BDCC_Final_Part1_Data_Cleaning_Kenny

December 2, 2024

1 ADSP Big Data and Cloud Computing Final Project

1.1 Part 1

1.2 Author: Jingkai Wang

1.2.1 Date: Nov 29, 2024

1.3 Start Code

1.4 Reading Git Final Project

[1]:
```
import os
import subprocess
import datetime
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from pyspark.sql import functions as F
from pyspark.sql.types import *
from pyspark.sql.functions import explode, length, expr
from pyspark.sql.functions import col, from_unixtime, to_date, regexp_extract

pd.set_option('display.max_rows', 100)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', None)
```

[2]:
```
spark.conf.set("spark.sql.repl.eagerEval.enabled",True)
```

[3]:
```
gcs_folder = 'gs://msca-bdp-data-open/final_project_git'
```

Check data size in GCS

[4]:
```
cmd = 'gsutil du -s -h ' + gcs_folder

p = subprocess.Popen(cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.
 ↪STDOUT, universal_newlines=True)
for line in p.stdout.readlines():
```

```
    print (f'Total directory size: {line}')

retval = p.wait() # Wait for the child process to terminate.
```

```
Total directory size: 1.36 TiB     gs://msca-bdp-data-open/final_project_git
```

1.4.1 Read Git data from GCS

Languages Programming languages by repository as reported by GitHub's https://developer.github.com/v3/repos/#list-languages API

[5]:
```
%%time

df_languages = spark.read.parquet(os.path.join(gcs_folder, 'languages'))
print(f'Records read from dataframe *languages*: {df_languages.count():,.0f}')
```

```
[Stage 1:==============================>                          (2 + 2) / 4]

Records read from dataframe *languages*: 3,325,634
CPU times: user 11.9 ms, sys: 401 µs, total: 12.3 ms
Wall time: 10.3 s
```

[6]:
```
df_languages.printSchema()
```

```
root
 |-- repo_name: string (nullable = true)
 |-- language: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- name: string (nullable = true)
 |    |    |-- bytes: long (nullable = true)
```

Licenses Open source license SPDX code for each repository as detected by https://developer.github.com/v3/licenses/

[7]:
```
%%time

df_licenses = spark.read.parquet(os.path.join(gcs_folder, 'licenses'))
print(f'Records read from dataframe *licenses*: {df_licenses.count():,.0f}')
```

```
Records read from dataframe *licenses*: 3,325,634
CPU times: user 3.74 ms, sys: 1.13 ms, total: 4.88 ms
Wall time: 1 s
```

[8]:
```
df_licenses.printSchema()
```

```
root
 |-- repo_name: string (nullable = true)
 |-- license: string (nullable = true)
```

Commits Unique Git commits from open source repositories on GitHub, pre-grouped by repositories they appear in.

[9]:
```
%%time

df_commits = spark.read.parquet(os.path.join(gcs_folder, 'commits'))
print(f'Records read from dataframe *commits*: {df_commits.count():,.0f}')
```

```
[Stage 11:>                                                         (0 + 1) / 1]

Records read from dataframe *commits*: 265,419,190
CPU times: user 101 ms, sys: 19.7 ms, total: 121 ms
Wall time: 32.6 s
```

[10]:
```
df_commits.printSchema()
```

```
root
 |-- commit: string (nullable = true)
 |-- tree: string (nullable = true)
 |-- parent: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- author: struct (nullable = true)
 |    |-- name: string (nullable = true)
 |    |-- email: string (nullable = true)
 |    |-- time_sec: long (nullable = true)
 |    |-- tz_offset: long (nullable = true)
 |    |-- date: struct (nullable = true)
 |    |    |-- seconds: long (nullable = true)
 |    |    |-- nanos: long (nullable = true)
 |-- committer: struct (nullable = true)
 |    |-- name: string (nullable = true)
 |    |-- email: string (nullable = true)
 |    |-- time_sec: long (nullable = true)
 |    |-- tz_offset: long (nullable = true)
 |    |-- date: struct (nullable = true)
 |    |    |-- seconds: long (nullable = true)
 |    |    |-- nanos: long (nullable = true)
 |-- subject: string (nullable = true)
 |-- message: string (nullable = true)
 |-- trailer: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- key: string (nullable = true)
```

```
 |    |    |-- value: string (nullable = true)
 |    |    |-- email: string (nullable = true)
 |-- difference: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- old_mode: long (nullable = true)
 |    |    |-- new_mode: long (nullable = true)
 |    |    |-- old_path: string (nullable = true)
 |    |    |-- new_path: string (nullable = true)
 |    |    |-- old_sha1: string (nullable = true)
 |    |    |-- new_sha1: string (nullable = true)
 |    |    |-- old_repo: string (nullable = true)
 |    |    |-- new_repo: string (nullable = true)
 |-- difference_truncated: boolean (nullable = true)
 |-- repo_name: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- encoding: string (nullable = true)
```

Contents Unique file contents of text files under 1 MiB on the HEAD branch.
Can be joined to `files` dataset using the id columns to identify the repository and file path.

