Introduzione:

La documentazione ricalcherà quelli che sono stati i due lavori presentati come proposta di progetto, e sarà articolata nel seguente modo:

- **Capitolo I**: verrà ripresa la parte inerente il tool di BI e saranno spiegati nel dettaglio i dati utilizzati e messi a disposizione dalla protezione civile, oltre alla metodologia adottata da noi per averli disponibili sempre alla versione aggiornata;
- **Capitolo II**: si mostreranno gli aspetti di analisi dati con commenti sul codice relativo, ripercorrendo nel dettaglio gli steps che hanno portato alla costruzione del motore di sentiment analysis, cercando di costruire un parallelismo tra l'avanzamento epidemiologico dell'epidemia e l'avanzamento della stessa all'interno di un contesto mediatico, mostrando, laddove sia possibile, in che direzione vira il sentimento generale dei tweet da parte degli utenti del social media Twitter, nei confronti della stessa.

Insieme alla seguente documentazione saranno forniti i seguenti file:

- Report sulla BI e l'introduzione al lavoro condotto con il tool MicroStrategy;
- Presentazione in formato PowerPoint sulla parte di BI;
- Un file intitolato 'dossier_covid_federicoII', in formato '.mstr', contenente il dossier di BI;
- Un file intitolato 'HW-SW-BIG-DATA', in formato .html riferente al notebook della piattaforma Databricks per la parte concernente la sentiment analysis;

COVID data Business Intelligence

Come riportato nella documentazione relativa alla Business Intelligence, la prima parte del lavoro è stata quella in cui si è descritto il tool MicroStrategy Desktop.

I dati utilizzati per l'analisi e la produzione delle dashboard sono stati quelli messi a disposizione dalla protezione civile italiana in un repository di GitHub accessibile tramite il seguente link. I dati messi a disposizione sono aggiornati ogni 24 ore dall'inizio dell'epidemia, ciò che è stato fatto per automatizzare il processo di estrazione ed aggiornamento degli stessi ai fini dell'analisi, è stato procedere alla creazione di uno script python, nel notebook Google Colaboratory, tramite il quale si scaricavano in locale i dataset in maniera aggiornata ogni volta che si lanciava il notebook.

La creazione dello stesso, riguardava delle analisi condotte ai fini statistici, ma il suo impiego si è reso necessario per l'accesso veloce agli ultimi dati aggiornati ogni volta, come appena detto (link al notebook Colab).

Una volta ottenuti i dati in locale, si è proceduto a:

- Creare un nuovo Dossier in MicroStrategy (che rappresenta l'ambiente di lavoro all'interno del tool), che per il progetto è intitolato `**dossier_covid_federicoII**`;

- Definire l'origine dati, come mostrato nelle figure 1-2 riportate di seguito:

