report

March 24, 2025

[1]:
```
#     1:
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
from pathlib import Path
from datetime import timedelta, datetime
from IPython.display import display, HTML

#
input_dir = Path('final_data')
moistures_file = input_dir / 'moistures_temps_mass.csv'
settings_file = input_dir / 'settings_optimized.csv'
alarms_segments_file = input_dir / 'alarms_segments.csv'

#
df = pd.read_csv(moistures_file)
settings_df = pd.read_csv(settings_file)
alarms_segments = pd.read_csv(alarms_segments_file)

#        DateTime          +3
df['DateTime'] = pd.to_datetime(df['Date'] + ' ' + df['Time'], format='%d-%m-%Y␣
 ↪%H:%M:%S') + timedelta(hours=3)
settings_df['DateTime'] = pd.to_datetime(settings_df['Date'] + ' ' +␣
 ↪settings_df['Time'], format='%d-%m-%Y %H:%M:%S') + timedelta(hours=3)
alarms_segments['Start'] = pd.to_datetime(alarms_segments['Start'])
alarms_segments['End'] = pd.to_datetime(alarms_segments['End'])

#
df = df.merge(settings_df[['DateTime', 'DROPS_SET_TIMER', 'SET_BURNERS_TEMP']],␣
 ↪on='DateTime', how='left')

#
def classify_moisture(row):
    grain_type = row['GRAIN_TYPE']
    moisture = row['perten_dry_Moisture']
    if pd.isna(grain_type) or pd.isna(moisture):
        return 'unknown'
```

```
    elif grain_type.lower() == 'raps':
        return 'overdry' if moisture < 8 else 'normal' if 8 <= moisture <= 9.5␣
 ↪else 'wet'
    else:
        return 'overdry' if moisture < 12.9 else 'normal' if 12.9 <= moisture␣
 ↪<= 14.5 else 'wet'

df['Moisture_Status'] = df.apply(classify_moisture, axis=1)

#
df['Moisture_Diff'] = df['perten_wet_Moisture'] - df['perten_dry_Moisture']
df['Incoming_Wet'] = df['Moisture_Diff'].apply(lambda x: x if x > 1 else None)
avg_incoming_wet = df[df['Incoming_Wet'].notna()].
 ↪groupby('GRAIN_TYPE')['perten_wet_Moisture'].agg(['mean', 'min']).
 ↪reset_index()
returned_mass = df[df['Moisture_Diff'].notna() & (df['Moisture_Diff'] <=␣
 ↪1)]['dry_mass'].sum()

#
def get_shift(datetime_obj):
    hour = datetime_obj.hour
    date = datetime_obj.date()
    if 8 <= hour < 20:
        shift_start = datetime(date.year, date.month, date.day, 8, 0)
        return shift_start, 'Day'
    else:
        if hour < 8:
            shift_start = datetime(date.year, date.month, date.day, 0, 0) -␣
 ↪timedelta(hours=4)
        else:
            shift_start = datetime(date.year, date.month, date.day, 20, 0)
        return shift_start, 'Night'

df['Shift_Start'], df['Shift_Type'] = zip(*df['DateTime'].apply(get_shift))
shift_productivity = df.groupby(['Shift_Start', 'Shift_Type'])['dry_mass'].
 ↪sum().reset_index()
shift_productivity['Shift_Start'] = pd.
 ↪to_datetime(shift_productivity['Shift_Start'])

#
df['Time_Diff'] = df['DateTime'].shift(-1) - df['DateTime']
df['Time_Diff'] = df['Time_Diff'].fillna(timedelta(seconds=0)).dt.
 ↪total_seconds() / 3600
mode_times = df.groupby('mode')['Time_Diff'].sum().reset_index()
total_work_time = mode_times[mode_times['mode'] != 'STOP']['Time_Diff'].sum()
```

```
#         :
dry_mass_stats = df.groupby('GRAIN_TYPE').agg({'dry_mass': 'sum',␣
 ↪'perten_dry_Moisture': ['min', 'mean']}).reset_index()
dry_mass_stats.columns = ['GRAIN_TYPE', 'Total_Dry_Mass', 'Min_Dry_Moisture',␣
 ↪'Mean_Dry_Moisture']
moisture_by_grain = df.groupby(['GRAIN_TYPE', 'Moisture_Status'])['dry_mass'].
