```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Sav
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
import pandas as pd
import numpy as np
# Create a synthetic dataset
data = {
  'Student_ID': range(1, 101),
  'Gender': np.random.choice(['Male', 'Female'], 100),
  'Age': np.random.randint(18, 25, 100),
  'Math_Score': np.random.randint(0, 101, 100),
  'English_Score': np.random.randint(0, 101, 100),
  'Attendance': np.random.uniform(60, 100, 100),
  'Final_Grade': np.random.randint(50, 100, 100),
  'Family_Income': np.random.randint(20, 100, 100)
}
# Create a DataFrame
df = pd.DataFrame(data)
# Introduce missing values (for demonstration purposes)
df.loc[::10, 'Math_Score'] = np.nan  # Missing values in Math_Score
df.loc[::5, 'Family_Income'] = np.nan  # Missing values in

# Introduce inconsistencies (age > 30, attendance > 100)
df.loc[5, 'Age'] = 32
df.loc[15, 'Attendance'] = 105
# Introduce outliers
df.loc[20, 'Math_Score'] = 200  # Outlier in Math_Score
df.loc[30, 'English_Score'] = -5  # Outlier in English_Score

# Display first few rows of the dataset
# Check for missing values
missing_values = df.isnull().sum()
import pandas as pd
import numpy as np
# Create a synthetic dataset
data = {
  'Student_ID': range(1, 101),
  'Gender': np.random.choice(['Male', 'Female'], 100),
  'Age': np.random.randint(18, 25, 100),
  'Math_Score': np.random.randint(0, 101, 100),
  'English_Score': np.random.randint(0, 101, 100),
  'Attendance': np.random.uniform(60, 100, 100),
  'Final_Grade': np.random.randint(50, 100, 100),
  'Family_Income': np.random.randint(20, 100, 100)
}
# Create a DataFrame
df = pd.DataFrame(data)
# Introduce missing values (for demonstration purposes)
df.loc[::10, 'Math_Score'] = np.nan  # Missing values in Math_Score
df.loc[::5, 'Family_Income'] = np.nan  # Missing values in

# Introduce inconsistencies (age > 30, attendance > 100)
df.loc[5, 'Age'] = 32
df.loc[15, 'Attendance'] = 105
# Introduce outliers
df.loc[20, 'Math_Score'] = 200  # Outlier in Math_Score
df.loc[30, 'English_Score'] = -5  # Outlier in English_Score

# Display first few rows of the dataset
# Check for missing values
```

```
missing_values = df.isnull().sum()
# Impute missing values for numeric variables with mean
df['Math_Score'] = df['Math_Score'].fillna(df['Math_Score'].mean())
df['Family_Income'] = df['Family_Income'].fillna(df['Family_Income'].mean())
# For categorical variable, impute with mode (most frequent value)
df['Gender'] = df['Gender'].fillna(df['Gender'].mode()[0])
# Fix inconsistencies
df['Age'] = df['Age'].apply(lambda x: x if x <= 30 else 25)  # Cap age

df['Attendance'] = df['Attendance'].apply(lambda x: x if x <= 100 else 100)  # Cap attendance to 100
# Check the dataset after cleaning
df.head()
df['Gender'] = df['Gender'].fillna(df['Gender'].mode()[0])
# Fix inconsistencies
df['Age'] = df['Age'].apply(lambda x: x if x <= 30 else 25)  # Cap age

df['Attendance'] = df['Attendance'].apply(lambda x: x if x <= 100 else
100)  # Cap attendance to 100
# Check the dataset after cleaning
df.head()
```

```
# Function to detect outliers using IQR
def detect_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[column] < lower_bound) | (df[column] > upper_bound)]

# Check for outliers in numeric columns
outliers_math = detect_outliers(df, 'Math_Score')
outliers_english = detect_outliers(df, 'English_Score')
# Display outliers
outliers_math, outliers_english

# Handle outliers by capping them
def cap_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df[column] = df[column].apply(lambda x: min(max(x, lower_bound), upper_bound))

# Cap outliers in Math_Score and English_Score
cap_outliers(df, 'Math_Score')
cap_outliers(df, 'English_Score')
# Check for outliers after capping
detect_outliers(df, 'Math_Score'), detect_outliers(df, 'English_Score')
```

```
# Before capping
print(df[['Math_Score', 'English_Score']].describe())

# Apply capping (your cap_outliers function)

# After capping
print(df[['Math_Score', 'English_Score']].describe())
```

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Apply log transformation to Family_Income
df['Log_Family_Income'] = np.log(df['Family_Income'])

# Check the distribution before and after transformation
plt.figure(figsize=(12, 6))

# Before transformation
plt.subplot(1, 2, 1)
sns.histplot(df['Family_Income'], kde=True, color='blue')
plt.title('Family Income (Original)')

# After transformation
```

```
plt.subplot(1, 2, 2)
sns.histplot(df['Log_Family_Income'], kde=True, color='green')
plt.title('Log Transformed Family Income')

plt.tight_layout()  # Adjusts subplot params for a tight layout
plt.show()
```