In [ ]:
```
# Import needed libraries

import findspark
findspark.init('/usr/hdp/2.6.5.0-292/spark2')

# Create a Spark Context which will be used for distributed data

import pyspark
sc = pyspark.SparkContext(appName="Twitter Topic Sentiment")

import string

import re as re

import nltk

import time

from pyspark.sql import SQLContext

from pyspark.sql.types import *

from pyspark.sql.functions import monotonically_increasing_id

from pyspark.mllib.util import MLUtils

from pyspark.ml.feature import RegexTokenizer, Tokenizer, StopWo

from pyspark.mllib.clustering import LDA, LDAModel

nltk.download('stopwords')

from nltk.corpus import stopwords

from pyspark.mllib.linalg import Vector as oldVector, Vectors as

from pyspark.ml.linalg import Vector as newVector, Vectors as ne

from pyspark.ml.feature import IDF

import numpy as np

import matplotlib.pyplot as plt

import pyspark.sql.functions as func
```

In [ ]:
```
# Create an SQL Context which will be used for sql like distribu

# As I get more familiar with what technology to use where I wil

# pyspark dataframes, and pandas dataframes

sqlContext = SQLContext(sc)
```

In [ ]:
```
# Hadoop is the filesystem being used. This is a three node virt

# Read in data from Hadoop

ITData = sc.textFile("hdfs:///user/vagrant/practicum/input")
```

In [ ]:
```
# Output sample of data

ITData.take(5)
```

In [ ]:
```
# Count number of records loaded to pyspark RDD

ITData.count()
```

In [ ]:
```
# By default, data is partitioned based on the data size

# Check the number of partitions created

ITData.getNumPartitions()
```

In [ ]:
```
# Twitter data was collected and batched in files with each file

# Extract the first file header from the dataset and display

# This will be used later to remove all headers from the dataset

header = ITData.first()
header
```

In [ ]:
```
# Filter all of the headers from the data set

# Count the number of records remaining in the data set

# If 10 files were read from Hadoop, this count should be 10 les

ITData_NoHeader = ITData.filter(lambda row : row != header)
ITData_NoHeader.count()
```

In [ ]:

```
# We now have an RDD with not header information

# In preparation for creating a dataframe from the RDD, create a

schema = StructType([
    StructField('timetext', StringType(), nullable=True),
    StructField('tweet_id', StringType(), nullable=True),
    StructField('tweet_source', StringType(), nullable=True),
    StructField('tweet_truncated', StringType(), nullable=True),
    StructField('tweet_text', StringType(), nullable=True),
    StructField('tweet_user_screen_name', StringType(), nullable
    StructField('tweet_user_id', StringType(), nullable=True),
    StructField('tweet_user_location', StringType(), nullable=Tr
    StructField('tweet_user_description', StringType(), nullable
    StructField('tweet_user_followers_count', StringType(), null
    StructField('tweet_user_statuses_count', StringType(), nulla
    StructField('tweet_user_time_zone', StringType(), nullable=T
    StructField('tweet_user_geo_enabled', StringType(), nullable
    StructField('tweet_user_lang', StringType(), nullable=True),
    StructField('tweet_coordinates_coordinates', StringType(), n
    StructField('tweet_place_country', StringType(), nullable=Tr
    StructField('tweet_place_country_code', StringType(), nullab
    StructField('tweet_place_full_name', StringType(), nullable=
    StructField('tweet_place_name', StringType(), nullable=True)
    StructField('tweet_place_type', StringType(), nullable=True)
])

# Create a dataframe from the RDD with schema

ITData_df = sqlContext.createDataFrame(ITData_NoHeader.map(lambd

ITData_df.printSchema()
```

In [ ]:

```
# First convert dataframe to rdd

# Use map lambda to select the tweet_text column and filter out

tweet = ITData_df.rdd.map(lambda x: x['tweet_text']).filter(lamb
```

In [ ]:

```
# Retrieve stop words. Note we may need to add to the stop words

StopWords = stopwords.words("english")
```

In [ ]:

```
# Further clean tweets, split them out into individual words, an

tokens = tweet.map(lambda document: document.strip().lower()) \
              .map(lambda document: re.split(" ", document)) \
              .map(lambda word: [x for x in word if x.isalpha()]
              .map(lambda word: [x for x in word if len(x) > 3])
              .map(lambda word: [x for x in word if x not in Sto
              .zipWithIndex()
```

In [ ]:
```
# tokens is an RDD, display the first 5 records

tokens.take(5)
```

In [ ]:
```
# Create a new dataframe from the above RDD, adding column names

tweet_df = sqlContext.createDataFrame(tokens, ["tweet_words", 'i
```

In [ ]:
```
# Display the first 5 records of the dataframe

tweet_df.show(5)
```

In [ ]:
```
# Prepare for Topic Modeling

print(time.strftime('%m%d%Y %H:%M:%S'))
cv = CountVectorizer(inputCol="tweet_words", outputCol="raw_feat
cvmodel = cv.fit(tweet_df)
print(time.strftime('%m%d%Y %H:%M:%S'))
```

In [ ]:
```
print(time.strftime('%m%d%Y %H:%M:%S'))
result_cv = cvmodel.transform(tweet_df)
print(time.strftime('%m%d%Y %H:%M:%S'))
```

In [ ]:
```
result_cv.show(1)
```

In [ ]:
```
rs = result_cv.rdd.map(lambda (x, y, z): (x, y, oldVectors.fromM
```

In [ ]:
```
rs_df = rs.toDF(['tweet_words', 'index', 'raw_features'])
```

In [ ]:
```
rs.take(1)
```

In [ ]:
```
rs_df.show(1)
```

In [ ]:
```
print(time.strftime('%m%d%Y %H:%M:%S'))
idf = IDF(inputCol="raw_features", outputCol="features")
idfModel = idf.fit(result_cv)
result_tfidf = idfModel.transform(result_cv)
print(time.strftime('%m%d%Y %H:%M:%S'))
```

