DATA MANAGEMENT FOR ANALYTICS - DNSC 6305

INDIVIDUAL ASSIGNMENT

GOKUL KUMAR KESAVAN - G25385029

In this task, we conducted a detailed analysis of individual contributions to Federal Committees such as Presidential, Senate and House committees for the period between June 20, 2024, and July 23, 2024. Using Apache Spark's distributed computing capabilities, we processed and analyzed large datasets to extract meaningful insights about the patterns and trends of political contributions during this time frame.

DATA COLLECTION

The data for this analysis was obtained from the Federal Election Commission (FEC) and consists of individual contributions to political committees.

We have used the 2023-2024 contributions dataset. The files of interest for this task include:

- itcont_2024_20240620_20240709.txt: Covers the period from June 20,2024, to July 09,2024.
- itcont_2024_20240710_20240723.txt: Covers the period from July 10,2024, to July 23,2024.

These files are delimited using the pipe symbol (|) and contain all contributions reported by individuals within the specified duration.

The Individual Data Dictionary provides detailed information about the fields present in the data files. Some of the key attributes include:

- CMTE_ID: Committee ID receiving the contribution.
- NAME: Name of the contributor.
- TRANSACTION_AMT: Contribution amount.
- CITY, STATE, ZIP_CODE: Geographical details.
- EMPLOYER and OCCUPATION: Employment details of the contributor.

To understand the structure of the data, the file header provided with the dataset is used. It defines the columns and their respective order, ensuring proper data mapping and processing.

DATA PROCESSING

We used an AWS EC2 t2.large instance with 60+ GB of data storage to process and analyze the data. This instance provides sufficient computational power and storage for handling large datasets.

In [1]:
```
#To make sure we are in the right directory
!pwd
```

```
/home/ubuntu
```

We began by uploading the zip file containing the two raw text documents (itcont_2024_20240620_20240709.txt and itcont_2024_20240710_20240723.txt) along with the header file indiv_header_file.csv

In [2]:
```
# Unzipping the file and storing them into a new folder using !unzip command
# -o : Overwrites existing files in the target directory
# -d : Specifies the destination directory where the contents will be extracted
!unzip -o Ind_Assign_AWS.zip -d ./Ind_Assign_AWS/
```

```
Archive:  Ind_Assign_AWS.zip
  inflating: ./Ind_Assign_AWS/itcont_2024_20240620_20240709.txt
  inflating: ./Ind_Assign_AWS/itcont_2024_20240710_20240723.txt
```

In [3]:
```
# Deleing the zip folder
!rm Ind_Assign_AWS.zip
```

The original data files were in (|) delimited format. To facilitate further processing, we convert these files into CSV format.

In [4]:
```
# !csvformat : A command from the csvkit library used to format and manipulate CSV
# -d "|": Specifies the delimiter used in the input file
# Ind_Assign_AWS/itcont_2024_20240620_20240709.txt : Path of the input file
# > : This symbol redirects the output of the csvformat command to a new file
# June2024.csv, July2024: Name of the ouput file
!csvformat -d "|" Ind_Assign_AWS/itcont_2024_20240620_20240709.txt > June2024.csv
!csvformat -d "|" Ind_Assign_AWS/itcont_2024_20240710_20240723.txt > July2024.csv
```

In [5]:
```
# !csvclean : A command from the csvkit library used to clean and validate CSV file
# -n : The "dry-run" option, which validates the file without actually modifying it
!csvclean -n June2024.csv
!csvclean -n July2024.csv
```

```
No errors.
No errors.
```

After validating the files, we concatenated the header file (indiv_header_file.csv) with the two CSV files (June2024.csv and July2024.csv) to create a single unified file named final.csv

In [6]:
```
!cat indiv_header_file.csv > final.csv

!cat June2024.csv >> final.csv
!cat July2024.csv >> final.csv
```

To verify the integrity of the combined dataset, we use the wc -l command to count the number of rows in the individual files and the final combined file

In [7]:
```
# wc : Word count, can also be used to count line, words or character in a file
# -l : It tells the wc to count the total number of lines in a given file
!wc -l indiv_header_file.csv
!wc -l Ind_Assign_AWS/itcont_2024_20240620_20240709.txt
!wc -l Ind_Assign_AWS/itcont_2024_20240710_20240723.txt
!wc -l final.csv
```

```
1 indiv_header_file.csv
1944602 Ind_Assign_AWS/itcont_2024_20240620_20240709.txt
1902192 Ind_Assign_AWS/itcont_2024_20240710_20240723.txt
3846795 final.csv
```

The total number of rows in Final.csv is the sum of all three files. This confirms that all rows from the individual files were successfully concatenated into the final.csv file

Removing intermediate files and directories to free up storage space and keep the workspace organized, as they are no longer needed after generating the final dataset.

In [8]:
```
# Deleting the unzipped folder
# !rm -r: Command to remove files or directories
!rm -r Ind_Assign_AWS

# Deleteing the header file and intermediate file generated after converting data f
!rm June2024.csv
!rm July2024.csv
!rm indiv_header_file.csv
```

DATA ANALYSIS PART 1: RDD PROCESSING

Every time we use Spark, we'll have to run the following steps:

In [9]:
```
import findspark
```

In [10]:
```
findspark.init()
```

The next two steps are required for the simple Python style of using Spark :

In [11]:
```
from pyspark import SparkContext
```

In [12]:
```
spark = SparkContext(appName= 'AWS_Assignment3')
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(ne
wLevel).
24/11/24 16:53:02 WARN NativeCodeLoader: Unable to load native-hadoop library for yo
ur platform... using builtin-java classes where applicable
```

In [13]:
```
spark
```

Out[13]: **SparkContext**

Spark UI

Version	`v3.5.2`
Master	`local[*]`
AppName	`AWS_Assignment3`

In [14]:
```
# Topstates: This is the variable that stores the RDD created from the final.csv fi
# spark.textFile(): Method provided by Apache Spark's RDD (Resilient Distributed Da
Topstates = spark.textFile('final.csv')
```

In [15]:
```
# Retrieve the header row of the dataset containing column names.
header = Topstates.first()
header
```

Out[15]:
```
'CMTE_ID,AMNDT_IND,RPT_TP,TRANSACTION_PGI,IMAGE_NUM,TRANSACTION_TP,ENTITY_TP,NAME,
CITY,STATE,ZIP_CODE,EMPLOYER,OCCUPATION,TRANSACTION_DT,TRANSACTION_AMT,OTHER_ID,TR
AN_ID,FILE_NUM,MEMO_CD,MEMO_TEXT,SUB_ID'
```

```
24/11/24 16:53:17 WARN GarbageCollectionMetrics: To enable non-built-in garbage coll
ector(s) List(G1 Concurrent GC), users should configure it(them) to spark.eventLog.g
cMetrics.youngGenerationGarbageCollectors or spark.eventLog.gcMetrics.oldGenerationG
arbageCollectors
```

Using code and the function 'add', we are going to find:

1. Top 10 States by Contribution Count:
   - Identifying the top 10 states based on the total 'count' of contributions.
2. Top 10 States by Contribution Amount:
   - Identifying the top 10 states based on the 'sum' of contribution amounts.

In [16]:
```
from operator import add
```

1. Top 10 States by Contribution Count

In [17]:
```
top10 = Topstates.filter(lambda row: row != header) \
    .map(lambda row: row.replace('"', '')) \
    .map(lambda row: row.split(",")) \
    .map(lambda cols: (cols[10], 1)) \
    .reduceByKey(add) \
    .takeOrdered(10, key=lambda pair: -pair[1])
for STATE, count in top10:
    print("{}\t{}".format(STATE, count))
```

```
[Stage 2:============================================>              (17 + 2) / 22]
```

```
CA      564941
TX      324187
FL      279517
NY      225839
WA      140701
PA      130224
VA      125452
IL      124756
AZ      114972
OH      109487
```

EXPLANATION OF THE ABOVE CODE:

1. filter(lambda row: row != header):

- Removes the header row from the dataset (Topstates).

2. map(lambda row: row.replace('"', '')):

- Removes double quotes (") from each row to clean the data.
- This step ensures no issues arise due to extra quotes during processing.

3. map(lambda row: row.split(",")):

- Splits each row into a list of columns using a comma delimiter (",").

4. map(lambda cols: (cols[10], 1)):

- Extracts the state information from column index 10 (The column representing the state).
- Maps each state to the value 1, marking each contribution as a single count.

5. reduceByKey(add):

- Groups all the contributions by the state (key) and sums up the counts (value) for each state.

6. takeOrdered(10, key=lambda pair: -pair[1]):

- Extracts the top 10 states based on the count of contributions in descending order.
- key=lambda pair: -pair[1] ensures sorting is done in descending order of the second element (the count).

7. for STATE, count in top10::

- Iterates over the top 10 states and their respective contribution counts.

8. print("{}\t{}".format(STATE, count)):

- Prints each state and its contribution count in a tab-separated format.

2. Top 10 States by Contribution Amount

In [18]:
```
def is_valid_float(value):
    try:
        float(value)
        return True
    except (ValueError, TypeError):
        return False
```

In [19]:
```
top10_tot_amount = Topstates.filter(lambda row: row != header) \
    .map(lambda row: row.replace('"', '')) \
    .map(lambda row: row.split(",")) \
    .filter(lambda cols: cols[15] is not None) \
    .filter(lambda cols: is_valid_float(cols[15])) \
    .map(lambda cols: (cols[10], float(cols[15]))) \
    .reduceByKey(add) \
    .takeOrdered(10, key=lambda pair: -pair[1])
for STATE, total in top10_tot_amount:
    print("{}\t{}".format(STATE, total))
```

```
[Stage 4:==========================================>                (16 + 2) / 22]
CA      64846446564.0
TX      55446914035.0
IL      34138029513.0
NY      30557933743.0
FL      26208594146.0
GA      23379784573.0
VA      21874918857.0
PA      21375698873.0
NJ      20531460523.0
MA      18481600069.0
```

EXPLANATION OF THE ABOVE CODE:

1. filter(lambda row: row != header):

- Removes the header row from the dataset (Topstates).

2. map(lambda row: row.replace('"', '')):

- Removes double quotes (") from each row to clean the data.
- This step ensures no issues arise due to extra quotes during processing.

3. map(lambda row: row.split(",")):

- Splits each row into a list of columns using a comma delimiter (",").

4. filter(lambda cols: cols[15] is not None):

- Ensures that the value in column 15 (transaction amount) is not None

5. filter(lambda cols: is_valid_float(cols[15])):

- Uses the helper function is_valid_float to check whether the value in column 15 is a valid floating-point number.
- Ensures that only numeric transaction amounts are processed.

6. map(lambda cols: (cols[10], float(cols[15]))):

- Extracts the state information from column 10 and the transaction amount from column 15.
- Converts the transaction amount into a floating-point number.

5. reduceByKey(add):

- Groups all the contributions by the state (key) and sums up the total transaction amount (value) for each state.

6. takeOrdered(10, key=lambda pair: -pair[1]):

- Extracts the top 10 states based on the total transaction amount in descending order.
- The negative sign (-pair[1]) ensures descending order by the second element (the total amount).

7. for STATE, count in top10::

- Iterates over the top 10 states and their respective total contribution counts.

8. print("{}\t{}".format(STATE, count)):

- Prints each state and its total contribution in a tab-separated format.

Summary and Insights: RDD Processing Analysis

Findings:

1. Top 10 States by Contribution Count:

- California (CA) leads the list with the highest number of contributions, followed by Texas (TX), Florida (FL), and New York (NY).
- States like Illinois (IL) and Pennsylvania (PA) also show significant contributions, which reflects their importance as swing states in political campaigns.

2. Top 10 States by Contribution Amount:

- California (CA) again dominates in total contribution amounts, with Texas (TX) close behind, contributing billions during the observed period.
- Illinois (IL), Florida (FL), and Georgia (GA) also indicating strong financial support from contributors in these states.
- The presence of Virginia (VA) and Pennsylvania (PA) in the top list suggests that they are strategic hubs for fundraising, possibly due to their proximity to Washington, D.C., and

their influence in national politics.

Patterns and Trends Observed:

1. Contribution Volume vs Amount:

- States like California and Texas consistently rank high in both contribution count and monetary contributions, indicating high political engagement and wealth distribution.

