hr_data_quality

October 13, 2025

1 Advanced HR Data Quality & Profiling Report

Objective: This notebook connects to the `aws_stage` database to perform an in-depth data quality analysis on the `raw__hr_kpi_t_sf_newsf_employees` table. It builds upon initial profiling by adding advanced, business-specific checks to identify logical inconsistencies in the HR data.

[1]:
```
import pyodbc
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from datetime import datetime

# Set plotting style and options
sns.set_style('whitegrid')
pd.set_option('display.max_columns', 100)
```

[2]:
```
# --- Connection Details ---
server = '172.22.74.254'
database = 'aws_stage'
username = 'extmertcan.coskun'
password = 'id3bGWpkLeDea4EAE4W9'
table_name = 'raw__hr_kpi_t_sf_newsf_employees'
schema_name = 'sf_odata'

# Construct the connection string
conn_str = (
    f'DRIVER={{ODBC Driver 17 for SQL Server}};'
    f'SERVER={server};'
    f'DATABASE={database};'
    f'UID={username};'
    f'PWD={password};'
    f'Encrypt=yes;'
    f'TrustServerCertificate=yes;'
)

# Establish connection
try:
```

```
    cnxn = pyodbc.connect(conn_str)
    cursor = cnxn.cursor()
    print("Connection to database established successfully!")
except Exception as e:
    print(f"Failed to connect to the database. Error: {e}")
```

```
Connection to database established successfully!
```

[7]:
```
query = f"SELECT * FROM [{schema_name}].[{table_name}]"

print("Loading full dataset from the database...")
df = pd.read_sql(query, cnxn)
print(f"Data loaded successfully with {len(df)} rows.")

# --- Data Preparation and Standardization ---
date_cols = ['start_date', 'job_start_date', 'job_end_date', 'end_date',␣
 ↪'date_of_birth',
             'initial_hire_date', 'seniority_base_date', 'db_upload_timestamp']

print("\nStandardizing all date columns to timezone-naive...")
for col in date_cols:
    if col in df.columns:
        # Step 1: Convert column to datetime objects, forcing errors into NaT
        df[col] = pd.to_datetime(df[col], errors='coerce')

        # Step 2: If the column has timezone info (is aware), convert it to UTC
        if df[col].dt.tz is not None:
            df[col] = df[col].dt.tz_convert('UTC')

            # Step 3: IMPORTANT - Remove the timezone info, making it naive but␣
 ↪standardized
            df[col] = df[col].dt.tz_localize(None)

print("Date standardization complete. All date columns are now timezone-naive.")
df.info()
```

```
Loading full dataset from the database...

C:\Users\mertc\AppData\Local\Temp\ipykernel_52080\1364919447.py:4: UserWarning:
pandas only supports SQLAlchemy connectable (engine/connection) or database
string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested.
Please consider using SQLAlchemy.
  df = pd.read_sql(query, cnxn)

Data loaded successfully with 7829 rows.

Standardizing all date columns to timezone-naive...
Date standardization complete. All date columns are now timezone-naive.
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 7829 entries, 0 to 7828
Columns: 120 entries, seq_number to db_upload_timestamp
dtypes: datetime64[ns](9), float64(4), int64(1), object(106)
memory usage: 7.2+ MB

C:\Users\mertc\AppData\Local\Temp\ipykernel_52080\1364919447.py:15: UserWarning:
Could not infer format, so each element will be parsed individually, falling
back to `dateutil`. To ensure parsing is consistent and as-expected, please
specify a format.
  df[col] = pd.to_datetime(df[col], errors='coerce')
```

[8]:
```python
# --- 5a. Missing Value Analysis ---
print("--- Missing Value Analysis ---")
missing_percentage = (df.isnull().sum() / len(df)) * 100
missing_df = pd.DataFrame({'column_name': df.columns, 'missing_percentage':␣
 ↪missing_percentage})
missing_df = missing_df[missing_df['missing_percentage'] > 0].
 ↪sort_values('missing_percentage', ascending=False)
print("Top 20 Columns with Missing Values (%):")
display(missing_df.head(20))
print('\n' + '='*80 + '\n')

# --- 5b. Date Field Range Analysis ---
print("--- Date Range Analysis ---")
date_summary = {col: {'Minimum Date': df[col].min(), 'Maximum Date': df[col].
 ↪max()} for col in date_cols if col in df.columns}
summary_df = pd.DataFrame.from_dict(date_summary, orient='index')
display(summary_df)
print('\n' + '='*80 + '\n')

# --- 5c. Categorical Data Distribution ---
print("--- Categorical Value Distributions ---")
categorical_cols_to_analyze = ['employee_status_en', 'workplace_en',␣
 ↪'payroll_company', 'gender']
for col in categorical_cols_to_analyze:
    if col in df.columns:
        print(f"\n--- Value Counts for '{col}' ---")
        print(df[col].value_counts(dropna=False).head(15))
print('\n' + '='*80 + '\n')

# --- 5d. Numeric Data Profiling ---
print("--- Descriptive Statistics for Numeric Columns ---")
numeric_df = df.select_dtypes(include=np.number)
if not numeric_df.empty:
    display(numeric_df.describe().T)
else:
    print("No numeric columns found.")
print('\n' + '='*80 + '\n')
```

```
--- Missing Value Analysis ---
Top 20 Columns with Missing Values (%):

                      column_name  missing_percentage
end_date                 end_date           99.808405
total_team_size   total_team_size           45.752970
team_member_size  team_member_size           45.752970
job_level               job_level            0.012773

================================================================================

--- Date Range Analysis ---

                              Minimum Date            Maximum Date
start_date          2004-07-13 00:00:00.000 2025-10-13 00:00:00.000
job_start_date      1996-06-10 00:00:00.000 2025-10-15 00:00:00.000
job_end_date        1753-01-01 00:00:00.000 2025-10-11 00:00:00.000
end_date            2025-10-13 00:00:00.000 2025-10-31 00:00:00.000
date_of_birth       1900-01-01 00:00:00.000 2025-07-01 00:00:00.000
initial_hire_date   1753-01-01 00:00:00.000 2025-10-15 00:00:00.000
seniority_base_date 1753-01-01 00:00:00.000 2025-10-15 00:00:00.000
db_upload_timestamp 2025-10-13 13:01:46.527 2025-10-13 13:01:46.527

================================================================================

--- Categorical Value Distributions ---

--- Value Counts for 'employee_status_en' ---
employee_status_en
Active              4248
Terminated          3575
Reported No Show       6
Name: count, dtype: int64

--- Value Counts for 'workplace_en' ---
workplace_en
VERİ AKTARIMI    2780
Site             2718
Central          1860
Corprate          470
None                1
Name: count, dtype: int64

--- Value Counts for 'payroll_company' ---
payroll_company
                                2780
REC ULUSLARARASI                1804
RÖNESANS ENDÜSTRİ TESİS          584
```

```
RÖNESANS YÖNETİM A.Ş.              277
RÖNESANS HOLDİNG                   234
RÖNESANS ÖZEL OKULLARI             215
RÖNESANS TÜRKMEN ŞUBE              196
RENAISSANCE INFRA CONSTRUCTION     164
RMI ULUSLARARASI İNŞ.TAAH.A.Ş      143
RÖNESANS GAYRİMENKUL               139
ÖZBEKİSTAN                         138
RNS Tesis Bakım Onarım AŞ          116
RCT Makine ve Güç Sistemleri        85
RSC RÖNESANS SATIN ALMA             78
BALLAST NEDAM INTERNATIONAL         75
Name: count, dtype: int64

--- Value Counts for 'gender' ---
gender
M    5807
F    2021
       1
Name: count, dtype: int64

================================================================================

--- Descriptive Statistics for Numeric Columns ---
                   count        mean         std    min  25%    50%    75%  \
seq_number        7829.0    1.178184    0.518492  1.000  1.0    1.0    1.0
total_team_size   4247.0    3.799388   29.377868  0.000  0.0    0.0    1.0
team_member_size  4247.0    0.989169    2.591405  0.000  0.0    0.0    1.0
job_level         7828.0  112.031170   86.567167  0.000  0.0  145.0  180.0
fte               7829.0    1.005306    0.144010  0.022  1.0    1.0    1.0

                      max
seq_number          7.000
total_team_size   1154.000
team_member_size    46.000
job_level          290.000
fte                 10.111

================================================================================
```

[9]:
```python
print("--- Logical Date Consistency Checks ---")

# Check 1: Job end date before job start date
invalid_end_dates = df[df['job_end_date'] < df['job_start_date']]
if not invalid_end_dates.empty:
```

```
    print(f"[ISSUE FOUND] {len(invalid_end_dates)} records where job_end_date␣
 ↪is before job_start_date.")
    display(invalid_end_dates[['user_id', 'job_start_date', 'job_end_date']].
 ↪head())
else:
    print("[OK] No records found where job_end_date is before job_start_date.")

# Check 2: Job start date before birth date
invalid_birth_dates = df[df['job_start_date'] < df['date_of_birth']]
if not invalid_birth_dates.empty:
    print(f"\n[ISSUE FOUND] {len(invalid_birth_dates)} records where␣
 ↪job_start_date is before date_of_birth.")
    display(invalid_birth_dates[['user_id', 'date_of_birth', 'job_start_date']].
 ↪head())
else:
    print("\n[OK] No records found where job_start_date is before date_of_birth.
 ↪")
print('\n' + '='*80 + '\n')
```

```
--- Logical Date Consistency Checks ---
[ISSUE FOUND] 4254 records where job_end_date is before job_start_date.
```

	user_id	job_start_date	job_end_date
2767	47002014	2004-07-13	1753-01-01
2768	47002030	2005-09-03	1753-01-01
2769	47010665	2017-05-24	1753-01-01
2770	47015152	2013-09-02	1753-01-01
2771	47011278	2023-07-01	1753-01-01

```
[OK] No records found where job_start_date is before date_of_birth.

================================================================================
```

[14]:
```
print("--- Employment Status Consistency Checks ---")

# The FIX: Create a timezone-NAIVE 'today' variable to match the standardized␣
 ↪columns
today = pd.to_datetime(datetime.now().date())

# Check 1: Active employees with a past end date
# This comparison now works because both df['end_date'] and today are␣
 ↪timezone-naive
active_with_past_end_date = df[(df['employee_status_en'] == 'Active') &␣
 ↪(df['end_date'] < today)]
if not active_with_past_end_date.empty:
```

```
    print(f"[ISSUE FOUND] {len(active_with_past_end_date)} 'Active' employees␣
 ↪have an end_date in the past.")
    display(active_with_past_end_date[['user_id', 'employee_status_en',␣
 ↪'end_date']].head())
else:
    print("[OK] No active employees found with a past end date.")

# Check 2: Terminated employees with no end date
terminated_no_end_date = df[(df['employee_status_en'] != 'Active') &␣
 ↪(df['end_date'].isnull())]
if not terminated_no_end_date.empty:
    print(f"\n[ISSUE FOUND] {len(terminated_no_end_date)} non-Active employees␣
 ↪are missing an end_date.")
    display(terminated_no_end_date[['user_id', 'employee_status_en',␣
 ↪'end_date']].head())
else:
    print("\n[OK] All non-Active employees have an end date.")
print('\n' + '='*80 + '\n')
```

```
--- Employment Status Consistency Checks ---
[OK] No active employees found with a past end date.

[ISSUE FOUND] 3575 non-Active employees are missing an end_date.
```

	user_id	employee_status_en	end_date
0	47045320	Terminated	NaT
1	47047553	Terminated	NaT
2	47050708	Terminated	NaT
3	47005344	Terminated	NaT
4	47008454	Terminated	NaT

```
================================================================================
```

[ ]:
```
print("--- ID and Hierarchy Integrity Checks ---")

# Check 1: Duplicate user_id entries
duplicate_users = df[df.duplicated(subset=['user_id'], keep=False)]
if not duplicate_users.empty:
    print(f"[ISSUE FOUND] {duplicate_users['user_id'].nunique()} user_ids have␣
 ↪duplicate records.")
    display(duplicate_users[['user_id', 'name', 'surname', 'start_date']].
 ↪sort_values('user_id').head())
else:
    print("[OK] All user_ids are unique.")

# Check 2: Employees who are their own manager
```