[11]:
```
%%time

df_contents = spark.read.parquet(os.path.join(gcs_folder, 'contents'))
print(f'Records read from dataframe *commits*: {df_contents.count():,.0f}')
```

```
[Stage 15:>                                                         (0 + 1) / 1]

Records read from dataframe *commits*: 281,191,977
CPU times: user 69.4 ms, sys: 26.9 ms, total: 96.2 ms
Wall time: 25.2 s
```

[12]:
```
df_contents.printSchema()
```

```
root
 |-- id: string (nullable = true)
 |-- size: long (nullable = true)
 |-- content: string (nullable = true)
 |-- binary: boolean (nullable = true)
 |-- copies: long (nullable = true)
```

Files File metadata for all files at HEAD.
Join with `contents` dataset on id columns to search text.

[13]:
```
%%time
```

```
df_files = spark.read.parquet(os.path.join(gcs_folder, 'files'))
print(f'Records read from dataframe *files*: {df_files.count():,.0f}')
```

```
[Stage 17:=================================================>(1079 + 1) / 1080]

Records read from dataframe *files*: 2,309,424,945
CPU times: user 14.6 ms, sys: 4.31 ms, total: 18.9 ms
Wall time: 6.45 s
```

[14]:
```
df_files.printSchema()
```

```
root
 |-- repo_name: string (nullable = true)
 |-- ref: string (nullable = true)
 |-- path: string (nullable = true)
 |-- mode: long (nullable = true)
 |-- id: string (nullable = true)
 |-- symlink_target: string (nullable = true)
```

1.5 Project code

1.5.1 Step 1: Discard irrelevant or obviously erroneous data

Most of the variable names should be self-explanatory, however data is deeply nested and will require detailed review in order to select the most appropriate data elements ### Step 2: Complete thorough EDA to identify which variables you can use to complete your analysis Any poorly populated or duplicate variables should be discarded

[15]:
```
%%time
# Set a sample to analyze data
sample = 0.1
df_languages = df_languages.sample(withReplacement=False, fraction=sample)
df_licenses = df_licenses.sample(withReplacement=False, fraction=sample)
df_commits = df_commits.sample(withReplacement=False, fraction=sample)
df_contents = df_contents.sample(withReplacement=False, fraction=sample)
df_files = df_files.sample(withReplacement=False, fraction=sample)
```

```
CPU times: user 1.8 ms, sys: 527 µs, total: 2.33 ms
Wall time: 21.4 ms
```

[16]:
```
# for languages
# Drop duplicates first
df_languages = df_languages.dropDuplicates()

# Explode the 'language' field to flatten the nested structure
```

```
df_languages_flattened = df_languages.withColumn('language',␣
 ↪explode('language'))

# Select the relevant fields from the flattened structure
df_languages_extracted = df_languages_flattened.select(
    'repo_name',
    col('language.name').alias('language'),
    col('language.bytes').alias('bytes'))

# Clean df_languages
df_languages_cleaned = df_languages_extracted.filter(
    (col('repo_name').isNotNull()) &
    (col('language').isNotNull()) &
    (col('bytes') > 0)
).dropDuplicates()

# show result
print('Records count', df_languages_cleaned.count())
df_languages_cleaned.printSchema()
df_languages_cleaned.limit(5)
```

```
Records count 782324
root
 |-- repo_name: string (nullable = true)
 |-- language: string (nullable = true)
 |-- bytes: long (nullable = true)
```

[16]:
```
+--------------------+--------+-------+
|           repo_name|language|  bytes|
+--------------------+--------+-------+
|NSAMR/uk.ac.nsamr…|     CSS|2781949|
|kirtgoh/gcc-vcg-p…|   Shell| 290218|
|      ChrisOHu/vimrc|  Erlang|  10020|
|gramic/rules_closure|Starlark| 250591|
|portefaix/docker-…|     HCL|    206|
+--------------------+--------+-------+
```

[17]:
```
# for licenses
# Drop duplicates first
df_licenses = df_licenses.dropDuplicates()

# Clean df_licenses
df_licenses_cleaned = df_licenses.filter(
```

```
    (col('repo_name').isNotNull()) &
    (col('license').isNotNull())
)

# print result
print('Records count', df_licenses_cleaned.count())
df_licenses_cleaned.printSchema()
df_licenses_cleaned.limit(5)
```

```
Records count 332254
root
 |-- repo_name: string (nullable = true)
 |-- license: string (nullable = true)
```

[17]:
```
+--------------------+------------+
|           repo_name|     license|
+--------------------+------------+
|bennie/perl-Text-…|artistic-2.0|
|ocefpaf/weather_a…|artistic-2.0|
|alexneri/umi-sono…|artistic-2.0|
|   squell/bb-scripts|artistic-2.0|
|         ehmicky/Koi|artistic-2.0|
+--------------------+------------+
```

