// Соединение двух файлов в один и парсинг в csv

CSVParser(txtSourseFilePath, paramSourseFilePath)

**INPUT:** txt file with energy data and mod, param file with observation data

**OUTPUT:** csv file with combined dataframe of observations

txtFile = Open(txtSourseFilePath)

paramFile = Open(paramSourseFilePath)

**while !**endOfData**(**txtFile) **or** !endOfData**(**paramFile)

txtFields = readFields(txtFile);

paramFields = readFields(paramFile);

**if** first

txtHeader = txtFields;

paramwHeader = paramFields;

**for** i=0 **to** i < Length(paramFields)

combinedData[paramHeader[i]] = paramFields[i];

**for** i=0 **to** i < Length(txtFields)

combinedData[txtHeader[i]] = txtFields[i];

resultFile.Write(combinedData);

**return** resultFile;

// Поиск повторяющихся строчек

FindRepeatLine(csvSourseFilePath)

**INPUT:** csv file with combined dataframe of observations

**OUTPUT:** csv file with combined dataframe of observations without repeating lines

csvFile = Open(csvSourseFilePath)

**while !**endOfData**(**csvFile)

line = readLine(csvFile);

**if** hashSet.Add(line)

resultFile.WriteLine(line);

**else**

foundRepeatCount += 1;

**return** resultFile;

// Поиск и заполнение пустых полей

FindEmptyField(csvSourseFilePath)

**INPUT:** csv file with combined dataframe of observations

**OUTPUT:** csv file with combined dataframe of observations without empty fields

csvFile = Open(csvSourseFilePath)

**while !**endOfData**(**csvFile)

fields = readFields(csvFile);

**if** **!**first

**for** i= DATA\_OFFSET **to** i < Length(fields)

**if** isNullOrEmpty(fields[i])

value = Parse(fields[i]);

valueList.Add(value);

**else**

valueList.Add(NULL);

lineList.Add(valueList);

**for** column= 0 **to** column < Length(lineList[0])

**for** row= 0 **to** row < Length(lineList)

**if** lineList[row][column] == NULL

**for** i= row **downto** i >= 0

**if** lineList[i][column] != NULL

aboveValue = lineList[i][column];

**break**;

**for** i= row **to** i < Length(lineList)

**if** lineList[i][column] != NULL

belowValue = lineList[i][column];

**break**;

**if** aboveValue == NULL **and** belowValue != NULL

**for** i= row **to** i < Length(lineList)

**if** lineList[i][column] != NULL **and** dataLineList[i][column] != belowValue

aboveValue = lineList[i][column];

**break**;

**if** aboveValue == NULL

aboveValue = belowValue;

**else if** belowValue == NULL **and** aboveValue != NULL

**for** i= row **downto** i >= 0

**if** lineList[i][column] != NULL **and** dataLineList[i][column] != aboveValue

belowValue = lineList[i][column];

**break**;

**if** belowValue == NULL

belowValue = aboveValue;

**else if** belowValue == NULL **and** aboveValue == NULL

**return** error;

lineList[row][column] = (aboveValue + belowValue) / 2;

csvFile = Open(csvSourseFilePath)

**while !**endOfData**(**csvFile)

fields = readFields(csvFile);

**if** lineCount == 0

csvHeader = fields;

**for** i=0 **to** i < Length(fields)

**if** i >= DATA\_OFFSET

**if** lineCount == 0

resultData[csvHeader[i]] = fields[i];

**else**

resultData[csvHeader[i]] = lineList[lineCount-1] [i - DATA\_OFFSET];

**else**

resultData[csvHeader[i]] = fields[i];

resultFile.Write(resultData);

**return** resultFile;

// Поиск аномалий и нормализация

Normalization(csvSourseFilePath)

**INPUT:** csv file with combined dataframe of observations

**OUTPUT:** csv file with combined dataframe of observations without anomaly

csvFile = Open(csvSourseFilePath)

**while !**endOfData**(**csvFile)

fields = readFields(csvFile);

**if** first

csvHeader = fields;

**else**

**for** i= DATA\_OFFSET **to** i < Length(fields)

value = Parse(fields[i]);

valueList.Add(value);

tableValues.Add(valueList);

**for** column= 0 **to** column < Length(tableValues[0])

**for** row= 0 **to** row < Length(tableValues)

values[row] = tableValues[row][column];

meanSquareDeviation = MeanSquareDeviation(values);

**for** row= 0 **to** row < Length(values)

normalizedValues[row][column] = ZNormalize(values[row], values, meanSquareDeviation);

csvFile = Open(csvSourseFilePath)

**while !**endOfData**(**csvFile)

fields = readFields(csvFile);

**if** lineCount == 0

csvHeader = fields;

**for** i=0 **to** i < Length(fields)

**if** i >= DATA\_OFFSET

**if** lineCount == 0 **or** (normalizedValues[lineCount - 1, i - DATA\_OFFSET] > -1 **and** normalizedValues[lineCount - 1, i - DATA\_OFFSET] < 1)

resultData[csvHeader[i]] = fields[i];

**else**

resultData[csvHeader[i]] = “”;

**if** lineCount == 0

resultNormalizedData[csvHeader[i]] = fields[i];

**else**

resultNormalizedData[csvHeader[i]] = normalizedValues[lineCount - 1, i - DATA\_OFFSET];

**else**

resultData[csvHeader[i]] = fields[i];

resultFile.Write(resultData);

resultNormalizedFile.Write(resultNormalizedData);

**return** resultFile;

// Вычисление z-нормализации

ZNormalize(value, valueRange, meanSquareDeviation)

**INPUT:** value, array with this value and mean square deviation

**OUTPUT:** normalized value

**return** (value – AverageValue(valueRange)) / meanSquareDeviation;

// Вычисление среднего квадратического отклонения

MeanSquareDeviation(valueRange)

**INPUT:** array

**OUTPUT:** mean square deviation

average = AverageValue(valueRange);

**for** i=0 **to** i < Length(valueRange)

sum += Pow(valueRange[i] – average, 2);

**return** Sqrt(sum / (Length(valueRange) - 1));

// Вычисление среднего арифметического занчения

AverageValue(valueRange)

**INPUT:** array

**OUTPUT:** average value

**for** i=0 **to** i < Length(valueRange)

sum += valueRange[i];

**return** sum / Length(valueRange);

// Поиск статистических выбросов

Emissions(csvSourseFilePath)

**INPUT:** csv file with combined dataframe of observations

**OUTPUT:** csv file with combined dataframe of observations without anomaly

csvFile = Open(csvSourseFilePath)

**while !**endOfData**(**csvFile)

fields = readFields(csvFile);

**if** !first

**for** i= DATA\_OFFSET **to** i < Length(fields)

value = Parse(fields[i]);

valueList.Add(value);

tableValues.Add(valueList);

**for** column= 0 **to** column < Length(tableValues[0])

**for** row= 0 **to** row < Length(tableValues)

dataRange.Add(tableValues[row][column]);

Sort(dataRange);

quartile1 = dataRange[Length(dataRange) / 4];

quartile3 = dataRange[Length(dataRange) \* 3 / 4];

midQuartile = quartile3 - quartile1;

aboveLine[column] = quartile3 + midQuartile \* 1.5;

belowLine[column] = quartile1 - midQuartile \* 1.5;

csvFile = Open(csvSourseFilePath)

**while !**endOfData**(**csvFile)

fields = readFields(csvFile);

**if** lineCount == 0

csvHeader = fields;

**for** i=0 **to** i < Length(fields)

**if** i >= DATA\_OFFSET

**if** lineCount == 0

resultData[csvHeader[i]] = fields[i];

**else**

**if** tableValues[lineCount - 1][i - DATA\_OFFSET] >= belowLine[i - DATA\_OFFSET] **and** tableValues[lineCount - 1][i - DATA\_OFFSET] <= aboveLine[i - DATA\_OFFSET]

resultData[csvHeader[i]] = tableValues[lineCount-1][i-DATA\_OFFSET];

**else**

resultData[csvHeader[i]] = “”;

**else**

resultData[csvHeader[i]] = fields[i];

resultFile.Write(resultData);

**return** resultFile;

// Разделение файла по фазе

Classificate(csvSourseFilePath)

**INPUT:** csv file with combined dataframe of observations

**OUTPUT:** csv files with combined dataframes of observations separated by phase

csvFile = Open(csvSourseFilePath)

**while !**endOfData**(**csvFile)

fields = readFields(csvFile);

**if** first

endRange = Length(fields) – 1;

**for** i= DATA\_OFFSET **to** i < Length(fields)

value = Parse(fields[i]);

**if** value == beginningValue

startRange = i;

**if** value == finishingValue

endRange = i;

**if** value >= beginningValue && value <= finishingValue

newFields.Add(fields[i]);

csvHeader = newFields

**else**

**for** i= startRange **to** i <= endRange

newFields.Add(fields[i]);

**if** !sections.Contains(fields[0])

sections.Add(newFields[0]);

tempPaths.Add(fields[0], GetTempFileName());

**for** i= 0 **to** i < Length(newFields)

resultData[csvHeader[i]] = csvHeader[i];

**for** i= 0 **to** i < Length(newFields)

resultData[csvHeader[i]] = newFields[i];

tempPath = tempPaths[newFields[0]];

resultFile.Write(resultData, tempPath);

**return** resultFile;