### \*\*Slide 1: H2D Price Calculator – Intro\*\*

\*\*Kernpunten\*\*:

- \*\*Python-systeem\*\* voor \*\*kostencalculatie 3D-prints\*\* + \*\*slijtage-analyse\*\*.

- \*\*Real-time analytics\*\*: Live monitoring van bv. nozzle-slijtage, materiaalverbruik.

- \*\*GUI (Tkinter)\*\* voor gebruiksvriendelijkheid.

\*\*Technische diepgang (data-analyse)\*\*:

1. \*\*Input-data\*\*:

- \*Gestructureerd\*: Printtijd (sec), filamentgewicht (gram), energieverbruik (kWh).

- \*Categorisch\*: Materiaaltype (PLA, ABS), printerconfiguratie.

2. \*\*Data-pipeline\*\*:

```python

# Voorbeeld: Kostencalculatie-engine (simplified)

def calculate\_cost(filament\_used\_g, print\_time\_h, energy\_cost\_per\_kwh):

material\_cost = filament\_used\_g \* material\_price\_per\_g # uit productdatabase

energy\_cost = (printer\_wattage \* print\_time\_h / 1000) \* energy\_cost\_per\_kwh

wear\_cost = nozzle\_wear\_per\_hour \* print\_time\_h # slijtagemodel

return material\_cost + energy\_cost + wear\_cost + overhead

```

3. \*\*Link naar data science\*\*:

- Dataset met historische prints → \*\*time-series analyse\*\* voor slijtagevoorspelling (ARIMA, LSTM).

---

### \*\*Slide 2: Systeemoverzicht (4 Modules)\*\*

\*\*Technische werking per module\*\*:

| Module | Data-input | Verwerking | Output | Data Science Link |

|--------------|------------|------------|--------|------------------|

| \*\*Calculator\*\* | `filament\_weight`, `time`, `temp` | Kostenberekening (zie Slide 1) | `Totaalprijs` | Regressiemodellen voor prijsoptimalisatie. |

| \*\*Configuratie\*\* | `materiaal\_parameters`, `printer\_instellingen` | Validatie (bijv. max. temperatuur checks) | `Configuratiebestand (JSON)` | Rule-based system → trainbaar met ML. |

| \*\*Producten\*\* | `Materiaaleigenschappen` (PLA: €0.02/g, ABS: €0.03/g) | Databasequery (SQLite) | `Beschikbare materialen` | Clustering voor materiaalgedrag. |

| \*\*Analyses\*\* | `Historische printjobs` | Aggregatie (bv. gem. kosten per maand) | `Rapporten (CSV)` | Dashboard met Plotly/Power BI. |

\*\*Belangrijk voor data-analist\*\*:

- \*\*Productdatabase\*\* is een \*\*gestructureerde dataset\*\* (bruikbaar voor PCA om materiaalcorrelaties te vinden).

- \*\*Analyses-module\*\* gebruikt \*\*pandas\*\* voor aggregaties (`df.groupby('materiaal').mean()`).

---

### \*\*Slide 3: Technische Architectuur\*\*

\*\*Dataflow-diagram (simplified)\*\*:

```mermaid

flowchart LR

GUI[Tkinter GUI] --> |JSON| Python[Python Engine]

Python --> |SQL Query| DB[(Product Database)]

Python --> |Pandas DataFrame| Analytics[Analytics Module]

Analytics --> |Matplotlib| Visuals[Grafieken]

```

\*\*Technische details\*\*:

- \*\*GUI\*\*: Verzamelt gebruikersinput → slaat op als JSON (bv. `{"filament\_weight": 50, "time": 2.5}`).

- \*\*Python Engine\*\*:

- Berekeningslogica gebruikt \*\*NumPy\*\* voor efficiënte operaties.

- Error handling met `try-except` voor ontbrekende data.

- \*\*Analytics Module\*\*:

- Gebruikt `matplotlib/seaborn` voor visualisaties (bv. slijtagecurve nozzle over tijd).

---

### \*\*Slide 4: Data Flow & Processing\*\*

\*\*Stapsgewijze data-transformatie\*\*:

1. \*\*GUI Input\*\*:

- Validatie met `pydantic` (bv. `assert filament\_weight > 0`).

2. \*\*Berekeningslaag\*\*:

- Parallelle verwerking met `multiprocessing` voor complexe jobs.

3. \*\*Analytics\*\*:

```python

# Voorbeeld: Materiaalgebruik analyseren

import pandas as pd

df = pd.read\_csv('print\_history.csv')

monthly\_usage = df.groupby(pd.to\_datetime(df['date']).dt.month)['filament\_used'].sum()

```

4. \*\*Export\*\*:

- CSV: `df.to\_csv('rapport.csv')` (compatibel met Power BI).

- Excel: Met `openpyxl` voor formattering.

\*\*Prestatiemetrics\*\*:

- \*\*<100ms responstijd\*\*: Bereikt door caching van veelgebruikte queries (LRU-cache).

- \*\*99.9% uptime\*\*: Logging met `logging`-module om crashes te tracken.

---

### \*\*Slide 5: Analytics Module & Data Visualisatie\*\*

\*\*Kernfocus\*\*:

- \*\*Real-time data insights\*\* uit 3D-printdata (materiaalgebruik, kosten, slijtage).

- \*\*Python-implementatie\*\* met bibliotheken als `pandas`, `Matplotlib/Seaborn`, en `Plotly`.

\*\*Technische details\*\*:

1. \*\*Data Pipeline\*\*:

```python

# Voorbeeld: Aggregatie van materiaalgebruik

import pandas as pd

df = pd.read\_csv('print\_history.csv')

top\_materials = df['material\_type'].value\_counts().head(10) # Top 10 materialen

```

- Output: Gestructureerde dataset voor visualisatie.

2. \*\*Link naar DS\*\*:

- Histogrammen/taartdiagrammen van `top\_materials` tonen \*\*materiaalpreferenties\*\* (gebruikt in marketingstrategieën).

---

### \*\*Slide 6: Geavanceerde Analytics – Materiaalgebruiksanalyse\*\*

\*\*Kernfocus\*\*:

- \*\*Diagnostische analyse\*\* van materiaalefficiëntie (bv. verspilling detecteren).

\*\*Technische uitwerking\*\*:

1. \*\*Data-transformatie\*\*:

```python

# Kosten per materiaalsoort berekenen

cost\_per\_material = df.groupby('material\_type')['total\_cost'].mean().sort\_values()

```

2. \*\*Visualisatie\*\*:

- \*\*Boxplots\*\* (`Seaborn.boxplot`) tonen kostenverschillen tussen materialen.

- \*\*DS-link\*\*: Uitschieters identificeren (bv. ABS duurder dan PLA?).

--

### \*\*Slide 7: Correlatie-analyse Implementatie\*\*

\*\*Kernfocus\*\*:

- \*\*Correlaties\*\* tussen variabelen (bv. gewicht vs. printtijd, temperatuur vs. slijtage).

\*\*Technische stappen\*\*:

1. \*\*Correlatiematrix\*\*:

```python

correlation\_matrix = df[['weight\_g', 'print\_time\_h', 'nozzle\_temp\_C']].corr()

```

- Output: Pearson-correlatiecoëfficiënten (-1 tot 1).

2. \*\*Visualisatie\*\*:

- \*\*Heatmap\*\* (`Seaborn.heatmap`) om sterke relaties te highlighten (bv. hogere temperatuur → kortere printtijd?).

3. \*\*DS-link\*\*:

- Sterke correlaties kunnen \*\*feature selection\*\* sturen voor ML-modellen.

---

## 🎙️ ****Slide 8 – Correlatie & Scatter Plot Analyse****

“We starten hier met een vrij klassieke, maar daarom niet minder krachtige analyse: een **scatter plot gecombineerd met een correlatie-analyse**. Wat je hier ziet, is een visuele voorstelling van de relatie tussen het gewicht van een 3D-geprint object en de bijbehorende printtijd.

