Spark

First configure the basic information for running Spark on Ukko node:

```
conf = (SparkConf()
        .setMaster("spark://ukko007:7077")
        .setAppName(AppName)
        .set("spark.rdd.compress", "true")
        .set("spark.broadcast.compress", "true")
        .set("spark.cores.max", 10)
        .set("spark.local.dir", TMPDIR))
sc = SparkContext(conf = conf)
```

1. First Question

To answer the first question, I first import the data as a RDD object, and then use functions max(), min(), mean(), variance() to calculate the maximum, minimum, average, variance value of the data set.

```
def statistics_summary(fn):
    data = sc.textFile(fn)
    data = data.map(lambda s: float(s))
    summary = []
    summary.append(data.mean())
    summary.append(data.max())
    summary.append(data.min())
    summary.append(data.variance())
    return summary
```

##

The output of first question:

```
Avg: 50.000970
variance: 833.340322
max: 100.000000
min: 0.000000
```

2. Second Question

Import the data set as a RDD object and then map each item in the data set as a key with a value 1. Then invoke function reduceByKey() and assign add as the partition function so that we can calculate how many times the key have appeared in the data set. Sort the returning RDD from reduceByKey() by descending and print the first item. The output is a random key in the RDD with a value 1 which indicates that the keys are not reduced at all. So we can say that this data set is not suitable for finding the mode since float values are quite different from each. Data set with lots of the same repeat values is much better for this task.

```
def find_mode(fn):
    data = sc.textFile(fn)
    data = data.map(lambda s: (float(s), 1))
    mode = data.reduceByKey(add).sortBy(lambda s: s[1], False).first()
    return mode
```

##

The output of second question:
('mode: ', (5.6049308, 6))

3. Third Question
Invoke histogram() function by parsing different bucket as parameters to answer a), b), and c). Since the bucket limits specified in the question, for example (0, 10], (10, 20] … (90, 100], do not coincide with the way histogram divide the buckets, for example [0, 10], [10, 20) … [90, 100]. Therefore I reverse the values to minus and the range to [-100, 0] and invoke histogram(), and calculate the number of keys of 0 minus it from the first bucket range [0, 10], [0, 1], or [0, 0.1].

```
def hist(fn, granularity, length):
    data = sc.textFile(fn)
    # reverse the values to minus and the range to [-100, 0] so that the bucket
    # limits are exact the same(left start point exclusive, right end point included)
    # as specified in the problem, which is the reverse of the bucket divison of
    # histogram() function provided by Spark
    data = data.map(lambda s: -float(s))
    i = -length
    buckets = []
    while (i <= 0):
        buckets.append(i)
        i = i + granularity
    hist = data.histogram(buckets)[1]
    hist.reverse()
    # make items which equal 0 exclusive in the histogram
    zero = data.filter(lambda s: s == 0).count()
    hist[0] = hist[0] - zero

    return hist
```

##
The output of third question: