BIG DATA Analytics & Apps

LAB - 3

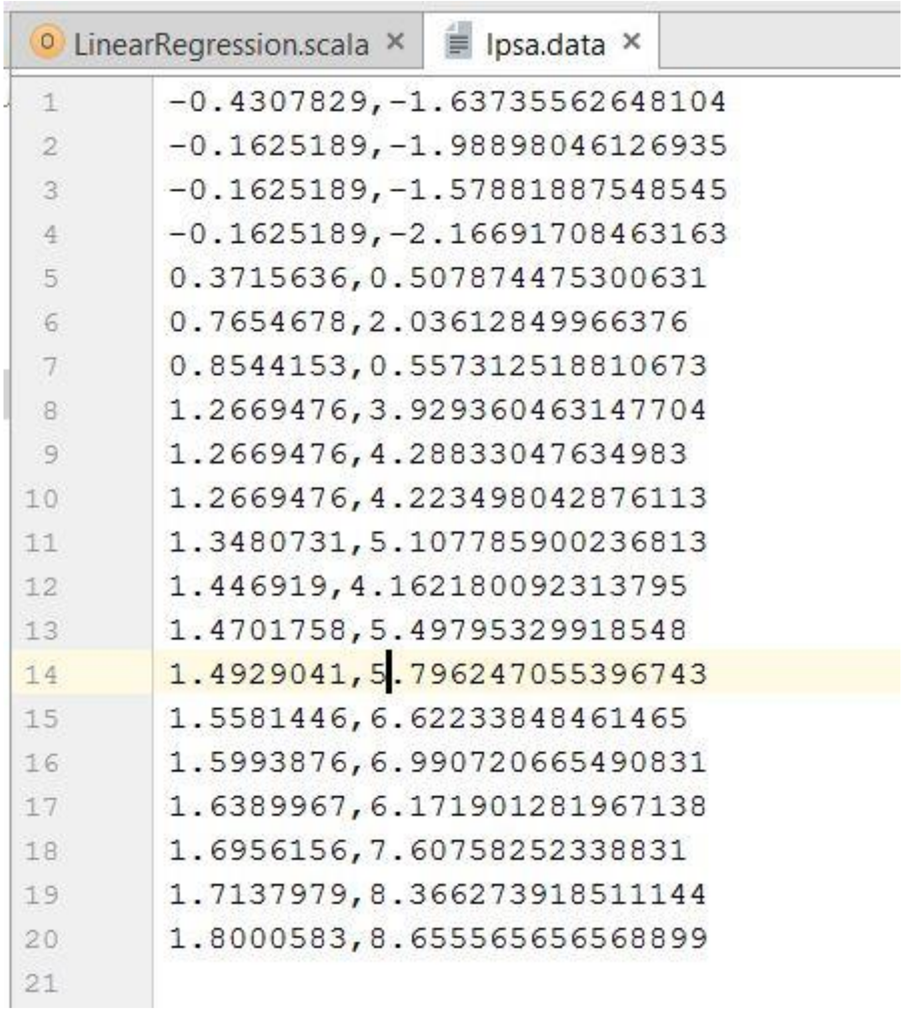
Task1:

1. Implement to build a linear regression model for selected two parameters for chimpanzee's daily movement, activities and interaction. Define your own datasets.

2. Implement K-Means clustering for the clusters of the chimpanzee's activities. Define your own data sets.

- Source file contains the information of two features related to chimpanzees.
- They are position corresponding to the chimpanzee in the zoo and second parameter is the sleeping coefficient.
- Based on out observation, chimpanzees will sleep at some places. So sleeping coefficient will vary based on place coefficient.
- Following two columns of data represent two variables corresponding to chimpanzees. First one is place coefficient and second one is sleeping coefficient

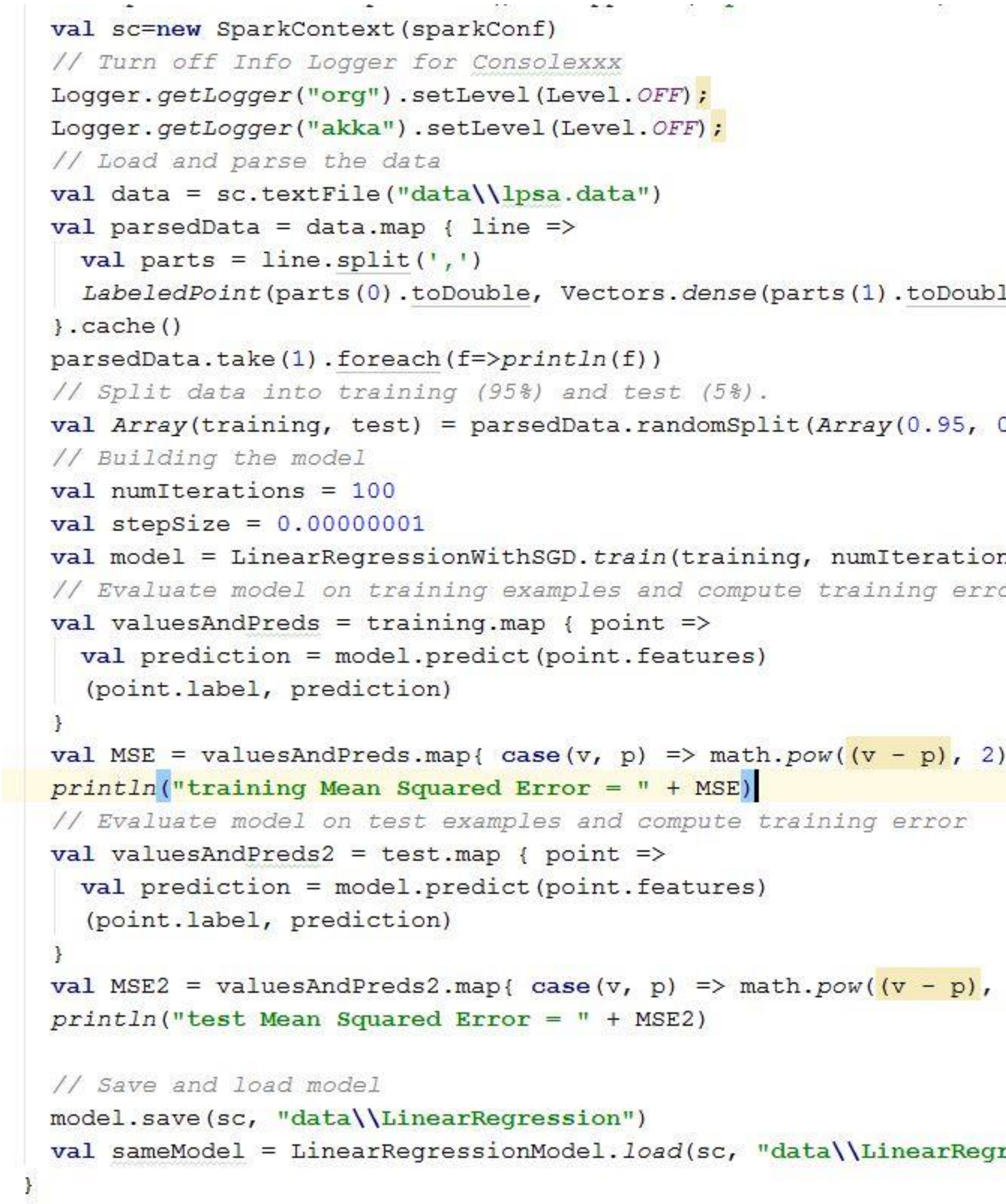
Input file



```
-0.4307829,-1.63735562648104
-0.1625189,-1.98898046126935
-0.1625189,-1.57881887548545
-0.1625189,-2.16691708463163
0.3715636,0.507874475300631
0.7654678,2.03612849966376
0.8544153,0.557312518810673
1.2669476,3.929360463147704
1.2669476,4.28833047634983
1.2669476,4.223498042876113
1.3480731,5.107785900236813
1.446919,4.162180092313795
1.4701758,5.49795329918548
1.4929041,5.796247055396743
1.5581446,6.62233848461465
1.5993876,6.990720665490831
1.6389967,6.171901281967138
1.6956156,7.60758252338831
1.7137979,8.366273918511144
1.8000583,8.655565656568899
```

