Professionaliseren van Streamlit Productie Dashboard

**Professionalisering van het Downtime Dashboard**

**Interfilling Downtime Dashboard – Verbeterplan**  
In dit rapport staan aanbevelingen om het bestaande Streamlit-dashboard professioneler, overzichtelijker en gebruiksvriendelijker te maken. We behandelen visuele lay-out, data-inhoud, UX/interactie, concrete codevoorbeelden en specifieke verbeterpunten in de huidige code.

**1. Visuele lay-out: modern en consistent design**

*Afbeelding: Voorbeeld van een professioneel dashboard met een duidelijke sidebar, statistiek-kaarten en donkere thema.*

Een professionele uitstraling begint bij een consistente lay-out en kleurgebruik. Enkele aanbevelingen:

* **Kleurenschema en theming:** Kies een beperkt palet dat past bij uw huisstijl (bijv. Interfilling-blauw/oranje) en zorg voor voldoende contrast​[blog.streamlit.io](https://blog.streamlit.io/designing-streamlit-apps-for-the-user-part-ii/#:~:text=Colors%20and%20contrast). Overweeg een **donker thema** met contrasterende accentkleuren voor een moderne look, zoals in het voorbeeld (donkerblauwe achtergrond met lichtblauwe accenten). Streamlit ondersteunt custom theming – u kunt via .streamlit/config.toml of st.set\_page\_config een thema instellen (primaire kleur, achtergrond, tekstkleur)​[blog.streamlit.io](https://blog.streamlit.io/designing-streamlit-apps-for-the-user-part-ii/#:~:text=Streamlit%20supports%20Theming%20and%20Dark,source%20code%29%20by%20Streamlit). Dit zorgt ervoor dat alle componenten (grafieken, knoppen, etc.) automatisch de juiste kleuren gebruiken.
* **Achtergrond en component-stijl:** In plaats van willekeurig CSS-klassen te stylen (die kunnen veranderen bij updates), kunt u globale stijlen instellen via thema’s of expliciete CSS selectors. Gebruik bijv. een **neutrale achtergrond** (wit of heel lichtgrijs/blauw) en plaats onderdelen op semi-transparante kaarten. U had al een lichte blauw-gradiënt als achtergrond; zorg dat tekst (donkerblauw/zwart) hier goed leesbaar op is​[blog.streamlit.io](https://blog.streamlit.io/designing-streamlit-apps-for-the-user-part-ii/#:~:text=Colors%20and%20contrast). Voor consistentie zouden de sidebar en main area verschillende tinten kunnen krijgen (bijv. sidebar iets donkerder dan de hoofdachtergrond, zoals in het voorbeeld).
* **Koppen en teksten:** Gebruik Streamlit’s ingebouwde titelformaten om hiërarchie aan te brengen​[blog.streamlit.io](https://blog.streamlit.io/designing-streamlit-apps-for-the-user-part-ii/#:~:text=There%20are%20several%20,written%20using%20different%20text%20sizing). Begin met st.title("Downtime Dashboard") of een mooie HTML-markup voor de header (in het voorbeeld is een icoon “🔔” voor de titel gebruikt). Vervolgens subheaders (st.subheader) voor secties als *Verdeling stilstand* of *Stilstand per reden*. Beperk het handmatig gebruiken van <div> voor titels tenzij nodig; Streamlit’s titels passen ook bij het thema.
* **Statistiek-kaarten (KPI’s):** De drie kerncijfers (totale stilstand, aantal stilstanden, langste stilstand) kunnen als **cards** vormgegeven worden voor extra nadruk. U deed dit deels met st.metric. Voeg eventueel een icoontje toe in de label (bijv. "💡 Totale stilstand (uren)") en style de metric-cards via CSS of thema. In het voorbeelddashboard worden 📌-icoontjes en korte beschrijvingen getoond op de kaarten. U kunt st.metric gebruiken met een tekstuele *delta* om extra context te tonen. Bijvoorbeeld:

python

KopiërenBewerken

col1, col2, col3 = st.columns(3)

col1.metric("📌 Totale stilstand (uren)", f"{round(total\_hours,2)}", help="Som van stilstand in uren")

col2.metric("📌 Aantal stilstanden", str(len(df\_filtered)), help="Totaal aantal storingen in selectie")

col3.metric("📌 Langste stilstand (min)", f"{max\_duration:.1f}", help="Langstdurende storing in selectie")

Hier hebben we een 📌-icoon toegevoegd en de help-parameter gebruikt voor een tooltip bij hover​[docs.streamlit.io](https://docs.streamlit.io/develop/api-reference/widgets/st.selectbox#:~:text=help%20). Met CSS kunt u deze kaarten verder opmaken (bijv. een donkerblauwe achtergrond en afgeronde hoeken zoals in het voorbeeld). Zorg voor voldoende padding en margin tussen de kaarten voor een luchtig uiterlijk.

