* create Database dataCollection;
* Use dataCollection;
* DROP TABLE if exists airValues;
* /\* Command to create table on top of azure data lake location \*/
* CREATE EXTERNAL TABLE airValues(
* Latitude STRING,
* Longitude STRING,
* DateObserved STRING,
* ParameterName STRING,
* AQI STRING,
* CategoryNumber STRING
* )
* Partitioned by(AQIDATE STRING)
* ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
* STORED AS TEXTFILE LOCATION 'adl://aqidatastorgae.azuredatalakestore.net/Test/AQIDATA';
* /\* Command to create table which performs typecasting and data cleaning \*/
* Create table calaqi(
* latitude double,
* longitude double,
* dateobserved string,
* parametername string,
* parametervalue int,
* categorynumber int,
* categoryname string
* )
* Partitioned by(collection\_date date)
* ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
* Stored as textfile
* /\* Command to filter pm2.5 data into a table \*/
* CREATE EXTERNAL TABLE calaqi\_pm25(
* dateobserved string,
* parametervalue int,
* categorynumber int,
* categoryname string,
* collection\_date string)
* ROW FORMAT DELIMITED
* FIELDS TERMINATED BY ','
* STORED AS TEXTFILE
* LOCATION 'hdfs://sandbox-hdp.hortonworks.com:8020/apps/hive/warehouse/datacollection.db/calaqi\_pm25'
* /\* Command to filter ozone data into a table \*/
* CREATE EXTERNAL TABLE calaqi\_ozone(
* dateobserved string,
* parametervalue int,
* categorynumber int,
* categoryname string,
* collection\_date string)
* ROW FORMAT DELIMITED
* FIELDS TERMINATED BY ','
* STORED AS TEXTFILE
* LOCATION 'hdfs://sandbox-hdp.hortonworks.com:8020/apps/hive/warehouse/datacollection.db/calaqi\_ozone'
* /\*Command to create table which store AQI value for that particular day \*/
* Create table calaqi\_value(
* Dateobserved string,
* Parametervalue int,
* Categorynumber int,
* Categoryname string)
* ROW FORMAT DELIMITED
* FIELDS TERMINATED BY ','
* STORED AS TEXTFILE
* LOCATION 'hdfs://sandbox-hdp.hortonworks.com:8020/apps/hive/warehouse/datacollection.db/calaqi\_value'
* /\* Inserting data into calaqi table \*/
* Insert overwrite table calaqi partition(collection\_date)
* select
* cast(regexp\_replace(latitude,'\\"','')as double) as latitude,
* cast(regexp\_replace(longitude,'\\"','')as double) as longitude,
* cast(regexp\_replace(dateobserved,'\\"','')as string) as dateobserved,
* cast(regexp\_replace(parametername,'\\"','')as string) as parametername,
* cast(regexp\_replace(aqi,'\\"','')as int) as aqi,
* cast(regexp\_replace(categorynumber,'\\"','') as int) as categorynumber,
* Case
* when categorynumber like '%1%' then 'Good'
* when categorynumber like '%2%' then 'Moderate'
* when categorynumber like '%3%' then 'Unhealthy for sensitive group'
* when categorynumber like '%4%' then 'Unhealthy'
* end,
* (from\_unixtime(unix\_timestamp(regexp\_replace(dateobserved,'\\"',''),"yyyy-MM-dd'T'HH:mm"),"yyyyMMdd")) as collection\_date
* from airvalues
* /\* Inserting data into calaqi\_pm25 table \*/
* Insert overwrite table calaqi\_pm25
* Select
* (from\_unixtime(unix\_timestamp(regexp\_replace(dateobserved,'\\"',''),"yyyy-MM-dd'T'HH:mm"),"yyyyMMdd")) as dateobserved,
* max(parametervalue) as parametervalue,
* Case
* when max(parametervalue) <51 then 1
