

# Eindrapport- Hitte

---

Tijn Jonker (500886175), Tjibbe van Dokkum (500867587), Robin van Rooij (500849670)

---

HvA - Minor Data Science – Designing Future Cities  
Opdrachtgever: lectoraat Klimaatbestendige Stad, Stephanie Erwin  
Docent: Pieter Bons  
Datum: 23 januari 2024

# Inhoud

- 
- Doel van het onderzoek
  - Gebruikte data
  - Methode
  - Model
  - Conclusies

# Doel van het onderzoek

---

- Een belangrijke oorzaak van hittestress in de stad is verharde ondergrond.
- De verharding van de ondergrond is op het moment alleen in kaart gebracht voor openbaar terrein, maar nog niet voor privé terrein.

**Hoofdvraag: Hoe kunnen we met openbare data voorspellen of privé gebied verhard of onverhard is?**

---



# Gebruikte data

---

- **Basisregistratie grootschalige topografie (BGT).** We kunnen toegang krijgen tot BGT-data via PDOK:
  - <https://www.pdok.nl/ogc-webservices/-/article/basisregistratie-grootschalige-topografie-bgt->
- **Infrarood luchtfoto (8cm) en RGB luchtfoto (25cm).** De luchtfoto's zijn beschikbaar via het Nationaal GeoRegister (NGR):
  - [https://nationalgeoregister.nl/geonetwork/srv/dut/catalog\\_.search#/metadata/ac33d7d4-10a1-4151-b742-7aa064734a3e](https://nationalgeoregister.nl/geonetwork/srv/dut/catalog_.search#/metadata/ac33d7d4-10a1-4151-b742-7aa064734a3e)



# Verhardingsdata

We hebben BGT data gebruikt om het publieke terrein in te delen als verhard (1) of onverhard (0). Elk terrein heeft een 'Fysiek voorkomen', dat aangeeft hoe verhard het is:

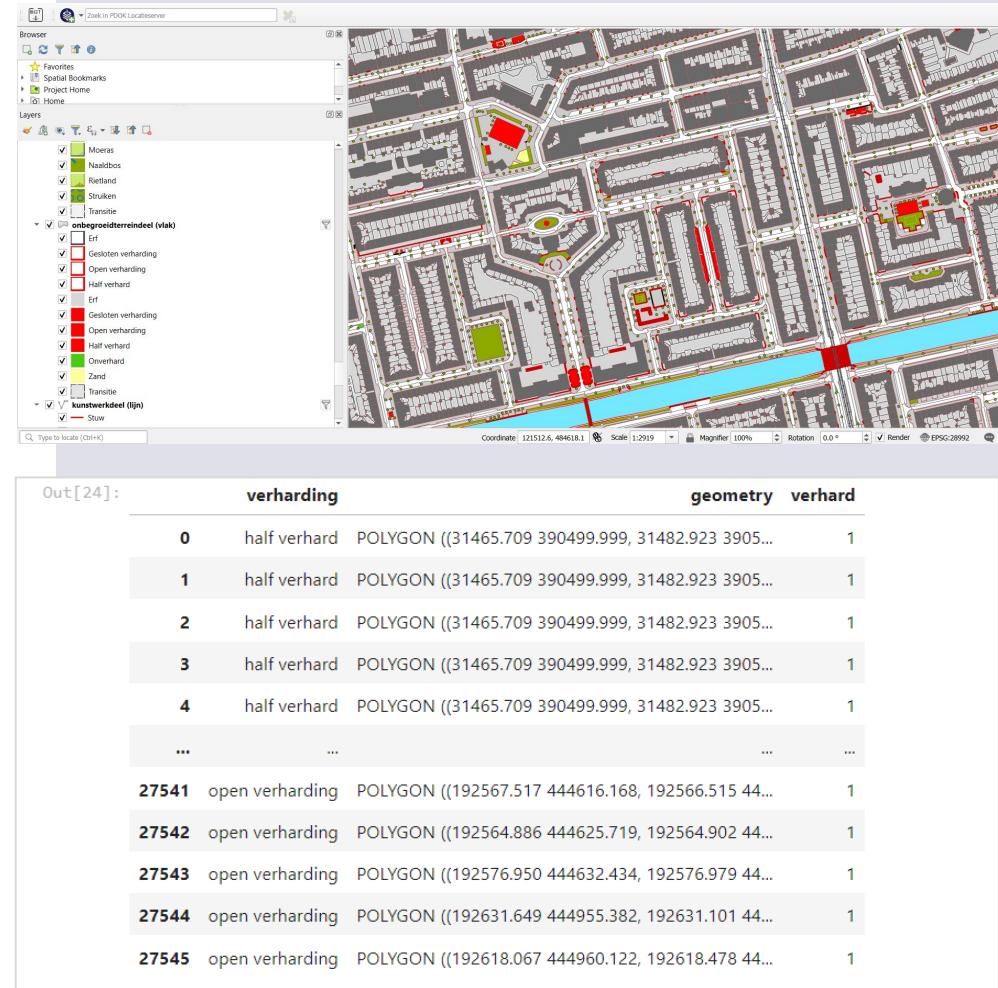
- Gesloten verharding
- Open verharding
- Half verhard
- Onverhard