* **Lay-out structuur:** Organiseer content in logische blokken. Gebruik kolommen (st.columns) om gerelateerde visualisaties naast elkaar te plaatsen​[blog.streamlit.io](https://blog.streamlit.io/designing-streamlit-apps-for-the-user-part-ii/#:~:text=Lay%20it%20out%20in%20columns). U deed dit al voor de donut-grafieken per lijn. Zorg dat alle grafieken dezelfde breedte krijgen (bijv. use\_container\_width=True bij Plotly charts) voor een nette uitlijning. Overweeg secties visueel te scheiden met subtiele lijnen of ruimte.
* **Logo en branding:** Plaats eventueel het bedrijfslogo bovenaan de sidebar of dashboard (bijv. st.sidebar.image("logo.png", use\_column\_width=True)). Dit geeft een professionele touch en maakt duidelijk wiens dashboard het is. In het voorbeeld staat linksboven een *Company Logo*. U kunt tevens een tagline of url ("Interfilling – www.proces360.com") als kleinere tekst toevoegen onder de titel voor branding.

Samengevat: stem de kleuren en componentstijl af op elkaar, benut kolommen en titels voor structuur en geef belangrijke cijfers een aantrekkelijke kaart-weergave. Dit creëert een overzichtelijk en modern ogend dashboard.

**2. Data-inhoud: relevante inzichten uit de CSV**

De CSV dataset (OEE\_Dashboard\_PowerBI\_Finaal.csv) bevat details over storingen (downtime) per order. Hiermee kunnen we diverse **beschrijvende statistieken** en visualisaties tonen:

* **Totale stilstand en aantallen:** Deze kerncijfers toont u al (totale stilstandstijd, aantal storingen, langste storing). Eventueel kunt u hier extra context aan toevoegen, zoals *gemiddelde duur per storing* of *totale stilstand als percentage van totale tijd*. In uw huidige data (alleen storingen) is het percentage moeilijk te bepalen zonder geplande productie, maar u berekent al een *Performance (%)* op basis van een 8u shift. Dit is een waardevol inzicht om dagelijks prestaties te tonen.
* **Stilstand per lijn (machine):** Dankzij de kolom Workflow kunnen we stilstanden per productielijn analyseren. U visualiseert dit al met de donutdiagrammen per lijn, verdeeld naar subType (bijv. technisch vs. organisatorisch). Dit geeft een snelle indruk welke lijn de meeste stilstand heeft en welk type oorzaak domineert per lijn. Zorg dat categorieën duidelijk gelabeld zijn – misschien de ‘null’ categorie uitsluiten of hernoemen (als subType soms leeg is). Een donut of pie per lijn is geschikt als globaal overzicht. Daarnaast kunt u een staafgrafiek “Stilstand per lijn” toevoegen die totale minuten per lijn vergelijkt (voor een snel totaaloverzicht).
* **Top 5 storingsredenen:** Analyseer de kolom *Reden* om te zien welke oorzaken het vaakst of het langst voorkomen. U had al een staafgrafiek *Stilstand per reden*, gesorteerd op totaal minuten. Zorg dat deze duidelijk getiteld is en eventueel beperk tot **Top N** redenen voor leesbaarheid. Top 5 is een goede keuze: anderen vallen daarna mogelijk in *“Overige”*. U kunt de top 5 redenen uitfilteren en desnoods de rest aggregeren als "Overig". Codevoorbeeld om top 5 redenen te berekenen:

python

KopiërenBewerken

top5 = (df\_filtered.groupby("Reden")["Duur\_min"].sum()

.nlargest(5).reset\_index())

fig\_top5 = px.bar(top5, x="Reden", y="Duur\_min",

title="Top 5 storingsredenen (min stilstand)")

st.plotly\_chart(fig\_top5, use\_container\_width=True)

Hieruit blijkt direct welke redenen de meeste downtime veroorzaken. Eventueel kunt u ook het **aantal storingen per reden** analyseren (in plaats van tijdsduur) om te zien of veel korte storingen vs. enkele lange storingen de oorzaak zijn.

* **Trends over tijd (dag/week/maand):** De tijdsdimensie is cruciaal. U heeft al een keuze *Dag/Week/Maand* waarmee de downtime per gekozen periode wordt getoond in een staafdiagram. Dit laat trends zien, bv. of bepaalde weken meer storingen hadden. Zorg dat deze grafiek duidelijk getiteld is (in uw code: title=f"Niet-geplande stilstand per {periode.lower()}" is prima). Overweeg om naast de staafdiagram van totale stilstand per periode ook **het aantal storingen per periode** te tonen (dit kan in dezelfde grafiek als tweede y-as, of als aparte grafiek/lijn). Zo ziet men of een toename in stilstand komt door meer storingen of langere storingen.
* **Details per order en langste stilstand:** U voegt nuttige detailanalyses toe, zoals de *Top 3 orders met hoogste stilstandstijd* en voor die orders de langste individuele storing. Dit is waardevolle informatie om te pinpointen welke productieorders problematisch waren. U zou dit kunnen uitbreiden naar **Top 5 orders** voor volledigheid, afhankelijk van hoeveel unieke orders er zijn. De tabel van langste storingen per top-order is goed; zorg eventueel dat tijdstippen mooi zijn opgemaakt (misschien alleen datum + tijd zonder seconden, voor leesbaarheid).
* **Downtime per werkdag (ma t/m vr):** U groepeert storingen per weekdag en toont top 3 orders per dag. Dit is interessante detailinfo, hoewel vrij uitgebreid. Let op de presentatie: dit levert vijf tabellen onder elkaar (één per werkdag met data). Overweeg dit in een compactere vorm te gieten, bv. met st.tabs voor elke dag, zodat de gebruiker per dag kan klikken in plaats van alle tabellen tegelijkertijd te zien. Bijvoorbeeld:

python

KopiërenBewerken

weekdays = ['Maandag','Dinsdag','Woensdag','Donderdag','Vrijdag']

dag\_tabs = st.tabs(weekdays)

for dag, tab in zip(weekdays, dag\_tabs):

with tab:

df\_dag = df\_filtered[df\_filtered['Weekdag']==dag]

# ... bereken df\_top\_dag en df\_top\_reason zoals nu ...

st.table(df\_top\_dag[['Ordernummer','Reden','Totale stilstand (min)','Totale stilstand (uur)']])

Hiermee heeft u per tab de top 3 orders van die dag​[docs.streamlit.io](https://docs.streamlit.io/develop/api-reference/layout/st.tabs#:~:text=Inserts%20a%20number%20of%20multi,between%20groups%20of%20related%20content). Dit houdt de UI compacter en de gebruiker kan specifieke dagen bekijken.

* **Performance en uitschieters:** Het overzicht *Performance (%)* per dag (gebaseerd op 8 uur min downtime) is handig om in één oogopslag te zien welke dagen goed/slecht gingen. De lijnchart die u tekent geeft dit duidelijk weer. U kunt nog **markers** of annotaties toevoegen op dagen die outlier zijn. U bepaalt al een drempel (gemiddelde + 1\*std) en filtert outliers. U zou die drempel als lijn in de grafiek kunnen plotten of de outlier-dagen in een afwijkende kleur markeren. Met Plotly kan dat door bijv. punten boven drempel een andere marker-kleur te geven. Ook is het goed dat u de outlier-drempel en melding toont (“Geen uitschieters 🎉” als none). Dit soort feedback draagt bij aan de gebruikerservaring.

Kortom, de dataset biedt mogelijkheden voor kern-KPI’s, oorzaakanalyses (Pareto), tijdtrends en detail-drilldowns. Prioriteer de visuele weergave van *belangrijkste inzichten*: totale stilstand en top oorzaken. Detailtabellen (per order, per dag) zijn waardevol maar kunnen eventueel onder tabs/expanders zodat ze het hoofdoverzicht niet overweldigen.

