용해탱크 제조DATA 분석

Microsoft
.NET

2팀 노용천 박지선 이지환 추형욱

목차

용해탱크 제조DATA 분석

- 제조데이터를 받아 이해하고 관리한다.
- DB/Api/CSV등 다양한방식으로 관리한다
- 제조데이터를 차트 시각화 및 분석.

데이터관리 /설계

DB/CSV

주제선정/목표설계

데이터값분석및 DB설계

MsSQL 연동하기

Csv파일를 통한 처리

필터링 시스템

Filter/Chart

DB연동후sql문처리

DB연동없이필터

필터링데이터 차트출력

분석

ANALYSIS

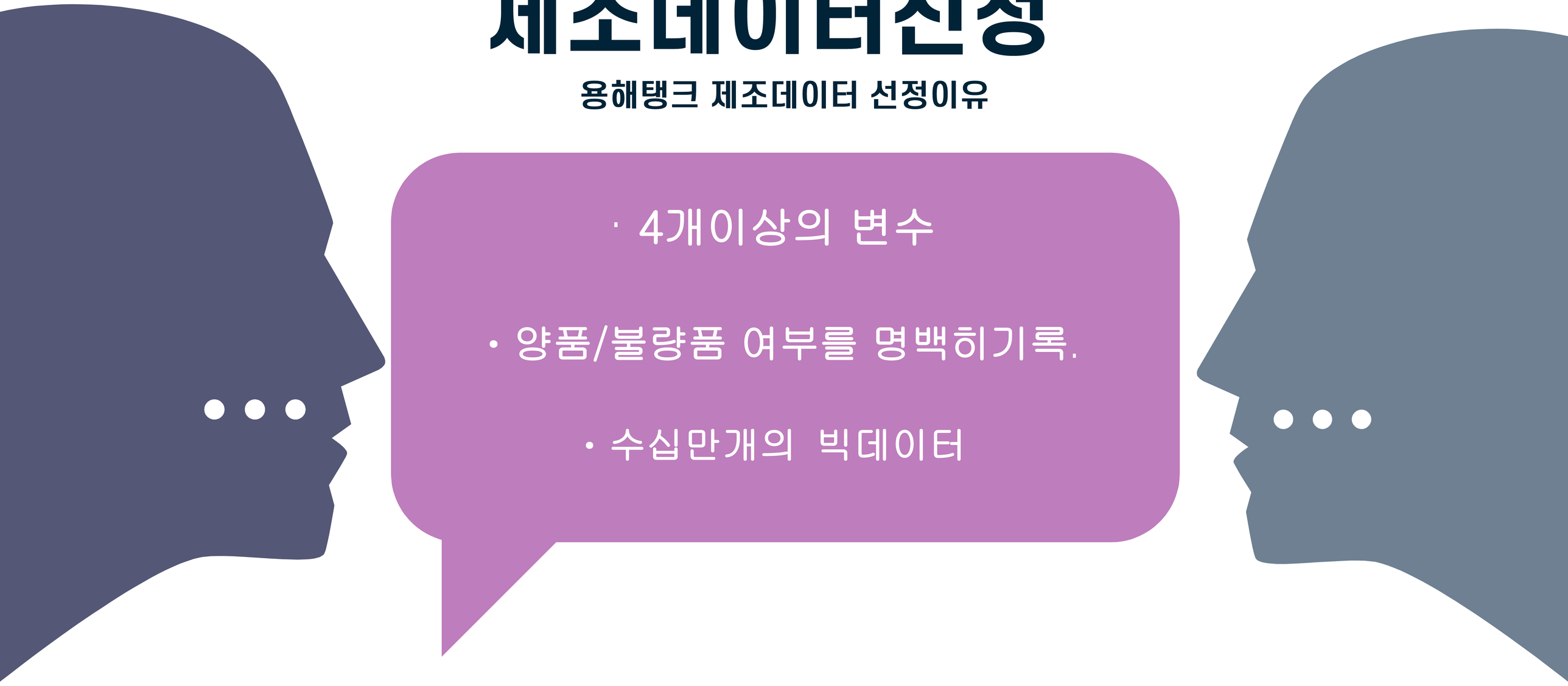
차트를통한분석

추가분석및 개선점

느낀점/Q&A

제조데이터선정

용해탱크 제조데이터 선정이유



목표 설계

필터링 / 차트 형태 선정

DB연동 및 CSV파일 읽기

차트의 시각화 작업

원 재료가 균일하게 혼합 되는 것이 매우 중요하며,
이에 따른 결과론적 품질 분석이 아닌 예견된 불량을
효율적으로 방지하기 위해 분석 데이터를 가져와
세부적 분석을 하여 성공과 실패의 지점을 파악하고,
원인 분석 및 PLC,DBMS등 제어와 수집, 저장 관리로
불량을 예견해 효율적으로 방지하는 것이 궁극적 목표이다.

C#프로젝트

역할 분배

PROJECT

노용천

· DB / 필터링
· CSV파일 읽기
· API 명세서 작업
· 차트 시각화 작업

박지선

· 차트 시각화 구상작업
· PPT제작 및 발표
· 회의록 정리 및 작성
· 요구 사항 정의서 작업

이지환

· DB작업
· 코드수정 및 문제점 보완작업
· 시연영상 제작
· 차트 시각화 세분화 작업

추형욱

· DB연동/Sql문을 통한조회/필터링
· 코드통합및 차트 출력조정
· 차트 시각화 세분화 작업
· ppt초안작성

DB설계

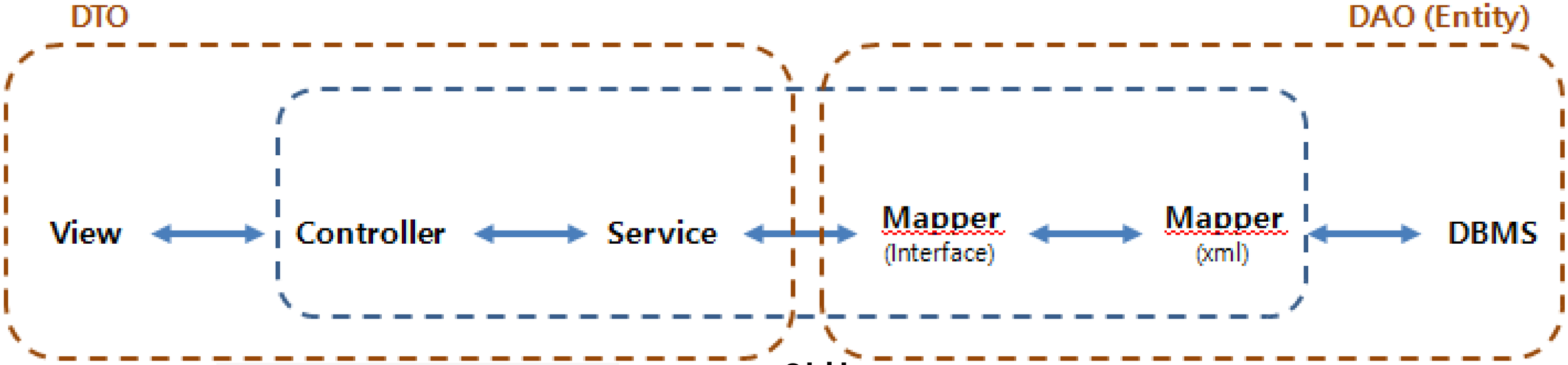
• 데이터 속성정의 표 :

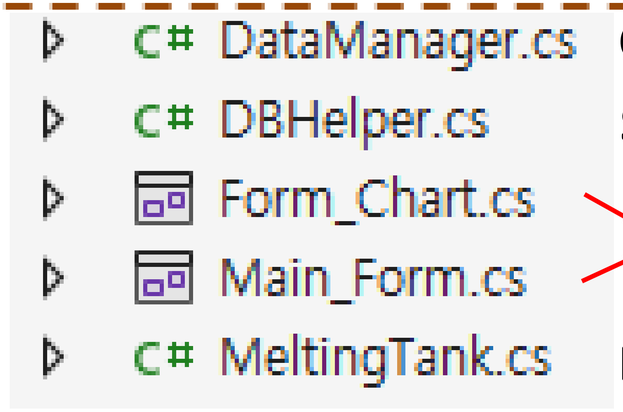
변수명	설 명	데이터 타입
STD_DT	날짜, 시간(YYYY-MM-DD HH:MM:SS)	object
NUM	인덱스	int64
MELT_TEMP	용해 온도	int64
MOTORSPEED	용해 교반속도	int64
MELT_WEIGHT	용해탱크 내용량(중량)	int64
INSP	생산품의 수분함유량(%)	float64
TAG	불량여부	object