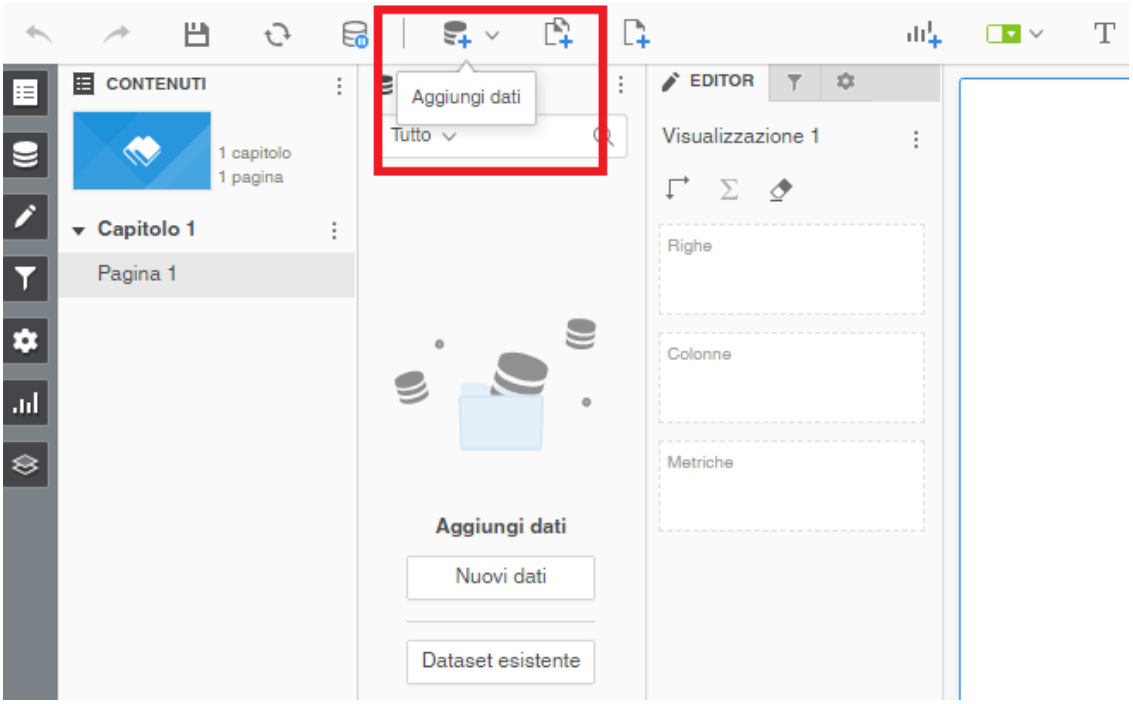


Figura 1 Aggiunta dati al dossier

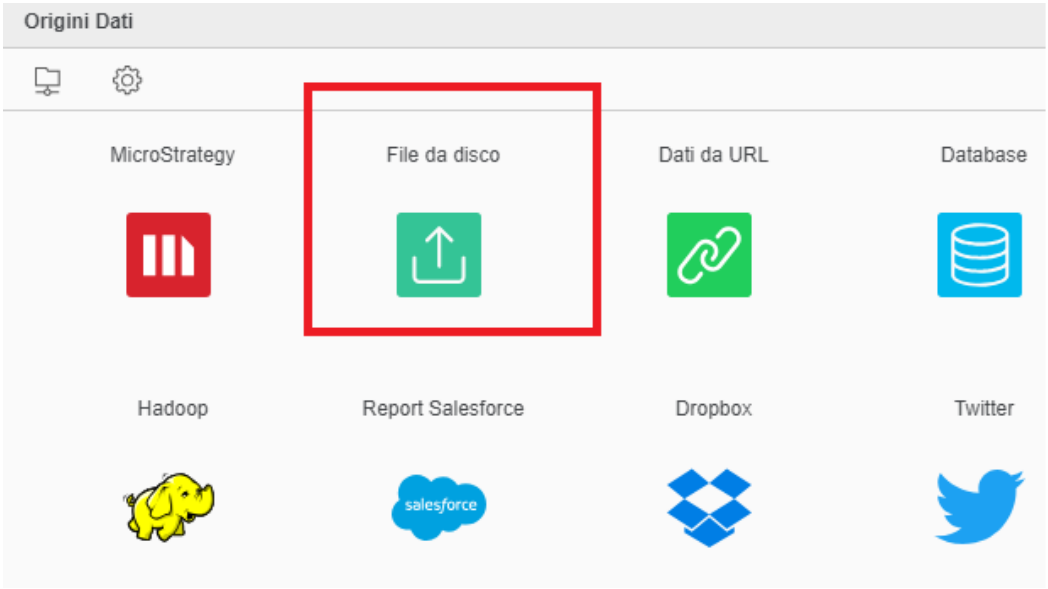


Figura 2 Selezione origine dati

I dati forniti dalla protezione civile, consistono di una time series rispettivamente:

- A livello nazionale;
- A livello regionale;
- A livello provinciale.

La time series parte dal giorno 30 gennaio (momento in cui sono stati segnalati i due turisti cinesi).

Guardando nel dettaglio ai 3 dataset, essi sono strutturati nel seguente modo:

- Nazionale (dpc-covid19-ita-andamento-nazionale.csv) ha 144 righe (equivalenti ai 114 giorni di permanenza del virus in Italia) ed ha le seguenti colonne:
    1. Data;
    2. Stato;
    3. Ricoverati con sintomi;
    4. Terapia intensiva;
    5. Totale ospedalizzati;
    6. Isolamento domiciliare;
    7. Totale positivi;
    8. Variazione totali positivi;
    9. Nuovi positivi;
    10. Dimessi guariti;
    11. Deceduti;
    12. Totale casi;
    13. Tamponi;
    14. Casi testati.
- Regionale (dpc-covid19-ita-regioni.csv):
    1. Data;
    2. Stato;
    3. Codice regione;
    4. Denominazione regione;
    5. Latitudine;
    6. Longitudine;
    7. Ricoverati con sintomi;
    8. Terapia intensiva;
    9. Totale ospedalizzati;
    10. Isolamento domiciliare;

11. Totale positivi;
12. Variazione totale positivi;
13. Nuovi positivi;
14. Dimessi guariti;
15. Deceduti;
16. Totale casi;
17. Tamponi;
18. Casi testati.

- Provinciale (dpc-covid19-ita-province.csv):
    1. Data;
    2. Stato;
    3. Codice regione;
    4. Denominazione regione;
    5. Codice provincia;
    6. Denominazione provincia;
    7. Sigla provincia;
    8. Latitudine;
    9. Longitudine;
    10. Totale casi.

Come riportato nel documento word riguardante la BI, l'avere a disposizione questi dataset ci ha permesso di definire tre capitoli, per avere una panoramica delle statistiche circa l'andamento Nazionale, Regionale e Provinciale per un totale di 11 dashboard, raffiguranti utili insights sull'avanzamento dell'epidemia del tempo oltre ad avere un quadro generale in termini di conseguenze, purtroppo dolorose, in termini di contagi e morti. Tra i dati a disposizione erano anche note le coordinate geografiche delle provincie colpite espresse come latitudine e longitudine le quali ci hanno permesso di costruire una mappa dettagliata dell'intero suolo Italiano che è stato interessato da casi di coronavirus (a titolo di esempio si faccia riferimento alla figura di seguito).