**3. UX en interactie: gebruiksgemak voor de gebruiker**

Een professioneel dashboard is niet alleen mooi, maar ook intuïtief in gebruik. Enkele UX-verbeteringen:

* **Eenduidige filters in de sidebar:** Momenteel is er een duplicaat *“Kies workflow (machine)”* filter zichtbaar (een boven en een onder in de sidebar – dit is verwarrend). Maak één centrale filtersectie. Gebruik de sidebar voor alle invoercontroles, zodat de gebruiker op één plek filters kan instellen​[blog.streamlit.io](https://blog.streamlit.io/designing-streamlit-apps-for-the-user-part-ii/#:~:text=The%20above%20layout%20has%20a,achieved%20with%20the%20following%20code). Groepeer verwante filters eventueel binnen st.expander("Filters") om de sidebar compact te houden​[discuss.streamlit.io](https://discuss.streamlit.io/t/how-to-create-this-type-of-beautiful-page-sidebars/52402#:~:text=Thanks%20for%20posting%21). In uw geval zijn er twee belangrijke filters: **datumbereik** en **lijn (Workflow)**. Deze kunnen als volgt geïmplementeerd worden bovenaan de sidebar:

python

KopiërenBewerken

st.sidebar.header("🔍 Filters")

# Kies workflow filter

workflows = ["Alle lijnen"] + sorted(df\_downtime["Workflow"].unique())

lijn\_selectie = st.sidebar.selectbox("Kies workflow (machine):", options=workflows, index=0)

# Datum bereik filter

min\_date = df\_downtime["Datum"].min()

max\_date = df\_downtime["Datum"].max()

datum\_bereik = st.sidebar.date\_input("Datum bereik:", value=(min\_date, max\_date),

min\_value=min\_date, max\_value=max\_date)

Met bovenstaande code krijgt u één drop-down voor de lijnselectie (met default “Alle lijnen”) en één datumrange-picker. Geen dubbele controles meer. Ook hebben we een st.sidebar.header *Filters* met een icoontje voor duidelijkheid.

* **Filtertoepassing op data:** Wanneer de gebruiker filters instelt, moeten alle visualisaties daarop reageren. Pas daarom direct na het ophalen van de data de filters toe op df\_downtime. Bijv.:

python

KopiërenBewerken

df\_filtered = df\_downtime.copy()

# Filter op gekozen lijn (indien niet 'Alle lijnen')

if lijn\_selectie != "Alle lijnen":

df\_filtered = df\_filtered[df\_filtered["Workflow"] == lijn\_selectie]

# Filter op gekozen datumbereik

start\_d, end\_d = pd.to\_datetime(datum\_bereik[0]), pd.to\_datetime(datum\_bereik[1])

df\_filtered = df\_filtered[(df\_filtered["Datum"] >= start\_d) & (df\_filtered["Datum"] <= end\_d)]

Zorg dat df\_filtered hiermee de basis is voor alle verdere berekeningen en grafieken. Dit betekent dat bijvoorbeeld de donuts **per lijn** dynamisch moeten omgaan met de lijnfilter:

* + Als *Alle lijnen* geselecteerd is, toon donuts voor elke lijn (zoals nu).
  + Als een specifieke lijn is geselecteerd, kunt u óf alleen die ene donut tonen, óf zelfs de hele sectie “Verdeling per subType (per lijn)” overslaan omdat de gebruiker al ingezoomd heeft op één lijn. Een oplossing is conditioneel weergeven:

python

KopiërenBewerken

if lijn\_selectie == "Alle lijnen":

# toon alle donuts (Alle lijnen, COSMO, VMPT1, VMPT5)

else:

# toon één donut voor de geselecteerde lijn of een aangepast grafiekje

st.subheader(f"Verdeling stilstand voor {lijn\_selectie}")

fig = px.pie(df\_filtered, names="subType", values="Duur\_min",

title=f"{lijn\_selectie} - verdeling oorzaken", hole=0.4)

st.plotly\_chart(fig, use\_container\_width=True)

Zo ziet de gebruiker bij selectie van een enkele machine direct de oorzaakverdeling voor die machine, in plaats van vier vrijwel identieke donuts. Dit voorkomt verwarring.

* **Tabs voor structuur:** Overweeg st.tabs te gebruiken om de resultaten te organiseren​[docs.streamlit.io](https://docs.streamlit.io/develop/api-reference/layout/st.tabs#:~:text=Inserts%20a%20number%20of%20multi,between%20groups%20of%20related%20content). Bijvoorbeeld een tabblad *Overzicht* met de kerncijfers, donuts en top 5 redenen, en een tabblad *Details* met de verdiepende info (top orders, per dag/week trend, etc.). Dit houdt de interface schoon: de gebruiker bekijkt in *Overzicht* de hoofdlijnen en kan doorklikken naar *Details* voor diepere analyse. Op een groot breedbeeldscherm kan alles onder elkaar, maar bij veel content is tabben prettig om niet eindeloos te scrollen. Implementatievoorbeeld:

python

KopiërenBewerken

tab1, tab2 = st.tabs(["📈 Overzicht", "🔎 Details"])

with tab1:

st.plotly\_chart(fig\_top5) # voorbeeld: Top 5 redenen grafiek

# ... overige overview elementen ...

with tab2:

st.plotly\_chart(fig\_tijd) # trend grafiek per periode

st.table(df\_order\_downtime) # detailtabel top orders

# ... etc ...

Let op dat *alle* tabs altijd worden gerenderd (ook als ze niet actief zijn)​[docs.streamlit.io](https://docs.streamlit.io/develop/api-reference/layout/st.tabs#:~:text=Warning), dus de performance moet dat aankunnen. In uw geval is dat geen probleem, de dataset is relatief klein.

