pyspark code cheat sheet

Week 1 word count example

```
from pyspark import SparkConf, SparkContex
from pyspark.sql import SparkSession

spark_conf = SparkConf()\
  .setMaster('local[*]')\
  .setAppName('Assignment_1v2')

spark = SparkSession.builder.config(conf = spark_conf).getOrCreate()
sc = spark.sparkContext

twitter_rdd = sc.textFile('twitter.txt')
counts = twitter_rdd.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.collect()
```

Week 2 parallel search

```
bank_rdd1 = bank_rdd.map(lambda line: line.split(','))
header = bank_rdd1.first()
bank_rdd1 = bank_rdd1.filter(lambda row: row != header)   #filter out header


bank_rdd1 = bank_rdd1.filter(lambda x: int(x[5])>1000 and int(x[5])<2000)


numPartitions = bank_rdd1.getNumPartitions()
print(f"Total partitions: {numPartitions}")

# glom(): Return an RDD created by coalescing all elements within each
partition into a list
partitions = bank_rdd1.glom().collect()
for index,partition in enumerate(partitions):
    print(f'------ Partition {index}:')
    for record in partition:
        print(record)

result_max_balance = bank_rdd_4.max(key=lambda x: int(x[5]))
```

```
# Round-robin data partitioning
df_round = df.repartition(5)
# Range data partitioning
df_range = df.repartitionByRange(5,"balance")
# Hash data partitioning
column_hash = "education"
df_hash = df.repartition(column_hash)
```

Week 3 parallel join

```
from pyspark.sql.functions import broadcast

# Use broadcast function to specify the use of BroadcastHashJoin algorithm
df_joined_broadcast = df_B.join(broadcast(df_A),df_A.id==df_B.id,how='inner')

df_dict_inner_summ =
df_dictionary.join(df_summer,df_dictionary.Code==df_summer.Country,how='inner')
```

Week 4 parallel aggregation

```
import pyspark.sql.functions as F

#### Aggregate the dataset by 'Year' and count the total number of athletes
using Dataframe
agg_attribute = 'Year'
df_count =
df_events.groupby(agg_attribute).agg(F.count(agg_attribute).alias('Total'))

#### Aggregate the dataset by 'Year' and count the total number of athletes
using SQL
sql_count = spark.sql('''
  SELECT year,count(*)
  FROM sql_events
  GROUP BY year
''')

df = df.withColumn('balance', col('balance').cast('integer'))
df = df.withColumn('Age',F.col('Age').cast(IntegerType()))

from pyspark.sql.types import IntegerType
df = df.withColumn('Height',df.Height.cast(IntegerType()))
df.filter((df.Season == 'Winter')) \
  .groupby('Country') \
  .agg(F.min('Height').alias('min_height'), \
       F.avg('Height').alias('avg_height'), \
       F.max('Height').alias('max_height')) \
  .sort('avg_height', ascending=False) \
```

```
  .show()
```

Week 5 spark ml

```
from pyspark.ml.feature import StringIndexer

df_ref = spark.createDataFrame(
    [(0, "a"), (1, "b"), (2, "c"), (3, "a"), (4, "a"), (5, "c")],
    ["id", "category"])

indexer = StringIndexer(inputCol="category", outputCol="categoryIndex")
indexed_transformer = indexer.fit(df_ref)
indexed = indexed_transformer.transform(df_ref)
indexed.show()

from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.classification import GBTClassifier
dc = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'attack',
maxDepth = 10)
gbt = GBTClassifier(featuresCol = 'features', labelCol = 'attack',  maxIter =
10)
```

stream

```
def connect_kafka_producer():
    _producer = None
    try:
        _producer = KafkaProducer(bootstrap_servers=['localhost:9092'],
                                  value_serializer=lambda x:
dumps(x).encode('ascii'),
                                  api_version=(0, 10))
    except Exception as ex:
        print('Exception while connecting Kafka.')
        print(str(ex))
    finally:
        return _producer

def connect_kafka_consumer():
    _consumer = None
    try:
         _consumer = KafkaConsumer(topic,
                                   consumer_timeout_ms=10000, # stop iteration
if no message after 10 sec
                                   auto_offset_reset='latest', # comment this
if you don't want to consume earliest
                                   bootstrap_servers=['localhost:9092'],
```

```
                                    value_deserializer=lambda x:
loads(x.decode('ascii')),
                                    api_version=(0, 10))
    except Exception as ex:
        print('Exception while connecting Kafka')
        print(str(ex))
    finally:
        return _consumer

import os
os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages org.apache.spark:spark-
streaming-kafka-0-10_2.12:3.0.0,org.apache.spark:spark-sql-kafka-0-
10_2.12:3.0.0 pyspark-shell'

from pyspark.sql import SparkSession
from pyspark.sql.functions import explode
from pyspark.sql.functions import split
from pyspark.sql import functions as F
from pyspark.sql.types import *

spark = SparkSession \
    .builder \
    .appName("Clickstream Analysis in Spark") \
    .getOrCreate()

topic = "clickstream"
df = spark \
  .readStream \
  .format("kafka") \
  .option("kafka.bootstrap.servers", "127.0.0.1:9092") \
  .option("subscribe", topic) \
  .load()

query_file_sink_p = df_process \
    .writeStream.format('parquet')\
    .outputMode('append')\
    .option('path', 'process.parquet') \
    .option('checkpointLocation', 'parquet/linux_process_log/checkpoint')\
    .start()

attack_count_dfm = predictions_dfm \
    .filter('prediction = 1') \
    .groupBy('machine', window(predictions_dfm.event_time, '2 minutes')) \
    .agg(approx_count_distinct('CMD_PID').alias('count')) \
    .select('machine','window','count') \
    .orderBy('machine','window')

query = df \
    .writeStream \
```

```
    .outputMode("append") \
    .format("console") \
    .start()

query_p = attack_count_dfp \
    .writeStream \
    .queryName("process_attack_count") \
    .outputMode("Complete") \
    .format("memory") \
    .trigger(processingTime='10 seconds') \
    .start()
```