2. Regional Influence:

- Most contributions come from states with larger economies or political importance, highlighting the political engagement, and campaign funding.

Reflection on Findings:

- The dominance of California and Texas is expected due to their large populations, economic strength and their strategic roles in presidential and congressional elections.
- The significant contributions from Virginia align with their political significance as battleground states for campaign activities.

Significance in Political Campaigns and Elections:

1. Impact on Campaign Strategies:

- States like California and Texas act as key fundraising locations, shaping where candidates focus their campaign efforts.
- The patterns suggest the need for targeted campaigning in high-contribution states to maximize financial support.

DATA ANALYSIS PART 2: DATA FRAME API

First we obtain a `SQLContext` frm our existing `SparkContext`

In [20]:
```
from pyspark import SQLContext
```

In [21]:
```
sqlc = SQLContext(spark)
```

```
/usr/local/lib/spark/python/pyspark/sql/context.py:113: FutureWarning: Deprecated in
3.0.0. Use SparkSession.builder.getOrCreate() instead.
  warnings.warn(
```

In [22]:
```
sqlc
```

Out[22]:
```
<pyspark.sql.context.SQLContext at 0x736523522840>
```

```
In [23]: # sqlc.read.csv: This method is  used to read a CSV file and create a Spark DataFra
         dataframe_api = sqlc.read.csv("final.csv", header=True, inferSchema=True)
```

```
In [24]: # The method printSchema() is used to display the schema of a Spark DataFrame in a
         dataframe_api.printSchema()
```

```
root
 |-- CMTE_ID: string (nullable = true)
 |-- AMNDT_IND: string (nullable = true)
 |-- RPT_TP: string (nullable = true)
 |-- TRANSACTION_PGI: string (nullable = true)
 |-- IMAGE_NUM: long (nullable = true)
 |-- TRANSACTION_TP: string (nullable = true)
 |-- ENTITY_TP: string (nullable = true)
 |-- NAME: string (nullable = true)
 |-- CITY: string (nullable = true)
 |-- STATE: string (nullable = true)
 |-- ZIP_CODE: string (nullable = true)
 |-- EMPLOYER: string (nullable = true)
 |-- OCCUPATION: string (nullable = true)
 |-- TRANSACTION_DT: integer (nullable = true)
 |-- TRANSACTION_AMT: integer (nullable = true)
 |-- OTHER_ID: string (nullable = true)
 |-- TRAN_ID: string (nullable = true)
 |-- FILE_NUM: integer (nullable = true)
 |-- MEMO_CD: string (nullable = true)
 |-- MEMO_TEXT: string (nullable = true)
 |-- SUB_ID: long (nullable = true)
```

```
In [25]: # describe() : Generates basic statistics for the specified columns and can be used
         dataframe_api.describe('CMTE_ID').show()
```

```
[Stage 7:==================================================>        (5 + 1) / 6]
+-------+---------+
|summary|  CMTE_ID|
+-------+---------+
|  count|  3846794|
|   mean|     NULL|
| stddev|     NULL|
|    min|C00000059|
|    max|C99002396|
+-------+---------+
```

1. The Top 10 contributor’s name, the committee ID the contributor contributed to and the transaction amount for all contributions above $5000 ordering them by the transaction amount in descending order

In [26]:
```python
dataframe_api.filter("TRANSACTION_AMT > 5000") \
    .select("NAME","CMTE_ID","TRANSACTION_AMT") \
    .orderBy("TRANSACTION_AMT", ascending=False) \
    .show(10)
```

```
[Stage 10:================================================>         (5 + 1) / 6]
+--------------------+---------+---------------+
|                NAME|  CMTE_ID|TRANSACTION_AMT|
+--------------------+---------+---------------+
|     MELLON, TIMOTHY|C00825851|       50000000|
|          ONE NATION|C00571703|       18400000|
|SECURING AMERICAN...|C00881805|       15000000|
|FUTURE FORWARD US...|C00669259|       15000000|
|CLEMENT, CHRISTIN...|C00857128|       12000000|
|BLOOMBERG, MICHAE...|C00495028|       10000000|
|     SINGER, PAUL E.|C00504530|       10000000|
|SINGER, PAUL ELLIOTT|C00571703|       10000000|
|BRICK BY BRICK FO...|C00631549|        5000000|
|AMERICAN ACTION N...|C00504530|        5000000|
+--------------------+---------+---------------+
only showing top 10 rows
```

EXPLANATION OF THE ABOVE CODE:

1. dataframe_api.filter("TRANSACTION_AMT > 5000"):

- Filters rows in the DataFrame dataframe_api where the TRANSACTION_AMT column is greater than 5000.