* **Dropdowns en radio-buttons:** De gebruiker kan met de huidige radio *Dag/Week/Maand* een tijdsresolutie kiezen – dit is prima. Radio-buttons zijn geschikt bij 2-5 exclusieve opties. U zou hier ook st.selectbox kunnen gebruiken als de lijst eventueel uitbreidt. Zorg dat de labels duidelijk zijn (bijv. *“Selecteer aggregatieperiode:”*). Overweeg een **dropdown voor het kiezen van een specifieke periode** om nader te bekijken. U heeft al selected\_periode = st.selectbox("Kies week/dag:", ...) staan voor de Pareto van top 3 redenen per periode. Dit is goed – het maakt de grafiek interactief zodat een gebruiker een specifieke week of dag kan uitlichten.
* **Tooltips en help-iconen:** Maak gebruik van de help parameter bij widgets (zoals getoond bij st.metric en st.selectbox hierboven)​[docs.streamlit.io](https://docs.streamlit.io/develop/api-reference/widgets/st.selectbox#:~:text=help%20). Hierdoor verschijnt een klein ⓘ-icoon naast de label. Bijvoorbeeld bij *Kies workflow (machine)* kunt u help="Filter op specifieke productielijn of toon Alle lijnen." toevoegen. De gebruiker die twijfelt, kan erover hoveren en direct een toelichting lezen. Ook bij complexe grafieken kunt u met st.caption onder de grafiek extra uitleg geven, of een st.expander("Uitleg") met tekst plaatsen als handleiding.
* **Interactie in grafieken:** Plotly grafieken bieden hover-informatie en zoomfunctionaliteit out-of-the-box, wat goed is. Zorg dat titels en as-labels duidelijk zijn, zodat ook zonder extra toelichting begrijpelijk is wat men ziet. Overweeg bij de *Performance (%)* lijnchart om weekenddagen te markeren of uitsluiten (aangezien alleen ma-vr relevant zijn in uw filtering). Dit kan door data van za/zo te filteren of door verticale vlakken aan te duiden, maar filtering is eenvoudiger: u filtert nu al op werkdagen 07:30-16:00, dus dat is in orde.
* **Gebruiksvriendelijkheid boven alles:** Test het dashboard vanuit eindgebruiker perspectief. Zijn de interacties logisch? Bijvoorbeeld, als men een specifieke week kiest in de Pareto-sectie, is het duidelijk dat eerst *Week* als periode gekozen moet zijn? U zou de *Kies week* selectbox conditioneel kunnen tonen **alleen** wanneer periode-selectie ‘Week’ is gekozen, om verwarring te voorkomen. Dit kan met een simpel if:

python

KopiërenBewerken

if periode\_selectie == "Week":

week = st.selectbox("Kies week:", sorted(df\_filtered["Week"].unique()))

# ... vervolgens Pareto voor die week ...

Zo verschijnt de controle alleen wanneer relevant.

Samenvattend: houd de interface simpel en intuïtief. Alle filteropties op één plek, content logisch gegroepeerd (via tabs of subheaders), en voldoende toelichting via tooltips of captions. Dit geeft de gebruiker de controle om zelf analyses te maken, zonder aan gebruiksvriendelijkheid in te boeten.

**4. Concrete codevoorbeelden voor verbeteringen**

Hieronder vatten we enkele belangrijke aanpassingen samen met code-snippets, zodat u ze direct in PyCharm kunt implementeren.

