", line 400, in match_module_callback
    self._make_module_from_path(filepath)
0    0
1    6
2    6
3    0
4    6
5    6
6    6
7    2
8    2
9    6
Name: cluster, dtype: int32
```

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Assuming you have your data in a variable called 'data' and the column to scale is 'column_to_scale'
# If not, replace these with your actual data and column name

k_range = range(2, 11) # Check silhouette score for k from 2 to 10
silhouette_scores = []

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=48)
    cluster_labels = kmeans.fit_predict(data_train_df[column_to_scale])
    silhouette_avg = silhouette_score(data_train_df[column_to_scale], cluster_labels)
    silhouette_scores.append(silhouette_avg)

# Create a DataFrame for the table
results_df = pd.DataFrame({'k': k_range, 'Silhouette Score': silhouette_scores})

# Display the table
print(results_df.to_markdown(index=False, numalign="left", stralign="left"))

# Plot the results
plt.plot(k_range, silhouette_scores, marker='o')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score for Different k')
plt.show()
```

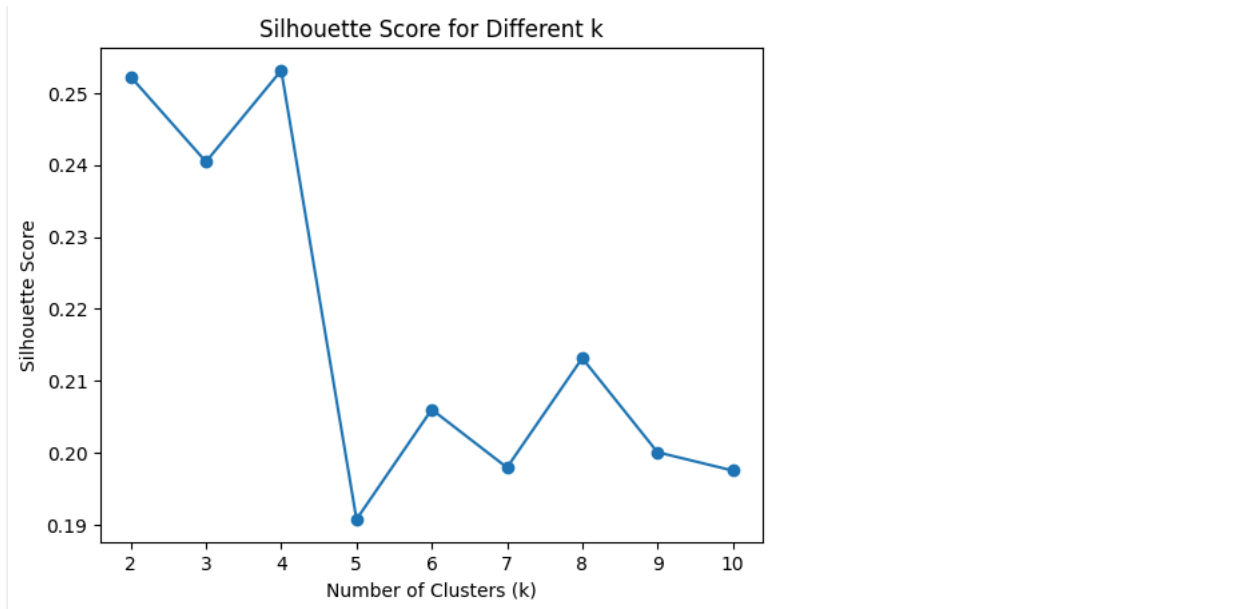
```
Exception ignored on calling ctypes callback function: <function _ThreadPoolInfo._find_modules_with_dl_iterate_phdr.<locals>.match_module_callback at 0x7f815af1ba60>
```

```
Traceback (most recent call last):  
  File "/databricks/python/lib/python3.11/site-packages/threadpoolctl.py", line 400, in match_module_callback  
    self._make_module_from_path(filepath)  
  File "/databricks/python/lib/python3.11/site-packages/threadpoolctl.py", line 515, in _make_module_from_path  
    module = module_class(filepath, prefix, user_api, internal_api)  
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^  
  File "/databricks/python/lib/python3.11/site-packages/threadpoolctl.py", line 606, in __init__  
    self.version = self.get_version()  
                  ^^^^^^^^^^^^^^^^^  
  File "/databricks/python/lib/python3.11/site-packages/threadpoolctl.py", line 646, in get_version  
    config = get_config().split()  
            ^^^^^^^^^^^^^^^^^
```

```
AttributeError: 'NoneType' object has no attribute 'split'
```

```
Exception ignored on calling ctypes callback function: <function _ThreadPoolInfo._find_modules_with_dl_iterate_phdr.<locals>.match_module_callback at 0x7f815af1ba60>
```

```
Traceback (most recent call last):  
  File "/databricks/python/lib/python3.11/site-packages/threadpoolctl.py", line 400, in match_module_callback  
    self._make_module_from_path(filepath)  
  File "/databricks/python/lib/python3.11/site-packages/threadpoolctl.py", line 515, in _make_module_from_path
```