 ↪sum().reset_index()

#
grain_types = df['GRAIN_TYPE'].dropna().unique()
correlations = {}
for grain in grain_types:
    grain_df = df[df['GRAIN_TYPE'] == grain].dropna(subset=['SET_BURNERS_TEMP',␣
 ↪'DROPS_SET_TIMER', 'ACTUAL_BURNERS_TEMP', 'perten_dry_Moisture', 'dry_mass'])
    if len(grain_df) < 2 or grain_df['SET_BURNERS_TEMP'].std() == 0 or␣
 ↪grain_df['DROPS_SET_TIMER'].std() == 0:
        correlations[grain] = {k: float('nan') for k in ['temp_moisture',␣
 ↪'timer_moisture', 'actual_temp_moisture', 'timer_mass', 'temp_mass']}
    else:
        correlations[grain] = {
            'temp_moisture': grain_df['SET_BURNERS_TEMP'].
 ↪corr(grain_df['perten_dry_Moisture']),
            'timer_moisture': grain_df['DROPS_SET_TIMER'].
 ↪corr(grain_df['perten_dry_Moisture']),
            'actual_temp_moisture': grain_df['ACTUAL_BURNERS_TEMP'].
 ↪corr(grain_df['perten_dry_Moisture']),
            'timer_mass': grain_df['DROPS_SET_TIMER'].
 ↪corr(grain_df['dry_mass']),
            'temp_mass': grain_df['SET_BURNERS_TEMP'].corr(grain_df['dry_mass'])
        }

#     (                PDF)
texts = {
    'en': {
        'title': "Grainstate: Grain Drying Analytics",
        'intro': "This report provides detailed analytics for grain drying␣
 ↪performance over the period {} - {} (Local Time, UTC+3), collected using␣
 ↪Perten AM5200A and Grainstate systems.",
        'summary': "Key Results",
        'total_mass': "Total dried mass: {:.0f} kg",
        'drops': "Number of drops: {}",
        'returned': "Returned wet grain: {:.0f} kg",
        'wet_moisture': "Incoming moisture by grain type (wet > dry by 1%):",
        'mode_pie': "Operating Modes (hours)",
        'mass_pie': "Dried Mass by Grain Type (%)",
        'moisture_bar': "Dried Mass by Moisture Status (kg)",
```

```
        'shift_line': "Dried Mass by Shift (kg, Day: 8:00-20:00, Night: 20:00-8:
 ↪00)",
        'dry_moisture_bar': "Dry Moisture by Grain Type (%)",
        'settings_scatter': "Set Temperature vs Dry Moisture",
        'alarms_timeline': "Dryer Alarms Timeline (Segments)",
        'operator_notes': "Operator Recommendations",
        'notes_intro': "Statistics-based guidelines for optimal drying:",
        'notes': {
            'temp_moisture': "Set Temperature vs Dry Moisture: {:.2f}. Higher␣
 ↪temperature dries better if negative (e.g., -0.5 = strong drying), wetter␣
 ↪grain if positive (e.g., 0.3 = less drying). NaN = insufficient data.",
            'timer_moisture': "Drop Interval vs Dry Moisture: {:.2f}. Longer␣
 ↪intervals dry better if negative (e.g., -0.4 = good drying), shorter␣
 ↪intervals leave wetter grain if positive (e.g., 0.2).",
            'actual_temp_moisture': "Actual Temperature vs Dry Moisture: {:.2f}.
 ↪ Hotter burns dry better if negative (e.g., -0.6 = strong effect), less␣
 ↪drying if positive.",
            'timer_mass': "Drop Interval vs Dried Mass: {:.2f}. Shorter␣
 ↪intervals increase mass if negative (e.g., -0.3), longer intervals increase␣
 ↪mass if positive.",
            'temp_mass': "Set Temperature vs Dried Mass: {:.2f}. Higher␣
 ↪temperature increases mass if positive (e.g., 0.4), less mass if negative.",
            'general': [
                "Frequent alarm segments (see timeline) indicate overloading-
reduce input rate.",
                "If incoming moisture is close to dry (<1% difference),␣
 ↪increase SET_BURNERS_TEMP by 5-10°C to avoid returns.",
                "Dryer worked {:.1f} hours (excluding STOP mode). Adjust input␣
 ↪rate if wet grain exceeds 20% per grain type.",
                "For Raps: overdry < 8%, normal 8-9.5%. For others: overdry <␣
 ↪12.9%, normal 12.9-14.5%."
