DATA CLEANING SUMMARY

This document outlines the step-by-step process followed to clean the dataset messy_Data.csv and includes any assumptions made during the cleaning process.

1- **Loading the Dataset**:

The dataset was loaded into the Jupyter Notebook using Pandas' read_csv() function for inspection.

Code:-

```
import pandas as pd
data = pd.read_csv('messy_Data.csv')
```

2- **Inspecting the Dataset**:

Basic functions like head(),shape, info(), and describe() were used to inspect the structure of the dataset, identify data types, and detect missing or inconsistent data.

Code:-

```
    data.head()
    data.info()
    Data.shape
    data.describe()
```

3-Handling Duplicates:

To ensure the dataset maintains a unique set of records,executed a thorough process for identifying and removing duplicate entries. The steps taken are as follows:

- **Check for Duplicate Values**: first assessed the dataset to determine the number of duplicate rows present.

  Code

  ```
  duplicate_count = data.duplicated().sum()
  ```

- **Remove Duplicate Rows**: proceeded to remove the duplicate rows while retaining the first occurrence of each unique record.

  Code

  ```
  data = data.drop_duplicates()
  ```

By following this structured approach,ensured that the dataset now consists solely of unique records, enhancing its quality for subsequent analysis.

4-Handling Missing Values

In the dataset, missing values were identified and addressed through a systematic approach. Below are the steps taken to handle missing data effectively:

- **Check for Missing Values**: began by assessing the dataset for any missing values across all columns.

```
missing_values = data.isnull().sum()
```

- **Identify Rows with Excessive Nulls**: checked for rows that had more than 5 missing values to determine if any could be considered irrelevant.

```
rows_with_many_nulls = data[data.isnull().sum(axis=1) > 5]
```

- **Remove Irrelevant Rows**: decided to drop rows with more than 5 missing values to clean the dataset.

```
data = data.dropna(thresh=len(data.columns) - 5)
```

- **Check Remaining Null Values**: After removing irrelevant rows, we checked for any remaining missing values.
- **Fill Missing Values in Specific Columns**:
  - **Name Column**: Missing values in the Name column were filled with the placeholder "Unknown".
    ```
    data['Name'] = data['Name'].fillna('Unknown')
    ```
  - **Age Column**: Missing values in the Age column were filled with the median age, a method chosen for its resilience against outliers.

    ```
    data['Age'] = data['Age'].fillna(data['Age'].median())
    ```

- **Department Column**: Missing values in the Department column were filled using the mode (most frequent value).

```
data['Department'] =      data['Department'].fillna(data['Department'].mode()[0])
```

- **Salary Column**: Missing values in the Salary column were filled using the mean salary of the specific department to ensure that department-related data was preserved.

```
data['Salary'] =data.groupby('Department')['Salary'].transform(lambda x: x.fillna(x.mean()))
```

- **Join date** column's missing values will be addressed separately after standardizing the date formats.

5-**Correcting Email Format:** To ensure the integrity of the email addresses in the dataset, implemented a systematic approach to identify and remove any invalid email formats. The following steps outline this process:

- **Define Email Validation Function**: began by creating a function to check the validity of email addresses using a regular expression (regex).

```
import re

def is_valid_email(email):

pattern    =    r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'

return re.match(pattern, email) is not None
```

- **Remove Rows with Invalid Emails**: To clean the dataset, filtered out the rows containing invalid email addresses.

```
data = data[data['Email'].apply(is_valid_email)]
```

As a result, all rows with invalid email addresses were removed from the dataset.

6-. Cleaning the Name Field

To ensure the names in the dataset are consistently formatted and free from extraneous titles and characters, implemented a cleaning process. Below are the detailed steps taken:

implemented a function that performs several cleaning tasks:

- Removes common titles (e.g., Mr., Mrs., Dr.).
- Eliminates non-alphabetical characters (except spaces).
- Trims extra spaces and leading/trailing whitespace.
- Capitalizes the first letter of each word (title case).

```
def clean_name(name):

    # Remove titles like Mr., Mrs., Dr., etc. (add more titles as necessary)

    titles = ['Mr', 'Mrs', 'Ms', 'Miss', 'Dr', 'Prof', 'Sir']

    name = re.sub(r'\b(?:' + '|'.join(titles) + r')\b\.?\s*', '', name, flags=re.IGNORECASE)

    # Remove any non-alphabetical characters (except spaces)

    name = re.sub(r'[^a-zA-Z\s]', '', name)

    # Remove multiple spaces and strip leading/trailing spaces

    name = re.sub(r'\s+', ' ', name).strip()

    # Capitalize the first letter of each word (title case)

    name = name.title()
```

```
return name

# Apply the cleaning function to the 'Name' column

df['Name'] = df['Name'].apply(clean_name)
```

7-Standardizing the 'Join Date' Column:

To standardize the date format in the **'Join Date'** column of the dataset, follow the steps below:

- **Check for Unique Date Formats**: First, inspect the unique values in the **'Join Date'** column to identify different date formats.
- **Convert Dates to a Consistent Format (YYYY-MM-DD)**: Use pd.to_datetime() to convert the **'Join Date'** column to a standardizedformat

```
df['Join Date'] = pd.to_datetime(df['Join Date'], errors='coerce', format='%Y-%m-%d')
```

HandleMissingValues

Use **forward filling** (ffill()) to fill missing values by propagating the previous non-null value downwards.
Then, apply **backward filling** (bfill()) to handle any remaining NaN values by propagating the next non-null value upwards.

```
# Forward fill missing values

df['Join Date'] = df['Join Date'].ffill()


# Backward fill any remaining missing values

df['Join Date'] = df['Join Date'].bfill()
```

By following these steps, the **'Join Date'** column has been successfully converted to a consistent **YYYY-MM-DD** format, and any missing values were handled using forward and backward filling methods.

8-Cleaning the 'Department' Column:

Define a Function to Extract the Base Department Name: The function **clean_department()** checks each department name and compares it with a list of valid department names. If the name starts with one of the valid names, it returns the clean version. Otherwise, it leaves the department name unchanged for later handling.

```
def clean_department(dept):
    # List of valid departments
    valid_departments = ['HR', 'Sales', 'Marketing', 'Engineering', 'Support']

    # Iterate over valid departments and check if the base name matches
    for valid_dept in valid_departments:
        if dept.startswith(valid_dept):
            return valid_dept  # Return the clean department name
    return dept  # If not found, return as-is (can handle later)
df['Department'] = df['Department'].apply(clean_department)
```

This function efficiently cleans the **'Department'** column by ensuring that only the valid base department names are retained, leaving non-standard entries for further handling if needed. The cleaned department names now include only **'Sales'**, **'Marketing'**, **'Support'**, **'HR'**, and **'Engineering'**.

9- **Handle salary noice:**

boxplot is used to visualize the distribution of salaries and identify potential outliers.

The **boxplot** reveals that the **'Salary'** column does not contain any noticeable outliers, suggesting a fairly consistent range of salary values across the dataset.

Documenting the Cleaning and Saving of the Dataset

After performing the necessary data cleaning steps on the dataset, it is sorted by the **'Unnamed: 0'** column, and then saved to a new CSV file called **cleaned_dataset.csv**.