- Source code of the Linear Regression Model.

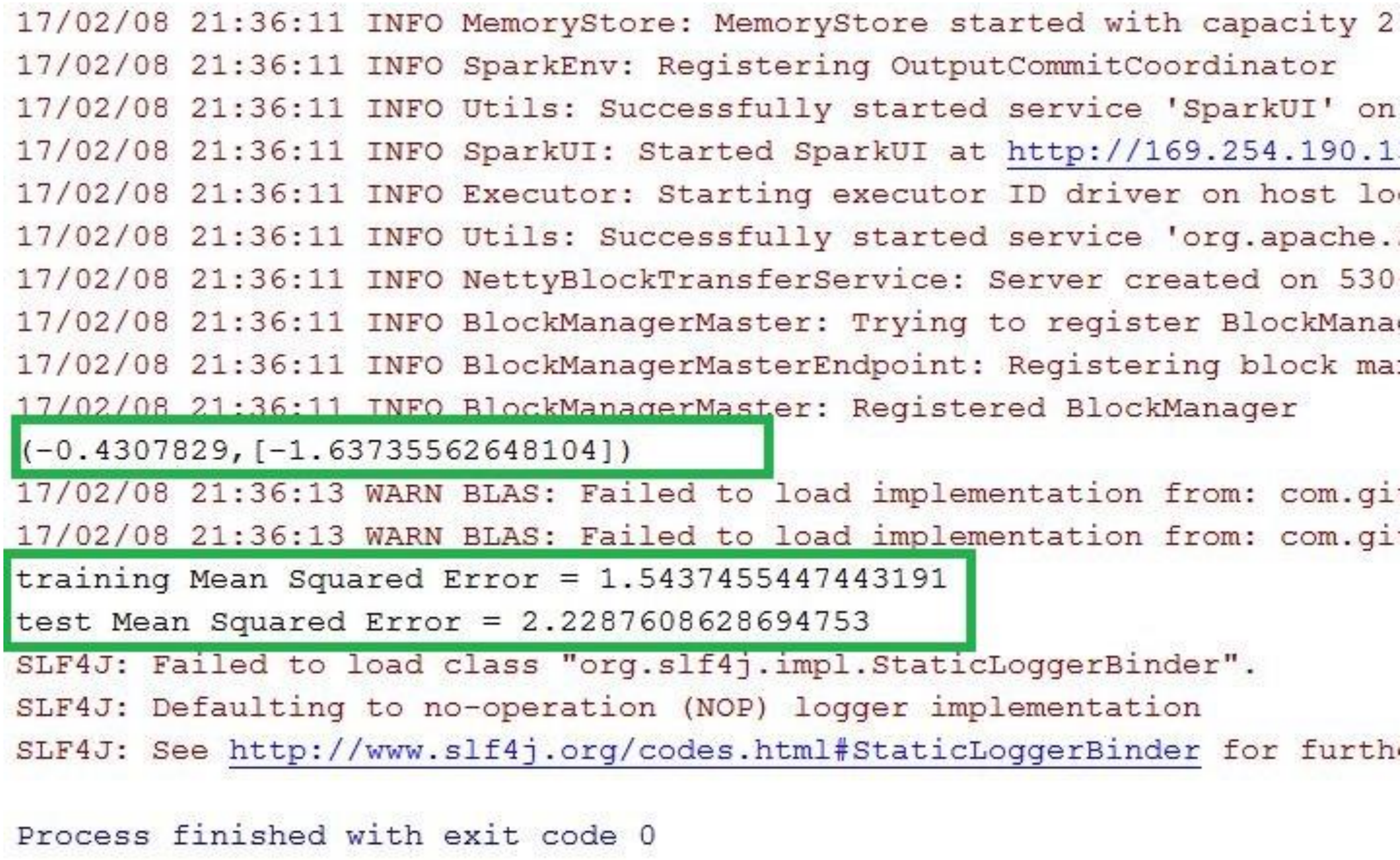
Code



```
  val sc=new SparkContext(sparkConf)
  // Turn off Info Logger for Consolexxx
  Logger.getLogger("org").setLevel(Level.OFF);
  Logger.getLogger("akka").setLevel(Level.OFF);
  // Load and parse the data
  val data = sc.textFile("data\\lpsa.data")
  val parsedData = data.map { line =>
    val parts = line.split(',')
    LabeledPoint(parts(0).toDouble, Vectors.dense(parts(1).toDoubl
  }.cache()
  parsedData.take(1).foreach(f=>println(f))
  // Split data into training (95%) and test (5%).
  val Array(training, test) = parsedData.randomSplit(Array(0.95, 0
  // Building the model
  val numIterations = 100
  val stepSize = 0.00000001
  val model = LinearRegressionWithSGD.train(training, numIteration
  // Evaluate model on training examples and compute training erro
  val valuesAndPreds = training.map { point =>
    val prediction = model.predict(point.features)
    (point.label, prediction)
  }
  val MSE = valuesAndPreds.map{ case(v, p) => math.pow((v - p), 2)
  println("training Mean Squared Error = " + MSE)
  // Evaluate model on test examples and compute training error
  val valuesAndPreds2 = test.map { point =>
    val prediction = model.predict(point.features)
    (point.label, prediction)
  }
  val MSE2 = valuesAndPreds2.map{ case(v, p) => math.pow((v - p),
  println("test Mean Squared Error = " + MSE2)

  // Save and load model
  model.save(sc, "data\\LinearRegression")
  val sameModel = LinearRegressionModel.load(sc, "data\\LinearRegr
}
```

- Following screenshot indicates the output of linear regression model.