Alle terreinen die in de eerste drie categorieën vallen zien wij als verhard. Als resultaat hebben we een GeoDataFrame met de columns 'geometry' en 'verhard'.

Ook hebben we de data gecleaned door:

- Het terrein "transitie" (niet verhard of onverhard) uit de dataset te halen
- Het terrein "erf" (privéterrein) uit de dataset te halen
- Alle NaN (missende) waardes te verwijderen

De verhardingsdata kan worden gedownload met QGIS.



## Voorbeelden verharding

### Verhard



### Half verhard



### Onverhard



Deze parkeerplaats heeft het fysieke voorkomen 'verhard' en wordt dus gecategoriseerd als verhard in onze trainingsdata

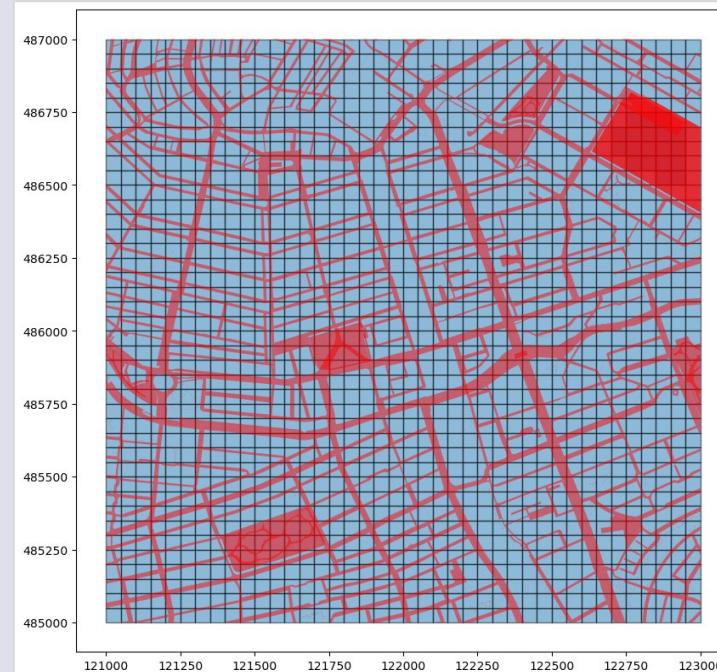
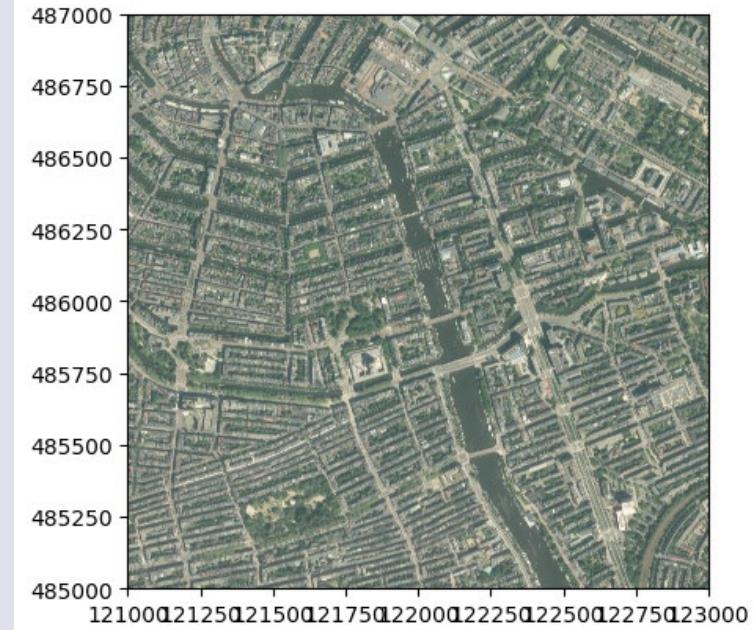
Deze parkeerplaats heeft het fysieke voorkomen 'half verhard' en wordt dus gecategoriseerd als verhard in onze trainingsdata