2. select("NAME", "CMTE_ID", "TRANSACTION_AMT"):

- Selects only the specified columns (NAME, CMTE_ID, and TRANSACTION_AMT) from the filtered DataFrame.
- The resulting DataFrame contains only these three columns.

3. orderBy("TRANSACTION_AMT", ascending=False):

- Sorts the filtered and selected DataFrame in descending order (ascending=False) based on the TRANSACTION_AMT column.

4. show(10):

- Displays the top 10 rows of the sorted DataFrame in the output.

2. The Top 10 name of the individual and their total contributions ordering them by the total transaction amount in descending order

In [27]:
```python
#imports the sum function from the pyspark.sql.functions module
from pyspark.sql.functions import sum
```

```
In [28]: dataframe_api.groupBy("NAME") \
             .agg(sum("TRANSACTION_AMT")) \
             .withColumnRenamed("sum(TRANSACTION_AMT)", "TOTAL_CONTRIBUTIONS") \
             .orderBy("TOTAL_CONTRIBUTIONS", ascending=False) \
             .show(10)
```

```
[Stage 13:>                                                         (0 + 2) / 2]
+--------------------+-------------------+
|                NAME|TOTAL_CONTRIBUTIONS|
+--------------------+-------------------+
|     MELLON, TIMOTHY|           55003300|
|          ONE NATION|           18400000|
|FUTURE FORWARD US...|           15190058|
|SECURING AMERICAN...|           15000000|
|          MUSK, ELON|           14950000|
|CLEMENT, CHRISTIN...|           12000564|
|     SINGER, PAUL E.|           10150000|
|BLOOMBERG, MICHAE...|           10014900|
|SINGER, PAUL ELLIOTT|           10000000|
|AMERICAN ACTION N...|            7500000|
+--------------------+-------------------+
only showing top 10 rows
```

EXPLANATION OF THE ABOVE CODE:

1. dataframe_api.groupBy("Name"):

- Groups the DataFrame rows by the NAME column.
- Each unique NAME (individual contributor) becomes a group.

2. agg(sum("TRANSACTION_AMT")):

- Performs an aggregation on each group using the sum function.
- Calculates the total of TRANSACTION_AMT for each contributor.

3. withColumnRenamed("sum(TRANSACTION_AMT)", "TOTAL_CONTRIBUTIONS")

- Renames the column sum(TRANSACTION_AMT) to TOTAL_CONTRIBUTIONS for better readability.

4. orderBy("TOTAL_CONTRIBUTIONS", ascending=False):

- Sorts the DataFrame in descending order of TOTAL_CONTRIBUTIONS.
- Ensures that contributors with the highest total contributions appear first.

4. show(10):

- Displays the top 10 rows of the sorted DataFrame in the output.

Summary and Insights: DATA FRAME API

Findings:

1. Top Contributors by Individual Transactions:

- Timothy Mellon emerges as the leading contributor, with a single donation of $50,000,000.

- One Nation follows with $18,400,000

2. Top Contributors by Total Contributions:

- Timothy Mellon tops again, with a staggering total contribution of over $55,003,300 indicating repeated and significant financial support to political committees.

Patterns and Trends Observed:

1. High Dependency on Wealthy Donors:

- Contributions are heavily skewed toward a few high-profile individuals and organizations, indicating a concentrated donor base.

2. Prominent Individuals and Entities:

- Contributors like Elon Musk and Timothy Mellon highlight the growing involvement of wealthy individuals in political funding.

Reflection on Findings:

- The dominance of wealthy individuals and prominent organizations in the dataset is expected given their historical involvement in campaign funding.
- Large donations are consistent with the fundraising trends observed in high-stakes elections, where major donors play a critical role.
- The presence of well-known figures like Elon Musk among the top contributors highlights the crossover between corporate influence and political campaigns.

Significance in Political Campaigns and Elections:

1. Fundraising Strategies:

- Understanding the contributions landscape allows campaigns to identify key donors and tailor their outreach efforts accordingly.
- Committees relying on large-scale donors must also balance the optics of donor dependency with grassroots support.

2. Electoral Influence:

- The concentration of financial power among a few individuals and entities may shape campaign strategies, resource allocation, and electoral outcomes.
- Transparency in political funding remains crucial for maintaining public trust and accountability.

3. Broader Implications:

- These findings underscore the evolving role of high-profile donors and organizations in shaping the political agenda and influencing policy discussions.
- The data highlights the need for continued monitoring and analysis of campaign contributions to ensure fair and transparent electoral processes.

DATA ANALYSIS PART 3: SQL WITH DATA FRAME

In [29]:
```
# We create a temporary SQL view named "dataframe_sql" from the DataFrame dataframe
# The allows to query the DataFrame using SQL syntax instead of PySpark DataFrame m
dataframe_api.createOrReplaceTempView("dataframe_sql")
```

In [30]:
```
# An SQL query to count the total number of rows in the dataframe_sql
sqlc.sql("SELECT COUNT(*) FROM dataframe_sql").show()
```

```
[Stage 14:=======================================>                   (4 + 2) / 6]
+--------+
|count(1)|
+--------+
| 3846794|
+--------+
```

1. The top 10 individual contributor’s name, the contributor's occupation, and the transaction amount ordering them by the transaction amount in descending order.

In [31]:
```
sqlc.sql("""
     SELECT
         NAME,
         OCCUPATION,
         TRANSACTION_AMT
     FROM
         dataframe_sql
     ORDER BY
         TRANSACTION_AMT DESC
    """).show(10)
```

```
[Stage 17:=================================================>         (5 + 1) / 6]
```

```
+--------------------+-------------+---------------+
|                NAME|   OCCUPATION|TRANSACTION_AMT|
+--------------------+-------------+---------------+
|     MELLON, TIMOTHY|  INVESTMENTS|       50000000|
|          ONE NATION|         NULL|       18400000|
|SECURING AMERICAN...|         NULL|       15000000|
|FUTURE FORWARD US...|         NULL|       15000000|
|CLEMENT, CHRISTIN...|SELF EMPLOYED|       12000000|
|BLOOMBERG, MICHAE...|      FOUNDER|       10000000|
|     SINGER, PAUL E.|    PRESIDENT|       10000000|
|SINGER, PAUL ELLIOTT|    PRESIDENT|       10000000|
|BRICK BY BRICK FO...|         NULL|        5000000|
|AMERICAN ACTION N...|         NULL|        5000000|
+--------------------+-------------+---------------+
only showing top 10 rows
```

EXPLANATION OF THE ABOVE CODE:

1. SELECT NAME, OCCUPATION, TRANSACTION_AMT:

- Filters rows in the DataFrame dataframe_api where the TRANSACTION_AMT column is greater than 5000.

2. SELECT("NAME", "CMTE_ID", "TRANSACTION_AMT"):

- Specifies the columns to be retrieved.

3. FROM dataframe_sql:

- Specifies the temporary SQL view (dataframe_sql) as the data source for the query.

4. ORDER BY TRANSACTION_AMT DESC:

- Sorts the results by the TRANSACTION_AMT column in descending order.

5. show(10):

- Displays the top 10 rows of the sorted DataFrame in the output.

6. sqlc.sql("""..."""):

- Executes the SQL query written within the triple quotes.

2. The top 10 committees based on their total individual contributions on a given date. Providing the committee ID, the transaction date and the transaction amount ordered by the total transaction amount in descending order.