* int64 = 정수, object = 문자, float64 = 실수

- 자료를제공하는측의 속성을따름
- 필요시 Table을 미리 생성해놓는다
- 경우에따라 c#에서 drop create를 관리하는경우도 생각해둔다

DB로 데이터 다루기





Controller 역할

Service 역할

View 역할

DTO 역할

CSV직접처리

```
private void button_Csv_Click(object sender, EventArgs e)
{
    using (OpenFileDialog openFileDialog = new OpenFileDialog())
    {
        openFileDialog.Filter = "CSV files (*.csv)|*.csv|All files (*.*)|*.*";

        if (openFileDialog.ShowDialog() == DialogResult.OK)
        {
            string filePath = openFileDialog.FileName;
            records = ReadCsvFile(filePath);
            dataGridView_Main.DataSource = records;

            // CSV 파일 로드 후 ComboBox 업데이트
            InitializeDateComboBoxes();
        }
    }
}
```

Csv파일 경로를 가져온 후
그 파일을 읽어들여 레코드화.
그 레코드로 리스트를 만듦.

```
private static List<Data> ReadCsvFile(string filePath)
{
    using (var reader = new StreamReader(filePath))
    using (var csv = new CsvReader(reader, new CsvConfiguration(CultureInfo.InvariantCulture)))
    {
        var records = csv.GetRecords<Data>();
        return new List<Data>(records);
    }
}
```

