Project Documentation: Data Cleaning for Layoffs Dataset

Project Title
Layoffs Data Cleaning and Preparation

Overview

This project focuses on cleaning a dataset containing information about layoffs across various companies. The dataset exhibited numerous issues such as duplicates, inconsistent naming conventions, and missing values. The primary goal of this project is to enhance the quality of the data, making it suitable for further analysis.

Dataset

The dataset used in this project can be accessed Here

Objectives

- Identify and remove duplicate records.
- Standardize data entries across relevant columns.
- Handle NULL values and blank entries effectively.
- Ensure data integrity and readiness for analysis.

Dataset Description

The dataset includes the following fields:

- **Company**: The name of the company where layoffs occurred.
- **Location**: The geographical location of the company.
- **Industry**: The sector in which the company operates.
- **Total Laid Off**: The total number of employees laid off.
- **Percentage Laid Off**: The percentage of the workforce laid off.
- **Date**: The date the layoff was announced.
- **Stage**: The stage of the layoff process.
- **Country**: The country where the company is based.
- **Funds Raised (Millions)**: Amount of funds raised by the company in millions.

Tools and Technologies Used

List the tools and technologies you used during the project.

- **Tools Used**:
    - SQL (MySQL)
    - Data Cleaning Techniques
    - Data Standardization
    - Date Formatting and Transformation

Project Description

- **Description**:
    - **Objective**: Clean a dataset of global layoffs to ensure it is ready for analysis.
    - **Key Steps**:
        - Created a staging table to preserve raw data.
        - Removed duplicate rows using SQL CTEs.
        - Standardized company names and industry categories.
        - Handled NULL values by filling missing data and deleting incomplete records.
        - Converted date columns from text to proper date formats.

Data Cleaning Steps

Step 1: Display All Records from the Raw Layoffs Table

This step fetches all the data from the original layoffs table to review its contents before cleaning.

```
SELECT *
FROM layoffs;
```

Step 2: Create a Staging Table

A new table, layoffs_staging, is created to hold the raw data temporarily for cleaning purposes.

```
CREATE TABLE layoffs_staging
LIKE layoffs;
```

Step 3: Insert All Data into the Staging Table

This query copies all data from the original layoffs table into the staging table for processing.

```
INSERT layoffs_staging
SELECT *
FROM layoffs;
```

Step 4: Check the Staging Table

Run this query to verify that all data has been successfully copied into the staging table.

```
SELECT *
FROM layoffs_staging;
```

Step 5: Remove Duplicates

A Common Table Expression (CTE) is created to find duplicates based on key columns.

```
WITH duplicate_cte AS (
   SELECT *,
   ROW_NUMBER() OVER(
      PARTITION BY company, industry, total_laid_off,
percentage_laid_off, `date`, stage, country, funds_raised_millions
   ) AS row_num
   FROM layoffs_staging
)
```

Step 6: Check for Duplicates

This query retrieves all entries from the CTE where the row number indicates a duplicate.

```
SELECT *
FROM duplicate_cte
WHERE row_num > 1;
```

Step 7: Check Specific Companies for Duplicates

This step involves checking for duplicate entries for specific companies ('Oda' and 'Casper') to understand their records better.

```
SELECT *
FROM layoffs_staging
WHERE company='Oda';

SELECT *
FROM layoffs_staging
WHERE company = 'Casper';
```

Step 8: Create a New Table for Removing Duplicates

A new table, layoffs_staging2, is created, including an extra column to identify duplicates by their row number.

```
CREATE TABLE `layoffs_staging2` (
  `company` text,
  `location` text,
  `industry` text,
  `total_laid_off` int DEFAULT NULL,
  `percentage_laid_off` text,
  `date` text,
  `stage` text,
  `country` text,
  `funds_raised_millions` int DEFAULT NULL,
  `row_num` INT
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

Step 9: Insert Data into the New Table

Copy data from the staging table into the new table while assigning a row number to identify duplicates.

```
INSERT INTO layoffs_staging2
SELECT *,
ROW_NUMBER() OVER(
   PARTITION BY company, industry, total_laid_off,
percentage_laid_off, `date`, stage, country, funds_raised_millions
) AS row_num
FROM layoffs_staging;
```

Step 10: Check the New Table

This query checks the contents of the new table to ensure data was correctly inserted.

```
SELECT *
FROM layoffs_staging2;
```

Step 11: Filter Rows Where Row Number > 1

Identify which rows are duplicates based on the row number generated in the previous step.

```
SELECT *
FROM layoffs_staging2
WHERE row_num > 1;
```

Step 12: Delete Duplicates from the New Table

This query removes duplicate rows from the new table based on the row number.

```
DELETE
FROM layoffs_staging2
WHERE row_num > 1;
```

Step 13: Check the Table After Deletion

Verify the contents of the new table to confirm that duplicates have been removed.

```
SELECT *
FROM layoffs_staging2;
```

Step 14: Standardizing Data

Identify any unwanted spaces in company names for standardization.

```
SELECT *
FROM layoffs_staging2;
```

Step 15: Trim Spaces from Company Names

Update the company names in the new table to remove leading and trailing spaces.

```
UPDATE layoffs_staging2
SET company = TRIM(company);
```

Step 16: Standardizing the Industry Column

Check the unique values in the industry column to find inconsistencies.