In [32]:
```
sqlc.sql("""
     SELECT
         CMTE_ID,
         TRANSACTION_DT,
        SUM(TRANSACTION_AMT) AS TOTAL_TRANSACTION_AMOUNT
    FROM
        dataframe_sql
    WHERE
        TRANSACTION_DT = 07232024
    GROUP BY
        CMTE_ID, TRANSACTION_DT
    ORDER BY
        TOTAL_TRANSACTION_AMOUNT DESC
""").show(10)
```

```
[Stage 18:=================================================>          (5 + 1) / 6]
+---------+--------------+------------------------+
|  CMTE_ID|TRANSACTION_DT|TOTAL_TRANSACTION_AMOUNT|
+---------+--------------+------------------------+
|C00401224|       7232024|                16219539|
|C00744946|       7232024|                 8930581|
|C00703975|       7232024|                 6353147|
|C00504530|       7232024|                 5025208|
|C00867937|       7232024|                 3665789|
|C00694323|       7232024|                 2657790|
|C00873893|       7232024|                 1208679|
|C00418897|       7232024|                 1000000|
|C00000935|       7232024|                  941803|
|C00010603|       7232024|                  765994|
+---------+--------------+------------------------+
only showing top 10 rows
```

EXPLANATION OF THE ABOVE CODE:

1. SELECT NAME, OCCUPATION, TRANSACTION_AMT:

- Filters rows in the DataFrame dataframe_api where the TRANSACTION_AMT column is greater than 5000.

2. SELECT CMTE_ID, TRANSACTION_DT, SUM(TRANSACTION_AMT) AS TOTAL_TRANSACTION_AMOUNT:

- Specifies the columns to be retrieved.

3. FROM dataframe_sql:

- Specifies the temporary SQL view (dataframe_sql) as the data source for the query.

4. WHERE TRANSACTION_DT = 07232024:

- Filters the data to include only rows where the TRANSACTION_DT (transaction date) equals July 23, 2024.

5. GROUP BY CMTE_ID, TRANSACTION_DT:

- Groups the data by the CMTE_ID (committee ID) and TRANSACTION_DT (transaction date).

- Within each group, the SUM(TRANSACTION_AMT) computes the total contribution amount

6. ORDER BY TRANSACTION_AMT DESC:

- Sorts the results by the TRANSACTION_AMT column in descending order.

7. show(10):

- Displays the top 10 rows of the sorted DataFrame in the output.

8. sqlc.sql("""...""")::

- Executes the SQL query written within the triple quotes.

Summary and Insights: SQL WITH DATA FRAME

Findings:

- The data analysis revealed that Timothy Mellon is a significant contributor, with a transaction amount of $50,000,000.

- Followed by contributions from committees such as One Nation with $18,400,000.

- These substantial contributions highlight the role of high-net-worth individuals and organizations in political funding.

- The SQL queries also identified committees receiving the highest contributions on specific dates, such as July 23, 2024, showing peaks in fundraising activities around critical campaign timelines.

Patterns and Trends Observed:

- Contributions are highly skewed, with a small number of individuals and committees making exceptionally large contributions. This trend reflects the unequal distribution of financial influence in political campaigns.
- Certain committees and contributors consistently appear at the top, indicating their significant and recurring role in funding.

Significance in Political Campaigns and Elections:

- These findings emphasize the pivotal role of financial contributions in shaping election campaigns. Large donors and committees with substantial funds can significantly influence political narratives and outcomes.

FINAL CONCLUSION

This data analysis project on individual contributions to Federal Committees covering presidential, senate and house committees has provided significant insights into the dynamics of political campaign funding during the period from June 20, 2024 to July 23, 2024. Through systematic data collection, processing and analysis using Apache Spark, we effectively uncovered key patterns and trends in contributions showcasing the power and efficiency of big data tools in handling large datasets.

The data processing phase demonstrated the utility of tools like Spark for efficient handling of large, complex datasets through RDDs, DataFrame APIs and SQL queries. The data analysis revealed key contributors and patterns highlighting the role of a few high net-worth individuals and influential committees in shaping the political funding landscape.

This analysis underscores the critical importance of transparency and data-driven decision-making in understanding political financing. By leveraging modern tools like Spark, we demonstrated the potential for analyzing large-scale datasets to derive actionable insights, which can be used to inform policy discussions and improve the integrity of political processes. Overall, this project showcases how big data and analytics can transform our understanding of complex domains such as campaign financing.

```
In [ ]:
```