Aan de hand van meerdere datapunten, elk representatief voor een andere printopdracht, konden we vaststellen dat er een **duidelijke positieve correlatie** is: zwaardere objecten duren langer om te printen. Klinkt logisch, toch? Maar het interessante zit in de details…”

“Door middel van een lineaire regressie – hier weergegeven met de trendlijn – hebben we een **R²-waarde van 0.82** kunnen berekenen. Dat betekent dat 82% van de variatie in printtijd verklaard kan worden door het gewicht. En dat is behoorlijk stevig. Voor wie vertrouwd is met machine learning: deze regressielijn kan dus ook meteen gebruikt worden in een voorspellend model.”

“Daarnaast heb ik een **clustering-algoritme toegepast (hier K-Means)** om onderliggende patronen op te sporen – bijvoorbeeld bepaalde groepen van prints die onverwacht lang duren ondanks hun lichte gewicht. Dat zou bijvoorbeeld kunnen wijzen op problemen met instellingen, een verkeerd gekozen printmodus of hardwareproblemen.”

“En het mooiste van dit alles: die inzichten kunnen we direct inzetten om **prijsmodellen te optimaliseren**. Want als je op basis van gewicht en tijd al een betrouwbare prijsindicatie kan geven, dan vermijd je onder- of overcalculatie en kun je efficiënter offertes opstellen.”

## 🎙️ ****Slide 9 – Code Deep Dive****

“Als we dan onder de motorkap kijken – en geloof me, dat doen we als technische mensen graag – zien we hoe die analyse opgebouwd is. Ik heb hier gewerkt met Python, uiteraard, en een combinatie van pandas, sklearn, en seaborn voor de visualisatie.

In deze code splits ik de logica mooi op: eerst data cleaning en filtering, daarna de regressie-analyse, en tot slot de clustering. Wat belangrijk is om op te merken: ik behandel uitzonderingen zorgvuldig, ik werk met duidelijke variabelen, en de functies zijn modulair geschreven zodat ze eenvoudig herbruikbaar zijn in andere scripts of als onderdeel van een groter systeem.

Dit stuk code is eigenlijk het hart van het dataprofileringsproces: hier wordt bepaald welke verbanden echt relevant zijn en welke slechts ruis zijn. En dat maakt het verschil tussen een analyse die overtuigt en eentje die gewoon mooi is.”

## 🎙️ ****Slide 10 – Dagelijkse Activiteit Analyse (Heatmap)****

“Deze slide is misschien nog interessanter vanuit een organisatorisch perspectief. Wat we hier gedaan hebben, is een **tijdsanalyse over de dagelijkse en wekelijkse activiteit** van gebruikers of systemen.

De heatmap toont op welke uren van de dag de meeste activiteit plaatsvindt – denk aan jobs die draaien, rapporten die opgevraagd worden of andere interacties met het systeem.

Je ziet heel duidelijk pieken in de namiddag – vooral op woensdag en vrijdag. En dat geeft ons directe stuurinformatie: als je bijvoorbeeld weet dat je systeem op vrijdag tussen 15:00 en 18:00 het zwaarst belast wordt, kun je daar rekening mee houden bij het plannen van onderhoud of bij het inschalen van compute resources.”

“Maar ook op menselijk niveau is dit bruikbaar: bijvoorbeeld om supportteams slimmer in te plannen of om SLA’s af te stemmen op reële piekmomenten. Het is zelfs mogelijk om hiermee gebruikersgedrag te voorspellen en zo tijdig capaciteit op te schalen of functies aan te bieden op het juiste moment.”

## 🎙️ ****Slide 11 – Codeachtergrond****

“Voor wie zich afvraagt hoe zo’n heatmap technisch tot stand komt: hier zie je het script dat ik daarvoor gebruikt heb. We werken opnieuw met pandas voor de aggregaties, matplotlib en seaborn voor de visualisatie.

Let vooral op hoe we hier eerst timestamps groeperen per uur en per weekdag. Daarna zetten we dat om naar een matrix die visueel weergegeven wordt als heatmap. Klein detail: we zorgen ervoor dat ook weekends zichtbaar zijn, zodat afwijkende patronen op zaterdag of zondag niet verloren gaan.”

