```cpp
void ReadCsv(std::string FileName, std::vector<std::string>& Head, std::vector<std::vector<double>>& Data) {
    Head.clear();
    Data.clear();

    std::ifstream ifs;

    ifs.open(FileName);
    if (!ifs.is_open()) return;

    std::string LineString = "";
    std::string Delimeter = ",";
    bool bHead = true;
    while (getline(ifs, LineString))
    {
        std::vector<double> RowData;

        unsigned int nPos = 0, nFindPos;
        do {
            nFindPos = LineString.find(Delimeter, nPos);
            if (nFindPos == std::string::npos) nFindPos = LineString.length();

            if (bHead)
                Head.push_back(LineString.substr(nPos, nFindPos - nPos));
            else
                RowData.push_back(std::stod(LineString.substr(nPos, nFindPos - nPos)));

            nPos = nFindPos + 1;
        } while (nFindPos < LineString.length());

        if (bHead) {
            bHead = false;
        }
        else
            Data.push_back(RowData);
    }

    ifs.close();
}
```

1. 파일 읽기

csv 파일은 첫 줄은 row의 수치들이 어떤 정보를 나타내는지를 담고 있고, 두 번째 줄부터 수치 정보들을 담고 있다. 따라서 첫 줄은 따로 읽어온다. Head에는 첫 줄의 내용을, Data에는 두 번째 줄 이후의 내용들을 저장해준다.

```cpp
    std::vector<double> mean(m_pRocLayer->m_Head.size() - 1);
    std::vector<double> sd(m_pRocLayer->m_Head.size() - 1);
    for (auto& read : m_pRocLayer->m_ReadData)
        for (int k = 0; k < mean.size(); ++k) {
            mean[k] += read[k];
            sd[k] += read[k] * read[k];
        }

    for (int k = 0; k < mean.size(); k++) {
        mean[k] /= m_pRocLayer->m_ReadData.size();

        sd[k] = sd[k] / m_pRocLayer->m_ReadData.size() - mean[k] * mean[k];
        sd[k] = sqrt(sd[k]);
    }

    int nPrintCnt = 0;
    for (auto& read : m_pRocLayer->m_ReadData) {
        for (auto& column : read) std::cout << column << ", ";
        std::cout << std::endl;

        if (nPrintCnt++ > 10) break;
    }

    for (auto& read : m_pRocLayer->m_ReadData)
        for (int k = 0; k < mean.size(); k++) read[k] = (read[k] - mean[k]) / sd[k];
```

2. Z Normalization

정규화는 원래 데이터에서 평균을 빼고 분산으로 나눠서 데이터의 평균이 0, 분산이 1이 되도록 한다. 키, 체중 등의 단위의 scale이 다를 것으로 예상되는 데이터들은 학습 과정에서 서로 다른 정도의 영향을 끼치게 된다. 즉, 단위에 의해 표현되는 수치 값이 큰 데이터가 학습 과정에 더 큰 영향을 끼친다. 따라서 이에 따른 학습의 편향성을 줄이고자 데이터들의 평균과 분산을 통일시켜 학습에 대한 영향의 편향성을 줄이기 위해 Z Normalization을 시행하게 된다.

```
    int nTrainCnt = 0, nTestCnt = 0;
    for (int i = 0; i < m_ReadData.size(); i++) {
        if (rand() % 5 < 4) {
            Train[i] = true;
            nTrainCnt++;
        }
        else {
            Train[i] = false;
            nTestCnt++;
        }
    }
```

3. 8 : 2로 훈련 자료와 실험 자료 구분

rand() 함수를 이용하여 % 5를 수행하여 나머지 값인 0, 1, 2, 3, 4로 고정한 뒤, 0, 1, 2, 3이면 훈련 자료, 4면 실험 자료로 설정하여 각각의 자료가 훈련 자료인지 실험 자료인지 레이블을 붙여준다.

```
    if (decision > nK / 2.) decision = 1;
    else decision = 0;

    if (decision == 1) {
        int real = (int)m_ReadData[i][m_ReadData[i].size() - 1];
        if (real == 1) nTP++;
        else nFP++;
    }
    else {
        int real = (int)m_ReadData[i][m_ReadData[i].size() - 1];
        if (real == 1) nFN++;
        else nTN++;
    }
}
std::cout << "True Positive Rate : " << (double)nTP / (double)(nTP + nFN) * 100 << "%\n";
std::cout << "False Positive Rate : " << (double)nFP / (double)(nTN + nFP) * 100 << "%\n";
std::cout << "Accuracy : " << (double)(nTP + nTN) / (double)(nTP + nFP + nFN + nTN) * 100 << "%\n";
```

4. k / 2를 임계치로 하여 TPR, FPR, Accuracy 계산

먼저 당뇨병 여부를 k / 2를 임계치로 하여 예측한다. 이후 혼동행렬을 이용하여 TP, FP, FN, TN을 구해준다. 당뇨병이라고 예측했는데 실제로 당뇨병이면 TP, 아니면 FP, 당뇨병이 아니라고 예측했는데 실제로 당뇨병이면 FN, 아니면 TN이다. True Positive Rate는 TP / TP + FN이고, False Positive Rate는 FP / TN + FP이다. Accuracy는 TP + TN / TP + FP + FN + TN이다.

```
srand((unsigned)time(0));
int nK = rand() % 13 + 2;
void CKhuGleRocLayer::ComputePositives()
{
    m_Positive.clear();

    for(int nThreshold = 0 ; nThreshold <= 100 ; nThreshold += 1)
    {
        double TP = 0, FP = 0;
        int nPositiveCnt = 0;
        for(auto &Data : m_Data)
        {
            if(Data.second >= nThreshold/100.)
            {
                if(Data.first == 1)
                    TP++;
                else
                    FP++;
            }

            if(Data.first == 1)
                nPositiveCnt++;
        }

        TP /= nPositiveCnt;
        FP /= (m_Data.size()-nPositiveCnt);

        m_Positive.push_back({TP, FP});
    }
}
```

```
void CKhuGleRocLayer::DrawBackgroundImage()
{
    for(int y = 0 ; y < m_nH ; y++)
        for(int x = 0 ; x < m_nW ; x++)
        {
            m_ImageBgR[y][x] = KgGetRed(m_bgColor);
            m_ImageBgG[y][x] = KgGetGreen(m_bgColor);
            m_ImageBgB[y][x] = KgGetBlue(m_bgColor);
        }

    int xx0, yy0, xx1, yy1;
    bool bFirst = true;
    for(auto &Positive : m_Positive)
    {
        xx1 = (int)(Positive.second * (m_nW-1));
        yy1 = m_nH-(int)(Positive.first * (m_nH-1))-1;

        if(!bFirst)
            CKhuGleSprite::DrawLine(m_ImageBgR, m_ImageBgG, m_ImageBgB, m_nW, m_nH,
                xx0, yy0, xx1, yy1, KG_COLOR_24_RGB(0, 255, 0));

        bFirst = false;

        xx0 = xx1;
        yy0 = yy1;
    }
}
```

5. Positive Sample의 비율을 Score로 고려하여 ROC

K를 [2, 15]의 임의의 값으로 지정한다. rand() 함수를 이용하여 범위 내의 임의의 값을 골라준다. 이후 ComputePositive 함수를 이용하여 Positive Sample들을 m_Positive에 모아주고, DrawBackGroundImage 함수를 이용하여 이를 그래프로 표현한다.