The shared document helps in performing the Name parsing Checks, identifying the Forage_Company_IDs mapped with the same Company_LinkedIn_URL, and getting all the basic stats of the LinkedIn Profile Details delivery file.

- Task 1: Forage_Company_IDs mapped to the same Company_LinkedIn_URL:
- Task 2: Cleaned Company_Website mapped to multiple Forage_Company_IDs:

The above 2 tasks are completed by using the below script, and we get an output file with the affected URLs

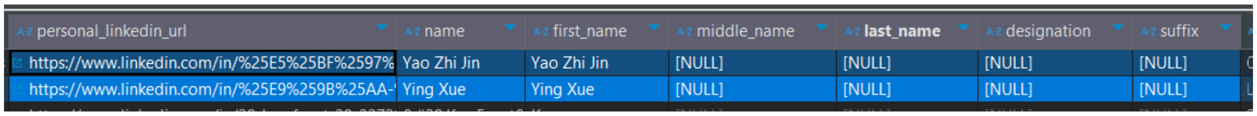
```
import zipfile
import pandas as pd
import os

# File paths
zip_path =
r"C:\Users\abhis\Downloads\company-contact-data-20250508-batch17-file1.zip"
extract_path = r"C:\Users\abhis\Downloads\company_contact_data"
output_path = r"C:\Users\abhis\Downloads\company_contact_data_output.xlsx"

# Unzip the file
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

# Assuming only one CSV inside -- get its path
csv_files = [f for f in os.listdir(extract_path) if f.endswith('.csv')]
if not csv_files:
    print("No CSV file found in the zip.")
    exit()

csv_path = os.path.join(extract_path, csv_files[0])

# Load CSV into DataFrame
df = pd.read_csv(csv_path, low_memory=False)

# Basic Stats
print(f"\n✅ File Loaded: {csv_files[0]}")
print(f"📊 Total Rows: {len(df)}")
print(f"📊 Total Columns: {len(df.columns)}\n")

# Number of unique companies
print(f"🏢 Unique Bisnow_Company_UUIDs:
```

```
{df['Bisnow_Company_UUID'].nunique()}")
print(f"🏢 Unique Forage_Company_IDs:
{df['Forage_Company_ID'].nunique()}\n")

# Number of unique people
print(f"🧑‍💼 Unique Personal LinkedIn URLs:
{df['Personal_Linkedin_URL'].nunique()}")
print(f"🧑‍💼 Unique Names: {df['Name'].nunique()}\n")

# Null value counts per column
print("📌 Null counts per column:")
null_counts = df.isnull().sum()
print(null_counts)

# Date ranges
if 'Last_Activity_Date' in df.columns:
    df['Last_Activity_Date'] = pd.to_datetime(df['Last_Activity_Date'],
errors='coerce')
    print(f"\n📅 Last_Activity_Date Range:
{df['Last_Activity_Date'].min().date()} to
{df['Last_Activity_Date'].max().date()}")

if 'Employment_Start_Date' in df.columns:
    df['Employment_Start_Date'] =
pd.to_datetime(df['Employment_Start_Date'], errors='coerce')
    print(f"📅 Employment_Start_Date Range:
{df['Employment_Start_Date'].min().date()} to
{df['Employment_Start_Date'].max().date()}")

# Task 1: Identify Forage_Company_ID having the same Company_LinkedIn_URL
print("\n🔍 Task 1: Forage_Company_IDs mapped to the same
Company_LinkedIn_URL:")

duplicate_company_urls =
df.groupby('Company_LinkedIn_URL')['Forage_Company_ID'].nunique()
duplicate_company_urls = duplicate_company_urls[duplicate_company_urls > 1]

if duplicate_company_urls.empty:
    print("✅ No duplicate mappings found. Each Company_LinkedIn_URL is
uniquely mapped to a Forage_Company_ID.")
    task1_df = pd.DataFrame(columns=['Company_LinkedIn_URL',
'Forage_Company_IDs'])
else:
    print(f"⚠️ Found {len(duplicate_company_urls)} Company_LinkedIn_URL(s)
linked to multiple Forage_Company_IDs:")
    records = []
    for url in duplicate_company_urls.index:
        ids = df.loc[df['Company_LinkedIn_URL'] == url,
'Forage_Company_ID'].unique()
        print(f"\n🔗 {url} -- Forage_Company_IDs: {list(ids)}")
        records.append({'Company_LinkedIn_URL': url, 'Forage_Company_IDs':
', '.join(map(str, ids))})
    task1_df = pd.DataFrame(records)

# Task 2: Clean Company_Website and find duplicate companies with different
Forage_Company_IDs
print("\n🔍 Task 2: Cleaned Company_Website mapped to multiple
Forage_Company_IDs:")

df['Cleaned_Company_Website'] =
df['Company_Website'].astype(str).str.lower()
df['Cleaned_Company_Website'] =
df['Cleaned_Company_Website'].str.replace(r'^https?://(www\.)?', '',
regex=True)
df['Cleaned_Company_Website'] =
df['Cleaned_Company_Website'].str.rstrip('/')

duplicate_websites =
df.groupby('Cleaned_Company_Website')['Forage_Company_ID'].nunique()
duplicate_websites = duplicate_websites[duplicate_websites > 1]

if duplicate_websites.empty:
    print("✅ No duplicate mappings found. Each Cleaned_Company_Website is
uniquely mapped to a Forage_Company_ID.")
    task2_df = pd.DataFrame(columns=['Cleaned_Company_Website',
'Forage_Company_IDs'])
else:
    print(f"⚠️ Found {len(duplicate_websites)} Cleaned_Company_Website(s)
linked to multiple Forage_Company_IDs:")
    records = []
    for site in duplicate_websites.index:
        ids = df.loc[df['Cleaned_Company_Website'] == site,
'Forage_Company_ID'].unique()
        print(f"\n🌐 {site} -- Forage_Company_IDs: {list(ids)}")
        records.append({'Cleaned_Company_Website': site,
'Forage_Company_IDs': ', '.join(map(str, ids))})
```

```
    task2_df = pd.DataFrame(records)

# Write to Excel with 2 sheets: Task 1 and Task 2
with pd.ExcelWriter(output_path, engine='xlsxwriter') as writer:
    task1_df.to_excel(writer, sheet_name='Task1_CompanyLiUrl_Dups',
index=False)
    task2_df.to_excel(writer, sheet_name='Task2_CompanyWeb_Dups',
index=False)

print(f"\n📤 Output file saved at: {output_path}")

# Optional cleanup: remove extracted files if you want
# import shutil
# shutil.rmtree(extract_path)
```