* when max(parametervalue) >50 and max(parametervalue) < 101 then 2
* when max(parametervalue) >100 and max(parametervalue) < 151 then 3
* when max(parametervalue) >150 then 4
* end,
* Case
* when max(parametervalue) <51 then 'Good'
* when max(parametervalue) >50 and max(parametervalue) < 101 then 'Moderate'
* when max(parametervalue) >100 and max(parametervalue) < 151 then 'Unhealthy for sensitive group'
* when max(parametervalue) >150 then 'Unhealthy'
* end,
* collection\_date
* From calaqi where parametername='PM2.5' group by (from\_unixtime(unix\_timestamp(regexp\_replace(date observed,'\\"',''),"yyyy-MM-dd'T'HH:mm"),"yyyyMMdd")),collection\_date;
* /\* Inserting data into calaqi\_ozone table \*/
* Insert overwrite table calaqi\_ozone
* Select
* (from\_unixtime(unix\_timestamp(regexp\_replace(dateobserved,'\\"',''),"yyyy-MM-dd'T'HH:mm"),"yyyyMMdd")) as dateobserved,
* max(parametervalue) as parametervalue,
* Case
* when max(parametervalue) <51 then 1
* when max(parametervalue) >50 and max(parametervalue) < 101 then 2
* when max(parametervalue) >100 and max(parametervalue) < 151 then 3
* when max(parametervalue) >150 then 4
* end,
* Case
* when max(parametervalue) <51 then 'Good'
* when max(parametervalue) >50 and max(parametervalue) < 101 then 'Moderate'
* when max(parametervalue) >100 and max(parametervalue) < 151 then 'Unhealthy for sensitive group'
* when max(parametervalue) >150 then 'Unhealthy'
* end,
* collection\_date
* From calaqi where parametername='OZONE' group by (from\_unixtime(unix\_timestamp(regexp\_replace(date observed,'\\"',''),"yyyy-MM-dd'T'HH:mm"),"yyyyMMdd")),collection\_date;
* /\* Inserting Data into Calaqi\_value table \*/
* Insert into calaqi\_value
* select o.dateobserved,
* greatest(o.parametervalue,p.parametervalue),
* Case
* when greatest(o.parametervalue,p.parametervalue) <51 then 1
* when greatest(o.parametervalue,p.parametervalue) >50 and greatest(o.parametervalue,p.parametervalue) < 101 then 2
* when greatest(o.parametervalue,p.parametervalue) >100 and greatest(o.parametervalue,p.parametervalue) < 151 then 3
* when greatest(o.parametervalue,p.parametervalue) >150 then 4
* end,
* Case
* when greatest(o.parametervalue,p.parametervalue) <51 then 'Good'
* when greatest(o.parametervalue,p.parametervalue) >50 and greatest(o.parametervalue,p.parametervalue)< 101 then 'Moderate'
* when greatest(o.parametervalue,p.parametervalue) >100 and greatest(o.parametervalue,p.parametervalue) < 151 then 'Unhealthy for sensitive group'
* when greatest(o.parametervalue,p.parametervalue) >150 then 'Unhealthy'
* end
* from calaqi\_ozone o full outer join calaqi\_pm25 p on (o.dateobserved=p.dateobserved);
* import com.cloudera.sparkts.models.\_
* import com.cloudera.sparkts.models.ARIMA.\_
* import breeze.linalg.\_
* import org.apache.spark.mllib.linalg.{Vector, Vectors}
* val lines = sc.textFile("hdfs://sandbox-hdp.hortonworks.com:8020/apps/hive/warehouse/datacollection.db/calaqi\_value/000000\_0")
* val records = lines.map(\_.split(","))
* val date = records.map(rec=>rec(0))
* val dates =date.collect()
* val tuples = records.map(rec=>rec(1))
* val tuples = records.map(rec=>rec(1).toDouble)
* val ts = tuples.collect()
* val trainingSize = (ts.length \* 0.85).toInt
* val trainingAmounts = new Array[Double](trainingSize)  
