DATA CLEANING TASK DOCUMENTATION

1. INTRODUCTION

This document describes various procedures and assumptions taken into consideration to identify and correct errors and inconsistencies in data to improve its quality. It also documents the process of cleaning a dataset named '*messy_data.csv*'. The goal is to ensure the data is accurate, consistent, and usable for analysis.

The '*datacleaning.ipynb*' script outlines a series of steps that address common data quality issues such as missing values, duplicates, incorrect formats, and outliers. By systematically applying these steps, the transformation of raw, messy data into a clean dataset is made possible.

2. OBJECTIVE

The key objectives are:

- Remove unwanted columns
- Validate and Correct Formats
- Handle missing values
- Remove Duplicates
- Clean text fields
- Standardize and correct values
- Saves the cleaned data

3. PROCEDURE

3.1. Load the Data:

Initially, the dataset '*messy_data.csv*' was downloaded. Using Anaconda as a platform, Jupyter Notebook was used as a tool for providing a web-based interface, with Python as the programming language. The dataset was loaded into a pandas DataFrame, and its initial structure was displayed

3.2. Inspect the Data:

An analysis of the data header and columns was conducted to find basic information and identify data quality issues. As a result, it was found that there was one unnamed column, which might be the result of the person who created the file not using index=False, as the unnamed column resembles an index value.

It was noted that there were a total of seven legitimate columns: 'ID', 'Name',' Age',' Email', 'Join Date', 'Salary', and 'Department'. Among these, six columns were found to be missing some values, except for the 'ID' column. The six columns had different counts of values, indicating that some values are missing, as shown in Fig. 3.2.1. The count of missing values is also displayed in Fig. 3.2.2.

```
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  11000 non-null  int64
 1   ID          11000 non-null  object
 2   Name        8667 non-null   object
 3   Age         9253 non-null   float64
 4   Email       9731 non-null   object
 5   Join Date   8808 non-null   object
 6   Salary      8761 non-null   float64
 7   Department  8745 non-null   object
dtypes: float64(2), int64(1), object(5)
memory usage: 687.6+ KB
Categorical columns : ['ID', 'Name', 'Email', 'Join Date', 'Department']
Numerical columns : ['Unnamed: 0', 'Age', 'Salary']
```

Fig. 3.2.1. Columns and their value count

```
MISSING VALUES BEFORE REMOVING ROWS:
 Unnamed: 0       0
ID                0
Name           2333
Age            1747
Email          1269
Join Date      2192
Salary         2239
Department     2255
dtype: int64
```

Fig. 3.2.2. Column and their missing value count

3.3. Handle Missing Values:

Before handling the missing values, the columns were categorized into two groups: categorical columns and numerical columns.

For categorical column handling, the missing rows were first filled with the string 'Unknown'. Later, it was found that the ‘Department’ column had some values named 'unknown'. All rows filled with 'Unknown' were removed, and values with the string 'unknown' in the Department column were also removed. As the ‘Department’ column contained string values, it was difficult to fill the missing values reliably. Some assumptions about using Naïve Bayes or Decision Tree were considered, but these methods wouldn’t be reliable if the remaining data was also full of inconsistencies, which was the case here.

Numerical values were then handled. Unlike the categorical values, the missing numerical values were filled with median values. The median was used as it is less prone to outliers. The median of the numerical column ‘Age’ was calculated and used to fill the missing values. The last numerical column with missing values was Salary, and it was handled differently. For this, the ‘Department’ column was taken into consideration. It was assumed that people in the same department have similar salaries.

There was a lot of noise in the ‘Department’ column, which was addressed in detail in section 3.7. It was found that there were a total of five main departments: Support, Sales, Marketing, Engineering, and HR. The median salary for each department was calculated and then assigned to the missing ‘Salary’ column values based on their department.

At the end, the count of missing values was displayed again, showing '0' for all columns.

3.4. Remove Duplicates:

ID', 'Name', 'Email', 'Join Date', and 'Department' were considered for processing after filling the missing values for 'Age' and 'Salary'. It was found that including 'Age' and 'Salary' at this stage could introduce new noise.

3.5 Correct Email Formats:

In this step, the script cleans a DataFrame by filtering out invalid email addresses. It defines a function to check the email format and then uses that function to filter the ‘Email’ column, keeping only rows with valid email addresses. The cleaned data is saved to a CSV file.

3.5. Clean Name Fields:

It standardizes the format by taking a name string as input. It removes common title prefixes like Mr., Ms., etc., as well as leading and trailing spaces, and capitalizes the first letter of each word. This function is then applied to the ‘Name’ column of a DataFrame, effectively cleaning all names in the column.

3.6. Standardize Date Formats:

This step cleans up the 'Join Date’ column in a DataFrame. It first converts all dates to a consistent format, handling any errors that might occur during conversion. Then, it reformats the valid dates into a common year-month-day format for better readability. Finally, it replaces any missing dates with the text ‘Unknown’ to provide a clear placeholder for missing information.

At the end, it displays the count of how many times each unique date appears in the column.

3.7. Correct Department Names:

This step was done earlier while handling the missing values in the numerical category, as it was needed to assign salaries according to departments. It was found that the ‘Department’ column had a number of issues such as spelling mistakes (e.g., HRx, Engineeringe, etc.) and strings like 'unknown'. There was a need to standardize department names using a predefined mapping.

The function takes dept as a parameter. It checks if dept is a string, converts it to lowercase, and then iterates through the keys of the departments dictionary. If dept starts with any key from departments, it returns the corresponding value (standardized department name). If no match is found, it returns dept unchanged.

The modified DataFrame is then saved to a CSV file named *'messy_data.csv*' without including the index column. Finally, the unique values and count of the column 'Department' are displayed (shown in **Fig. 3.7.1.**).

```
VALUES COUNT IN DEPARTMENT COLUMN AFTER HANDLING:
 Department
Support        1781
Sales          1777
Marketing      1765
Engineering    1717
HR             1705
Name: count, dtype: int64
```

Fig. 3.7.1. Department values and its occurring counts after handling

3.8. Handle Salary Noise:

In this phase, a function is involved to handle cases where salaries start with a dot, round floating-point salaries, and correct outlier salaries to fall within specified limits.

After applying this function, the 'Salary' column in the DataFrame will contain cleaned and standardized salary values suitable for further analysis or processing. Salary values are cleaned and standardized to ensure they fall within a reasonable range. The minimum salary is set to 30,000, and the maximum salary to 200,000. If a salary exceeds the maximum, it is replaced with the maximum salary. If a salary is lower than the minimum, it is replaced with the minimum salary.

3.9. Save the Cleaned Dataset:

The resulting data after cleaning is converted to another CSV file named `*cleaned_dataset.csv*` for further analysis. This allows for analysis of the new dataset in case any errors occurred while saving the changes during different data cleaning procedures. A script named `*check_cleaned_data.ipynb*` is used for this purpose.