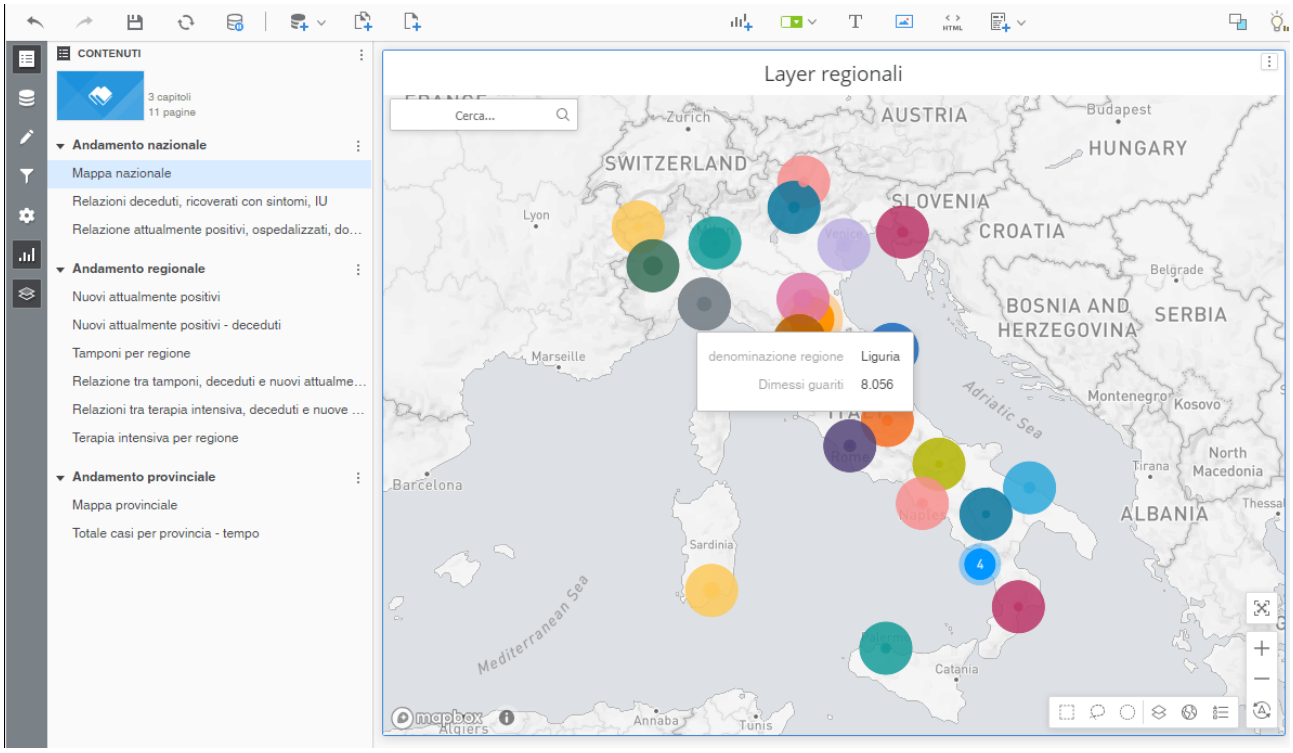


Figura 3 Visione delle regioni italiane interessate da contagi da coronavirus

Per una visione più dettagliata di tutto l'organico del lavoro, si rimanda al dossier tramite il quale visualizzare le dashboard, accessibile dal file .mstr.

Analisi dello stream con databricks

Questa seconda parte è volta alla costruzione di un motore di sentiment analysis, con l'obiettivo di *proof of concepts*, al fine di illustrare la fattibilità e le tecniche necessarie per incrociare ed evidenziare il dilagare dell'epidemia da coronavirus tra le persone contestualmente al suo impatto all'interno del contenuto informativo contenuto nei tweet dell'omonimo social media Twitter.

In aggiunta a quanto detto, vi è da dire, che tale parte del progetto vuole anche essere volto alla costruzione di un tipico ambiente di lavoro in ambito "Big data", servendosi delle architetture HW e SW alla base di un processo di lavoro nel suddetto ambiente.

Con quanto segue, intendiamo trasmettere e mostrare il processo logico-operativo che ha portato alla costruzione della seconda parte del progetto.

Il punto di partenza è stato quello di voler procedere preformando un *real time streaming process,* dei tweet.

TWITTER API KEYS

Per ottenere i tweet si è provveduto ad ottenere le *API keys* di Twitter, iscrivendosi alla piattaforma Twitter Developer, le API keys rappresentano un set di 4 codici, necessari per essere accreditati da Twitter ed accedere al DB dei tweet. Le immagini di seguito chiarificano quanto detto:

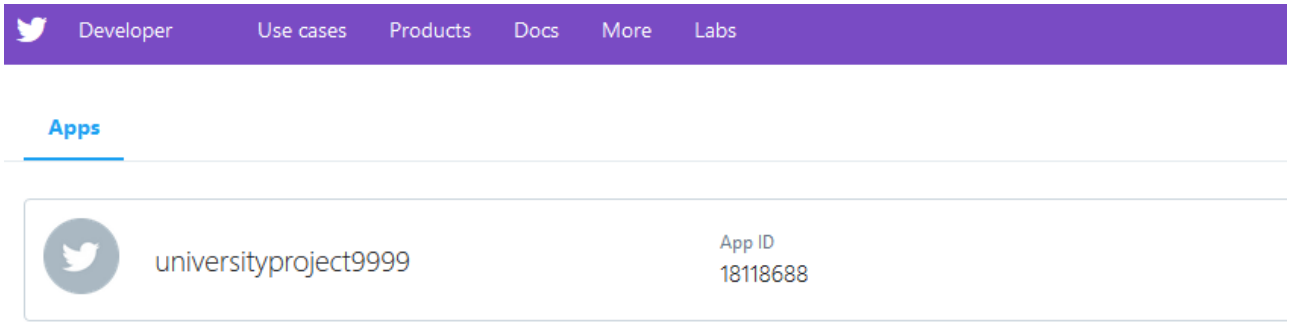


Figura 4 Fase I dell'accreditamento

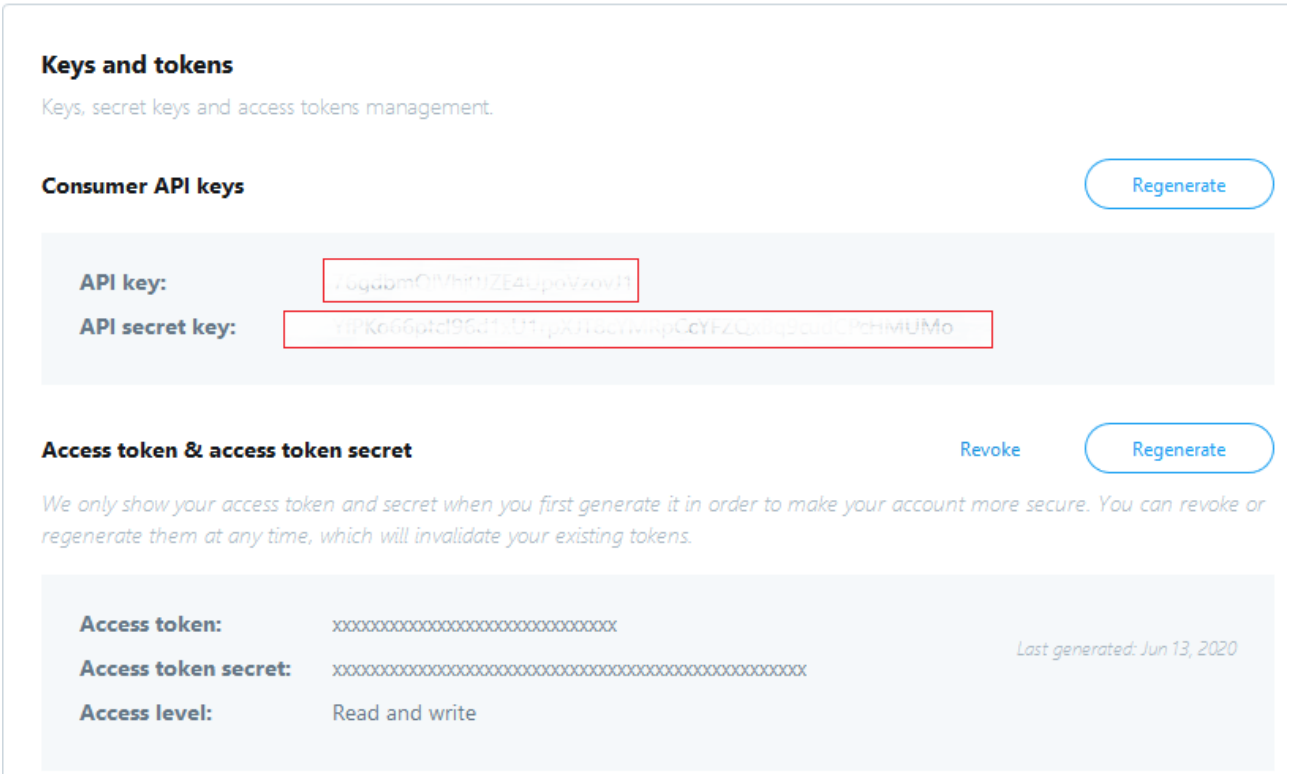


Figura 5 Esempio blocco Twitter API Keys

CREAZIONE ISTANZA EC2 IN AWS

Ci siamo avvalsi di Amazon Web Service (AWS) per la creazione di una istanza EC2 (Amazon Elastic Compute Cloud), che offre capacità di elaborazione variabile grazie alla sua scalabilità, infatti una volta configurata ed istanziata la VM abbiamo fatto uso del servizio *Kinesis Data Stream* che dà la possibilità di maneggiare flussi di dati in real time, catturando in modo continuo gigabyte di dati al secondo. I dati in questione nel nostro caso, sono il flusso di tweet pubblicati dagli utenti.

CONNESSIONE ALL'ISTANZA EC2 E SCRAPING DEI TWEET

Abbiamo effettuato la connessione in remoto all'istanza EC2, tramite l'emulatore di di terminale PuTTY. Una volta effettuata la connessione in remoto alla macchina virtuale, abbiamo lanciato il seguente script python, definito come twitter_kinesis_data.py:

```
class TweetStreamListener(StreamListener):
    # on success
    def on_data(self, data):
        # decode json
        tweet = json.loads(data)
        # print(tweet)
        if "text" in tweet.keys():
            payload = {'id': str(tweet['id']),
                                'tweet':
                                 str(tweet['text'].encode('utf8',
                                 'replace')),
                                'ts': str(tweet['created_at']),
            },
            print(payload)
            try:
                put_response = kinesis_client.put_record(
                                StreamName=stream_name,
                                Data=json.dumps(payload),

                PartitionKey=str(tweet['user']['screen_name']))
            except (AttributeError, Exception) as e:
                print (e)
                pass
        return True
    # on failure
    def on_error(self, status):
        print(status)
stream_name = ''  # fill the name of Kinesis data stream you created
```

La classe `TweetStreamListener` eredita dalla classe `StreamListener` presente in tweepy. Questa eredità ci permette di inizializzare `StreamListener` ed, al contempo, di andare ad ottenere, tramite il metodo `on_data`, il cuore pulsante del nostro del nostro streamer, un payload costituito dall' id del tweet, il tweet in sé ed, infine, il timestamp di quando lo stesso è stato creato.

Al fine di lanciare lo script `twitter_kinesis_data.py`, abbiamo bisogno di inserire, nello script, elementi come:

- `consumer_key;`
- `consumer_secret;`
- `access_token;`
- `access_token_secret;`

Inoltre, abbiamo bisogno di inserire:

- Il nome dello stream;
- Il nome della regione nella quale è situato la nostra istanza EC2;
- L'id della chiave d'accesso di EC2;
- La chiave segreta di accesso dell'istanza di EC2.

Questo processo, ci ha poi permesso di:

- Creare e lanciare la kinesis pipeline;
- Fare il fetching dei tweet filtrati per hashtag e regione di provenienza.

I dati raccolti vengono poi mandati come flusso a kinesis.

CREAZIONE NOTEBOOK IN DATABRICKS

Abbiamo provveduto a:

- Definire e creare la SparkSession:

```
from pyspark.sql import SparkSession
  from pyspark.sql.functions import explode
  from pyspark.sql.functions import split

  spark = SparkSession.builder\
                        .master("local")\
                        .appName("Structured Streaming")\
                        .getOrCreate()
```

- Definire la connessione tra Kinesis e SparkStreaming, unitamente allo schema dei dati in arrivo:

```
from pyspark.sql.types import StructType, StringType
pythonSchema = StructType() \
               .add("id", StringType(), True) \
               .add("tweet", StringType(), True) \
               .add("ts", StringType(), True

awsAccessKeyId = "*****************" # valore access key
awsSecretKey = "********************"   # valore secret key
kinesisStreamName = "***********"  # nome kinesis stream
kinesisRegion = "***********"
kinesisDF = spark \
               .readStream \
               .format("kinesis") \
               .option("streamName", kinesisStreamName)\
               .option("region", kinesisRegion) \
               .option("initialPosition", "LATEST") \
               .option("format", "json") \
               .option("awsAccessKey", awsAccessKeyId)\
               .option("awsSecretKey", awsSecretKey) \
               .option("inferSchema", "true") \
               .load()
```

AWS Kinesis offre anche delle metriche di controllo e monitoraggio dei dati in ingresso, visualizzabili con il seguente codice:

```
df = kinesisDF \
  .writeStream \
  .format("memory") \
  .outputMode("append") \
  .queryName("tweets")  \
  .start()
```

Per chiarezza, si riporta di seguito un'immagine di riferimento all'output del codice:

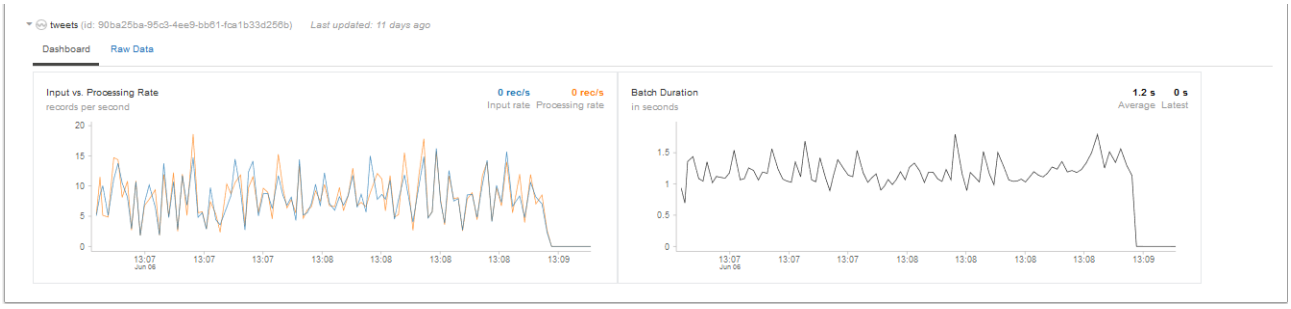


Figura 6 Metriche offerte da Kinesis per il controllo dell'injection dei tweet

I dati in ingresso tramite lo snippet di codice di sopra, vengono strutturati in Spark con la logica degli RDD

VISUALIZZAZIONE DEI DATI

Impostato tutto il set-up operativo, comprendente tutta l'architettura SW, abbiamo potuto lanciare la prima query SparkSQL per visualizzare dati:

```
tweets = spark.sql("select cast(data as string) from tweets")
```

I tweet, sono organizzati in formato JSON, al fine di averli in forma di DataFrame, abbiamo usato una User Defined Function per fare il parsing dei tweet e renderli compatibili con il formato Spark definito nella variabile pythonSchema, andando a creare uno Spark Structured Streaming Data Frame:

```
from pyspark.sql.functions import col
import json
from pyspark.sql.functions import UserDefinedFunction
def parse_tweet(text):
    data = json.loads(text)
    id = data[0]['id']
    ts = data[0]['ts']
    tweet = data[0]['tweet']
    return (id, ts, tweet)

# Define your function
getID = UserDefinedFunction(lambda x: parse_tweet(x)[0], StringType())
getTs = UserDefinedFunction(lambda x: parse_tweet(x)[1], StringType())
getTweet = UserDefinedFunction(lambda x: parse_tweet(x)[2], StringType())
# Apply the UDF using withColumn
tweets = (tweets.withColumn('id', getID(col("data")))
                .withColumn('ts', getTs(col("data")))
                .withColumn('tweet', getTweet(col("data")))
         )
```

Successivamete, andiamo a creare uno Spark Structured Streaming Data Frame che contenga un inizio di analisi testuale, abbiamo usato la libreria textblob, andando a definire la UDF `get_sentiment`.

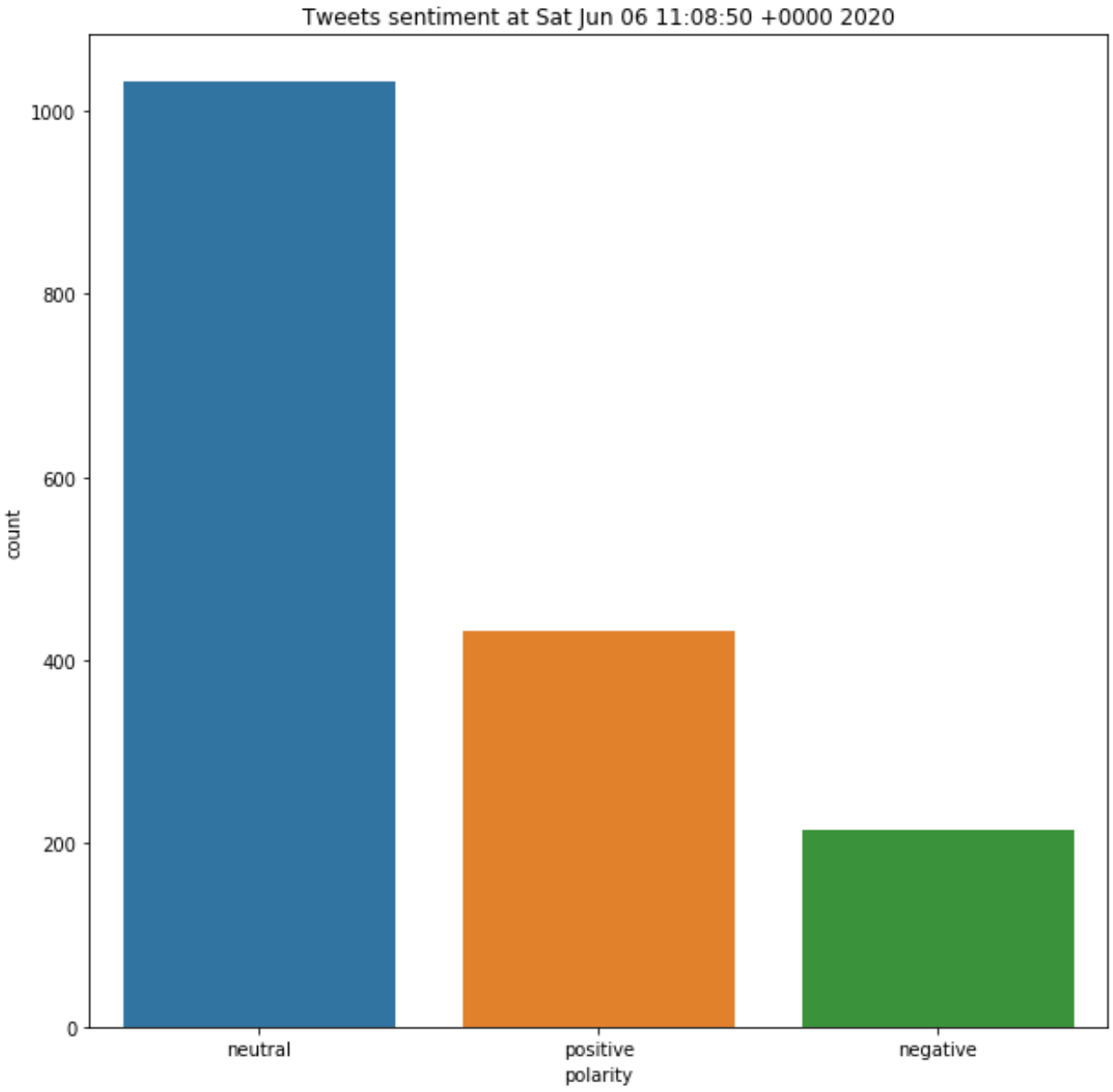
```
import textblob
def get_sentiment(text):
    from textblob import TextBlob
    tweet = TextBlob(text)
    if tweet.sentiment.polarity < 0:
      sentiment = "negative"
    elif tweet.sentiment.polarity == 0:
        sentiment = "neutral"
    else:
        sentiment = "positive"
    return sentiment
```

Questa funzione accetta in input il testo. Per velocizzare questa fase, abbiamo convertito lo Spark Data Frame in pandas Data Frame, utilizzando il metodo apply per mappare `get_sentiment` ad ogni riga del campo tweet del Data Frame.

L'output è definito come segue:

	data	id	ts	tweet	polarity
0	[{"id": "1269223863780589568", "tweet": "b'RT ...	1269223863780589568	Sat Jun 06 11:05:24 +0000 2020	b'RT @WHO: Media briefing on #COVID19 with @Dr...	neutral
1	[{"id": "1269223864984383488", "tweet": "b'Cap...	1269223864984383488	Sat Jun 06 11:05:25 +0000 2020	b'Cappuccino e cornetto? Cappuccino e brioche?...	neutral
2	[{"id": "1269223865630236679", "tweet": "b'RT ...	1269223865630236679	Sat Jun 06 11:05:25 +0000 2020	b'RT @WHO: WHO updated guidance on the use of ...	positive
3	[{"id": "1269223867274399744", "tweet": "b'RT ...	1269223867274399744	Sat Jun 06 11:05:25 +0000 2020	b'RT @BerndPulverer: Calling on @TheLancet &am...	neutral
4	[{"id": "1269223867484119041", "tweet": "b'RT ...	1269223867484119041	Sat Jun 06 11:05:25 +0000 2020	b'RT @BeachMilk: Trump mentioned U.V. light \x...	positive

Per quanto riguarda, invece, una visualizzazione dei dati relativi ad un primo sentiment a giugno 2020, abbiamo usato il countplot di seaborn, andando a generare il seguente grafico:



Com'è possibile vedere, al 6 giugno il sentiment prevalente è quello di neutralità. Come sarà evidente anche nella batch analysis, tuttavia, abbiamo che tale elemento è biased a causa di una non adeguata pulizia dei dati, essendo che qui abbiamo voluto semplicemente enfatizzare le potenzialità dell'analisi stream di twitter.

ANALISI BATCH DEI DATI

In questa sezione andiamo a creare, invece, il processo che preleva i dati storici da twitter, li parsa come pandas Data Frame e poi li analizza per mesi. Al fine di fare ciò, abbiamo creato uno script usando TwitterAPI che inserisce delle date custom e prende i dati solo fino a quella data. Per fare ciò, abbiamo dovuto definire un'app sul portale twitter dev, per poi usare la stessa per prelevare i dovuti elementi.

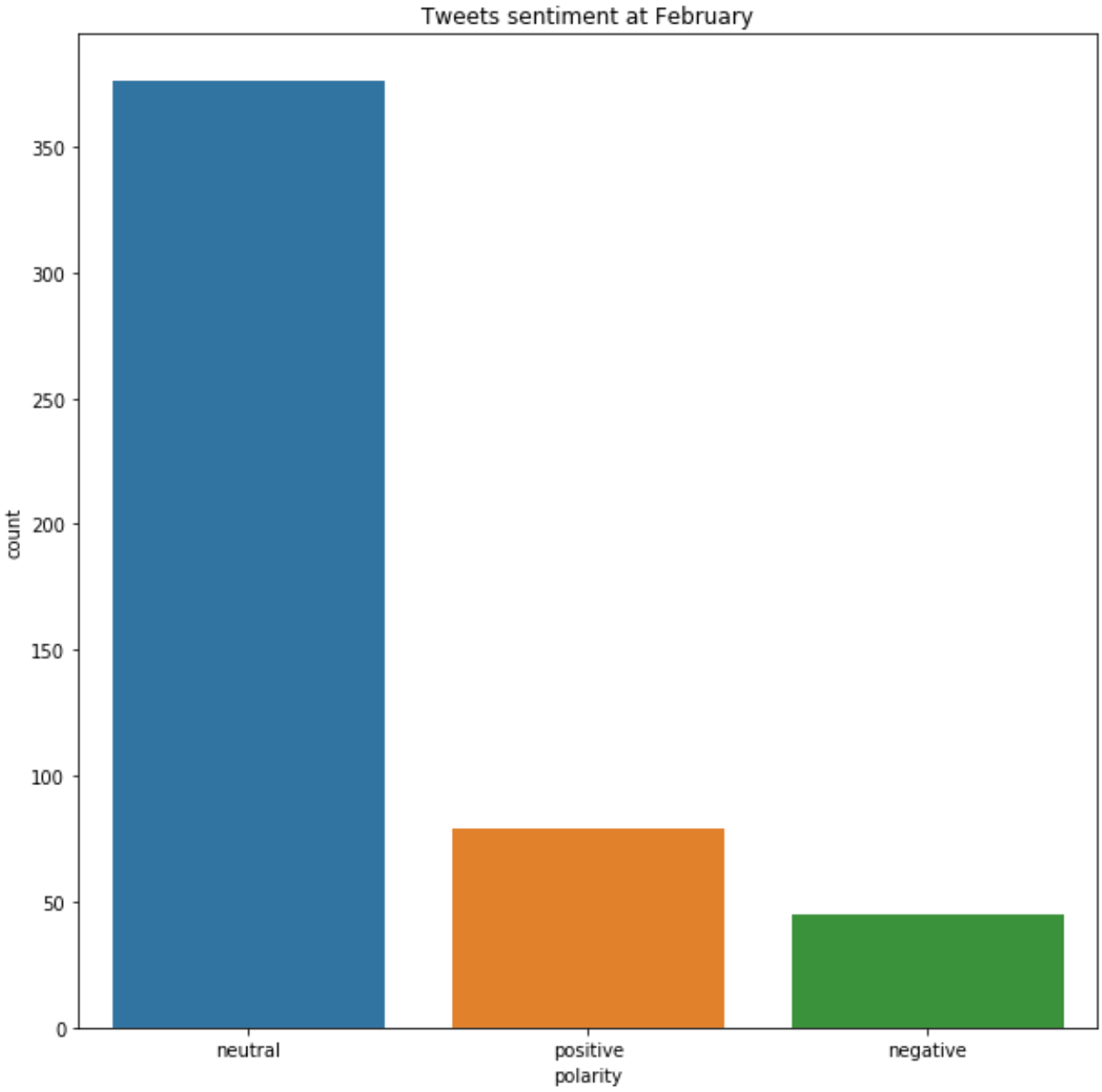
Questo processo si muove, come visibile sotto, mediante il metodo request. Questi conterrà nell'header elementi come query, toDate, ecc. Noi abbiamo usato il toDate come elemento per andare a filtrare i primi 100 tweet per ogni data scelta.