        for(i <- 0 until trainingSize)
* {  
          trainingAmounts(i) = ts(i)  
        }
* import org.apache.spark.mllib.linalg.{Vector, Vectors}
* val actual = Vectors.dense(trainingAmounts)  
        val period = ts.length - trainingSize  
        val model = ARIMA. fitModel(100, 0, 0,actual)
* println("best-fit model ARIMA(" + model.p + "," + model.d + "," + model.q + ") AIC=" + model.approxAIC(actual) )  
        val predicted = model.forecast(actual, period)  
        var totalErrorSquare = 0.0  
        for (i <- (predicted.size - period) until predicted.size) {  
          val errorSquare = Math.pow(predicted(i) - ts(i), 2)  
          println("Predicted= "+predicted(i) + "\t should be \t"+" Actual= " + ts(i) + "\t Error Square = " + errorSquare)  
          totalErrorSquare += errorSquare  
        }  
        println("Root Mean Square Error: " +Math.sqrt(totalErrorSquare/period))  
      println("Categorization of predicted models")
* var goodp = 0
* val predictedCat=Array.ofDim[String](predicted.size)
* var moderatep = 0
* var unhealthysp = 0
* var unhealthyp = 0
* var good = 0
* var actualCat=Array.ofDim[String](predicted.size)
* var predictedCat=Array.ofDim[String]( predicted.size)
* var moderate = 0
* var unhealthys = 0
* var unhealthy = 0
* var x=0
* for(i <- (predicted.size - period) until predicted.size){
* if (predicted(i) < 51){
* println(dates(i)+" "+predicted(i)+" Good")
* goodp =goodp+1
* predictedCat.update(x,"Good")
* }
* if((predicted(i) < 101) && (predicted(i) > 50)){
* println(dates(i)+" "+ predicted(i)+" Moderate")
* moderatep =moderatep+1
* predictedCat.update(x, "Moderate")
* }
* if ((predicted(i) < 151) && (predicted(i) > 100)){
* println(dates(i)+" "+ predicted(i)+" Unhealthy for sensitive group")
* unhealthysp =unhealthysp+1
* predictedCat.update(x,"Unhealthy for sensitive people")
* }
* if ((predicted(i) > 150)){
* println(dates(i)+" "+ predicted(i)+" Unhealthy" )
* unhealthyp =unhealthyp+1
* predictedCat.update(x, "Unhealthy")
* }
* x=x+1
* }
* var x=0
* for(i <- (predicted.size - period) until predicted.size){
* if (ts (i) < 51){
* //println(i+" "+ts(i)+" Good")
* good =good+1
* actualCat.update(x, "Good")
* }
* if((ts (i) < 101) && (ts (i) > 50)){
* //println(i+" "+ ts (i)+" Moderate")
* moderate =moderate+1
* actualCat.update(x,"Moderate")
* }
* if ((ts (i) < 151) && (ts (i) > 100)){
* //println(i+" "+ ts (i)+" Unhealthy for sensitive group")
* unhealthys =unhealthys+1
* actualCat.update(x,"Unhealthy for sensitive group")
* }
* if ((ts (i) > 150)){
* //println(i+" "+ ts (i)+" Unhealthy" )
* unhealthy =unhealthy+1
* actualCat.update(x,"Unhealthy")
* }
* x=x+1
* }
* var errorCount=0
* var errorCountGood=0
* var errorCountModerate=0
* for (i <- 0 until actualCat.size){
* if(predictedCat(i) != actualCat(i)){
* errorCount=errorCount+1
* if(predictedCat(i)== "Moderate")
* {
* errorCountModerate= errorCountModerate+1
* }
* if(predictedCat(i)== "Good")
* {
* errorCountGood= errorCountGood+1
* }
* }
* }
* println("Error Count percentage " +(errorCount\*100/predictedCat.size))
* println("Accuracy for Category Good " +( 100 - ( errorCountGood \*100)/good))
* println("Accuracy for Category Moderate " + (100- ( errorCountModerate \*100/ moderatep)))