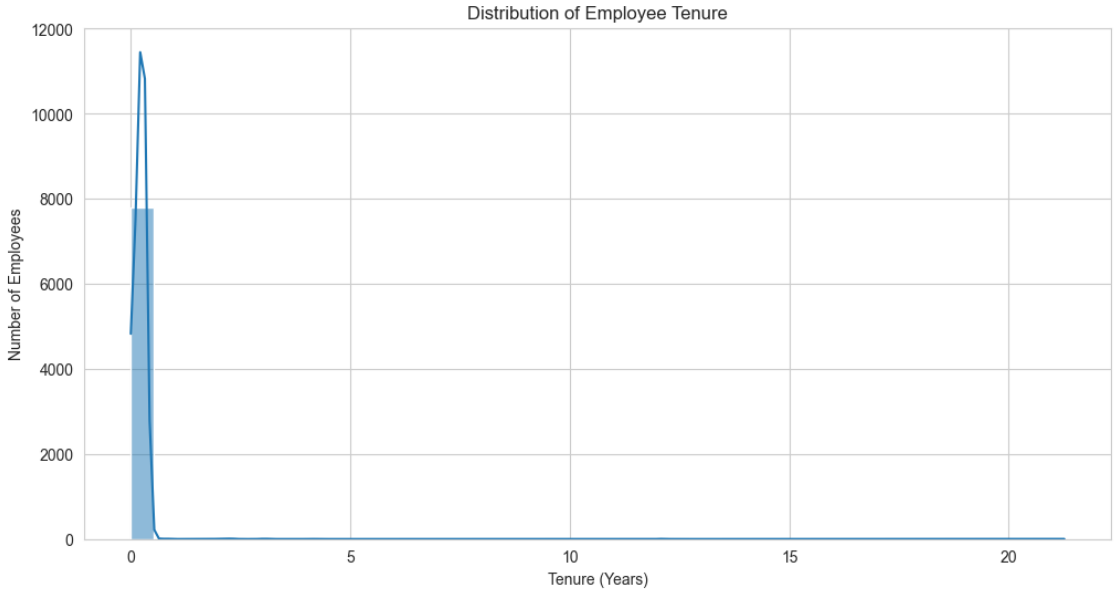
```
df['manager_user_id'] = pd.to_numeric(df['manager_user_id'], errors='coerce')
df['user_id'] = pd.to_numeric(df['user_id'], errors='coerce')
self_managed = df[df['user_id'] == df['manager_user_id']]
if not self_managed.empty:
    print(f"\n[ISSUE FOUND] {len(self_managed)} employees are listed as their␣
 ↪own manager.")
    display(self_managed[['user_id', 'name', 'surname', 'manager_user_id']].
 ↪head())
else:
    print("\n[OK] No employees are listed as their own manager.")
print('\n' + '='*80 + '\n')
```

```
--- ID and Hierarchy Integrity Checks ---
[OK] All user_ids are unique.

[OK] No employees are listed as their own manager.

================================================================================
```

[15]:
```
print("--- Employee Tenure Analysis ---")

# Calculate tenure in years. For active employees, use today as the end date.
end_date_for_tenure = df['end_date'].fillna(today)
df['tenure_days'] = (end_date_for_tenure - df['start_date']).dt.days
df['tenure_years'] = df['tenure_days'] / 365

# Check for negative tenure
negative_tenure = df[df['tenure_years'] < 0]
if not negative_tenure.empty:
    print(f"[ISSUE FOUND] {len(negative_tenure)} employees have negative tenure.
 ↪")
    display(negative_tenure[['user_id', 'start_date', 'end_date',␣
 ↪'tenure_years']].head())
else:
    print("[OK] No employees with negative tenure found.")

# Display descriptive statistics for tenure
print("\nDescriptive Statistics for Employee Tenure (in Years):")
display(df['tenure_years'].describe())

# Visualize the distribution
plt.figure(figsize=(12, 6))
sns.histplot(df['tenure_years'].dropna(), bins=40, kde=True)
plt.title('Distribution of Employee Tenure')
plt.xlabel('Tenure (Years)')
plt.ylabel('Number of Employees')
```

```
plt.show()
```

```
--- Employee Tenure Analysis ---
[OK] No employees with negative tenure found.

Descriptive Statistics for Employee Tenure (in Years):

count    7829.000000
mean        0.223782
std         0.507016
min         0.000000
25%         0.115068
50%         0.282192
75%         0.282192
max        21.265753
Name: tenure_years, dtype: float64
```



[16]:
```
try:
    cnxn.close()
    print("Database connection closed.")
except NameError:
    print("No active database connection to close.")
```

```
Database connection closed.
```

[ ]: