Data Cleaning

Comprehensive Data Cleaning & Exploratory Analysis of Job Market Trends

Anu Sharma, Cindy Guzman, Gavin Boss

# Looks for the duration of job postings

This was to ensure that the date format for the data was being pulled in the correct way, this is set up to pull the format of M/D/YYYY, this also makes sur ethat it sorts between expired and posted and fills any values that don’t represent either **EXPIRED** or **POSTED** with **DURATION** value.

from pyspark.sql.functions import col, to\_date, datediff, when
#| label: duration
#| include: true

# Use a flexible date format
date\_cols = ["POSTED", "EXPIRED", "MODELED\_EXPIRED"]
for colname in date\_cols:
 df = df.withColumn(colname, col(colname).cast("string"))
 df = df.withColumn(colname, to\_date(col(colname), "M/d/yyyy"))

# --- Step 2: Compute durations safely ---
df = df.withColumn(
 "DURATION",
 when(col("DURATION").isNull(), datediff("EXPIRED", "POSTED")).otherwise(col("DURATION"))
).withColumn(
 "MODELED\_DURATION",
 when(col("MODELED\_DURATION").isNull(), datediff("MODELED\_EXPIRED", "POSTED")).otherwise(col("MODELED\_DURATION"))
)

# Column Cleaning

This is to ensure that we are only sorting columns that could potentially be of use, this isn’t every column that was used, but we cut out columns that we knew wouldn’t be relevant for the data that we were going to be incorporating into the datasets. While these columns could have been refined even further to minimize the size of the dataset, but we were able to minimize the way that the EC2 utilized the data frames to incorporate and read the data to where we had the data cleaned just in case we needed it.

# Imports
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, regexp\_replace

# All columns that are being cleaned
text\_columns = [
 "TITLE\_RAW", "BODY", "SKILLS\_NAME",
 "SPECIALIZED\_SKILLS\_NAME",
 "CERTIFICATIONS\_NAME",
 "COMMON\_SKILLS\_NAME",
 "SOFTWARE\_SKILLS\_NAME", "URL",
 "EDUCATION\_LEVELS\_NAME", "LIGHTCAST\_SECTORS\_NAME",
 "CIP6\_NAME", "CIP4\_NAME", "CIP2\_NAME",
 "TITLE\_NAME", "TITLE\_CLEAN",
 "COMPANY\_NAME", "COMPANY\_RAW",
 "ONET", "ONET\_NAME", "ONET\_2019", "ONET\_2019\_NAME",
 "SOC\_2021\_2\_NAME", "SOC\_2021\_3\_NAME", "SOC\_2021\_4\_NAME", "SOC\_2021\_5\_NAME",
 "LOT\_CAREER\_AREA\_NAME", "LOT\_OCCUPATION\_NAME", "LOT\_SPECIALIZED\_OCCUPATION\_NAME",
 "LOT\_OCCUPATION\_GROUP\_NAME", "LOT\_V6\_SPECIALIZED\_OCCUPATION\_NAME",
 "LOT\_V6\_OCCUPATION\_NAME", "LOT\_V6\_OCCUPATION\_GROUP\_NAME", "LOT\_V6\_CAREER\_AREA\_NAME",
 "NAICS\_2022\_6\_NAME"
]

# clean text columns
def clean\_text\_columns(df, columns):
 """
 Cleans text-heavy columns by:
 - Removing brackets, quotes, and newlines
 - Replacing multiple spaces with single space
 - Standardizing commas with proper spacing
 """
 pattern\_cleanup = r'[\[\]\n{}"]' # remove brackets, newlines, braces, quotes
 for c in columns:
 df = df.withColumn(c, regexp\_replace(col(c), pattern\_cleanup, ""))
 df = df.withColumn(c, regexp\_replace(col(c), r'\s{2,}', ' ')) # multiple spaces → 1 space
 df = df.withColumn(c, regexp\_replace(col(c), r'\s\*,\s\*', ', ')) # standardize commas
 return df

# Apply cleaning
df = clean\_text\_columns(df, text\_columns)

# Preview cleaned text (Don't need to render for the project just for testing)
# df.select(text\_columns).show(5, truncate=100)

# Targeted column-specific cleaning

This code targets the Education Levels, Location, and Remote work Labels to clean up the dataframes in order to maintain the naming conventions needed for the datasets.

from pyspark.sql.functions import col, regexp\_extract, when, regexp\_replace
#| label: targeted\_cleaning
#| include: true

# Education levels → keep digits
df = df.withColumn("EDUCATION\_LEVELS", regexp\_extract("EDUCATION\_LEVELS", r'(\d+)', 1))

# Location cleanup
df = df.withColumn("LOCATION", regexp\_replace(col("LOCATION"), r"\s\*\n\s\*", " "))
df = df.withColumn("LOCATION", regexp\_replace(col("LOCATION"), r"[{}]", ""))

# Standardize remote work labels
df = df.withColumn(
 "REMOTE\_TYPE\_NAME",
 when(col("REMOTE\_TYPE\_NAME").isin("[None]", "Not Remote") | col("REMOTE\_TYPE\_NAME").isNull(), "On-Site")
 .when(col("REMOTE\_TYPE\_NAME") == "Hybrid Remote", "Hybrid")
 .when(col("REMOTE\_TYPE\_NAME") == "Remote", "Remote")
 .otherwise(col("REMOTE\_TYPE\_NAME"))
)

# Save the cleaned dataset

This saves the dataset, and saves it under our **/data** folder which is where we save our lightcast.csv, and our new cleaned dataframe also gets saved here and can be pulled from later when it is time to use the data for creating models.