#Final Project
#Group Number: group 51
#Student name: HaoYu Tan, WeiGuang Chen, JianHeng Chen
#Student ID: 1006147386, 999187259,1005680746
#Instructor: Dr. Maher Elshakankiri
#Course code: INF 1340
#Course name: Programming for Data Science
#Program: MI
#Faculty of Information
#University of Toronto

API Documentation

This API provides functions for analyzing socio-economic data related to the Human Development Index (HDI), Life Expectancy, Mean Years of Schooling, Expected Years of Schooling, and Gross National Income(GNI) per capita. It offers various functionalities, including data summarization, histogram generation, sorting, boxplot creation, statistical analysis, outlier detection, correlation analysis, and linear regression modeling.

Requirements
—Python
—Libraries: numpy, pandas, matplotlib, seaborn, scipy.stats, sklearn.

Functions in the Project
—Build-in Functions for comprehensive analysis of Human Development Index dataset:
statistical analysis, data visualization, sorting and ranking, outliers detection, correlation analysis, mapping and quantile analysis, linear regression modeling,

Detail of API Documentation

1. Data Cleaning:

Purpose:
The script is designed to clean and preprocess data from an Excel file for analysis, focusing on enhancing data readability and structure.

Input:
Excel file named 'HDR.xlsx'.
Output:
Cleaned data saved as a CSV file 'cleaned_data_final.csv'.

Arguments:
header: Specifies the row number to use as the column names.
columns: A list of column names to be dropped from the DataFrame.
new_names: A dictionary mapping old column names to new ones.
errors='coerce': Instructs pandas to set invalid parsing as NaN.

Returns:
A DataFrame with cleaned and structured data ready for analysis.

Code Example:

```
import pandas as pd
import numpy as np
df = pd.read_excel('HDR.xlsx', header=3)
df.drop(columns=['Unnamed: 3', 'Unnamed: 5', 'Unnamed: 7', 'Unnamed: 9', 'Unnamed: 11', 'Unnamed: 13'], inplace=True)
new_names = {...}  # Dictionary with new column names
df.rename(columns=new_names, inplace=True)
df['index'] = pd.to_numeric(df['index'], errors='coerce')
df = df[pd.notna(df['index'])]
df.drop('index', axis=1, inplace=True)
df.to_csv('cleaned_data_final.csv', index=False)
```

Descriptive Analysis:

2. Descriptive Analysis (summarize_statistics)

Purpose: Provides a basic statistical summary of the dataset excluding specific columns.

Input:
file_path (str): Path to the CSV file (e.g., “cleaned_data_final.csv").
Processing:
Reads the CSV file into a DataFrame.
Sets the maximum columns displayed.
Drops specified columns like 'HDI Rank

Output:
First 10 rows of the DataFrame.
Data types of each column.
Summary statistics (count, mean, std, min, quartiles, max) for each column, excluding specified columns like 'HDI Rank’.

Arguments:
file_path (str): Path to the CSV file.

Returns: None. Prints the DataFrame head, data types, and descriptive statistics.

Code Example:

```
import pandas as pd
def summarize_statistics(file_path):
    data = pd.read_csv(file_path, index_col='Country')
    pd.set_option('display.max_columns', None)
    print(data.head(10))
    print(data.dtypes)
    print(data.drop('HDI Rank', axis=1).describe())
```

3. Data Visualization (create_histograms)

Purpose: Generates histograms for selected columns in the dataset.

Input:
data (DataFrame): The DataFrame containing the data.
columns (list): List of column names to generate histograms for.

Output:
Histogram plots for each specified column.

Arguments:
data (DataFrame): The DataFrame containing the data.

Returns: None. Displays histograms.

Code Example:

```
def create_histograms(data, columns):
    for column in columns:
        data[[column]].plot(kind='hist', title=f'{column} histogram')
```

4. Statistical Analysis (statistical_analysis)

Purpose: Performs statistical analysis including histogram plotting, Q-Q plots, and calculation of skewness and kurtosis.

Input:
data (DataFrame): The DataFrame containing the data.
column (str): The column name for which the analysis is performed.

Output:
Histogram, Q-Q plot, skewness, kurtosis, and Shapiro-Wilk test results for the specified column.

Arguments:
data (DataFrame): The DataFrame containing the data.

Returns: None. Displays plots and prints statistics.

Code Example:

```
import matplotlib.pyplot as plt
import scipy.stats as stats

def statistical_analysis(data, column):
    plt.hist(data[column], bins=20, edgecolor='black')
    stats.probplot(data[column], dist="norm", plot=plt)
    skewness = data[column].skew()
    kurtosis = data[column].kurtosis()
    shapiro_test = stats.shapiro(data[column])
print(f"Skewness: {skewness}, Kurtosis: {kurtosis}, Shapiro-Wilk   Test: {shapiro_test}”)
```

5. Boxplots and Outliers (create_boxplots)

Purpose: Creates box plots for different indicators and identifies outliers.

Input:
data (DataFrame): The DataFrame containing the data.
columns (list): List of column names to create box plots for.

Output:
Box plots for each specified column.

Arguments:
data (DataFrame): The DataFrame containing the data.

Returns: None. Displays box plots.

Code Example:

```
import seaborn as sns
def create_boxplots(data, columns):
    for column in columns:
        sns.boxplot(data=data, y=column)
        plt.title(f'Box Plot of {column}')
        plt.show()
```

6. Sorting and Ranking (sort_and_rank)

Purpose: Sorts and ranks countries based on HDI within different income groups.

Input:
data (DataFrame): The DataFrame containing the data.

Output:
Sorted and ranked data, top 10 countries in each income group based on HDI.

Arguments:
data (DataFrame): The DataFrame containing the data.

Returns: None. Prints top 10 countries in each income group.

Code Example:

```
data['Income Group'] = ['High' if gni > 50000 else 'Middle' if gni > 20000 else 'Low' for gni in data['GNI per capita']]
sorted_data = pd.concat([sorted_data, income_group_data.sort_values('HDI', ascending=False)], ignore_index=True)
```

Diagnostic Analysis

7. Diagnostic Analytics - Top 10 Country Similarities (calculate_top10_statistics)

Purpose: Analyze top 10 countries in each income group for similarities in HDI, life expectancy, etc.

Input:

CSV file containing HDI data (cleaned_data_final.csv).

Output:
A DataFrame with mean, standard deviation, and range for HDI, life expectancy, schooling years, and GNI per capita.

Arguments:
data (DataFrame): DataFrame containing the socio-economic data.
Returns:
Dictionary with statistics for the top 10 countries in each income group.

Code Example:
```
# Load data and assign income groups
data = pd.read_csv('cleaned_data_final.csv')
data['Income Group'] = ['High' if gni > 50000 else 'Middle' if gni > 20000 else 'Low' for gni in data['Gross national income (GNI) per capita']]

# Sort and analyze top 10 countries in each group
sorted_data = data.sort_values(['Income Group', 'Human Development Index (HDI)'], ascending=[True, False])
top10_stats = calculate_top10_statistics(sorted_data)
stats_df = pd.DataFrame(top10_stats)
```

8. Outlier Detection (find_outliers)

Purpose: Identify outliers in the dataset for specified variables.

Input:
DataFrame data.

Output:
Outlier information printed for each variable.

Arguments:
data (DataFrame): DataFrame containing the data.
column (str): Name of the column to check for outliers.

Returns:
DataFrame containing outliers.

Code Example:

```
for variable in ['HDI', 'Life expectancy', 'Years of schooling', 'GNI per capita']:
    outliers = find_outliers(data, variable)
    print(f"Outliers in {variable}:")
    for index, row in outliers.iterrows():
        print(f"   Country: {row['Country']}, GNI Rank: {row['GNI Rank']}, HDI Rank: {row['HDI Rank']}")
```

9. Correlation Analysis (correlation_analysis)

Purpose: Calculate correlation between HDI and other variables.

Input:
DataFrame data.

Output:
Correlation values between HDI and other variables.

Arguments:
data (DataFrame): DataFrame containing the data.

Returns:
Series of correlation values.

Code Example:

```
correlation_matrix = data[['HDI', 'Life expectancy', 'Years of schooling', 'GNI per capita']].corr()
hdi_correlation = correlation_matrix['HDI']
print("Correlation with HDI:")
print(hdi_correlation)
```

10. Ranking Analysis

Purpose: Calculates and assigns ranks based on life expectancy and schooling, and compares these with HDI ranks.

Input:
data (DataFrame): The DataFrame containing HDI and related socio-economic data.

Output:

Updated DataFrame with new columns for life expectancy and schooling ranks, and their comparison with HDI ranks.

Arguments:
None. The function operates directly on the DataFrame.

Returns:
None. The DataFrame is modified in place.

Code Example:
data['Life Expectancy Rank'] = data['Life expectancy at birth'].rank(ascending=False)
Similar for other rankings and categorizations

11. Categorization Analysis

Purpose: Defines quantile thresholds for HDI, life expectancy, and schooling to categorize data.

Input:
data (DataFrame): The DataFrame containing HDI and related socio-economic data.

Output:
Categorized DataFrame with new columns for HDI, life expectancy, and schooling categories.
Arguments:
None. The function operates directly on the DataFrame.

Returns:
None. The DataFrame is modified in place.

Code Example:

hdi_quantiles = data['Human Development Index (HDI)'].quantile([0.25, 0.50, 0.75])
data['HDI Category'] = pd.cut(data['Human Development Index (HDI)'], bins=[0, ...], labels=['Low', 'Medium', 'High', 'Very High'])
Similar categorization for life expectancy and schooling

12. Linear Regression Analysis (linear_regression)

Purpose
Performs linear regression to analyze the relationship between 'Life expectancy at birth' and the Human Development Index (HDI).

Input:
data: DataFrame containing the HDI and socio-economic indicators.

Output:
Coefficient of determination (R^2), model intercept, and slope are printed. A scatter plot with the regression line is displayed.

Arguments
x1:the independent variable for example ('Life expectancy at birth').
y: Series or array of the dependent variable (HDI).

Returns
The fitted Linear Regression model.

Code Example:

```
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Linear Regression model
model = LinearRegression().fit(data[['Life expectancy at birth']].values, data['HDI'])

# Output and plot
print('R²:', model.score(data[['Life expectancy at birth']].values, data['HDI']))
plt.scatter(data[['Life expectancy at birth']], data['HDI'])
plt.plot(data[['Life expectancy at birth']], model.predict(data[['Life expectancy at birth']]), color='red')
plt.xlabel('Life Expectancy')
plt.ylabel('HDI')
plt.show()
```

Predictive Analytics

13. Predictive analytics(using train-test model)

Purpose:
Evaluates the relationship between various socio-economic indicators and the Human Development Index (HDI) using linear regression.

Input:
x: Independent variable(s) data.
y: Dependent variable (HDI) data.

Output:
Printed linear regression model details, predictions for the test dataset, and the model's R^2 score.

Arguments:
x_train, x_test: Training and testing data for independent variables.
y_train, y_test: Training and testing data for the dependent variable.
test_size: Fraction of the data to be used for testing.
random_state: Seed used by the random number generator.

Returns:
The fitted Linear Regression model.
HDI value predictions for the test set.
R^2 score indicating model performance.

Code Example:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Data split and model fitting
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=1)
model = LinearRegression().fit(x_train, y_train)

# Model evaluation
y_pred = model.predict(x_test)
print('Model:', model)
print('Predicted HDI:', y_pred)
print('R² score:', model.score(x_test, y_test))
```