Implementation of the code:

%pip install pyspark

%pip install findspark

import findspark

findspark.init()

from pyspark import SparkConf, SparkContext

# from stop\_words import get\_stop\_words

from collections import Counter

import re

import string

from pyspark.rdd import RDD

conf = SparkConf().setAppName("Tweets app")

sc = SparkContext(conf=conf)

Tweets\_RDD = sc.textFile("/content/Trump\_Tweet.txt")

#a

count\_char = Tweets\_RDD \

    .flatMap(lambda t: [char. upper()for char in t if re.match(r'[a-zA-Z]', char)]) \

    .filter(lambda x: x) \

    .countByValue()

count\_sorted = sorted(count\_char.items())

for char, count in count\_sorted:

    print(f"'{char}': {count}")

#b

Count\_word = Tweets\_RDD\

    .flatMap(lambda t: re.sub(r"[^a-zA-Z ]", "", t).split(" "))\

    .map(lambda x: x.lower())\

    .filter(lambda x: x != "")\

    .countByValue()

Count\_sorted = sorted(Count\_word.items(), key=lambda x: -x[1])

Count\_sorted

#c

gram2count = Tweets\_RDD\

    .flatMap(lambda t: [" ".join(t.translate(str.maketrans('', '', string.punctuation)).lower().split()[i:i+2]) for i in range(len(t.translate(str.maketrans('', '', string.punctuation)).lower().split())-1)])\

    .filter(lambda x: x!="")\

    .countByValue()

gram2count

#d

%pip install stop\_words

import stop\_words

%pip install pyspark

import pyspark

wordCountsRDD = sc.parallelize(Count\_word)

stop\_words = set(stop\_words.get\_stop\_words('english'))

words\_count\_filtered = {word: count for word, count in Count\_word.items() if word not in stop\_words}

sorted\_collection = sorted(words\_count\_filtered.items(), key=lambda x: x[1], reverse=True)

for word, count in sorted\_collection[:20]:

    print(word, count)

## a. Count the number of occurrences of each alphabetic character

In this task, we utilized PySpark to read the tweet data from a file and performed character-level processing to count the occurrences of each alphabetic character. The code implementation involves the following steps:  
  
1. Data Loading: We loaded the tweet data into a Spark RDD (`Tweets\_RDD`) using the `textFile` function.  
  
2. Character Counting: We applied a flatMap transformation on the RDD to split each tweet into individual characters. Within the flatMap function, we used a lambda expression to iterate over each character and converted it to uppercase for case-insensitivity. Additionally, we used a regular expression (`re.match`) to filter out non-alphabetic characters.  
  
3. Aggregation: The filtered characters were then aggregated using the `countByValue` function to count the occurrences of each character.  
  
4. Sorting: Finally, we sorted the counts to present the results in ascending order.

## b. Count the number of occurrences of each word

For this task, we processed the tweet data to count the occurrences of each word. The code implementation includes the following steps:  
  
1. Data Preprocessing: We applied preprocessing steps to clean the tweet text, including removing punctuation and splitting the text into words.  
  
2. Word Normalization: We normalized the words to lowercase using the `map` function to ensure case-insensitivity.  
  
3. Word Counting: Using a flatMap transformation followed by the `countByValue` function, we counted the occurrences of each word in the tweet data.  
  
4. Sorting: The word counts were sorted in descending order to identify the most frequently occurring words.

## c. Count the number of occurrences of 2-grams

In this task, we generated 2-grams from the tweet data to capture sequences of two adjacent words. The code implementation involves the following steps:  
  
1. 2-gram Generation: We applied string transformations to remove punctuation and split the tweet text into words. We then used list slicing to create pairs of consecutive words, forming 2-grams.  
  
2. Filtering: Non-empty 2-grams were filtered using a lambda function to remove any empty sequences.  
  
3. Counting: We applied the `countByValue` function to count the occurrences of each 2-gram in the tweet data.

## d. Obtain the top 20 most frequently occurred words excluding stop words

For this task, we filtered out common stop words to focus on meaningful content and identified the top 20 most frequently occurring words. The code implementation includes the following steps:  
  
1. Stop Words Removal: We imported the `stop\_words` package to obtain a set of common English stop words.  
  
2. Stop Words Filtering: We filtered out stop words from the word counts using a list comprehension and set intersection operation.  
  
3. Sorting: The filtered word counts were sorted in descending order based on frequency.  
  
4. Top 20 Selection: We selected the top 20 words with the highest frequencies to identify key terms in the tweet data.

E . brief summary of the style of Trump’s tweet based on the frequently used words

Based on the frequently used words in the dataset of Donald Trump's tweets, the style of his tweeting is assertive and direct, often focusing on personal opinions, declarations, and responses to current events. The words "great," "thanks," and "will" suggest a promotional and sometimes appreciative tone, whereas the frequent use of "president," "america," and "make" (likely part of "make America great again") reflect his political messaging.

Additionally, the presence of words like "new," "like," and "people" indicates a focus on engaging or addressing the public directly. The names mentioned (such as "obama" and "donald") likely pertain to discussions about political figures or himself, emphasizing a personal or confrontational element in his communication style.