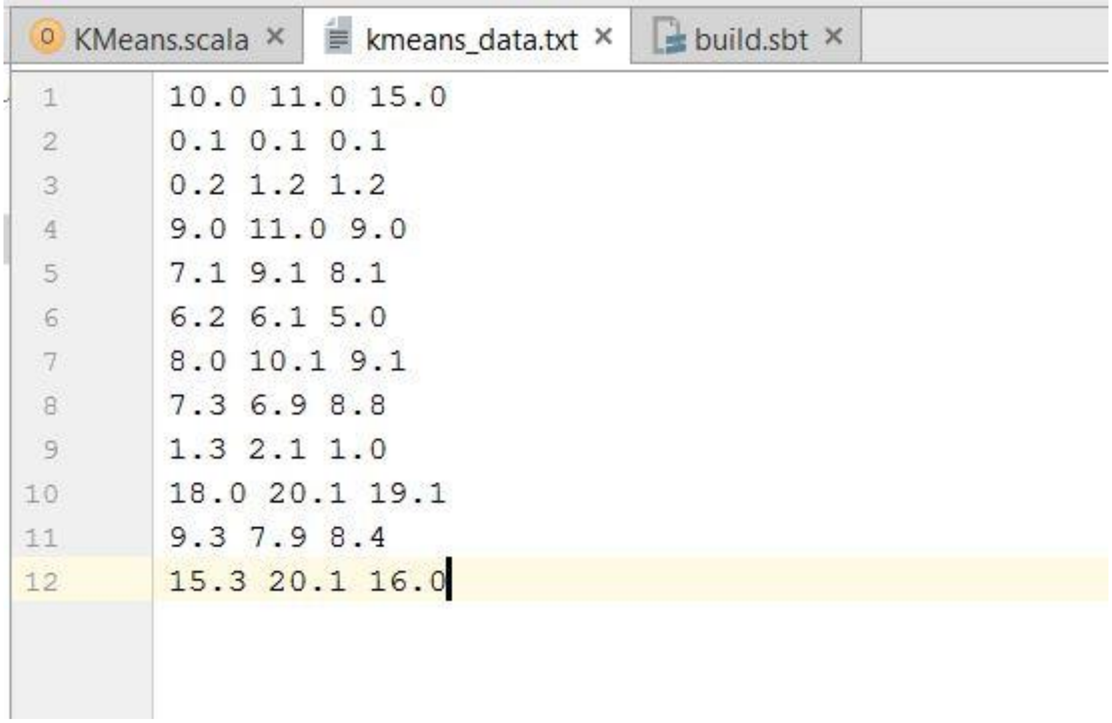
Output file



```
17/02/08 21:36:11 INFO MemoryStore: MemoryStore started with capacity 2
17/02/08 21:36:11 INFO SparkEnv: Registering OutputCommitCoordinator
17/02/08 21:36:11 INFO Utils: Successfully started service 'SparkUI' on
17/02/08 21:36:11 INFO SparkUI: Started SparkUI at http://169.254.190.1
17/02/08 21:36:11 INFO Executor: Starting executor ID driver on host lo
17/02/08 21:36:11 INFO Utils: Successfully started service 'org.apache.
17/02/08 21:36:11 INFO NettyBlockTransferService: Server created on 530
17/02/08 21:36:11 INFO BlockManagerMaster: Trying to register BlockMana
17/02/08 21:36:11 INFO BlockManagerMasterEndpoint: Registering block ma
17/02/08 21:36:11 INFO BlockManagerMaster: Registered BlockManager
(-0.4307829,[-1.63735562648104])
17/02/08 21:36:13 WARN BLAS: Failed to load implementation from: com.gi
17/02/08 21:36:13 WARN BLAS: Failed to load implementation from: com.gi
training Mean Squared Error = 1.5437455447443191
test Mean Squared Error = 2.2287608628694753
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for furth

Process finished with exit code 0
```

- Following screenshot indicates the input file for K-Means Algorithm. This input file specifies the coefficients of sleeping, feeding and playing parameters of chimpanzees.