C#프로젝트

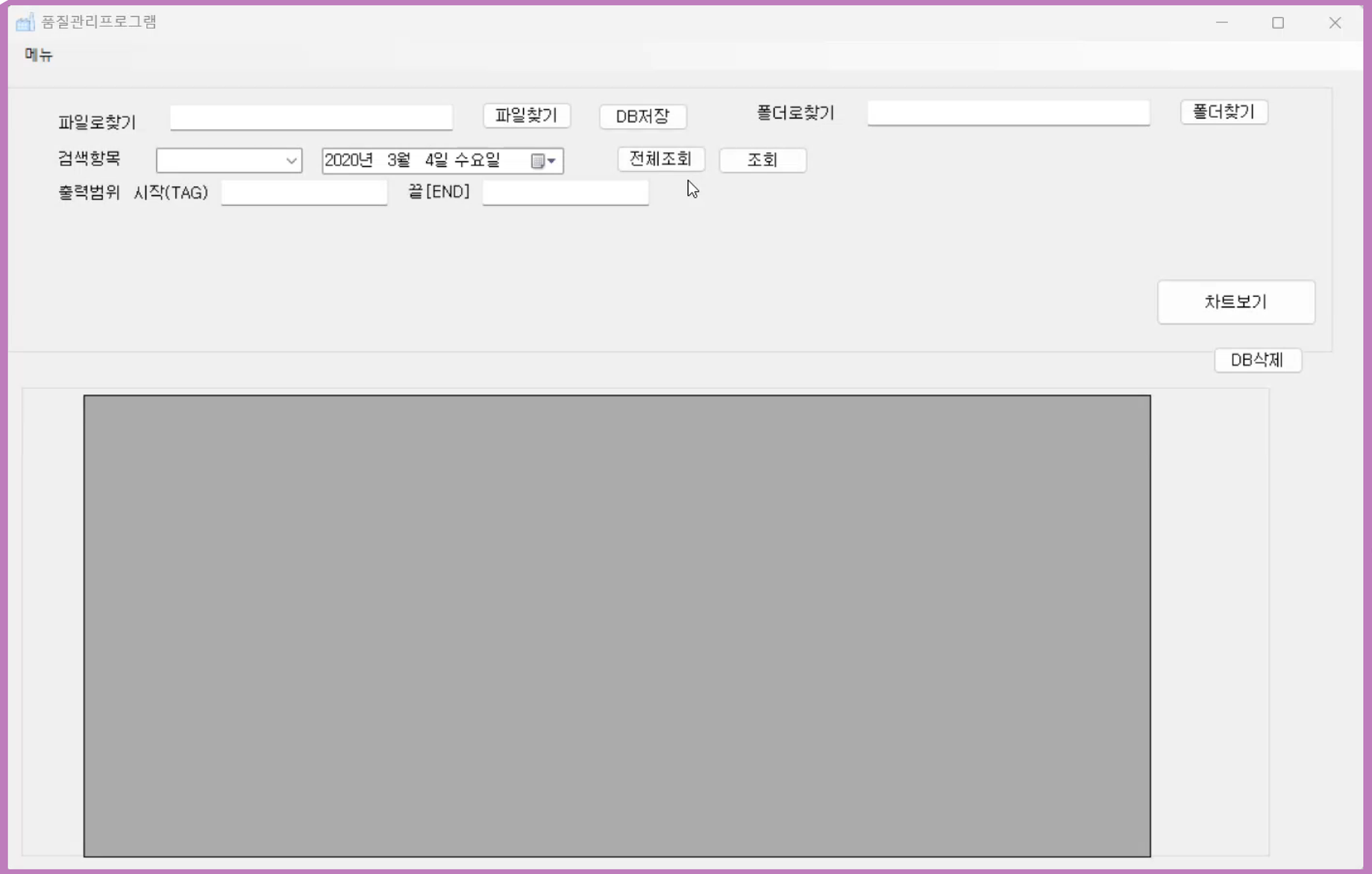
CSV직접처리

Csv파일을 로드 후 데이터상의 날짜를 추출하여 ComboBox에 추가 및 초기에 선택된 날짜 설정

```
private void button_Csv_Click(object sender, EventArgs e)
{
    using (OpenFileDialog openFileDialog = new OpenFileDialog())
    {
        openFileDialog.Filter = "CSV files (*.csv)|*.csv|All files (*.*)|*.*";

        if (openFileDialog.ShowDialog() == DialogResult.OK)
        {
            string filePath = openFileDialog.FileName;
            records = ReadCsvFile(filePath);
            dataGridView_Main.DataSource = records;

            // CSV 파일 로드 후 ComboBox 업데이트
            InitializeDateComboBoxes();
        }
    }
}
```

```
// ComboBox 초기화 메서드
참조 2개
private void InitializeDateComboBoxes()
{
    // records가 로드되어 있다고 가정하고, 그 중에서 날짜 목록을 추출하여 ComboBox에 추가
    if (records != null && records.Count > 0)
    {
        // 중복을 제거한 날짜 목록 추출
        var dateList = records.Select(r => r.STD_DT.Date).Distinct().OrderBy(d => d).ToList();

        // ComboBox에 날짜 목록 추가
        foreach (var date in dateList)
        {
            startDateComboBox.Items.Add(date.ToString("yyyy-MM-dd"));
            endDateComboBox.Items.Add(date.ToString("yyyy-MM-dd"));
        }

        // 첫 번째 아이템 선택 (선택 사항)
        startDateComboBox.SelectedIndex = 0;
        endDateComboBox.SelectedIndex = endDateComboBox.Items.Count - 1;
```

시연 영상


품질관리프로그램
메뉴
파일로찾기
파일찾기
DB저장
폴더로찾기
폴더찾기
검색항목
2020년 3월 4일 수요일
전체조회
조회
출력범위 시작(TAG)
끝[END]
차트보기
DB삭제

자료 필터링

```
if (filteredRecords == null || filteredRecords.Count == 0)
{
    MessageBox.Show("필터링된 데이터가 없습니다.");
    return;
}
```

필터링 안했을때

```
DateTime startDate, endDate;

if (!DateTime.TryParse(startDateComboBox.SelectedItem?.ToString(), out startDate))
{
    MessageBox.Show("유효한 시작 날짜를 선택하세요.");
    return;
}

if (!DateTime.TryParse(endDateComboBox.SelectedItem?.ToString(), out endDate))
{
    MessageBox.Show("유효한 종료 날짜를 선택하세요.");
    return;
}

// 시작 날짜보다 크거나 같고, 종료 날짜보다 작거나 같은 데이터 필터링
filteredRecords = records.Where(record => record.STD_DT.Date >= startDate && record.STD_DT.Date <= endDate).ToList();
dataGridView_Main.DataSource = filteredRecords;
```