            ]
        },
        'conclusion': "Why Grainstate?",
        'conclusion_text': "Real-time drying optimization. Contact us for a␣
 ↪demo: info@grainstate.com"
    }
}
t = texts['en']
```

[2]:
```
#     2:
#
total_mass = t['total_mass'].format(df['dry_mass'].sum())
drops = t['drops'].format(len(df['DROPS_SCORE'].dropna().unique()))
returned = t['returned'].format(returned_mass)
```

```
intro = t['intro'].format(df['DateTime'].min().strftime('%d-%m-%Y'),␣
 ↪df['DateTime'].max().strftime('%d-%m-%Y'))

#
wet_moisture_fig = px.bar(avg_incoming_wet, x='GRAIN_TYPE', y=['mean', 'min'],
                          title='Incoming Moisture (%)', barmode='group',␣
 ↪text_auto='.1f')
wet_moisture_fig.update_layout(font={'size': 14})

mode_fig = px.pie(mode_times, names='mode', values='Time_Diff',␣
 ↪title=t['mode_pie'], hole=0.4,
                  color_discrete_sequence=px.colors.qualitative.Pastel)
mode_fig.update_layout(font={'size': 14})

mass_fig = px.pie(dry_mass_stats, names='GRAIN_TYPE', values='Total_Dry_Mass',␣
 ↪title=t['mass_pie'], hole=0.4,
                  color_discrete_sequence=px.colors.qualitative.Plotly)
mass_fig.update_traces(textinfo='percent+label+value', texttemplate='%{label}:␣
 ↪%{value:.0f} kg (%{percent})')
mass_fig.update_layout(font={'size': 14})

moisture_fig = px.bar(moisture_by_grain, x='GRAIN_TYPE', y='dry_mass',␣
 ↪color='Moisture_Status',
                      title=t['moisture_bar'], barmode='group', text_auto='.0f',
                      color_discrete_map={'overdry': '#e74c3c', 'normal':␣
 ↪'#2ecc71', 'wet': '#3498db'})
moisture_fig.update_layout(font={'size': 14})

shift_fig = px.bar(shift_productivity, x='Shift_Start', y='dry_mass',␣
 ↪color='Shift_Type',
                   title=t['shift_line'], text_auto='.0f',
                   color_discrete_map={'Day': '#3498db', 'Night': '#2ecc71'})
shift_fig.update_layout(font={'size': 14}, xaxis={'tickangle': -45})

dry_moisture_fig = px.bar(dry_mass_stats, x='GRAIN_TYPE',␣
 ↪y=['Mean_Dry_Moisture', 'Min_Dry_Moisture'],
                          title=t['dry_moisture_bar'], barmode='group',␣
 ↪text_auto='.1f')
dry_moisture_fig.update_layout(font={'size': 14})

scatter_fig = px.scatter(df, x='SET_BURNERS_TEMP', y='perten_dry_Moisture',␣
 ↪size='DROPS_SET_TIMER',
                         title=t['settings_scatter'], color='GRAIN_TYPE',
                         labels={'SET_BURNERS_TEMP': 'Temperature (°C)',␣
 ↪'perten_dry_Moisture': 'Moisture (%)'},
                         color_discrete_sequence=px.colors.qualitative.Plotly)
```

```
scatter_fig.update_layout(font={'size': 14})

if alarms_segments.empty:
    alarms_timeline_fig = go.Figure()
    alarms_timeline_fig.update_layout(title=t['alarms_timeline'],␣
 ↪xaxis_title="Time", yaxis_title="Alarm Type")
else:
    alarms_timeline_fig = go.Figure()
    alarm_types = alarms_segments['Alarm_Type'].unique()
    colors = px.colors.qualitative.Plotly[:len(alarm_types)]
    for i, alarm_type in enumerate(alarm_types):
        alarm_data = alarms_segments[alarms_segments['Alarm_Type'] ==␣
 ↪alarm_type]
        for _, row in alarm_data.iterrows():
            alarms_timeline_fig.add_trace(go.Scatter(
                x=[row['Start'], row['End']],
                y=[i, i],
                mode='lines',
                line=dict(width=10, color=colors[i]),
                name=alarm_type,
                showlegend=True if row.name == alarm_data.index[0] else False
            ))
    alarms_timeline_fig.update_layout(
        title=t['alarms_timeline'], xaxis_title="Time", yaxis_title="Alarm␣
 ↪Type",
        yaxis={'tickmode': 'array', 'tickvals': list(range(len(alarm_types))),␣
 ↪'ticktext': alarm_types},
        font={'size': 14}, height=400
    )

#         (   ,              )
corr = {}
all_df = df.dropna(subset=['SET_BURNERS_TEMP', 'DROPS_SET_TIMER',␣
 ↪'ACTUAL_BURNERS_TEMP', 'perten_dry_Moisture', 'dry_mass'])
if len(all_df) >= 2 and all_df['SET_BURNERS_TEMP'].std() != 0 and␣
 ↪all_df['DROPS_SET_TIMER'].std() != 0:
    corr = {
        'temp_moisture': all_df['SET_BURNERS_TEMP'].