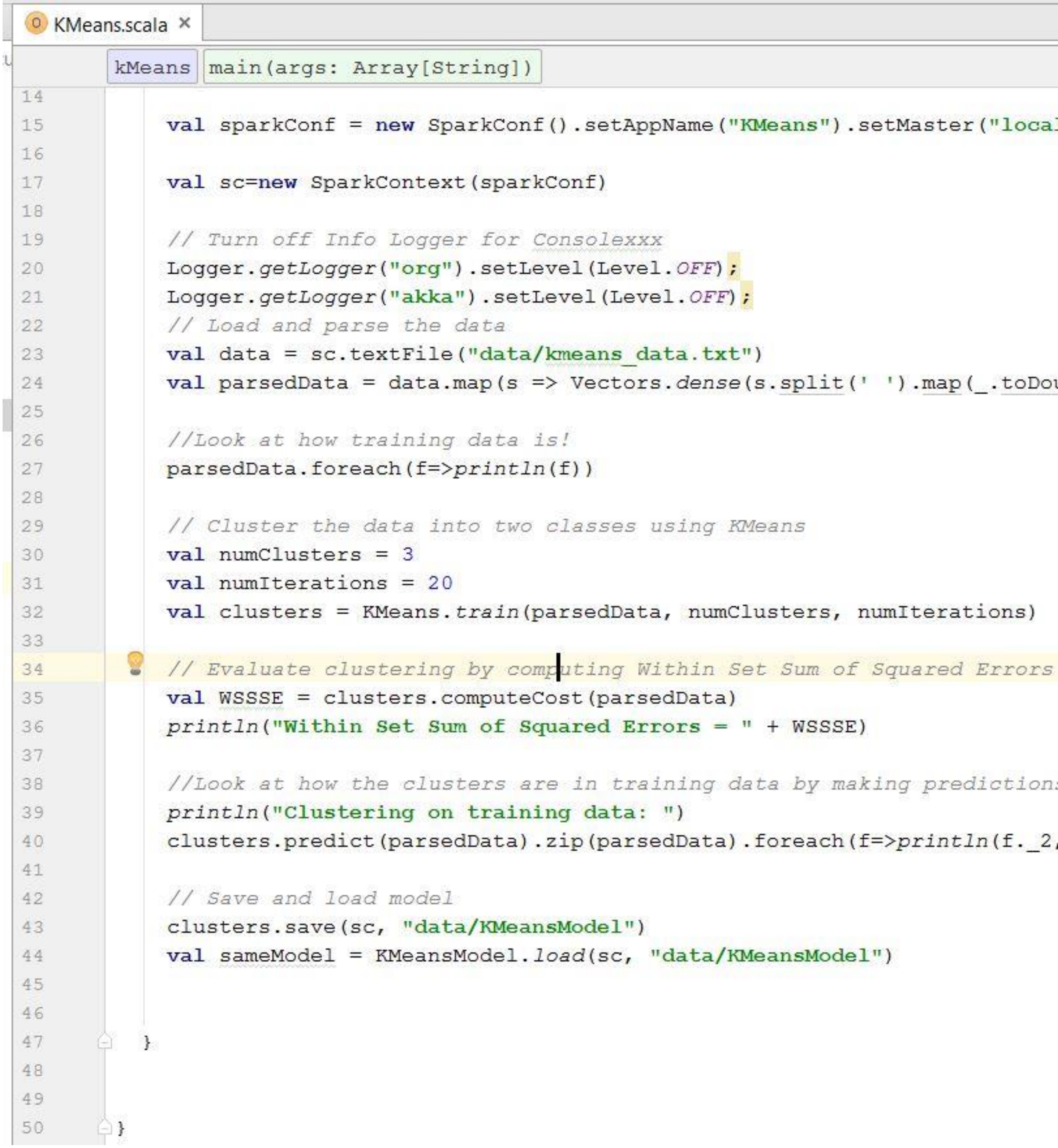
Input file for K-Means Program.



```
10.0 11.0 15.0
0.1 0.1 0.1
0.2 1.2 1.2
9.0 11.0 9.0
7.1 9.1 8.1
6.2 6.1 5.0
8.0 10.1 9.1
7.3 6.9 8.8
1.3 2.1 1.0
18.0 20.1 19.1
9.3 7.9 8.4
15.3 20.1 16.0
```

- Following screenshot corresponds to the K-Means Program and I have taken k as 3. Given inputs are the coefficients of sleeping, feeding and playing parameters of chimpanzees and this K-Means algorithms will classify them to 3 classes.

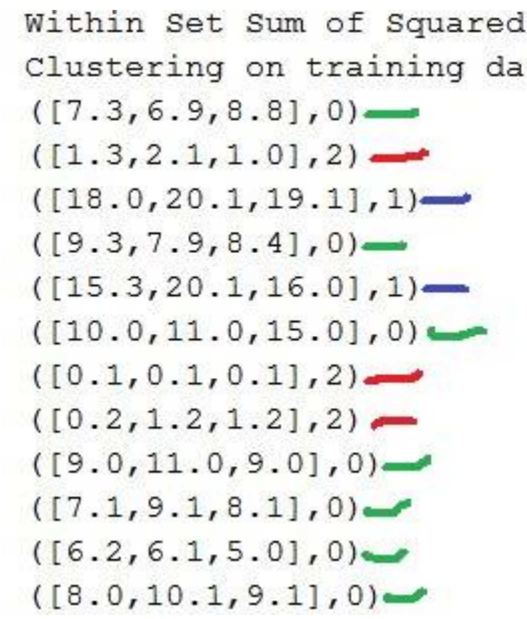
K-Means Source code



```
    val sparkConf = new SparkConf().setAppName("KMeans").setMaster("local

    val sc=new SparkContext(sparkConf)

    // Turn off Info Logger for Consolexxx
    Logger.getLogger("org").setLevel(Level.OFF);
    Logger.getLogger("akka").setLevel(Level.OFF);
    // Load and parse the data
    val data = sc.textFile("data/kmeans_data.txt")
    val parsedData = data.map(s => Vectors.dense(s.split(' ').map(_.toDou

    //Look at how training data is!
    parsedData.foreach(f=>println(f))

    // Cluster the data into two classes using KMeans
    val numClusters = 3
    val numIterations = 20
    val clusters = KMeans.train(parsedData, numClusters, numIterations)

    // Evaluate clustering by computing Within Set Sum of Squared Errors
    val WSSSE = clusters.computeCost(parsedData)
    println("Within Set Sum of Squared Errors = " + WSSSE)

    //Look at how the clusters are in training data by making prediction
    println("Clustering on training data: ")
    clusters.predict(parsedData).zip(parsedData).foreach(f=>println(f._2,

    // Save and load model
    clusters.save(sc, "data/KMeansModel")
    val sameModel = KMeansModel.load(sc, "data/KMeansModel")


  }


}
```

- Following image shows the output from K-Means algorithm. We can identify the data is classified to three classes. Three classes are 0,1 and 2.I have highlighted the different classes with different colors.