```
SELECT DISTINCT industry
FROM layoffs_staging2
ORDER BY 1;

SELECT *
FROM layoffs_staging2
WHERE industry LIKE 'Crypto%';
```

Step 17: Update Inconsistent Industry Names

Standardize all variations of 'Crypto' to a single entry.

```
UPDATE layoffs_staging2
SET industry = 'Crypto'
WHERE industry LIKE 'Crypto%';
```

Step 18: Review the Location and Country Columns

Check the unique entries in the location and country columns for any irregularities.

```
SELECT DISTINCT location
FROM layoffs_staging2;

SELECT DISTINCT country
FROM layoffs_staging2;
```

Step 19: Fix the Trailing Period in 'United States'

Update the country names by removing any trailing periods for consistency.

```
UPDATE layoffs_staging2
SET country = TRIM(TRAILING '.' FROM country)
WHERE country LIKE 'United States%';
```

Step 20: Convert Date Column from Text to Date Type

This updates the date column, converting the text representation to a standard date format.

```
UPDATE layoffs_staging2
SET `date` = STR_TO_DATE(`date`, '%m/%d/%Y');
```

Step 21: Change the Column Type to DATE

Modify the date column's data type to ensure it is recognized as a date type in the database.

```
ALTER TABLE layoffs_staging2
MODIFY COLUMN `date` DATE;
```

Step 22: Check for NULLs and Blanks

Identify records where both total_laid_off and percentage_laid_off are NULL.

```
SELECT *
FROM layoffs_staging2
WHERE total_laid_off IS NULL
AND percentage_laid_off IS NULL;

SELECT *
FROM layoffs_staging2
WHERE industry IS NULL
OR industry = '';
```

Step 23: Fill NULL Values for Industry

Convert empty strings in the industry column to NULL for uniformity and populate the NULL industry values for 'Airbnb'.

```
UPDATE layoffs_staging2
SET industry = NULL
WHERE industry = '';

UPDATE layoffs_staging2 AS t1
JOIN layoffs_staging2 AS t2
ON t1.company = t2.company
SET t1.industry = t2.industry
WHERE t1.industry IS NULL AND t2.industry IS NOT NULL;
```

Step 24: Verify that NULLs are Filled

Check if the industry values for 'Airbnb' have been successfully populated.

```
SELECT *
FROM layoffs_staging2
WHERE company = 'Airbnb';

SELECT *
FROM layoffs_staging2
WHERE industry IS NULL
OR industry = '';
```

Step 25: Delete Rows with NULL Total Laid Off and Percentage Laid Off

Remove any records where both total_laid_off and percentage_laid_off are NULL.

```
DELETE
FROM layoffs_staging2
WHERE total_laid_off IS NULL AND percentage_laid_off IS
NULL;
```

Step 26: Verify Deletion

Confirm that no rows remain with NULL values for total_laid_off and percentage_laid_off after the deletion.

```
SELECT *
FROM layoffs_staging2
WHERE total_laid_off IS NULL AND percentage_laid_off IS
NULL;
```

Step 27: Final Check of the Cleaned Data

Retrieve the final state of the cleaned data in the staging table.

```
SELECT *
FROM layoffs_staging2;
```

Step 28: Drop the Row Number Column

Remove the row_num column from the final table since it has served its purpose.

```
ALTER TABLE layoffs_staging2
DROP COLUMN row_num;
```

Final Step: Display the Finalized Cleaned Data

Display the cleaned data in the staging table after all processing steps have been completed.

```
SELECT *
FROM layoffs_staging2;
```

Challenges Faced

1. **Data Inconsistency**: The dataset contained various naming conventions, which required significant standardization efforts.
2. **Handling NULL Values**: Identifying and appropriately filling or removing NULL values was essential to maintaining data integrity.
3. **Duplicate Entries**: The presence of duplicates made it necessary to devise a robust strategy for identifying and eliminating them.

Outcomes

- The final dataset is clean, consistent, and ready for analysis.
- **Missing Values Addressed**: Any NULL or blank values were either appropriately filled or removed to ensure the completeness of the dataset.
- **Irrelevant Data Removed**: Unnecessary or irrelevant columns have been dropped, reducing noise in the dataset and enhancing its suitability for analysis.
- **Date Format Standardized**: The date column has been converted to a consistent format, allowing for accurate date-based analyses.
- **Geographical Consistency**: Country names and locations have been standardized, which is crucial for any location-based analysis or visualization.

As a result, the cleaned dataset is now an optimized, accurate, and reliable version of the raw data. It is well-structured, making it ideal for performing further data analysis or visualization tasks.

Key Learnings

- **Data Cleaning Efficiency**: Applying SQL's powerful functions such as ROW_NUMBER(), TRIM(), and STR_TO_DATE() effectively streamlined the process of identifying duplicates, inconsistencies, and formatting issues.
- **Importance of Data Validation**: Continuously validating changes through each step helped avoid data loss or incorrect transformations.
- **Real-World Data Challenges**: Handling missing, inconsistent, and redundant data highlighted the importance of robust cleaning processes in any data pipeline.

Conclusion

This project demonstrated the critical steps required to clean and prepare a large, unstructured dataset for analysis. By following a systematic approach, all key issues with the data were addressed, leading to a well-organized and reliable dataset that can be used for various analytical purposes.

This project also showcases a strong grasp of SQL techniques and the ability to solve common data quality problems in real-world scenarios.