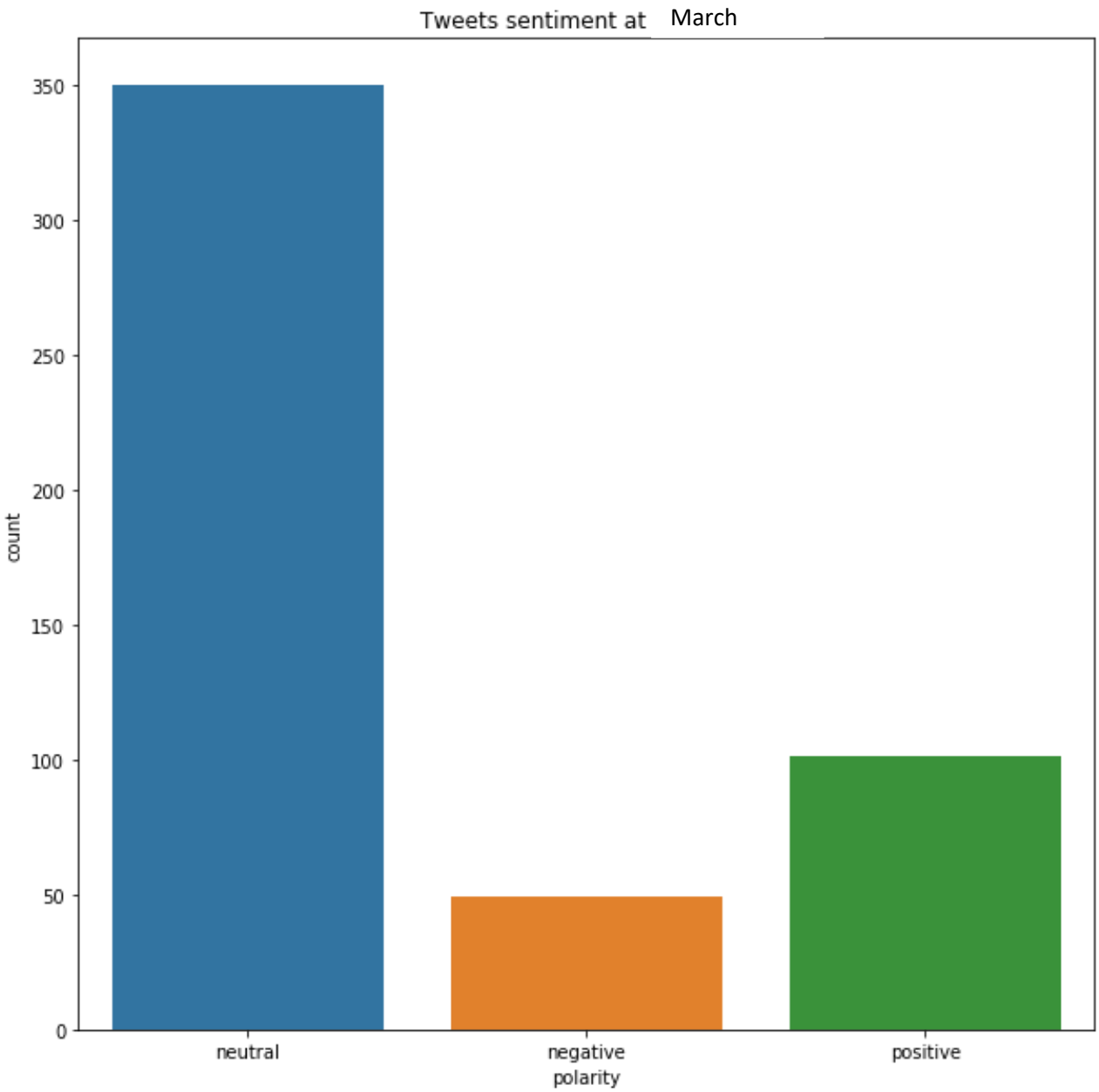
```
SEARCH_TERM = ['#coronavirus']

PRODUCT = 'fullarchive'

LABEL = 'DMBS2'


api = TwitterAPI(consumer_key,

             consumer_secret,

             access_token,

             access_token_secret)

hist_tweets=[]

for i in
["202002220000","202002230000","202002240000","202002250000","20200226000
0","202003220000","202003230000","202003240000","202003250000","202003260
000","202004220000","202004230000","202004240000","202004250000","2020042
60000"]:

  r = api.request('tweets/search/%s/:%s' % (PRODUCT, LABEL),

              {'query':SEARCH_TERM,

              'toDate':i,

              }

              )

  if r.status_code != 200:

    raise Exception("error on API %s"%(r.status_code))

  for item in r:

payload={'id':str(item['id']),'tweet':str(item['text']),'created_at':str(
item['created_at'])}

    if 'next' not in json:

        break

    hist_tweets.append(payload)

    next = json['next']
```

