**Spark Data frame for US\_Election\_Demo**

**=================**

**Upload the given CSV files on hdfs using FTP**

**Run the following code on zeppelin console by adding the username provided.**

**hdfs dfs -chmod -R 777 /user/edureka\_2**

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

import org.apache.spark.storage.StorageLevel

import scala.io.Source

import scala.collection.mutable.HashMap

import java.io.File

import org.apache.spark.sql.Row

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

import scala.collection.mutable.ListBuffer

import org.apache.spark.util.IntParam

import org.apache.spark.util.StatCounter

import org.apache.spark.rdd.RDD

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.SparkConf

import org.apache.spark.sql.SQLContext

import org.apache.spark.rdd.\_

import org.apache.spark.ml.feature.VectorAssembler

import org.apache.spark.ml.clustering.KMeans

import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}

import org.apache.spark.mllib.linalg.Vectors

val sqlContext = new org.apache.spark.sql.SQLContext(sc)

import sqlContext.implicits.\_

import sqlContext.\_

val schema = StructType(Array(StructField("state", StringType, true), StructField("state\_abbr", StringType, true), StructField("county", StringType, true), StructField("fips", StringType, true), StructField("party", StringType, true), StructField("candidate", StringType, true), StructField ("votes", IntegerType, true), StructField("fraction\_votes", DoubleType, true)))

**//add path to the csv file uploaded on hdfs**

val df = spark.read.option("header","true").schema(schema).csv("hdfs://nameservice1/user/edureka\_253770/primary\_results.csv")

val df\_R = df.filter($"party" === "Republican")

val df\_D = df.filter($"party" === "Democrat")

df\_R.createOrReplaceTempView("election")

val temp = spark.sql("SELECT \* FROM election INNER JOIN (SELECT fips as b, MAX(fraction\_votes) AS a FROM election GROUP BY fips) groupedtt WHERE election.fips = groupedtt.b AND election.fraction\_votes = groupedtt.a")

temp.createOrReplaceTempView("election1")

val temp = spark.sql("SELECT state, state\_abbr, county, fips, party, candidate, votes, fraction\_votes FROM election1")

val r\_winner = temp

r\_winner.createOrReplaceTempView("republican")

val temp = spark.sql("select state, candidate, count(candidate) as countyswon from republican group by state, candidate")

val d\_state = temp

d\_state.createOrReplaceTempView("state")

val schema1 = StructType(Array( StructField("fips", StringType, true), StructField("area\_name", StringType, true), StructField("state\_abbreviation", StringType, true), StructField("Population\_2014", IntegerType, true), StructField("Population\_2010\_Apr1", IntegerType, true), StructField("Change\_in\_Population\_percent", DoubleType, true), StructField("Population\_2010", IntegerType, true), StructField ("Persons\_under\_5", DoubleType, true), StructField("Persons\_under\_18", DoubleType, true), StructField ("Persons\_65\_years\_over", DoubleType, true), StructField("Female\_persons\_percent", DoubleType, true), StructField("White\_alone", DoubleType, true), StructField("Black\_or\_African\_American\_alone", DoubleType, true), StructField("American\_Indian\_and\_Alaska\_Native\_alone", DoubleType, true), StructField("Asian\_alone", DoubleType, true), StructField("Native\_Hawaiian\_and\_Other\_Pacific\_Islander\_alone", DoubleType, true), StructField("Two\_or\_More\_Races", DoubleType, true), StructField("Hispanic\_or\_Latino", DoubleType, true), StructField("White\_alone\_not\_Hispanic\_or\_Latino", DoubleType, true), StructField("Living\_in\_same\_house\_1\_year\_&\_over", DoubleType, true), StructField("Foreign\_born\_persons", DoubleType, true), StructField("Language\_other\_than\_English\_spoken\_at\_home", DoubleType, true), StructField("High\_school\_graduate\_or\_higher", DoubleType, true), StructField("Bachelor\_degree\_or\_higher", DoubleType, true), StructField("Veterans", IntegerType, true), StructField("Mean\_travel\_time\_to\_work", DoubleType, true), StructField("Housing\_units", IntegerType, true), StructField ("Homeownership\_rate", DoubleType, true), StructField("Housing\_units\_in\_multi\_unit\_structures", DoubleType, true), StructField("Median\_value\_of\_owner\_occupied\_housing\_units", IntegerType, true), StructField("Households", IntegerType, true), StructField("Persons\_per\_household", DoubleType, true), StructField("Per\_capita\_money\_income", IntegerType, true), StructField("Median\_household\_income", IntegerType, true), StructField("Persons\_below\_poverty\_level", DoubleType, true), StructField("Private\_nonfarm\_establishments", IntegerType, true), StructField("Private\_nonfarm\_employment", IntegerType, true), StructField("Private\_nonfarm\_employment\_percentage\_change", DoubleType, true), StructField ("Nonemployer\_establishments", IntegerType, true), StructField("Total\_number\_of\_firms", IntegerType, true), StructField("Black\_owned\_firms", DoubleType, true), StructField("American\_Indian\_and\_Alaska\_Native\_owned\_firms", DoubleType, true), StructField("Asian\_owned\_firms", DoubleType, true), StructField("Native\_Hawaiian\_and\_Other\_Pacific\_Islander\_owned\_firms", DoubleType, true), StructField ("Hispanic\_owned\_firms", DoubleType, true), StructField("Women\_owned\_firms", DoubleType, true), StructField("Manufacturers\_shipments", DoubleType, true), StructField("Merchant\_wholesaler\_sales", DoubleType, true), StructField("Retail\_sales", DoubleType, true), StructField("Retail\_sales\_per\_capita", IntegerType, true), StructField("Accommodation\_and\_food\_services\_sales", IntegerType, true), StructField("Building\_permits", IntegerType, true), StructField("Land\_area\_in\_square\_miles", DoubleType, true), StructField("Population\_per\_square\_mile", DoubleType, true)))