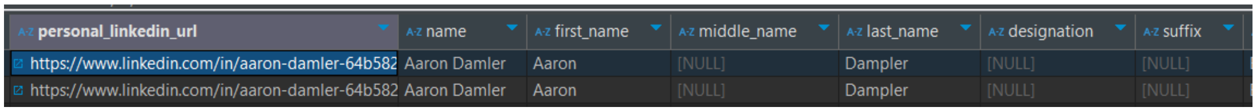
Next, we wanted to perform certain checks on the Name Parsing part, basically want to validate our name parser so that we are not manually finding the rows having issues.

- **Task 3** - To identify the rows where the name is not parsed completely into the respective columns. Example from the BN_Mortgage delivery
  Possible logic - if words in the name column >1, then more than one column from first_name, middle_name, last_name, designation, and suffix should be populated.



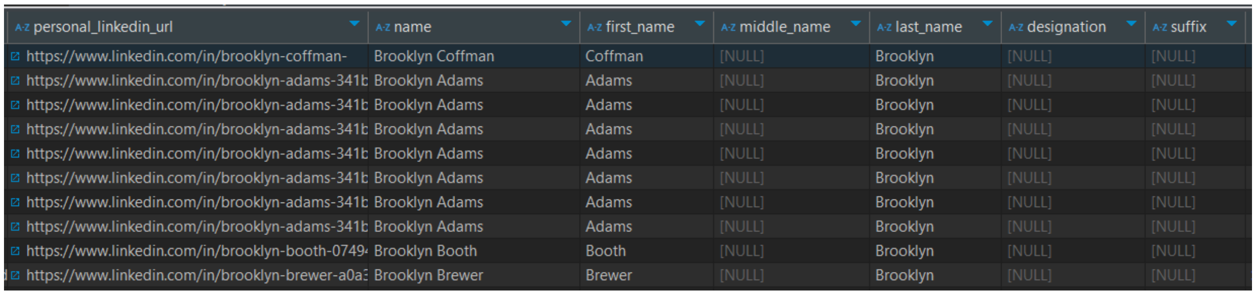
personal_linkedin_url	name	first_name	middle_name	last_name	designation	suffix
https://www.linkedin.com/in/%25E5%25BF%2597%	Yao Zhi Jin	Yao Zhi Jin	[NULL]	[NULL]	[NULL]	[NULL]
https://www.linkedin.com/in/%25E9%259B%25AA-	Ying Xue	Ying Xue	[NULL]	[NULL]	[NULL]	[NULL]

- **Task 4** - To identify the rows where there is a name but nothing parsed to the name columns, first_name, middle_name, last_name, designation, and suffix
- **Task 5** - To identify the rows where any of the values in the parsed columns first_name, middle_name, last_name, designation, and suffix are not present in the name column.



personal_linkedin_url	name	first_name	middle_name	last_name	designation	suffix
https://www.linkedin.com/in/aaron-damler-64b582	Aaron Damler	Aaron	[NULL]	Dampler	[NULL]	[NULL]
https://www.linkedin.com/in/aaron-damler-64b582	Aaron Damler	Aaron	[NULL]	Dampler	[NULL]	[NULL]

- **Task 5** - To identify the rows where designation or suffix are not parsed into their respective columns, designation and suffix, and are part of the name column, first_name, middle_name, last_name, or if the designation values are present in suffix or vice versa.
- **Task 6** - To identify nicknames(generally present in “” or ()) and mark rows where they are parsed in the name columns, first_name, middle_name, last_name, we would also need to discriminate between designations present in parentheses, which will act as false positives.
- **Task 7** - To identify rows where first_name, middle_name, and last_name are interchanged while parsing.



personal_linkedin_url	name	first_name	middle_name	last_name	designation	suffix
https://www.linkedin.com/in/brooklyn-coffman-	Brooklyn Coffman	Coffman	[NULL]	Brooklyn	[NULL]	[NULL]
https://www.linkedin.com/in/brooklyn-adams-341b	Brooklyn Adams	Adams	[NULL]	Brooklyn	[NULL]	[NULL]
https://www.linkedin.com/in/brooklyn-adams-341b	Brooklyn Adams	Adams	[NULL]	Brooklyn	[NULL]	[NULL]
https://www.linkedin.com/in/brooklyn-adams-341b	Brooklyn Adams	Adams	[NULL]	Brooklyn	[NULL]	[NULL]
https://www.linkedin.com/in/brooklyn-adams-341b	Brooklyn Adams	Adams	[NULL]	Brooklyn	[NULL]	[NULL]
https://www.linkedin.com/in/brooklyn-adams-341b	Brooklyn Adams	Adams	[NULL]	Brooklyn	[NULL]	[NULL]
https://www.linkedin.com/in/brooklyn-adams-341b	Brooklyn Adams	Adams	[NULL]	Brooklyn	[NULL]	[NULL]
https://www.linkedin.com/in/brooklyn-adams-341b	Brooklyn Adams	Adams	[NULL]	Brooklyn	[NULL]	[NULL]
https://www.linkedin.com/in/brooklyn-booth-0749	Brooklyn Booth	Booth	[NULL]	Brooklyn	[NULL]	[NULL]
https://www.linkedin.com/in/brooklyn-brewer-a0a3	Brooklyn Brewer	Brewer	[NULL]	Brooklyn	[NULL]	[NULL]

We have a working script for the name parsing checks, which also consists of Task1 and Task2 -

```python
import zipfile
import pandas as pd
import os
import re
from openpyxl import Workbook
from openpyxl.utils.dataframe import dataframe_to_rows

# --- File paths ---
zip_path =
r"C:\Users\abhis\Downloads\company-contact-data-20250508-batch17-file1.zip"
extract_path = r"C:\Users\abhis\Downloads\company_contact_data"
output_path = r"C:\Users\abhis\Downloads\company_contact_data_output.xlsx"
qa_output_path = r"C:\Users\abhis\Downloads\Name_Parsing_QA_Output.xlsx"

# --- Step 1: Unzip the file ---
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

# --- Step 2: Identify CSV file ---
csv_files = [f for f in os.listdir(extract_path) if f.endswith('.csv')]
if not csv_files:
    print("No CSV file found in the zip.")
    exit()
csv_path = os.path.join(extract_path, csv_files[0])

# --- Step 3: Load CSV ---
df = pd.read_csv(csv_path, low_memory=False)

# --- Step 4: Basic stats ---
print(f"\n✅ File Loaded: {csv_files[0]}")
print(f"📊 Total Rows: {len(df)} | 📊 Total Columns: {len(df.columns)}")
```

```
for col in ['Bisnow_Company_UUID', 'Forage_Company_ID',
'Personal_Linkedin_URL', 'Name']:
    if col in df.columns:
        print(f"🔍 Unique {col}s: {df[col].nunique()}")

print("\n📌 Null counts per column:")
print(df.isnull().sum())

date_columns = ['Last_Activity_Date', 'Employment_Start_Date']
for col in date_columns:
    if col in df.columns:
        df[col] = pd.to_datetime(df[col], errors='coerce')
        print(f"📅 {col} Range: {df[col].min().date()} to
{df[col].max().date()}")

# --- Task 1: Duplicate Company_LinkedIn_URL mappings ---
print("\n🔍 Task 1: Duplicate Company_LinkedIn_URLs")
duplicate_urls =
df.groupby('Company_LinkedIn_URL')['Forage_Company_ID'].nunique()
duplicate_urls = duplicate_urls[duplicate_urls > 1]

task1_df = pd.DataFrame(columns=['Company_LinkedIn_URL',
'Forage_Company_IDs'])
if not duplicate_urls.empty:
    records = [{'Company_LinkedIn_URL': url,
                'Forage_Company_IDs': ', '.join(map(str,
df.loc[df['Company_LinkedIn_URL'] == url, 'Forage_Company_ID'].unique()))}
               for url in duplicate_urls.index]
    task1_df = pd.DataFrame(records)
    print(task1_df)
else:
    print("✅ No duplicates found.")

# --- Task 2: Clean Company_Website and check for duplicate mappings ---
print("\n🔍 Task 2: Duplicate Cleaned Company_Websites")
df['Cleaned_Company_Website'] = (df['Company_Website'].astype(str)
                                 .str.lower()
                                 .str.replace(r'^https?://(www\.)?', '',
regex=True)
                                 .str.rstrip('/'))

duplicate_sites =
df.groupby('Cleaned_Company_Website')['Forage_Company_ID'].nunique()
duplicate_sites = duplicate_sites[duplicate_sites > 1]

task2_df = pd.DataFrame(columns=['Cleaned_Company_Website',
'Forage_Company_IDs'])
if not duplicate_sites.empty:
    records = [{'Cleaned_Company_Website': site,
                'Forage_Company_IDs': ', '.join(map(str,
df.loc[df['Cleaned_Company_Website'] == site,
'Forage_Company_ID'].unique()))}
               for site in duplicate_sites.index]
    task2_df = pd.DataFrame(records)
    print(task2_df)
else:
    print("✅ No duplicates found.")

# --- Save Task 1 & 2 results ---
with pd.ExcelWriter(output_path, engine='xlsxwriter') as writer:
    task1_df.to_excel(writer, sheet_name='Task1_CompanyLiUrl_Dups',
index=False)
    task2_df.to_excel(writer, sheet_name='Task2_CompanyWeb_Dups',
index=False)
print(f"\n📤 Output file saved at: {output_path}")

# --- QA Checks on Name Parsing ---

designation_list = ['CEO', 'Founder', 'Manager', 'PhD']
suffix_list = ['Jr', 'Sr', 'II', 'III']

if 'Personal_Linkedin_URL' not in df.columns:
    df['Personal_Linkedin_URL'] = None

main_cols = ['Personal_Linkedin_URL', 'Name', 'First_Name', 'Middle_Name',
'Last_Name', 'Designation', 'Suffix']

