Extracting the file First and Displaying It:

Code:

```
import pandas as pd


# Load the CSV file into a DataFrame
df = pd.read_csv('https://drive.google.com/file/d/11_-FZbtU0PcrP87Xz3Mf5WMEYJ-Sd-Z1/view?usp=drive_link')


# Display the head of the DataFrame
df_head = df.head()
print(df_head)


# Display basic statistics of the DataFrame
df_description = df.describe(include='all', datetime_is_numeric=True)
print(df_description)
```

1. Missing Values:

Question**:** Are there any missing values in the dataset, and if so, how should they be handled for each indicator?

Code:

```
# Check for missing values in the dataset
df_missing_values = df.isnull().sum()
print(df_missing_values)
```

Description

The dataset has missing values in the 'Age' and 'Rating' columns. Here are some strategies to handle them:

Age: With 7 missing values, options include:

Removing rows with missing 'Age' if the dataset is large enough.

Imputing missing values using statistical methods like mean, median, or mode.

Using model-based imputation if 'Age' is correlated with other variables.

Rating: With only 1 missing value, options include:

Removing the row with the missing 'Rating'.

Imputing the missing value with the mean or median of 'Rating'.

The choice of strategy should consider the size of the dataset, the importance of the missing values, and the potential impact on the analysis.

2. Data Types:
Question: What are the data types of each indicator, and do they align with their expected types (e.g., numerical, categorical)?

Code:

```
# Check the data types of each column in the dataframe

df_dtypes = df.dtypes

print(df_dtypes)
```

Output

The data types for each indicator are as follows:

Index: Integer (int64)

Age: Floating point number (float64)

Salary: Object (string)

Rating: Floating point number (float64)

Location: Object (string)

Established: Integer (int64)

Easy Apply: Object (string)

3. Outliers:
Question: Identify potential outliers in numerical indicators (e.g., Age, Salary, Rating). Should outliers be removed or adjusted?

In age , Salary and rating there are no outliers given .

4. Salary Formatting:
Question: Examine the format of the Salary column. Does it require any formatting or standardization for consistent analysis?

The 'Salary' column contains salary ranges in a string format with a dollar sign and a 'k' to denote thousands. For consistent analysis, this column requires formatting to convert these ranges into a numerical format. Here are some steps we could take:

- Remove the dollar sign and 'k' characters.
- Split the range into two separate numbers.
- Convert the strings to integers.
- Optionally, calculate the average of the range for each entry to represent the salary as a single number.

Code:

```
from tqdm import tqdm
import pandas as pd
tqdm.pandas()

# Convert the salary column into a numerical format
def convert_salary_range_to_avg(salary_str):
    # Remove the dollar sign and split the range
    range_vals = salary_str.replace('$', '').replace('k', '').split('-')
    # Convert the split values to integers and calculate the average
    avg_salary = sum([int(val) for val in range_vals]) / len(range_vals)
    return avg_salary

df['Salary'] = df['Salary'].progress_apply(convert_salary_range_to_avg)

# Display the head of the dataframe to show the updated 'Salary' column
print(df.head())
```

Description

The 'Salary' column has been successfully converted to a numerical format, with each salary range replaced by its average value. Here are the first few entries with the updated 'Salary' values:

Index 0: Average Salary is 71.5

Index 1: Average Salary is 60.5

Index 2: Average Salary is 83.0

Index 3: Average Salary is 29.5

The dataset is now ready for consistent numerical analysis of the 'Salary' indicator.

5. Location Standardization:

Question: Check the consistency of location entries. Do they need standardization, and how can this be achieved?

10. Handling Special Characters:

Question: Examine all text-based columns (e.g., Location). Are there special characters or inconsistencies that need cleaning?

['India,In' 'New York,Ny' 'India In' 'Australia Aus']

The 'Location' entries show some inconsistencies and may require standardization. For example, 'India,In' and 'India In' refer to the same location but are formatted differently. To achieve consistency:

Standardize the country and state abbreviation separation by choosing one format (e.g., comma, space, or another character).

Ensure the country and state abbreviations are consistent (e.g., 'In' vs 'IN' for India).

Replace any space (' ') with a comma (',') if it's not already present.

Ensure that country and state abbreviations are consistently uppercase.

Trim any trailing spaces.

Code:

```
# Standardize the 'Location' column


def standardize_location(location):
    # Replace space with a comma if it's not already present
    location = location.replace(' ', ',')
    # Split by comma to separate country and state
    parts = location.split(',')
    # Ensure that country and state abbreviations are uppercase and strip any trailing spaces
    standardized_parts = [part.strip().upper() for part in parts]
    # Rejoin the parts with a comma
    standardized_location = ','.join(standardized_parts)
    return standardized_location
```

```
df['Location'] = df['Location'].progress_apply(standardize_location)


# Display the unique values of the 'Location' column after standardization
print(df['Location'].unique())
```

Description

['INDIA,IN' 'NEW,YORK,NY' 'AUSTRALIA,AUS']

The 'Location' column has been standardized. The unique values now show consistent formatting with commas separating the country and state abbreviations, which are all in uppercase. This standardization will allow for more consistent analysis of location-based data.

6. Established Column:

Question: Explore the Established column. Are there any inconsistencies or anomalies that need to be addressed?

The 'Established' column contains some anomalies, such as entries with a value of -1, which is not a valid year for establishment. These entries may need to be addressed, either by replacing them with a valid year, a placeholder for missing data, or removing them, depending on the context and the importance of this column in the analysis.

Then replacing the -1.0 with valid year in established column

Code:

```
# Replace the -1.0 value with a valid year in the 'Established' column


def replace_negative_year(year):
    if year == -1.0:
        # Replace -1.0 with a valid placeholder year (e.g., 1900)
        return 1900
    else:
        return year


df['Established'] = df['Established'].progress_apply(replace_negative_year)


# Display the unique values of the 'Established' column after replacement
```

```
print(df['Established'].unique())
```

7. Easy Apply Indicator:
Question: Analyze the Easy Apply column. Does it contain boolean values or need transformation for better analysis?

12. Easy Apply Transformation:
Question: If the Easy Apply column contains non-boolean values, how can it be transformed into a usable format?

Code:

```
['TRUE' '-1']
```

The 'Easy Apply' column contains two unique values: 'TRUE' and '-1'. The value 'TRUE' is a boolean, but '-1' is not and may need to be transformed into a boolean for consistent analysis.

```
# Transform the -1 value in the 'Easy Apply' column to a valid boolean value


def transform_easy_apply(value):
    if value == '-1':
        return False
    else:
        return True


df['Easy Apply'] = df['Easy Apply'].progress_apply(transform_easy_apply)


# Display unique values in the 'Easy Apply' column after transformation
print(df['Easy Apply'].unique())
```

Now the 'Easy Apply' column has been transformed to contain only valid boolean values: True and False.

8. Rating Range:
Question: Investigate the range of values in the Rating column. Does it fall within expected rating scales, and how should outliers be treated?

The 'Rating' column contains values ranging from -1.0 to 7.8, with some entries as (not a number). Typically, rating scales do not include negative values and are within a range of 0 to 5

Code:

```
# Scale the rating value between 0 to 5 using the mean
```

```
# Calculate the mean of the 'Rating' column
rating_mean = df['Rating'].mean()

# Scale the rating values using the mean
scaled_ratings = df['Rating'].apply(lambda x: min(5, max(0, x * (5/rating_mean))) if x != -1 else -1)

# Display the scaled rating values
scaled_ratings
```

Description

The 'Rating' values have been scaled to a range of 0 to 5, based on the mean of the original ratings, with the exception of the original -1 values which have been preserved as -1 in the scaled data.

Now replacing the -1.0 value with the valid mean value .

Code:

```
# Calculate the mean of the valid ratings (excluding -1)
valid_ratings_mean = df[df['Rating'] != -1]['Rating'].mean()

# Replace -1 values with the calculated mean
import numpy as np
df['Rating'] = df['Rating'].replace(-1, np.nan).fillna(valid_ratings_mean)
# Check the replacement
print(df['Rating'].describe())
```

Description

The -1.0 values in the 'Rating' column have been successfully replaced with the mean of the valid ratings. The updated statistics for the 'Rating' column are now displayed.

9. Age Distribution:

Question: Check the distribution of values in the Age column. Are there any unusual entries, and how might they impact analysis?

The 'Age' column has a count of 22 entries. There are unique values present, including , which indicates missing data. The presence of ages such as 13 might be unusual depending on the context, as it could represent underage individuals in a dataset that should consist of adults. These entries could

impact analysis, especially if the dataset is meant to represent a population of a certain age range, such as working-age adults.

Now correcting the missing value and replacing the minimum value by 18 .

Code:

```
# Correct the missing value and ensure the minimum value is 18


# Replace missing values in the 'Age' column with the mean of the existing valid ages

valid_ages_mean = df['Age'].mean()

df['Age'] = df['Age'].fillna(valid_ages_mean)


# Ensure the minimum value is 18

import numpy as np

df['Age'] = np.maximum(df['Age'], 18)


# Check the updated 'Age' column

age_summary_updated = df['Age'].describe()

age_summary_updated
```

Description

The 'Age' column has been updated to correct missing values and ensure the minimum age is 18. The mean age is now approximately 39.22 years, with the youngest age being 18.

11. Data Integrity:

Question: Ensure data integrity by cross-referencing entries. For instance, does the Established column align with the Age column?

Code:

```
# Cross-reference the 'Established' column with the 'Age' column

# Display the 'Established' and 'Age' columns

established_age = df[['Established', 'Age']]

established_age
```

Description

To ensure data integrity, we've cross-referenced the 'Established' and 'Age' columns. The data shows some entries that may require further investigation, such as an 'Established' year that seems too old relative to the 'Age' of the individual or entity. Further steps should be taken to validate these entries.

Code:

```
# Calculate the age that each entry should be if they were established in the 'Established' year

# Current year is 2023


# Calculate the expected age

expected_age = 2023 - df['Established']


# Create a boolean mask where the expected age is less than 18 or greater than the 'Age' column

invalid_entries_mask = (expected_age < 18) | (expected_age > df['Age'])


# Apply the mask to the dataframe to filter invalid entries

invalid_entries = df[invalid_entries_mask]


# Display the invalid entries

invalid_entries[['Established', 'Age']]
```

13. Location Accuracy:

Question: Assess the accuracy of location entries. Are there misspelled or ambiguous locations that require correction?

Code:

```
# Check the unique values in the 'Location' column

df['Location'].value_counts()
```

14. Handling Categorical Data:

Question: For categorical indicators, consider encoding or transforming them into a format suitable for analysis.

Description

The 'Location' column contains entries with a combination of uppercase letters and commas. To assess the accuracy, we would need to standardize the format, possibly by removing commas and ensuring proper capitalization of location names. If there are misspelled or ambiguous locations, they would require correction.

15. Consistent Rating Scale:
Question: Ensure a consistent rating scale in the Rating column. Should it be normalized or adjusted for uniform analysis?

To ensure a consistent rating scale in the 'Rating' column, we should normalize or adjust the values for uniform analysis. The ratings appear to be on different scales, with some values exceeding the typical maximum of 5 for rating systems. Here's the plan to normalize the ratings:

- Identify the maximum value in the 'Rating' column.
- Divide all ratings by this maximum value to normalize them to a scale of 0 to 1.
- Multiply the normalized ratings by 5 to adjust them to a standard 0 to 5 scale.

Code:

```
# Normalize the 'Rating' column to a 0-5 scale
max_rating = df['Rating'].max()
# Normalize ratings to a 0-1 scale
df['Normalized Rating'] = df['Rating'] / max_rating
# Adjust to a 0-5 scale
df['Normalized Rating'] *= 5

# Check the transformation
normalized_ratings = df[['Rating', 'Normalized Rating']].head()
normalized_ratings
```

	Rating	Normalized Rating
0	5.4	3.461538461538462
1	3.5	2.2435897435897436
2	4.2	2.7457264957264966
3	4.4	2.820512820512821
4	6.4	4.102564102564103

Description

The 'Rating' column has been normalized and adjusted to a standard 0 to 5 scale. The new 'Normalized Rating' column reflects this transformation, making the ratings uniform for analysis.