**//add path to the csv file uploaded on hdfs**

val df1 = sqlContext.read.option("header","true").schema(schema1).csv("hdfs://nameservice1/user/edureka\_253770/county\_facts.csv")

df1.createOrReplaceTempView("facts")

val temp = spark.sql("SELECT facts.fips as fips, republican.state as state, facts.state\_abbreviation as state\_abbreviation, area\_name, candidate, Persons\_65\_years\_over, Female\_persons\_percent, White\_alone, Black\_or\_African\_American\_alone, Asian\_alone, Hispanic\_or\_Latino, Foreign\_born\_persons, Language\_other\_than\_English\_spoken\_at\_home, Bachelor\_degree\_or\_higher, Veterans, Homeownership\_rate, Median\_household\_income, Persons\_below\_poverty\_level, Population\_per\_square\_mile FROM facts INNER JOIN republican ON CAST(facts.fips AS INT) = CAST(republican.fips AS INT)")

val df\_facts = temp

df\_facts.createOrReplaceTempView("winner\_facts")

val bc = df\_facts.filter($"candidate" === "Ben Carson")

val dt = df\_facts.filter($"candidate" === "Donald Trump")

val jk = df\_facts.filter($"candidate" === "John Kasich")

val mr = df\_facts.filter($"candidate" === "Marco Rubio")

val tc = df\_facts.filter($"candidate" === "Ted Cruz")

val wbc = bc.withColumn("w\_bc", lit(1)).withColumn("w\_dt", lit(0)).withColumn("w\_jk", lit(0)).withColumn("w\_mr", lit(0)).withColumn("w\_tc", lit(0))

val wdt = dt.withColumn("w\_bc", lit(0)).withColumn("w\_dt", lit(1)).withColumn("w\_jk", lit(0)).withColumn("w\_mr", lit(0)).withColumn("w\_tc", lit(0))

val wjk = jk.withColumn("w\_bc", lit(0)).withColumn("w\_dt", lit(0)).withColumn("w\_jk", lit(1)).withColumn("w\_mr", lit(0)).withColumn("w\_tc", lit(0))

val wmr = mr.withColumn("w\_bc", lit(0)).withColumn("w\_dt", lit(0)).withColumn("w\_jk", lit(0)).withColumn("w\_mr", lit(1)).withColumn("w\_tc", lit(0))

val wtc = tc.withColumn("w\_bc", lit(0)).withColumn("w\_dt", lit(0)).withColumn("w\_jk", lit(0)).withColumn("w\_mr", lit(0)).withColumn("w\_tc", lit(1))

wbc.createOrReplaceTempView("wbc")

wdt.createOrReplaceTempView("wdt")

wjk.createOrReplaceTempView("wjk")

wmr.createOrReplaceTempView("wmr")

wtc.createOrReplaceTempView("wtc")

val result1 = spark.sql("SELECT \* FROM wbc UNION ALL SELECT \* FROM wdt")

result1.createOrReplaceTempView("result1")

val result2 = spark.sql("SELECT \* FROM wjk UNION ALL SELECT \* FROM wmr")

result2.createOrReplaceTempView("result2") val result3 = spark.sql("SELECT \* FROM result1 UNION ALL SELECT \* FROM result2")

result3.createOrReplaceTempView("result3")

val result = spark.sql("SELECT \* FROM result3 UNION ALL SELECT \* FROM wtc")

result.createOrReplaceTempView("result")

val featureCols = Array("Persons\_65\_years\_over", "Female\_persons\_percent", "White\_alone","Black\_or\_African\_American\_alone", "Asian\_alone", "Hispanic\_or\_Latino", "Foreign\_born\_persons", "Language\_other\_than\_English\_spoken\_at\_home", "Bachelor\_degree\_or\_higher", "Veterans", "Homeownership\_rate", "Median\_household\_income", "Persons\_below\_poverty\_level", "Population\_per\_square\_mile", "w\_bc", "w\_dt", "w\_jk", "w\_mr", "w\_tc")

val rows = new VectorAssembler().setInputCols(featureCols).setOutputCol("features").transform(result)

val kmeans = new org.apache.spark.ml.clustering.KMeans().setK(4).setFeaturesCol("features").setPredictionCol("prediction")

val model = kmeans.fit(rows)

model.clusterCenters.foreach(println)

val categories = model.transform(rows)

categories.createOrReplaceTempView("c")