K-Means output



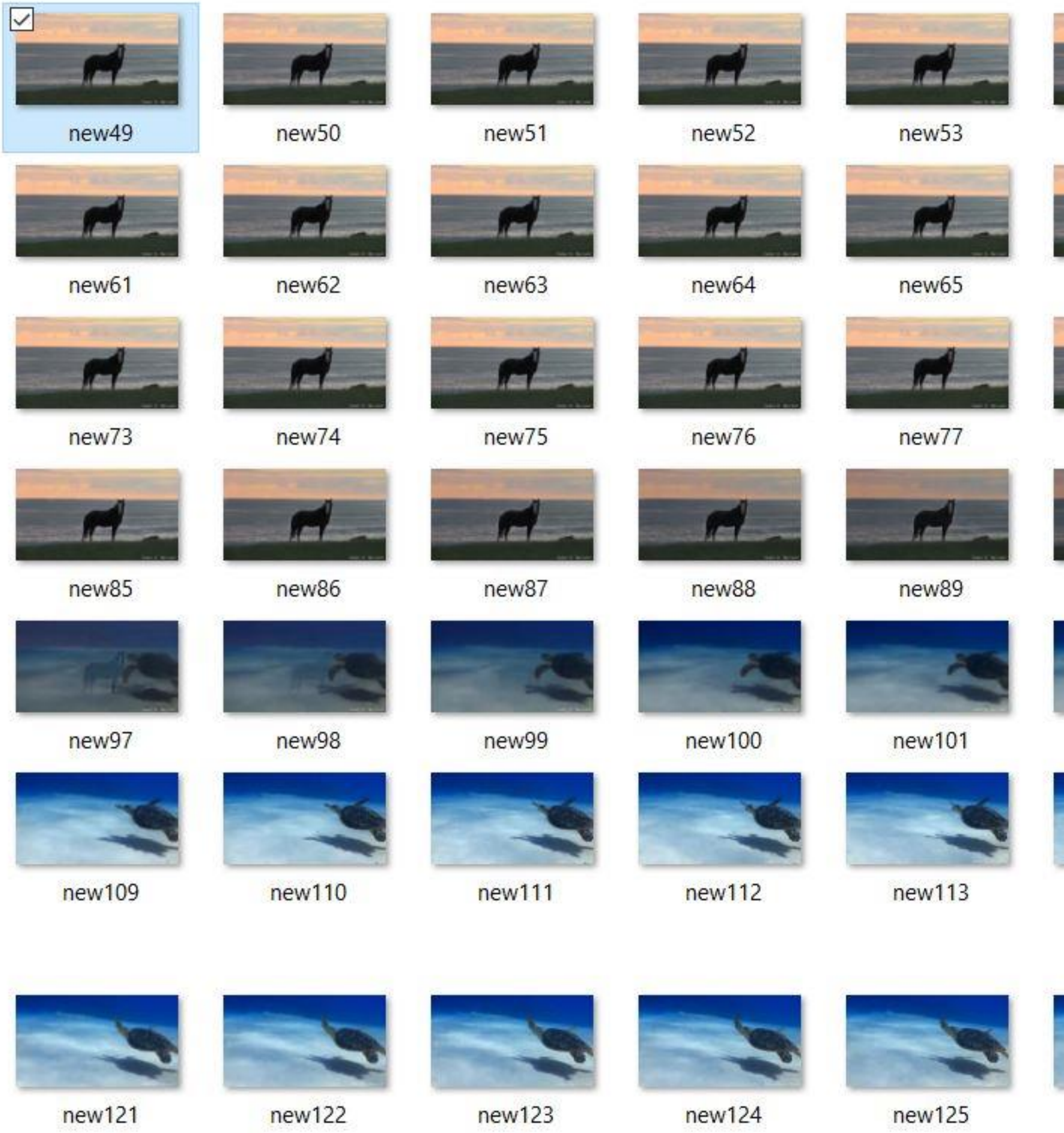
```
Within Set Sum of Squared Errors = 99.47571428571463
Clustering on training data:
([7.3,6.9,8.8],0)
([1.3,2.1,1.0],2)
([18.0,20.1,19.1],1)
([9.3,7.9,8.4],0)
([15.3,20.1,16.0],1)
([10.0,11.0,15.0],0)
([0.1,0.1,0.1],2)
([0.2,1.2,1.2],2)
([9.0,11.0,9.0],0)
([7.1,9.1,8.1],0)
([6.2,6.1,5.0],0)
([8.0,10.1,9.1],0)
```

Task 2

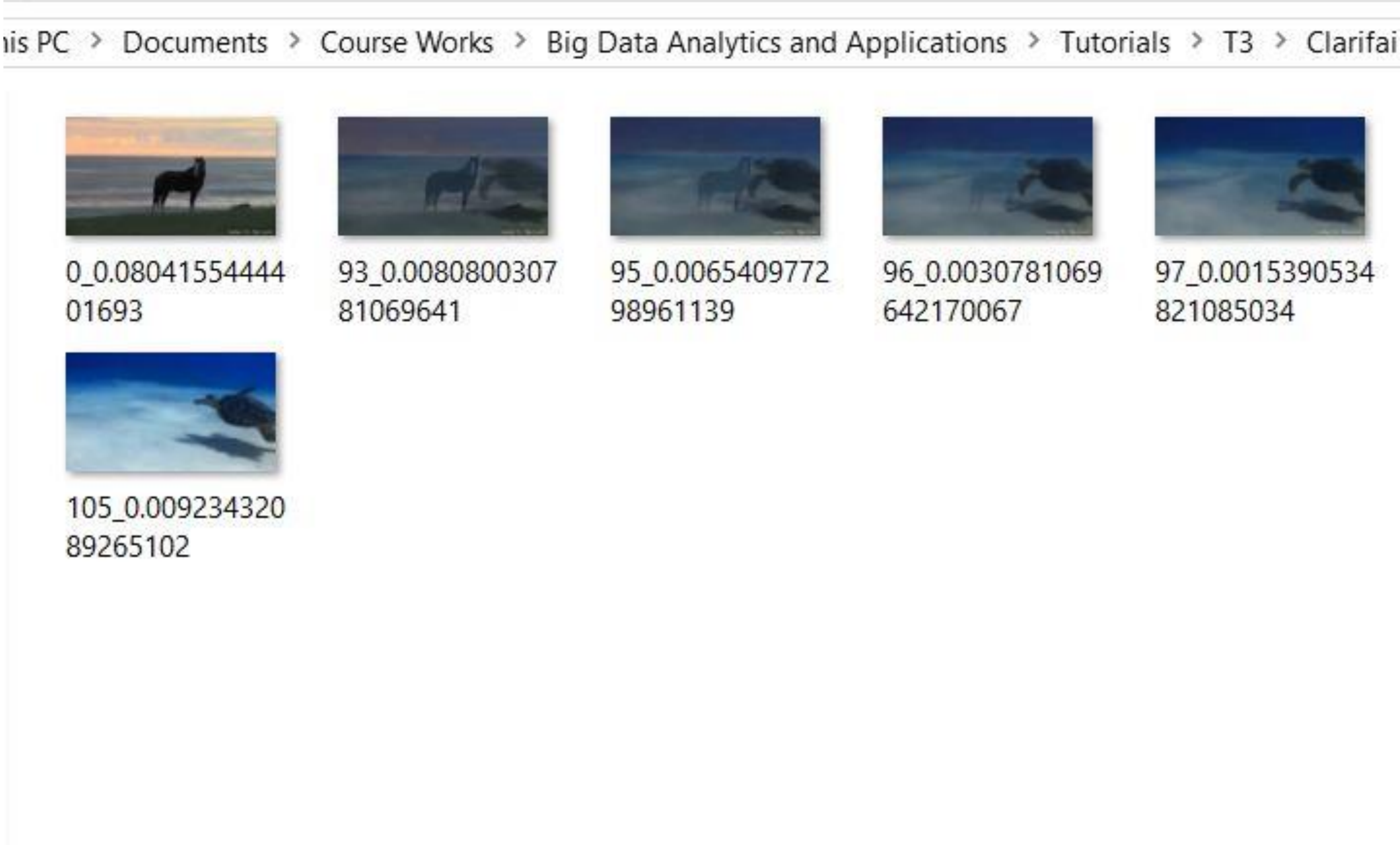
- Build a simple application to give the summary of a video by using Clarifai API. Using OpenImg Library to the key-frame images from the clarifai API.

- I have taken a sample video and i have given this video as input to KeyFrameDetection source file and it divides the frames and identifies major differential frames among all the frames.Following screenshots corresponds to generated frames and generated major frames.

Generated Frames

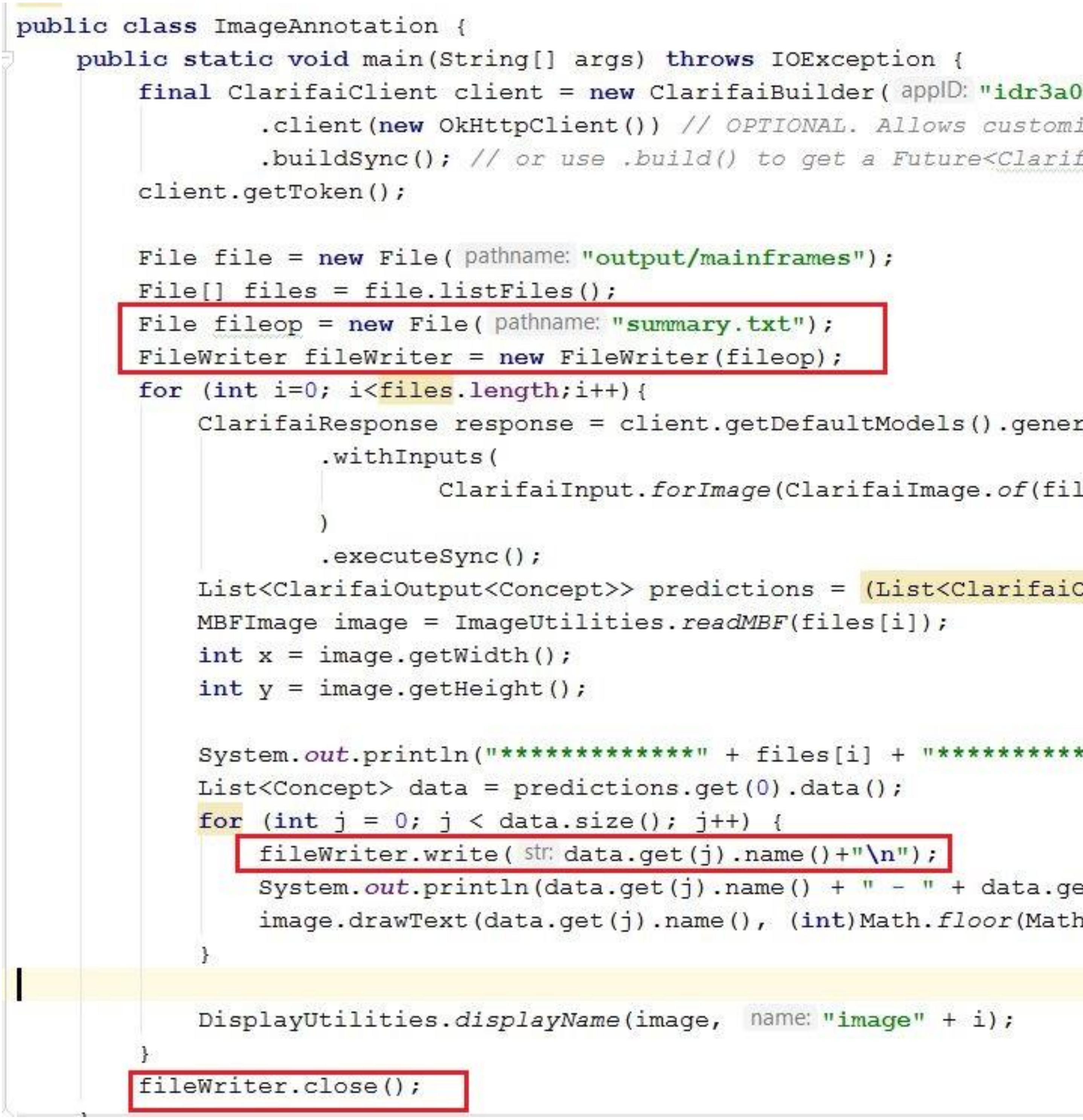


Generated MainFrames



- Generated Mainframes were given to ImageAnnotation code and by using clarifai api,major contents of the mainframes will be identified.Following is the screenshot of the ImageAnnotation code and it contains clarifai api. Highlighted code indicates the process of saving text from clarifai api to a text file to obtain wordcount summary.

ImageAnnotation source code



```
public class ImageAnnotation {
    public static void main(String[] args) throws IOException {
        final ClarifaiClient client = new ClarifaiBuilder( appID: "idr3a0
                .client(new OkHttpClient()) // OPTIONAL. Allows customi
                .buildSync(); // or use .build() to get a Future<Clarif
        client.getToken();

        File file = new File( pathname: "output/mainframes");
        File[] files = file.listFiles();
        File fileop = new File( pathname: "summary.txt");
        FileWriter fileWriter = new FileWriter(fileop);
        for (int i=0; i<files.length;i++){
            ClarifaiResponse response = client.getDefaultModels().gener
                    .withInputs(
                            ClarifaiInput.forImage(ClarifaiImage.of(fil
                    )
                    .executeSync();
            List<ClarifaiOutput<Concept>> predictions = (List<ClarifaiC
            MBFImage image = ImageUtilities.readMBF(files[i]);
            int x = image.getWidth();
            int y = image.getHeight();

            System.out.println("*************" + files[i] + "**********
            List<Concept> data = predictions.get(0).data();
            for (int j = 0; j < data.size(); j++) {
                fileWriter.write( str: data.get(j).name()+"\n");
                System.out.println(data.get(j).name() + " - " + data.ge
                image.drawText(data.get(j).name(), (int)Math.floor(Math
            }

            DisplayUtilities.displayName(image,  name: "image" + i);
        }
        fileWriter.close();
```

- Major contents of the mainframe are identified and displayed on those images in the form of text.

Output from Clarifai api on images

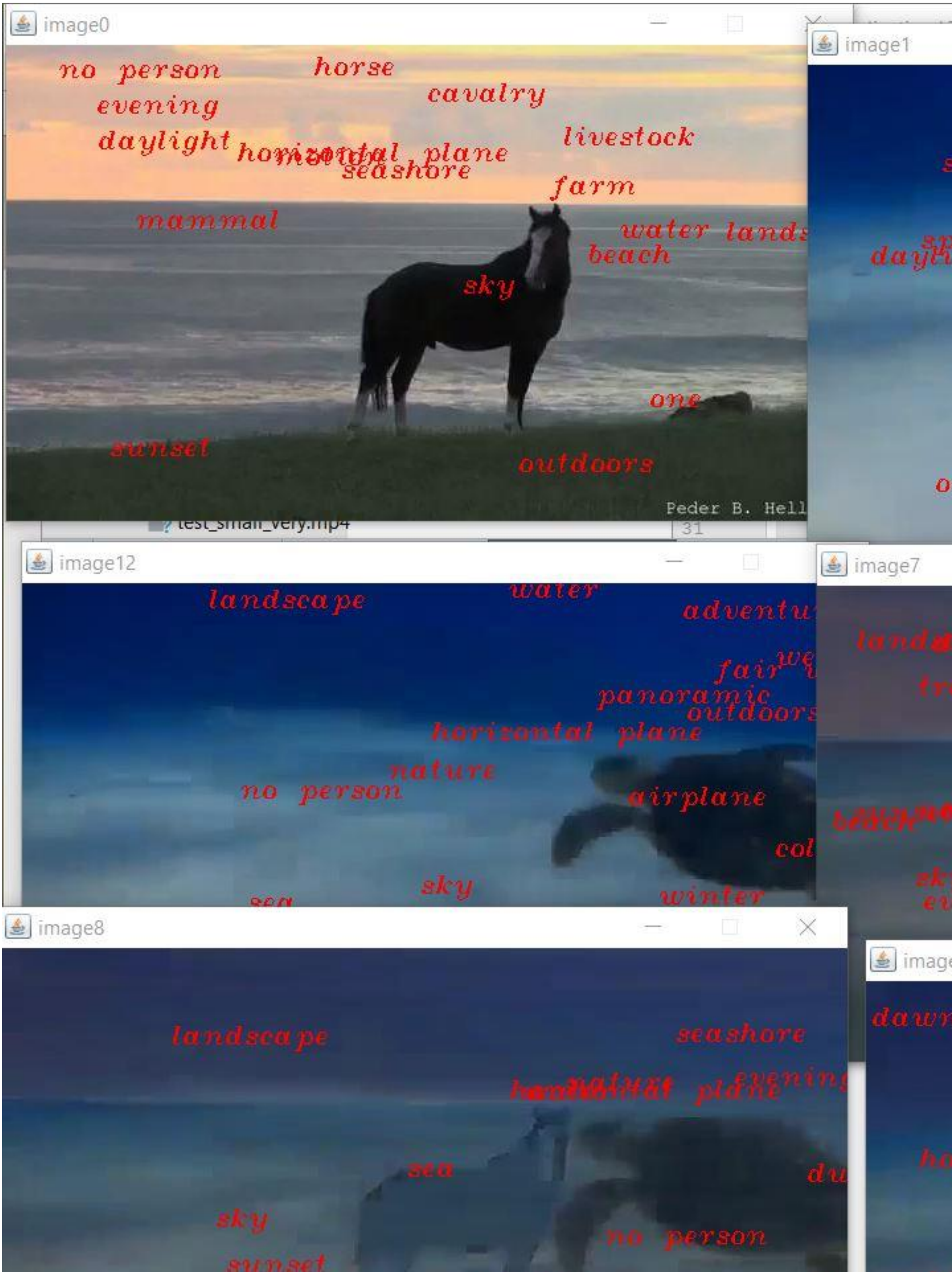
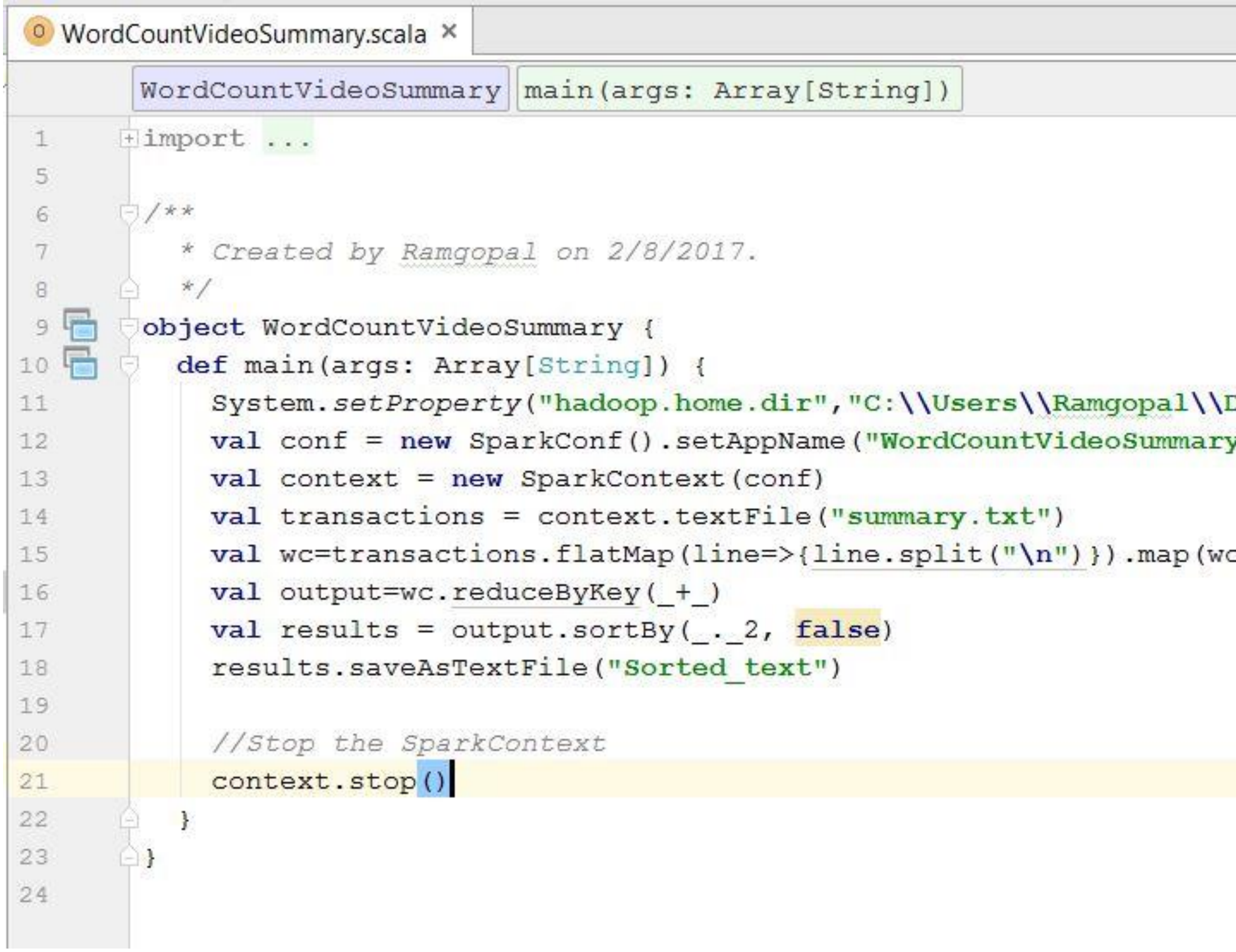


image0
no person
horse
evening
cavalry
daylight
horizontal plane
livestock
seashore
farm
mammal
water
beach
sky
one
sunset
outdoors
Peder B. Hell
test_small_very.mp4
31
image1
image12
landscape
water
adventu
fair
panoramic
outdoors
horizontal plane
nature
no person
airplane
col
sky
winter
sea
image7
image8
landscape
seashore
nature
plane
evening
sea
sky
no person
sunset
dawn

- I have saved the major words from clarifai api result to text file and ran word count program on the text file. This word count program will return the count of each word in descending order.Following image is the source code of wordcount program.

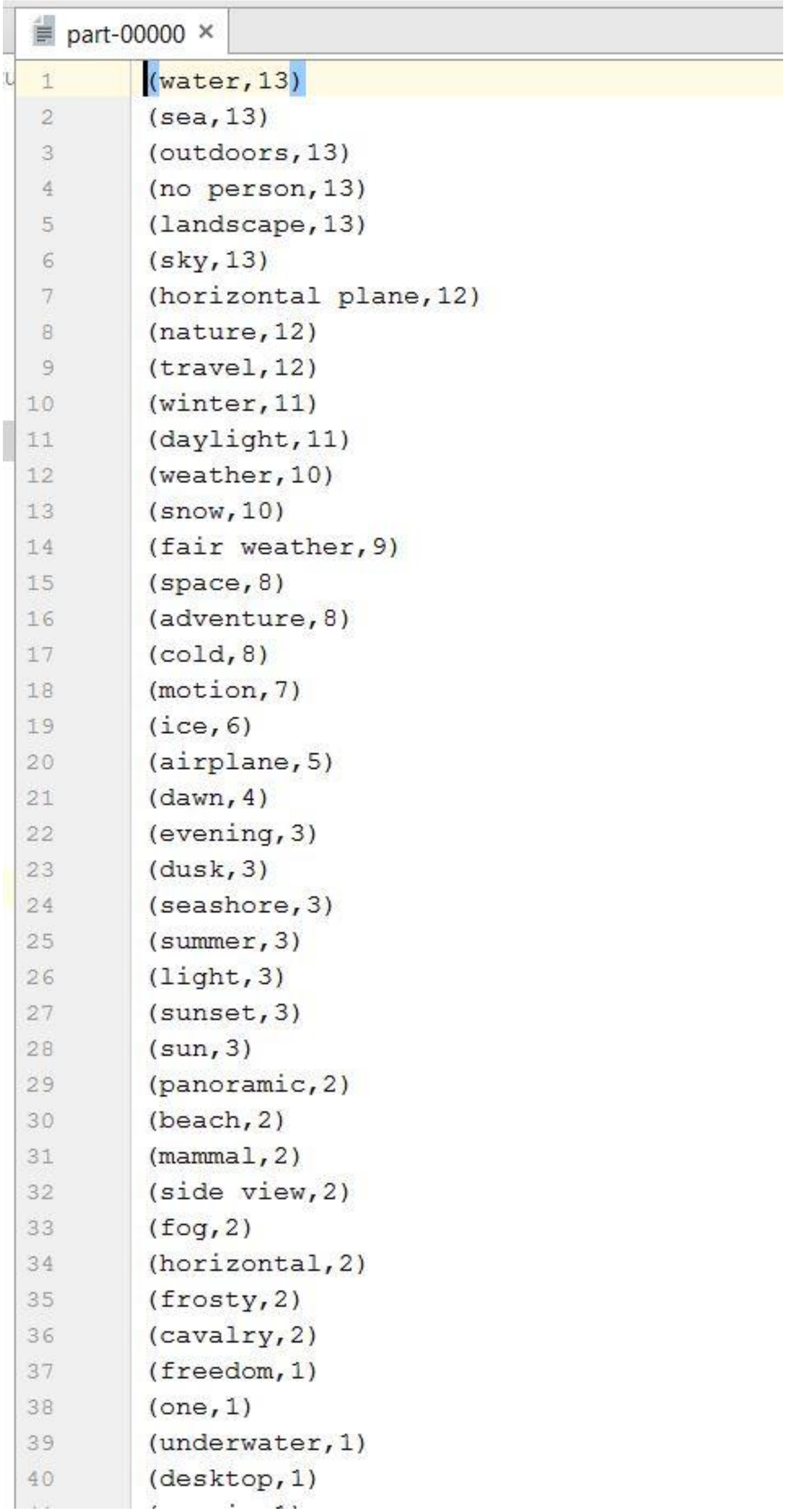
Word count program for video summary



```
import ...

/**
  * Created by Ramgopal on 2/8/2017.
  */
object WordCountVideoSummary {
  def main(args: Array[String]) {
    System.setProperty("hadoop.home.dir","C:\\Users\\Ramgopal\\D
    val conf = new SparkConf().setAppName("WordCountVideoSummary
    val context = new SparkContext(conf)
    val transactions = context.textFile("summary.txt")
    val wc=transactions.flatMap(line=>{line.split("\n")}).map(wc
    val output=wc.reduceByKey(_+_)
    val results = output.sortBy(_._2, false)
    results.saveAsTextFile("Sorted_text")

    //Stop the SparkContext
    context.stop()
  }
}

```

- Word count program was implemented on the results of clarifai api and the following image corresponds to the text associated with video frames. This summary is shown in the descending order of the words for the given video.

Video Summary



```
(water,13)
(sea,13)
(outdoors,13)
(no person,13)
(landscape,13)
(sky,13)
(horizontal plane,12)
(nature,12)
(travel,12)
(winter,11)
(daylight,11)
(weather,10)
(snow,10)
(fair weather,9)
(space,8)
(adventure,8)
(cold,8)
(motion,7)
(ice,6)
(airplane,5)
(dawn,4)
(evening,3)
(dusk,3)
(seashore,3)
(summer,3)
(light,3)
(sunset,3)
(sun,3)
(panoramic,2)
(beach,2)
(mammal,2)
(side view,2)
(fog,2)
(horizontal,2)
(frosty,2)
(cavalry,2)
(freedom,1)
(one,1)
(underwater,1)
(desktop,1)
```