 ↪corr(all_df['perten_dry_Moisture']),
        'timer_moisture': all_df['DROPS_SET_TIMER'].
 ↪corr(all_df['perten_dry_Moisture']),
        'actual_temp_moisture': all_df['ACTUAL_BURNERS_TEMP'].
 ↪corr(all_df['perten_dry_Moisture']),
        'timer_mass': all_df['DROPS_SET_TIMER'].corr(all_df['dry_mass']),
        'temp_mass': all_df['SET_BURNERS_TEMP'].corr(all_df['dry_mass'])
    }
```

```
notes_list = ""
for key, value in corr.items():
    if pd.isna(value):
        notes_list += f"<li>{t['notes'][key].split(':')[0]}: Not enough data or no variation.</li>"
    else:
        notes_list += f"<li>{t['notes'][key].format(value)}</li>"
notes_list += "".join([f"<li>{t['notes']['general'][i].format(total_work_time) if i == 2 else t['notes']['general'][i]}</li>" for i in range(len(t['notes']['general']))])
```

[3]:
```
#     3:       HTML    PDF
html_content = f"""
<html>
<head>
    <title>{t['title']}</title>
    <style>
        body {{ font-family: Arial; padding: 20px; background-color: #f9f9f9; }}
        h1 {{ color: #2c3e50; text-align: center; font-size: 32px; }}
        h3 {{ color: #34495e; font-size: 24px; }}
        p, li {{ color: #34495e; font-size: 16px; line-height: 1.5; }}
        .summary {{ background-color: #ecf0f1; padding: 15px; border-radius: 5px; }}
    </style>
</head>
<body>
    <h1>{t['title']}</h1>
    <p>{intro}</p>

    <h3>{t['summary']}</h3>
    <div class="summary">
        <p>{total_mass}</p>
        <p>{drops}</p>
        <p>{returned}</p>
        <p>{t['wet_moisture']}</p>
        {wet_moisture_fig.to_html(full_html=False)}
    </div>

    <h3>{t['mode_pie']}</h3>
    {mode_fig.to_html(full_html=False)}

    <h3>{t['mass_pie']}</h3>
    {mass_fig.to_html(full_html=False)}

    <h3>{t['moisture_bar']}</h3>
    {moisture_fig.to_html(full_html=False)}
```

```
    <h3>{t['shift_line']}</h3>
    {shift_fig.to_html(full_html=False)}

    <h3>{t['dry_moisture_bar']}</h3>
    {dry_moisture_fig.to_html(full_html=False)}

    <h3>{t['settings_scatter']}</h3>
    {scatter_fig.to_html(full_html=False)}

    <h3>{t['alarms_timeline']}</h3>
    {alarms_timeline_fig.to_html(full_html=False)}

    <h3>{t['operator_notes']}</h3>
    <p>{t['notes_intro']}</p>
    <ul>{notes_list}</ul>

    <h3>{t['conclusion']}</h3>
    <p>{t['conclusion_text']}</p>
</body>
</html>
"""

with open('report.html', 'w') as f:
    f.write(html_content)

display(HTML(html_content))

#     4:          PDF
print("                Codespaces:")
print("jupyter nbconvert --to pdf report.ipynb")
```

```
<IPython.core.display.HTML object>
```

```
                Codespaces:
jupyter nbconvert --to pdf report.ipynb
```