**A) Sidebar filters implementeren (zonder dubbels):**

python

KopiërenBewerken

# Sidebar filters

st.sidebar.header("🔍 Filters")

workflows = ["Alle lijnen"] + sorted(df\_downtime["Workflow"].unique())

lijn = st.sidebar.selectbox("Kies lijn/machine:", workflows, index=0, help="Filter op specifieke lijn")

# Datum bereik met default volledige range

min\_date = df\_downtime["Datum"].min()

max\_date = df\_downtime["Datum"].max()

start\_dt, end\_dt = st.sidebar.date\_input("Datum bereik:", (min\_date, max\_date),

min\_value=min\_date, max\_value=max\_date)

# Zorg dat altijd twee waardes (start, end) uit date\_input komen

if isinstance(start\_dt, list) or isinstance(start\_dt, tuple):

start\_date, end\_date = pd.to\_datetime(start\_dt[0]), pd.to\_datetime(start\_dt[1])

else:

start\_date, end\_date = pd.to\_datetime(start\_dt), pd.to\_datetime(end\_dt)

Deze code plaatst een kopje *Filters* in de sidebar, vervolgens een selectbox voor *lijn* en een datumrange-picker. De help teksten geven extra uitleg. We converteren de output van date\_input naar pandas datetime voor robuustheid.

**B) Data filtering toepassen:**

python

KopiërenBewerken

# Pas filters toe op dataframe

df\_filtered = df\_downtime.copy()

# 1. Filter op datumrange

df\_filtered = df\_filtered[(df\_filtered["Datum"] >= start\_date.date()) &

(df\_filtered["Datum"] <= end\_date.date())]

# 2. Filter op lijn (indien niet 'Alle lijnen')

if lijn != "Alle lijnen":

df\_filtered = df\_filtered[df\_filtered["Workflow"] == lijn]

# 3. Exclude storingen buiten 07:30-16:00 op weekdagen

df\_filtered = df\_filtered[

(df\_filtered["Starttijd"].dt.weekday < 5) &

(df\_filtered["Starttijd"].dt.time >= datetime.time(7, 30)) &

(df\_filtered["Starttijd"].dt.time < datetime.time(16, 0))

]

Hier wordt eerst op datum gefilterd, dan op lijn, en dan de uitsluiting van avond/nacht/weekend toegepast. We gebruiken .date() om ervoor te zorgen dat we datetime-objecten goed vergelijken met datum (dit voorkomt de eerder opgetreden TypeError). Merk op: als u df["Datum"] al als datetime64 hebt (met 00:00 tijd), kunt u direct vergelijken met start\_date zonder .date().

**C) Metric cards met stijl en iconen:**

python

KopiërenBewerken

st.markdown("""<style>

.card {background-color: #eef4ff; padding:15px; border-radius:10px; margin-bottom:10px}

.card h3 {margin:0; color:#003366;}

.card p {margin:0; font-size:24px; font-weight:bold; color:#003366;}

.small {font-size:0.8em; color: #555;}

</style>""", unsafe\_allow\_html=True)

# Voorbeeld metric card

total\_hrs = df\_filtered["Duur\_min"].sum()/60

st.markdown(f"""

<div class="card">

<h3>📊 Totale stilstand</h3>

<p>{total\_hrs:.2f} uur</p>

<span class="small">in de geselecteerde periode</span>

</div>

""", unsafe\_allow\_html=True)

Bovenstaande HTML/CSS creëert een herbruikbare *card*-stijl (lichtblauwe achtergrond, afgeronde hoeken). Vervolgens een voorbeeld van hoe u een dergelijke card invult met een icoon, titel, waarde en kleine toelichting. U kunt dit in kolommen zetten: bijv. drie <div class="card"> in drie kolommen voor de drie KPI’s. Hoewel st.metric eenvoudiger is, geeft dit fine-grained controle over opmaak.

**D) Top 5 redenen berekenen en weergeven:**

python

KopiërenBewerken

# Top 5 redenen tabel

top5 = (df\_filtered.groupby("Reden", dropna=False)["Duur\_min"].sum()

.nlargest(5).reset\_index())

top5["Duur\_uur"] = (top5["Duur\_min"]/60).round(1)

top5.columns = ["Reden", "Stilstand (min)", "Stilstand (uur)"]

st.subheader("🥇 Top 5 storingsredenen")

st.dataframe(top5)

Dit groeperingscommando verzamelt de top 5 oorzaken op basis van totale stilstand in minuten. We voegen ook een kolom uren toe voor perspectief. Met dropna=False zorgen we dat eventuele lege redenen als *NaN* meegerekend worden (optioneel: u kunt die eruit filteren vooraf als ze niet relevant zijn). st.dataframe geeft een scrollbare tabel; u kunt ook st.table gebruiken voor een statische weergave.