[18]:
```
# for commits
# Extract commit_date and timestamp
df_commits_cleaned = df_commits.select(
    col('commit'),
    col('author.name').alias('author_name'),
    col('author.email').alias('author_email'),
    from_unixtime(col('author.date.seconds'), 'yyyy-MM-dd HH:mm:ss').
 ↪alias('author_date'),
    col('committer.name').alias('committer_name'),
    col('committer.email').alias('committer_email'),
    from_unixtime(col('committer.date.seconds'), 'yyyy-MM-dd HH:mm:ss').
 ↪alias('committer_date'),
    col('subject'),
    col('message'),
    expr('repo_name[0]').alias('repo_name')
)

# drop the duplicate
df_commits_cleaned = df_commits_cleaned.dropDuplicates()
```

```
# Clean df_commits
df_commits_cleaned = df_commits_cleaned.filter(
    (col('commit').isNotNull()) &
    (col('repo_name').isNotNull()) &
    (col('author_name').isNotNull()) &
    (col('message').isNotNull()) &
    (col('author_date') < F.current_date()) &
    (col('committer_date') < F.current_date()) &
    (length(F.col('message')) > 5)  # Exclude trivial messages
)

# print result
#print('Records count', df_commits_cleaned.count())
df_commits_cleaned.printSchema()
#df_commits_cleaned.limit(5)
```

```
root
 |-- commit: string (nullable = true)
 |-- author_name: string (nullable = true)
 |-- author_email: string (nullable = true)
 |-- author_date: string (nullable = true)
 |-- committer_name: string (nullable = true)
 |-- committer_email: string (nullable = true)
 |-- committer_date: string (nullable = true)
 |-- subject: string (nullable = true)
 |-- message: string (nullable = true)
 |-- repo_name: string (nullable = true)
```

[19]:
```
# For contents
# Clean df_contents
df_contents_cleaned = df_contents.filter(
    (F.col('content').isNotNull()) &
    (~F.col('binary')) &
    (F.col('size') > 100) &  # Minimum size
    (F.col('size') < 1048576)  # Maximum size: 1 MB
).dropDuplicates()


# print result
#print('Records count', df_contents_cleaned.count())
df_contents_cleaned.printSchema()
#df_contents_cleaned.limit(5)
```

```
root
 |-- id: string (nullable = true)
 |-- size: long (nullable = true)
```

```
 |-- content: string (nullable = true)
 |-- binary: boolean (nullable = true)
 |-- copies: long (nullable = true)
```

[20]:
```
# For file
# drop the column that all are null
df_files_cleaned = df_files.drop('symlink_target')

# Drop duplicates
df_files_cleaned = df_files_cleaned.dropDuplicates()

# Clean df_files
df_files_cleaned = df_files_cleaned.filter(
    (col('repo_name').isNotNull()) &
    (col('path').isNotNull()) &
    (col('id').isNotNull())
)

# print result
#print('Records count', df_files_cleaned.count())
df_files_cleaned.printSchema()
#df_files_cleaned.limit(5)
```

```
root
 |-- repo_name: string (nullable = true)
 |-- ref: string (nullable = true)
 |-- path: string (nullable = true)
 |-- mode: long (nullable = true)
 |-- id: string (nullable = true)
```

[21]:
```
# Export the cleaned data
# storage path in general
cleaned_data = 'gs://msca-bdp-students-bucket/notebooks/jingkaiw/cleaned_data'

# Save the languages data
path_languages = os.path.join(cleaned_data, 'df_languages_cleaned')
df_languages_cleaned.write.mode('overwrite').parquet(path_languages) # ensure␣
 ↪save multiple times

# Save licenses data
path_licenses = os.path.join(cleaned_data, 'df_licenses_cleaned')
df_licenses_cleaned.write.mode('overwrite').parquet(path_licenses) # ensure␣
 ↪save multiple times

# Save commites data
```

```
path_commits = os.path.join(cleaned_data, 'df_commits_cleaned')
df_commits_cleaned.write.mode('overwrite').parquet(path_commits) # ensure save␣
↪multiple times

# Save contents data
path_contents = os.path.join(cleaned_data, 'df_contents_cleaned')
df_contents_cleaned.write.mode('overwrite').parquet(path_contents) # ensure␣
↪save multiple times

# Save files data
path_files = os.path.join(cleaned_data, 'df_files_cleaned')
df_files_cleaned.write.mode('overwrite').parquet(path_files) # ensure save␣
↪multiple times
```

[ ]:

[ ]:

[22]:
```
import datetime
import pytz

datetime.datetime.now(pytz.timezone('US/Central')).strftime("%a, %d %B %Y %H:%M:
↪%S")
```

[22]: 'Sun, 01 December 2024 18:59:20'