날짜 필터링 설정 시 시작 날짜부터 종료 날짜 까지 필터링

필터된 자료 출력 – 원 그래프

```
// 전체 데이터 수
int totalCount = okRecords.Count() + ngRecords.Count();

// 전체 데이터가 없는 경우 처리
if (totalCount == 0)
{
    MessageBox.Show("필터링된 데이터가 없습니다.");
    return;
}

// OK와 NG의 평균 온도에 따른 퍼센트 계산
double okPercent = Math.Round((okAvgTemp / (okAvgTemp + ngAvgTemp)) * 100, 2);
double ngPercent = Math.Round((ngAvgTemp / (okAvgTemp + ngAvgTemp)) * 100, 2);
```

원그래프로 표시할 데이터의 비율을
퍼센테이지로 나타냄

필터된 자료 출력 – 원 그래프

```
// 차트 데이터 설정
chart_OKNG.Series.Clear();
Series series = chart_OKNG.Series.Add("MELT_TEMP");
series.ChartType = SeriesChartType.Pie;
series.Points.AddXY($"OK{okPercent}%", okPercent);
series.Points.AddXY($"NG{ngPercent}%", ngPercent);
```

X(그래프에 띄울 데이터 명)　Y(그래프에 띄울 값 ,부채꼴 넓이)

```
// 차트 타이틀 설정
chart_OKNG.Titles.Clear();
chart_OKNG.Titles.Add($"기간에 따른 양품과 불량품 비율");
```

필터된 자료 출력 – 막대그래프

```
// STD_DT 별 OK와 NG의 개수를 저장할 Dictionary
Dictionary<DateTime, Tuple<int, int>> dateCounts = new Dictionary<DateTime, Tuple<int, int>>();

// 데이터를 STD_DT 별로 그룹화하고 OK와 NG의 개수를 계산
foreach (var record in filteredRecords)
{
    DateTime stdDt = record.STD_DT.Date; // 날짜만 사용하기 위해 시간 부분 제거
    if (!dateCounts.ContainsKey(stdDt))
    {
        dateCounts[stdDt] = new Tuple<int, int>(0, 0);
    }

    if (record.TAG == "OK")
    {
        dateCounts[stdDt] = new Tuple<int, int>(dateCounts[stdDt].Item1 + 1, dateCounts[stdDt].Item2);
    }
    else if (record.TAG == "NG")
    {
        dateCounts[stdDt] = new Tuple<int, int>(dateCounts[stdDt].Item1, dateCounts[stdDt].Item2 + 1);
    }
}
```

필터된 자료 출력 - 산점도

```
// 데이터를 NUM 별로 구분하여 온도 데이터 추가
foreach (var record in filteredRecords)
{
    int num = record.NUM;

    if (record.TAG == "OK")
    {
        if (!tempOKByNum.ContainsKey(num))
        {
            tempOKByNum[num] = new List<double>();
        }
        tempOKByNum[num].Add(record.MELT_TEMP);
    }
    else if (record.TAG == "NG")
    {
        if (!tempNGByNum.ContainsKey(num))
        {
            tempNGByNum[num] = new List<double>();
        }
        tempNGByNum[num].Add(record.MELT_TEMP);
    }
}
```

```
seriesOK.ChartType = SeriesChartType.FastPoint; // 양품은 산점도로 표시
seriesNG.ChartType = SeriesChartType.FastPoint; // 불량은 산점도로 표시
```

```
// 시리즈 추가
chart_Temp.Series.Add(seriesOK);
chart_Temp.Series.Add(seriesNG);

// 축 설정
chart_Temp.ChartAreas[0].AxisX.Interval = 1; // NUM이 정수형이므로 1 간격 설정
chart_Temp.ChartAreas[0].AxisX.Title = "날짜순"; // X 축 제목 설정
chart_Temp.ChartAreas[0].AxisY.Minimum = 0; // Y 축 최소값 설정
chart_Temp.ChartAreas[0].AxisY.Maximum = 800; // Y 축 최대값 설정
```

```
foreach (var num in tempNGByNum.Keys)
{
    if (tempNGByNum.ContainsKey(num))
    {
        foreach (var temp in tempNGByNum[num])
        {
            seriesNG.Points.AddXY(num, temp);
        }
    }
}
```

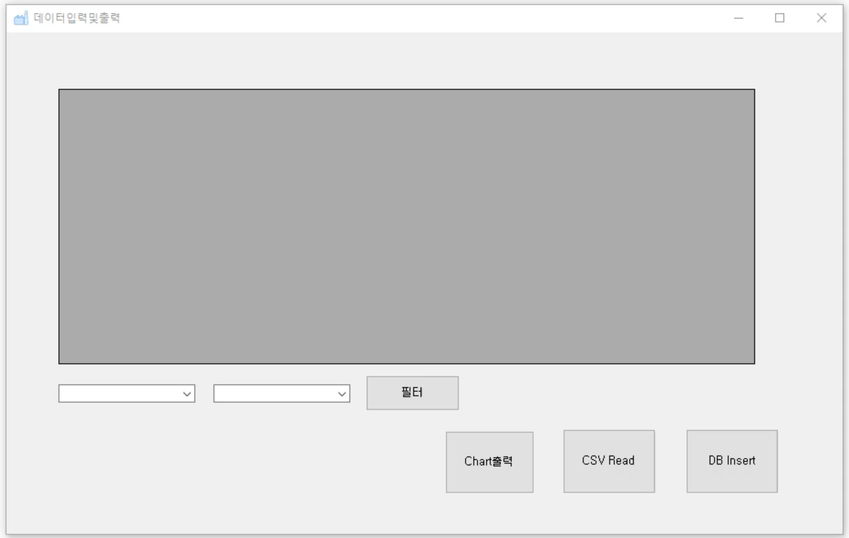
필터된 자료 출력 - 값 계산

```
// 중앙값 계산 함수
참조 1개
private double CalculateMedianInsp(List<float> sortedValues)
{
    int count = sortedValues.Count;
    if (count % 2 == 0)
    {
        // 짝수개인 경우 중앙에 있는 두 값의 평균
        int midIndex = count / 2;
        return (sortedValues[midIndex - 1] + sortedValues[midIndex]) / 2.0;
    }
    else
    {
        // 홀수개인 경우 중앙값
        return sortedValues[count / 2];
    }
}
```

중앙값

C#프로젝트

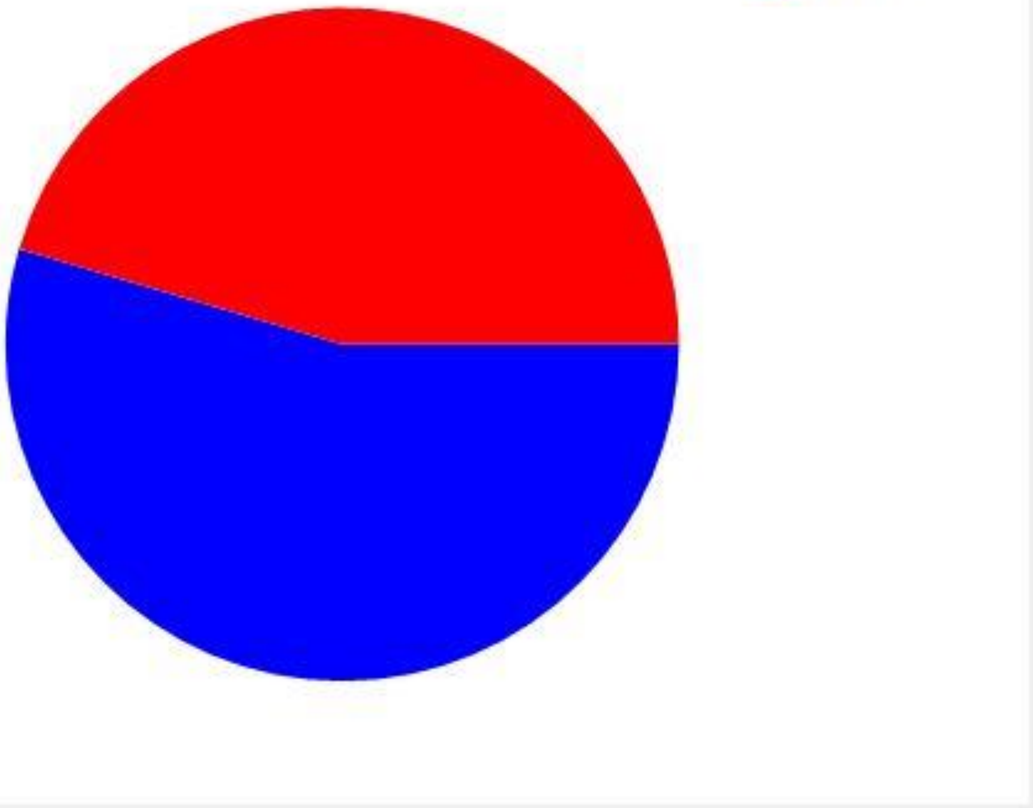
시연영상

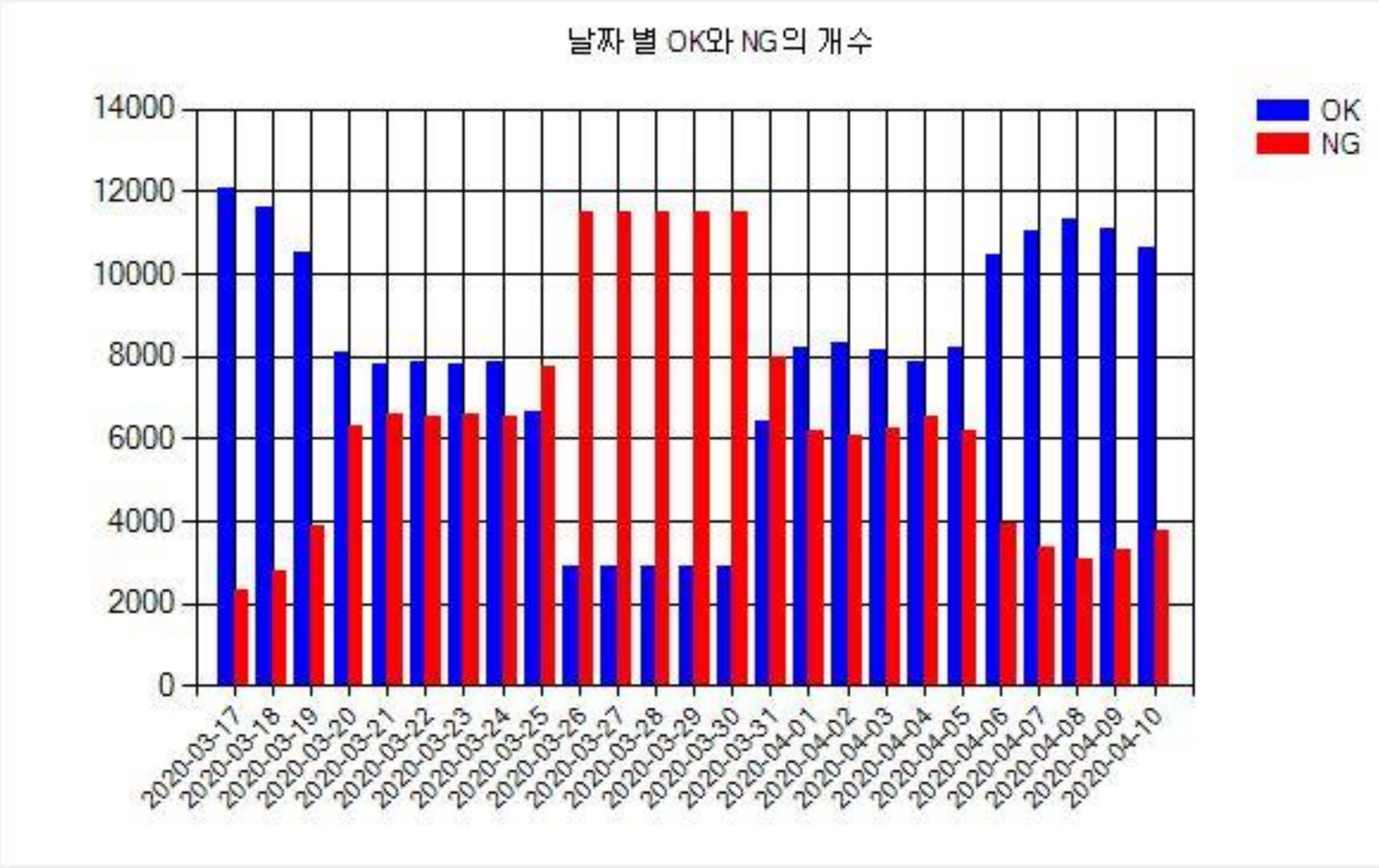


필터링 및 차트출력

Chart 분석

C#프로젝트





데이터분석

차트를 통해 문제 원인을 분석

ANALYSIS

자료 분석

Result

산점도 그래프로 보았을때 전체적으로 온도가 대략 320~480정도일때 불량이 자주 일어 나는걸 알수 있었고
모터스피드의 경우에도 대략 240~270일때 주로 오류가 자주 일어나는걸 알수있었다. 중량의 경우에는 값이 양품과 불량품 균등하게 되어있어서 시각적으로는 구분하기 어려웠으며, 수분함유량의 경우에도 값이 거의 일정하기 때문에 원하는 결과를 도출하는데에는 문제가 있었다
결론적으로 온도가 모터스피드는 불량을 내는데 미세하지만 영향을 끼친다는걸 시각적으로 알수 있었다.

온도의 경우 약 400도 구간에 가장 많이 발생했다.
전처리 공정에서 분말 및 액상 원재료를 정제수에 용해/ 혼합후, 후공정에서 다시 분말화 하기 때문에 원 재료가 균일하게 혼합 되는 것이 매우 중요하다.
원 재료가 온도 변화 및 용해 되지 못한 알갱이등에 균일하지 못해 불량이 발생한다.
또한 한가지 공정만으로는 불량 여부가 판단하기 어려운 부분이 있어 종합적으로 봤을 경우 작업자의 숙련도에 따른 영향 또한 불량에 많은 영향이 있을 것으로 판단 된다.

NG가 많은날에는 모터스피드가 약 250보다 낮은 경향