Based on Elbow method as well scores calculated from Silhouette , K =4 suggest well defined clusters

43

```
# Assuming you have your data in a variable called 'data' and the column to scale is 'column_to_scale'
# If not, replace these with your actual data and column name

# Set the optimal k value (you can change this based on your analysis)
optimal_k = 4

# Fit the KMeans model with the optimal k
kmeans = KMeans(n_clusters=optimal_k, random_state=48)
data_df['cluster_optimized'] = kmeans.fit_predict(data_train_df[column_to_scale]) # Assign cluster labels to a new column 'clus
```

```
File "/databricks/python/lib/python3.11/site-packages/threadpoolctl.py", line 606, in __init__
    self.version = self.get_version()
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/databricks/python/lib/python3.11/site-packages/threadpoolctl.py", line 646, in get_version
    config = get_config().split()
    ^^^^^^^^^^^^^^^^^^^^^^^^^
AttributeError: 'NoneType' object has no attribute 'split'
Exception ignored on calling ctypes callback function: <function _ThreadPoolInfo._find_modules_with_dl_iterate_phdr.<locals>.match_module_callback at 0x7f81539544a0>
Traceback (most recent call last):
  File "/databricks/python/lib/python3.11/site-packages/threadpoolctl.py", line 400, in match_module_callback
    self._make_module_from_path(filepath)
```

44

```
data_df.columns
```

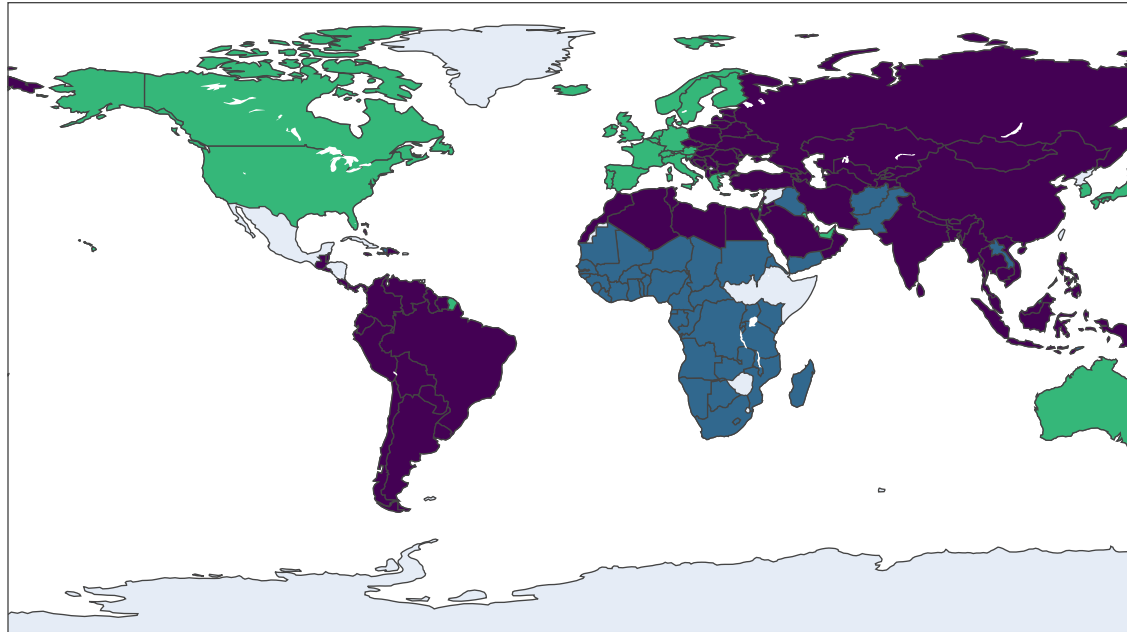
```
Index(['country', 'child_mort', 'exports', 'health', 'imports', 'income',
      'inflation', 'life_expec', 'total_fer', 'gdpp', 'cluster_optimized'],
      dtype='object')
```

