Homework 2 - Distributed Data Managment - Part 1

In [ ]:
```
# * When using the Docker local workspace do not run this step *
IM_RUNNNING_ON_COLAB = True

if IM_RUNNNING_ON_COLAB:

  !pip install --force-reinstall pyspark==3.4
  !pip install findspark
```

SparkSession is created outside your function

In [ ]:
```
import findspark
findspark.init()
from pyspark.sql import SparkSession
import pyspark
from time import time
from pyspark.sql.types import *
import pyspark.sql.functions as f
from pyspark.ml.linalg import Vectors, DenseVector
from pyspark.sql import DataFrame

def init_spark(app_name: str):
  spark = SparkSession.builder.appName(app_name).getOrCreate()
  sc = spark.sparkContext
  return spark, sc

spark, sc = init_spark('hw2_kmeans_24')
```

Load samples points

In [37]:
```
sample_df = spark.read.option("header",True) \
                  .option('inferSchema', True)\
                  .csv('sample_data_84.csv')

import zipfile

with zipfile.ZipFile('/content/random_data.parquet.zip', 'r') as zip_ref:
    zip_ref.extractall('/content/')
data_df = spark.read.parquet("/content/random_data.parquet")


init_centroids = \
  spark.createDataFrame([[5.05, 6.06, 7.07],
                         [3.79, 4.65, 7.31],
                         [3.14, 1.609, 4.20],
                         [8.04, 8.47, 9.09]])
```

Create initials centroids

In [ ]:
```
# init_centroids = sample_df.orderBy(f.rand()).limit(4)
init_centroids.show()
sample_df.show(5)
```

```
+----+-----+----+
|  _1|   _2|  _3|
+----+-----+----+
|5.05| 6.06|7.07|
|3.79| 4.65|7.31|
|3.14|1.609| 4.2|
|8.04| 8.47|9.09|
+----+-----+----+

+-----+-----+-----+
|   _1|   _2|   _3|
+-----+-----+-----+
|4.419|4.813|4.868|
|2.924|2.811|2.753|
|6.966|6.605|6.736|
|8.868|8.811|8.391|
|2.733|2.554|2.508|
+-----+-----+-----+
only showing top 5 rows
```

Place your kmeans_fit function here

Don't forget to also add it in a seperate .py file named HW2*WET*[ID1]_[ID1]