**E) Tabs voor overzicht en detail:**

python

KopiërenBewerken

tab\_overview, tab\_detail = st.tabs(["Overzicht", "Details"])

with tab\_overview:

# Kerncijfers en belangrijke grafieken

st.columns(3)[0].metric("Totale stilstand (uur)", f"{total\_hrs:.2f}")

st.plotly\_chart(fig\_top5, use\_container\_width=True)

# ... meer overzichtscharts ...

with tab\_detail:

st.plotly\_chart(fig\_tijd, use\_container\_width=True) # trend per periode

st.table(df\_order\_downtime) # top orders tabel

# ... overige details ...

Dit creëert twee tabbladen. In *Overzicht* plaatsen we de belangrijkste info, in *Details* de aanvullende analyses. Dit voorkomt een te lange één pagina scroll. De gebruiker navigeert gemakkelijk tussen de tabs voor gerelateerde content​[docs.streamlit.io](https://docs.streamlit.io/develop/api-reference/layout/st.tabs#:~:text=Inserts%20a%20number%20of%20multi,between%20groups%20of%20related%20content).

Deze codevoorbeelden illustreren de kernpunten van de verbeteringen. Uiteraard moeten ze worden geïntegreerd en verfijnd binnen uw volledige script, maar ze bieden een concrete basis om op voort te bouwen. Test telkens stap voor stap of de output klopt met de verwachting.

**5. Verbeterpunten in de huidige code**

Op basis van de analyse van uw huidige codebestand *dashboard.py* zijn dit de belangrijkste verbeterpunten en how-to’s:

* **Verwijder dubbele workflow-filter:** In de huidige sidebar wordt “Kies workflow (machine)” twee keer getoond (één hardcoded boven, één via workflow\_select later) – dit is verwarrend (zie rode cirkel in de screenshot). Los dit op door slechts één filtercomponent te gebruiken. Advies: verplaats de definitie workflow\_select = st.sidebar.selectbox(...) naar direct onder de datumfilter, en gebruik diezelfde workflow\_select overal. Verwijder eventuele tweede aanroep. Hiermee is de sidebar duidelijk en enkelvoudig.
* **Filter volgorde en variabele gebruik:** Er zit een bug in de code rond df\_filtered. U maakt mask\_date maar past het niet direct toe, en gebruikt df\_filtered voordat het gedefinieerd is. Correctie: maak eerst df\_filtered = df\_downtime[mask\_date].copy(). Pas daarna de shift-tijd filter toe *op df\_filtered*. In code:

python

KopiërenBewerken

mask\_date = (df\_downtime["Datum"] >= start\_date) & (df\_downtime["Datum"] <= end\_date)

df\_filtered = df\_downtime[mask\_date].copy()

# dan de df\_filtered = df\_filtered[...] met weekday/time filter

Dit voorkomt zowel een NameError als het per ongeluk filteren van een onjuiste dataframe. Werk dus steeds met df\_filtered als resultaat van alle filters.

* **TypeError bij datumvergelijking oplossen:** De fout *“Invalid comparison between dtype=datetime64 and date”* trad waarschijnlijk op doordat df\_downtime["Datum"] objecten van type datetime.date bevat, terwijl u vergeleek met een Timestamp. Om dit te voorkomen, zorg dat beide kanten van de vergelijking hetzelfde type zijn. U deed al pd.to\_datetime(date\_range[0]) voor start\_date, maar als Datum kolom date objecten heeft, is het beter start\_date.date() te gebruiken *of* de hele Datum kolom naar datetime64 om te zetten. Bijvoorbeeld tijdens het inlezen:

python

KopiërenBewerken

df["Datum"] = pd.to\_datetime(df["Starttijd"].dt.date)

dan is Datum een echte datetime (uur: 00:00) en werken vergelijkingen soepel. Of eenvoudiger:

python

KopiërenBewerken

start\_date = pd.to\_datetime(date\_range[0]).date()

end\_date = pd.to\_datetime(date\_range[1]).date()

mask = (df\_downtime["Datum"] >= start\_date) & (df\_downtime["Datum"] <= end\_date)

Vergelijking date vs date. Pas dit consistent toe. Hiermee verdwijnt de getoonde traceback【13†】.