# Task 3: Name has >1 word, but <2 parsed values
def task_3(row):
    words = len(str(row['Name']).split())
    parsed = [row[c] for c in ['First_Name', 'Middle_Name', 'Last_Name',
'Designation', 'Suffix']]
    filled = sum(pd.notnull(x) and str(x).strip() != '' for x in parsed)
    return words > 1 and filled < 2
```

```
task3_df = df[df.apply(task_3,
axis=1)].drop_duplicates(subset='Personal_Linkedin_URL')

# Task 4: Name present but all 5 parsed fields empty
def task_4(row):
    if pd.isnull(row['Name']) or str(row['Name']).strip() == '':
        return False
    return all(pd.isnull(row[c]) or str(row[c]).strip() == '' for c in
['First_Name', 'Middle_Name', 'Last_Name', 'Designation', 'Suffix'])

task4_df = df[df.apply(task_4,
axis=1)].drop_duplicates(subset='Personal_Linkedin_URL')

# Task 5: Values Not in Name -- check if First_Name, Middle_Name, Last_Name
exist in Name
def value_not_in_name(row):
    for col in ['First_Name', 'Middle_Name', 'Last_Name', 'Designation',
'Suffix']:
        val = str(row[col]).strip() if pd.notna(row[col]) else ''
        name = str(row['Name']).strip()
        if val and val not in name:
            return True  # flag issue if any parsed value isn't in Name
    return False

task5_df = df[df.apply(value_not_in_name,
axis=1)].drop_duplicates(subset='Personal_Linkedin_URL')

# Task 6: Misplaced designation/suffix in wrong columns
def task_6(row):
    for c in ['First_Name', 'Middle_Name', 'Last_Name']:
        val = str(row[c]).strip()
        if val in designation_list + suffix_list:
            return True
    if str(row['Designation']).strip() in suffix_list or
str(row['Suffix']).strip() in designation_list:
        return True
    return False

task6_df = df[df.apply(task_6,
axis=1)].drop_duplicates(subset='Personal_Linkedin_URL')

# Task 7: Nickname in wrong columns
def task_7(row):
    name = str(row['Name'])
    nicknames = re.findall(r'\((.*?)\)|\"(.*?)\"|\'(.*?)\'', name)
    nicknames = [x for t in nicknames for x in t if x]
    for c in ['First_Name', 'Middle_Name', 'Last_Name', 'Designation',
'Suffix']:
        val = str(row[c]).strip()
        if val in nicknames and val not in designation_list + suffix_list:
            return True
    return False

task7_df = df[df.apply(task_7,
axis=1)].drop_duplicates(subset='Personal_Linkedin_URL')

# Task 8: Swapped First and Last Names
def task_8(row):
    if pd.isnull(row['Name']) or pd.isnull(row['First_Name']) or
pd.isnull(row['Last_Name']):
        return False
    words = str(row['Name']).split()
    if len(words) >= 2:
        return (row['First_Name'].strip() == words[-1] and
row['Last_Name'].strip() == words[0])
    return False

task8_df = df[df.apply(task_8,
axis=1)].drop_duplicates(subset='Personal_Linkedin_URL')

# --- Output QA Task Results with task names inline ---
print("\n--- QA Issues Summary ---")
task_dfs = [task3_df, task4_df, task5_df, task6_df, task7_df, task8_df]
task_labels = [
    "Task 3 (Low Parse Count)",
    "Task 4 (All Parse Missing)",
    "Task 5 (Values Not in Name)",
    "Task 6 (Misplaced Titles)",
    "Task 7 (Nickname Issues)",
    "Task 8 (Name Swapped)"
]

# Define the columns to include in QA output
qa_columns = ['Personal_Linkedin_URL', 'Name', 'First_Name', 'Middle_Name',
'Last_Name', 'Designation', 'Suffix']
```

```python
for task_name, df in zip(task_labels, task_dfs):
    print(f"{task_name}: {len(df)} issues")
    if not df.empty:
        print(df[qa_columns].head(3))
    else:
        print("No issues detected.")

# Save QA Output with only the specified columns
with pd.ExcelWriter(qa_output_path, engine='xlsxwriter') as writer:
    for df, sheet_name in zip(task_dfs, task_labels):
        df[qa_columns].to_excel(writer, sheet_name=sheet_name, index=False)
print(f"\n📤 Output file saved at: {qa_output_path}")
```

The next step is to do Slack integration, as we have it for us_address_parser.