중량은 전체 날짜별, NG다수의 범위내에서 비슷한 값을 보여서 비교가 미미
온도는 약 320~560 구간에서 NG가 다량 발생
수분함유량은 전체구간에서 다 비슷하므로
비교하는데에 영향이 적음

온도에서는 특정구간에서 불량품이 많이 발생해서 유의미한 데이터로 보이며 모터 속도도 특정 구간에 불량품이 몰려있어 참고 사항 정도는 될듯하다
무게는 상당히 불량품이 넓은 범위로 퍼져있어 유의미한 결과라고 생각 되지 않는다.
수분의경우는 전구간에서 차이가 없었고, 주어진 데이터만으로 판단 할 때는 온도와 일부 모터스피드가 애매한 중간구간에서 발생했다. 실질적으로 영향을 끼친 데이터는 온도뿐 이라고추정 된다.

느낀점

PROJECT

노용천

데이터와 주제에 따라 데이터
에 맞는 시각화 표현이 어려웠다
데이터활용의 방향성을 잘 잡아서
해야 한다는걸 알수 있었다.

박지선

한 주제의 제조데이터를 가지고
어떻게 표현할지가 힘들었다.
다양한 형식의 차트와
그걸 시각화하기 위한 코드의
방향성을 잡는 걸 깨달았다.

이지환

데이터의 가져오고 하는것을 다른 프로젝트
때에도 시도해보았었지만 실제 공장의
데이터값을 가지고 분석하는것은 또다른
문제라는것을 알수있는 기회였다

추형욱

대량의 데이터를 관리하는것이
쉽지않다는것을 느꼈고 가져온
데이터와 실제 결과물간의
연관성을 찾아내기도힘들다는것을
알았다 빅데이터의중요성에대해
다시한번느낄수있었다

Thank You 감사합니다

Microsoft
.NET