“En net als bij de correlatieanalyse, is de code opgesplitst in duidelijke stappen: voorbereiden, aggregeren, visualiseren. Zo blijft het schaalbaar, aanpasbaar en klaar voor integratie in grotere dashboards of backends.”

## ****🎙️ Slide 12 – Kwaliteitsanalyse met Boxplot & Z-score****

“In deze analyse kijken we naar kwaliteitsmetingen – en specifieker: naar de **consistentie en uitschieters** in de resultaten van 3D-prints. Want zelfs als het gemiddelde goed lijkt, kunnen afwijkingen flink wat problemen veroorzaken, zeker als het gaat om precisieprints.

De **boxplot** hier toont ons de spreiding van een bepaalde kwaliteitsparameter – denk aan bijvoorbeeld wanddikte of maatafwijking – over meerdere prints. Het mooie van de boxplot is dat je in één oogopslag ziet wat de mediaan is, de interkwartielafstand én welke metingen buiten de verwachte range vallen (outliers dus).”

“Maar om het objectiever te maken, heb ik hier ook gebruikgemaakt van de **Z-score**. Die berekent hoe ver een datapunt afwijkt van het gemiddelde, in standaarddeviaties. Een Z-score van bijvoorbeeld 3 betekent: dit resultaat ligt drie standaarddeviaties boven het gemiddelde – dus hoogstwaarschijnlijk een echte uitschieter, en niet gewoon toevallig een afwijking.”

“Wat blijkt: een paar testseries hebben systematisch outliers. En dat is waardevolle info. Want dan weet je dat je moet gaan kijken naar batchproblemen, filamentkwaliteit, of misschien zelfs een specifieke printer die herhaaldelijk te veel afwijkt.”

## 🎙️ ****Slide 13 – Printfout Classificatie met Decision Tree****

“En dan duiken we nu de wereld van **machine learning** in – maar geen zorgen, we houden het begrijpelijk. Wat je hier ziet is een **decision tree** die getraind is op foutmeldingen en sensorwaardes om te voorspellen welke soort printfout is opgetreden.

Denk aan warping, stringing, nozzle clogs… Al die fouten kunnen veroorzaakt worden door verschillende combinaties van omstandigheden. De decision tree helpt ons om die combinaties te **structureren en interpreteren**.”

“Stel je voor: je hebt een foutmelding ‘underextrusion’ en de temperatuur schommelt. Dan kan het model leren dat dit meestal wijst op verstopte nozzles. Door de boomstructuur zien we **transparant** welke kenmerken het meest bepalend zijn. In tegenstelling tot black-box modellen zoals deep learning, geeft een decision tree je dus **duidelijkheid én uitleg**.”

“In de praktijk gebruiken we dit model om snel de oorzaak van een fout te achterhalen zonder manuele troubleshooting. Het model heeft een accuratesse van meer dan 90%, en als bonus: het helpt ook bij **voorspellend onderhoud**, want we zien fouten al aankomen voordat ze plaatsvinden.”

## 🎙️ ****Slide 14 – Slicing Parameter Analyse & Optimalisatie****

“Tot slot – en dit is er eentje voor de echte printfanaten – de **analyse van slicing parameters**. Want de manier waarop je een model ‘sliced’ (in lagen opsplitst) heeft een enorme impact op de printkwaliteit, snelheid én stabiliteit.

Hier heb ik slicing-instellingen zoals layer height, infill, wall count, en print speed geëvalueerd tegenover de outputkwaliteit en printtijd. We doen dat via een combinatie van **multivariate analyse en visuele feedbackloops**.”

“Het interessante is: sommige instellingen beïnvloeden elkaar kruislings. Bijvoorbeeld: een hogere printsnelheid leidt niet altijd tot lagere kwaliteit – tenzij je ook met dunnere wanden werkt. Dan wél. Door deze afhankelijkheden zichtbaar te maken, kunnen we dus **intelligente presets genereren**.”

“En uiteindelijk gaat het om **slimmer slicen**: voor een bepaald type print kunnen we het systeem automatisch de optimale instellingen laten kiezen. Dat is niet alleen handig voor productiviteit, maar ook voor duurzaamheid – want minder mislukte prints = minder verspilling.”