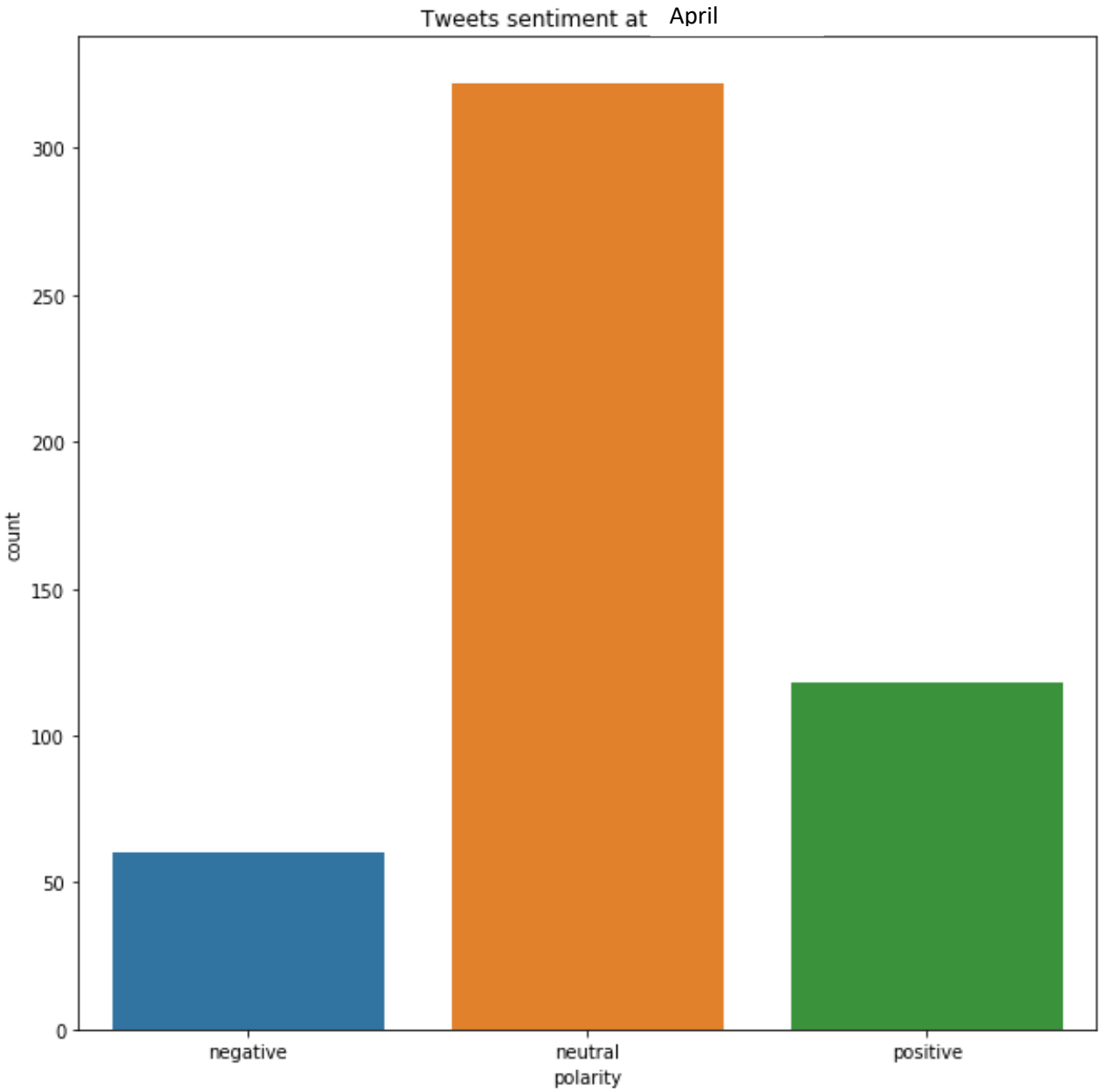
Da hist_tweets, andiamo a generare un pandas DataFrame, il cui head è il seguente:

created_at	id	tweet	polarity
Wed Mar 25 23:59:54 +0000 2020	1242964458361716737	Está partiendo desde Miami un avión de @Helidosa con destino a China que traerá l a ayuda anunciada para combatir el… https://t.co/FOYoEI7nJ6	neutral
Wed Mar 25 23:59:54 +0000 2020	1242964457854287872	RT @fatourgente: Governo dos Estados Unidos orienta que seus cidadãos deixem o Brasil imediatamente #coronavírus https://t.co/thYkPHWHsn	neutral
Wed Mar 25 23:59:54 +0000 2020	1242964457787207681	RT @SenTedCruz: The Chinese Communist Party did everything it could to keep the origin & spread of #coronavirus a secret. Now that thousand…	negative
Wed Mar 25 23:59:54 +0000 2020	1242964457611018241	#Creatividad en caricaturas. [☎] Esta es la forma más divertida para hacerle frente a la contingencia mundial del… https://t.co/8t42I1GOgn	neutral

Infine, abbiamo subsettato i dati per Febbraio, Marzo ed Aprile, generando i grafici qui sotto:




Tweets sentiment at March
350
300
250
200
150
100
50
0
count
neutral
negative
positive
polarity



Com'è possibile constatare, al di là di un'inversione inspiegata dei colori, è possibile notare che il sistema riesce a:

- Immagazzinare i dati;
- Renderli in Spark o pandas per ulteriori analisi;
- Effettua un'analisi, seppure molto sommaria, del sentiment.