45

```
# Create the world map
fig = px.choropleth(data_df,
                    locations='country',
                    locationmode='country names',
                    color='cluster_optimized',
                    hover_data=['gdpp', 'life_expec', 'total_fer'],
                    color_continuous_scale='Viridis',
                    title='Visulizing the countries based on selected clusters',
                    width=1200,
                    height=800)

fig.show()
```

Visualizing the countries based on selected clusters



7. Evaluation Matrix

```
# Assuming Y_test and Y_pred are your true and predicted labels respectively

results = []
for k in range(2, 11): # Example range
    kmeans = KMeans(n_clusters=k, random_state=48)
    kmeans.fit(datatrain_df[column_to_scale])

    inertia = kmeans.inertia_
    labels = kmeans.labels_
    silhouette_avg = silhouette_score(datatrain_df[column_to_scale], labels)

    results.append([k, inertia, silhouette_avg])

# Accuracy
#accuracy = accuracy_score(Y_test, Y_pred)
#print(f"Accuracy: {accuracy}")

# Precision
#precision = precision_score(Y_test, Y_pred, average='macro') # or 'micro', 'weighted'
#print(f"Precision: {precision}")

# Recall
#recall = recall_score(Y_test, Y_pred, average='macro') # or 'micro', 'weighted'
#print(f"Recall: {recall}")

# F1-score
#f1 = f1_score(Y_test, Y_pred, average='macro') # or 'micro', 'weighted'
#print(f"F1-score: {f1}")

# Confusion Matrix
#cm = confusion_matrix(Y_test, Y_pred)
#print("Confusion Matrix:")
#print(cm)

# Classification Report
#cr = classification_report(Y_test, Y_pred)
#print("Classification Report:")
#print(cr)
```



```

exception ignored on calling ctypes callback function: <function _inreadpoolinto._find_modules_with_dlopen_iterate_pndr.<locals>.match_module_callback at 0x7f81539127a0>
Traceback (most recent call last):
  File "/databricks/python/lib/python3.11/site-packages/threadpoolctl.py", line 400, in match_module_callback
    self._make_module_from_path(filepath)
  File "/databricks/python/lib/python3.11/site-packages/threadpoolctl.py", line 515, in _make_module_from_path
    module = module_class(filepath, prefix, user_api, internal_api)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/databricks/python/lib/python3.11/site-packages/threadpoolctl.py", line 606, in __init__
    self._version = self._get_version()

```

48

```

# Print the results in a table
headers = ["k", "Inertia", "Silhouette Score"]
print(tabulate(results, headers=headers, tablefmt="fancy_grid")) # Use "fancy_grid" for a grid-like table

```

k	Inertia	Silhouette Score
2	1216.56	0.252237
3	998.052	0.240437
4	866.288	0.253132
5	798.433	0.190767
6	704.634	0.206032
7	646.887	0.197967
8	591.877	0.213158
9	556.605	0.20009
10	526.484	0.197546

As we can see from the results, K=4 provides best results. At K=4, Silhouette Score is maximum and the slope on elbo drops down significantly.