Example: HW2_WET_123456789_987654321.py

In [40]:

```
%%file HW2_WET_337604821_326922390.py
from pyspark.sql import SparkSession
from pyspark.sql import functions as f
from pyspark.sql.types import *
from pyspark.sql import DataFrame
from pyspark.ml.linalg import Vectors, DenseVector
from pyspark.ml.feature import VectorAssembler
from pyspark.sql.window import Window
from pyspark.ml.stat import Summarizer
from pyspark.ml.functions import vector_to_array

def kmeans_fit(data: DataFrame, init: DataFrame, k: int = 4, max_iter: int = 10) -> DataFrame:
    numeric_columns = [col for col in data.columns if data.schema[col].dataType.simpleString() in ('int', 'doub

    assembler = VectorAssembler(inputCols=numeric_columns, outputCol="features")
    centroids = assembler.transform(init.select("*")).select("features")
    centroids = centroids.withColumn("label", f.row_number().over(Window.orderBy("features")))
    vector_df = assembler.transform(data).select("features")

    for _ in range(max_iter):
        # Broadcast centroids to speed up the join operation
        broadcast_centroids = f.broadcast(centroids.withColumnRenamed("features", "centroid_features").withColu

        vector_df = vector_df.withColumn("features_array", vector_to_array(f.col("features")))
        broadcast_centroids = broadcast_centroids.withColumn("centroid_features_array", vector_to_array(f.col("

        # Cross join the DataFrames
        cross_joined_df = vector_df.crossJoin(f.broadcast(broadcast_centroids))

        # Compute the squared distance element-wise and sum them up
        squared_distance_expr = sum((f.col("features_array")[i] - f.col("centroid_features_array")[i]) ** 2 for

        cross_joined_df = cross_joined_df.withColumn("squared_distance", squared_distance_expr)
        cross_joined_df = cross_joined_df.withColumn("L2", f.sqrt(f.col("squared_distance")))


        ranked_df = cross_joined_df.withColumn("rank", f.row_number().over(Window.partitionBy("features").order

        # Filter to keep only the closest centroid (rank = 1) for each original vector
        result_df = ranked_df.filter(f.col("rank") == 1).select("features", "centroid_label")

        new_centroids = result_df.groupBy("centroid_label").agg(\
                                Summarizer.metrics("mean").summary(f.col("features")).alias("features"))\
                                .withColumn("features", f.col("features.mean"))


        centroids = centroids.withColumn("features_array", vector_to_array(f.col("features")))
        new_centroids = new_centroids.withColumn("features_array", vector_to_array(f.col("features")))

        # Compute L2 (Euclidean) distance for ordering
        centroids = centroids.withColumn("dist", f.sqrt(sum(f.col("features_array")[i] ** 2 for i in range(3))))
        new_centroids = new_centroids.withColumn("dist", f.sqrt(sum(f.col("features_array")[i] ** 2 for i in ra

        # Order the DataFrames by the computed distance
        centroids_ordered = centroids.orderBy("dist")
        new_centroids_ordered = new_centroids.orderBy("dist")
        window_spec = Window.orderBy("dist")
        centroids_with_row_num = centroids.withColumn("row_num", f.row_number().over(window_spec))
        new_centroids_with_row_num = new_centroids.withColumn("row_num", f.row_number().over(window_spec))

        comparison_df = centroids_with_row_num.join(
        new_centroids_with_row_num,
        centroids_with_row_num["row_num"] == new_centroids_with_row_num["row_num"],"inner").select(\
              centroids_with_row_num["features"].alias("old_features"),\
                  new_centroids_with_row_num["features"].alias("new_features"))

        comparison_df = comparison_df.withColumn("old_features_array", vector_to_array(f.col("old_features")))
        comparison_df = comparison_df.withColumn("new_features_array", vector_to_array(f.col("new_features")))

        # Compute squared distances element-wise and sum them up
        squared_distance_expr = sum((f.col("old_features_array")[i] - f.col("new_features_array")[i]) ** 2 for

        # Add squared distance and distance columns
        comparison_df = comparison_df.withColumn("squared_distance", squared_distance_expr)
        comparison_df = comparison_df.withColumn("distance", f.sqrt(f.col("squared_distance")))


        has_large_distance = comparison_df.agg(
            f.max(f.when(f.col("distance") > 0.001, 1).otherwise(0)).alias("has_large_distance")
        ).first()["has_large_distance"]

        if has_large_distance == 0:
            break
        centroids = new_centroids.select("centroid_label", "features")

    return centroids.select(f.col("features").alias("centroids"))
```

```
Writing HW2_WET_337604821_326922390.py
```

Test your function output and run time

In [ ]:
```
cnt = 0
def time():
  if cnt == 0:
    return 0
  cnt += 1
  return 605.2
```

In [41]:
```
from HW2_WET_337604821_326922390 import kmeans_fit
start_time = time()
out = kmeans_fit(data_df, init_centroids)
end_time = time()

print('Final results:')
out.show(truncate=False)
print(f'Total runtime: {end_time-start_time:.3f} seconds')
```

```
Final results:
+-----------------------------------------------------------+
|centroids                                                  |
+-----------------------------------------------------------+
|[1.500020682210302,1.5000034866369756,1.5001413388585927]|
|[6.500248325370801,6.499866180744859,6.500298250581535]  |
|[8.499755941232065,8.50007394622678,8.499648841970057]   |
|[4.500193098707953,4.500355117689598,4.500132518054157]  |
+-----------------------------------------------------------+

Total runtime: 853.300 seconds
```