**FINAL PROJECT SCRIPTS**

//Analysis of Harddrive data from Backblaze

import org.apache.spark.sql.SparkSession

import org.apache.spark.sql.functions.\_

import org.apache.spark.sql.expressions.Window

val spark=SparkSession.builder().appName("Spark dataframe eaxample").config("spark.some.config.option","value").getOrCreate()

import spark.implicits.\_

var data=spark.read.option("sep",",").option("header","true").csv("C:/BigData/~data/FP-Data-2020/Drivestats-\*.csv.gz")

//count of lines

val linecount= data.count()

//Parquet implementation

data.write.parquet("C:/BigData/~data/project.parquet")

//Reading values from parquet

val parquetProject = spark.read.parquet("C:/BigData/~data/project.parquet")

//selecting most used columns in the data

val data\_useful = parquetProject.select(

col("date"),

col("serial\_number"),

col("model"),

col("capacity\_bytes"),

col("failure"),

col("smart\_1\_raw"),col("smart\_5\_raw"),col("smart\_9\_raw"),col("smart\_187\_raw"),

col("smart\_188\_raw"),col("smart\_194\_raw"),col("smart\_197\_raw"),

col("smart\_198\_raw"))

data\_useful.show(5)

//Adding an additional column for manufacturer based on model

import org.apache.spark.sql.functions.{when, \_}

val data\_MFG = data\_useful.withColumn("manufacturer", when(col("model") like "ST%","Seagate")

.when(col("model") like "Hitachi%","HGST/Hitachi")

.when(col("model") like "HGST%","HGST/Hitachi")

.when(col("model") like "TOSHI%","TOSHIBHA")

.when(col("model") like "Seagate%","Seagate")

.when(col("model") like "Dell%","Dell")

.when(col("model") like "WDC%","WDC")

.otherwise("Unknown"))

data\_MFG.show(5)

//To show Capacity bytes in more user friendly

val data\_mod = data\_MFG.withColumn("capacity\_bytes\_TB", when(col("capacity\_bytes") like "-1%","not\_detected")

.when(col("capacity\_bytes") like "12000138625024%","12TB")

.when(col("capacity\_bytes") like "500107862016%","500GB")

.when(col("capacity\_bytes") like "480036847616%","480GB")

.when(col("capacity\_bytes") like "480103981056%","480GB")

.when(col("capacity\_bytes") like "240057409536%","240GB")

.when(col("capacity\_bytes") like "250059350016%","240GB")

.when(col("capacity\_bytes") like "14000519643136%","14TB")

.when(col("capacity\_bytes") like "6001175126016%","6TB")

.when(col("capacity\_bytes") like "4000787030016%","4TB")

.when(col("capacity\_bytes") like "16000900661248%","16TB")

.when(col("capacity\_bytes") like "10000831348736%","10TB")

.when(col("capacity\_bytes") like "8001563222016%","8TB")

.when(col("capacity\_bytes") like "2000398934016%","2TB")

.when(col("capacity\_bytes") like "18000207937536","18TB")

.when(col("capacity\_bytes") like "1000204886016%","1TB"))

data\_mod.show(5)

//Handling null values to avoid inconsistency

//val df = data\_mod.na.drop(Seq("smart\_194\_raw"))

val df= data\_mod.na.fill(0)

//creating a temporary table for the dataframe

df.createOrReplaceTempView("query\_data")

//Drive temperature

val drive\_temp = spark.sql("""

select

manufacturer,

model,

capacity\_bytes\_TB,

max(failure) as failure,

Avg(smart\_194\_raw) as temperature

from query\_data

group by manufacturer,model, capacity\_bytes\_TB

order by manufacturer, capacity\_bytes\_TB

""").cache()

drive\_temp.show(5)

drive\_temp.createOrReplaceTempView("temperature")

%sql

select \* from temperature

//Calculating Annual Failure rate

//drivedays = the number of days all of the drives being observed were operational during the period of observation.

val drivedays = spark.sql("SELECT manufacturer, model, capacity\_bytes\_TB, count(date) AS drive\_days "+

"FROM query\_data "+

"GROUP BY model, manufacturer, capacity\_bytes\_TB")

drivedays.show(5)

//Drive Failures is the number of drives that failed during the period of observation

val failure\_bymodel = spark.sql("SELECT manufacturer, model,capacity\_bytes\_TB, count(\*) AS failures, avg(smart\_194\_raw) as temperature "+

"FROM query\_data "+

"WHERE failure = 1 "+

"GROUP BY model, manufacturer, capacity\_bytes\_TB")

failure\_bymodel.show(5)

val total\_datacount\_bymodel = spark.sql("SELECT manufacturer, model, capacity\_bytes\_TB, count(\*) AS count "+

"FROM query\_data "+

"WHERE date = '2020-12-31'"+

"GROUP BY model, manufacturer, capacity\_bytes\_TB")

total\_datacount\_bymodel.show(5)

total\_datacount\_bymodel.registerTempTable("model\_count")

failure\_bymodel.registerTempTable("model\_failures")

drivedays.registerTempTable("drivedays")

//since we are considering 2020 data, Annual Failure Rate can be calculated using 366 days instead of 365 days

val failure\_rates = spark.sql("SELECT drivedays.manufacturer AS manufacturer, drivedays.model AS model, "+

"drivedays.capacity\_bytes\_TB AS capacity\_bytes, "+

"drivedays.drive\_days AS drivedays, "+

"model\_failures.failures AS failures, "+

"model\_failures.temperature AS temperature, "+

"100.0 \* (1.0 \* failures) / (drive\_days / 366.0) AS annual\_failure\_rate "+

"FROM drivedays, model\_failures, model\_count "+

"WHERE drivedays.model = model\_failures.model "+

"AND model\_count.model = model\_failures.model "+

"ORDER BY model")

failure\_rates.show(5)

failure\_rates.registerTempTable("Annual\_failure\_rate")

%sql

select \* from Annual\_failure\_rate

//Harddrive failure prediction

//Back blaze consider 5 smart values inorder to predict the drive failure. If one of these values is greater than one that drive may potentially fail in near future.

//Since it is very important to protect the data, failure prediction gives a better insight of when to replace a drive.

//Here each of those 5 smart raw values are being analysed seperately to see whether the failure of drive depends on these values.

//Smart 5

val Reallocated\_Sector\_Count = spark.sql("SELECT model, smart\_5\_raw, failure "+

"FROM query\_data "+

"WHERE smart\_5\_raw >=1 "+

"group by model, smart\_5\_raw,failure")

Reallocated\_Sector\_Count.show(5)

val Reported\_Uncorrectable\_Errors = spark.sql("SELECT model, smart\_187\_raw, failure "+

"FROM query\_data "+

"WHERE smart\_187\_raw >=1 ")

Reported\_Uncorrectable\_Errors.show(5)

val Command\_Timeout = spark.sql("SELECT model, smart\_188\_raw, failure "+

"FROM query\_data "+

"WHERE smart\_188\_raw >=1 ")

Command\_Timeout.show(5)

val Current\_Pending\_Sector\_Count = spark.sql("SELECT model, smart\_197\_raw, failure "+

"FROM query\_data "+

"WHERE smart\_197\_raw >=1 ")

Current\_Pending\_Sector\_Count.show(5)

val Offline\_Uncorrectable = spark.sql("SELECT model, smart\_198\_raw, failure "+

"FROM query\_data "+

"WHERE smart\_198\_raw >= 100 ")

Offline\_Uncorrectable.show(5)