* **Performanceoptimalisatie:** Voor de huidige dataset zal performance geen groot probleem zijn, maar let op inefficiënties:
  + U berekent sommige aggregaties dubbel. Bijvoorbeeld df\_filtered\_global wordt aangemaakt voor de Pareto per periode, terwijl df\_filtered al een subset is. U kunt gewoon df\_filtered hergebruiken nadat u de periodekolom hebt toegevoegd, in plaats van opnieuw te filteren op datum. Hetzelfde geldt voor filtering per workflow: als u df\_filtered al op lijn filtert, is een aparte loop met df\_cat = df\_filtered\_global[df...] redundant – u kunt direct op df\_filtered opereren.
  + Gebruik .copy() na filteren als u nieuwe kolommen toevoegt (zoals Weekdag of Periode) om SettingWithCopyWarning te vermijden. U doet dit deels, maar bv. df\_filtered['Weekdag'] = ... kan een waarschuwing geven als df\_filtered een slice was. Door df\_filtered = df\_filtered.copy() voordat u nieuwe kolommen maakt, voorkomt u dit.
  + @st.cache\_data wordt al gebruikt bij het inladen van CSV – goed. Overweeg dit ook bij zware berekeningen indien nodig. Bijvoorbeeld een functie die de aggregaten per periode berekent kan gecachet worden op basis van (periode\_selectie, lijn\_selectie) keys. Echter, pas op dat caching niet complicerend wordt; vaak is de huidige performance toereikend.
* **Foutafhandeling en robuustheid:** U vangt al het scenario op dat het CSV-bestand niet gevonden is (st.error + st.stop()). Denk ook aan:
  + **Geen data na filter:** Als filters zó streng zijn dat df\_filtered leeg wordt (bijv. toekomstige datumrange of een lijn zonder storingen in die periode), is het goed de gebruiker hierover te informeren. U kunt checken if df\_filtered.empty: st.warning("Geen data voor deze selectie.") en evt. st.stop(). In plaats van dat de rest van de visualisaties lege grafieken tonen of fouten geven, stopt u netjes en toont u een boodschap.
  + **Niet-beschikbare kolommen:** Als de CSV structuur ooit wijzigt (kolomnamen), faalt nu de hernoeming en drop van eerste kolom. U kunt dit robuuster maken door in pd.read\_csv bijvoorbeeld index\_col=0 te gebruiken als die eerste kolom een index is, of door kolommen op naam te hernoemen (dan krijgt u een KeyError als een kolom ontbreekt, die u kunt opvangen). Voor nu is dit wellicht overkill, maar in productie-omgeving is het nuttig.
  + **Gebruik st.experimental\_memo/st.cache\_data:** U heeft dit correct toegepast op de data-inleesfunctie. Zorg ervoor dat bij veranderingen (nieuwe CSV uploaden) de cache wordt gewist of opnieuw ingelezen – eventueel st.cache\_data(clear\_on\_change=True) of een *“Refresh data”* knop die de functie opnieuw aanroept.
* **Code-organisatie:** Overweeg uw code op te splitsen in logische onderdelen of modules. Bijvoorbeeld de *Downtime Dashboard* en *Order Target Calculator* zouden gescheiden kunnen worden door Streamlit’s multipage functionaliteit (meerdere Python-files in een pages/ directory), of behoud de radio-switch maar verplaats de implementatie van elke pagina naar een aparte functie om het hoofdbestand leesbaarder te maken. Dit maakt onderhoud en verdere professionalisering makkelijker. U zou een functie render\_downtime\_dashboard(df) kunnen maken die alle bovenstaande elementen tekent gegeven een gefilterde dataframe, en apart een render\_order\_calculator(). In de hoofdcode roept u dan simpelweg if choice == "...": render\_downtime\_dashboard(df\_downtime).
* **Opschonen van restcode:** In de huidige code staat een functie pareto\_tabel(df, totaal) gedefinieerd die vervolgens niet meer wordt aangeroepen. Dergelijke resten kunnen verwijderd worden om verwarring te voorkomen. Houd alleen code die nodig is of documenteer duidelijk waarom het er staat. Schone code is ook een vorm van professionalisering.

Door bovenstaande verbeterpunten door te voeren, wordt uw dashboard code betrouwbaarder, leesbaarder en makkelijker uit te breiden. Combineer dit met de visuele en UX-aanpassingen, en u krijgt een modern dashboard dat waardevolle downtime-inzichten geeft op een professionele manier. Veel succes met implementeren en finetunen in PyCharm – stap voor stap deze adviezen invoeren zal leiden tot een aanzienlijk verbeterd eindresultaat!​[blog.streamlit.io](https://blog.streamlit.io/designing-streamlit-apps-for-the-user-part-ii/#:~:text=Theming%2C%20Colors%20and%20Contrast%20Ratios)​[blog.streamlit.io](https://blog.streamlit.io/designing-streamlit-apps-for-the-user-part-ii/#:~:text=Lay%20it%20out%20in%20columns)