In [ ]:
```
# Run the LDA Topic Modeler

# Note the time before and after is printed in order to find out

print(time.strftime('%m%d%Y %H:%M:%S'))
num_topics = 10
max_iterations = 20
lda_model = LDA.train(rs_df['index', 'raw_features'].rdd.map(lis
print(time.strftime('%m%d%Y %H:%M:%S'))
```

In [ ]:
```
vocabArray = cvmodel.vocabulary
```

In [ ]:
```
# Set the top number of topics to write to spark

wordNumbers = 20
topicIndices = sc.parallelize(lda_model.describeTopics(maxTermsP
```

In [ ]:
```
def topic_render(topic):
    terms = topic[0]
    result = []
    for i in range(wordNumbers):
        term = vocabArray[terms[i]]
        result.append(term)
    return result
```

In [ ]:
```
print(time.strftime('%m%d%Y %H:%M:%S'))
topics_final = topicIndices.map(lambda topic:
                               topic_render(topic)).collect()
print(time.strftime('%m%d%Y %H:%M:%S'))
```

In [ ]:
```
# Display topics

for topic in range(len(topics_final)):
    print("Topic" + str(topic) + ":")
    for term in topics_final[topic]:
        print(term)
    print('\n')
```

In [ ]:
```
# The above above relates topics to the terms I searched in Twit

# For sentiment analysis, I would like to rate the actual search

# For this I will build a python array with those search terms

search_terms = ["machine_learning", "computer_programmer", "data
                "data_scientist", "systems_engineer", "data_anal
                "web_programmer", "automation_engineer", "data_p
                "software_engineer", "software_developer", "info
                "business_intelligence", "enterprise_architect",
                "information_technology", "data", "java", "iot",
                "etl", "devops", "cloud", "developer", "programm

search_terms
```

In [ ]:
```
# Python function to search for topics within a tweet

# Function will return the topic and the related tweet or NA is

def SearchTopics(topics, tweet_text):
    for term in topics:
        result = tweet_text.find(term)
        if result > -1:
            return term, tweet_text
    return 'NA', tweet_text
```

In [ ]:
```
# While removing stopwords helps obtain valid topics it will not

# With topics in hand, topics_final, we will use tweets where st

tweet.take(5)
```

In [ ]:
```
# Search each tweet for topics returning only tweets that match

# SearchTopics will return both the topic and the related tweet

# Sentiment will be done on these tweets

topic_tweet = tweet.map(lambda x: SearchTopics(search_terms, x))
```

In [ ]:
```
# Display 5 topic tweet combinations

topic_tweet.take(10)
```

In [ ]:
```
# Setup sentiment analysis

import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')
```

In [ ]:
```
# Python function to print the sentiment scores

# This function will have topic and related tweet as in put

# This function will perform sentiment analysis and output topic

# Also note this function will only return the compound portion

# Revert sigpipe to default behavior

def print_sentiment_scores(topic, sentence):
    snt = SentimentIntensityAnalyzer().polarity_scores(sentence)
    print("{:-<40} {}".format(sentence, str(snt)))
    print(str(snt))
    return(topic, sentence, str(snt.get('compound')))
```

In [ ]:
```
# Retrieve sentiment for each topic, tweet

topic_tweet_sentiment = topic_tweet.map(lambda x: print_sentimen
```

In [ ]:
```
# Display sentiment

topic_tweet_sentiment.take(10)
```

In [ ]:
```
# Assign the topic and sentiment only

topic_tweet_sentiment_pair = topic_tweet_sentiment.map(lambda x:
```

```
In [ ]: # Display topic, sentiment combination

        topic_tweet_sentiment_pair.take(10)
```

```
In [ ]: # Convert to dataframe naming columns

        topic_tweet_sentiment_pair_df = topic_tweet_sentiment_pair.toDF(
```

```
In [ ]: # Display dataframe

        topic_tweet_sentiment_pair_df.show(5)
```

```
In [ ]: # Count sentiment records

        topic_tweet_sentiment_pair_df.count()
```

```
In [ ]: # Create panda dataframe based on topic, sentiment dataframe

        # This dataframe will enable us to plot highs, lows, and means

        pdf1 = topic_tweet_sentiment_pair_df.toPandas()
```

```
In [ ]: # Check new dataframe types

        pdf1.dtypes
```

```
In [ ]: # Sentiment is currently of type object, needs to be float

        # Convert sentiment datatype to float

        pdf1['sentiment'] = pdf1.sentiment.astype(float)

        # Check datatypes

        pdf1.dtypes

        # list new panda dataframe

        pdf1
```

```
In [ ]: # Describe data

        pdf1.describe()
```

```
In [ ]: topic_groups = pdf1.groupby('topic')
        topic_groups.size()
```

```
In [ ]: sentiment_terms1 = ['ai', 'data', 'tecnology', 'cloud']
```

```
In [ ]: pdf2 = pdf1[pdf1.topic.isin(sentiment_terms1)]
        pdf2
```

```
In [ ]: # Histograms

        pdf2['sentiment'].plot(kind='hist', bins=5)
```

```
In [ ]: pdf2_plot = topic_groups.plot(kind='bar')
```

```
In [ ]: # Boxplot sentiments by topic

        pdf2.boxplot(by='topic', column=['sentiment'], grid=False)
```

```
In [ ]:
```