Deze grasvelden naast de weg hebben het fysieke voorkomen 'onverhard' en wordt dus gecategoriseerd als onverhard in onze trainingsdata

# Zonal statistics

---

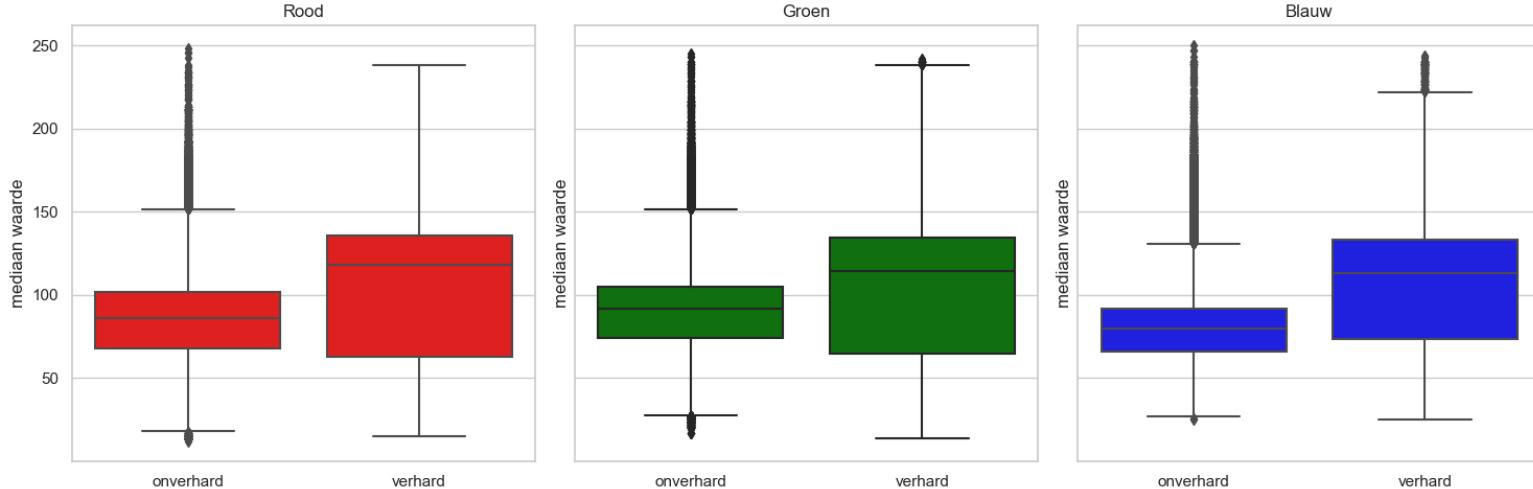
- We hebben de BGT polygonen opgedeeld in een grid van vierkanten van 2 bij 2 meter.
- Daarna hebben we de luchtfoto's ingeladen. Hiermee berekenen we voor elk vierkant de mediaan van de kleurwaardes (r, g, b) uit de foto's (rgb, ir).
- De vierkanten die binnen een polygon van de BGT lagen waarvan de verharding bekend is, hebben deze medianen als waarde meegekregen



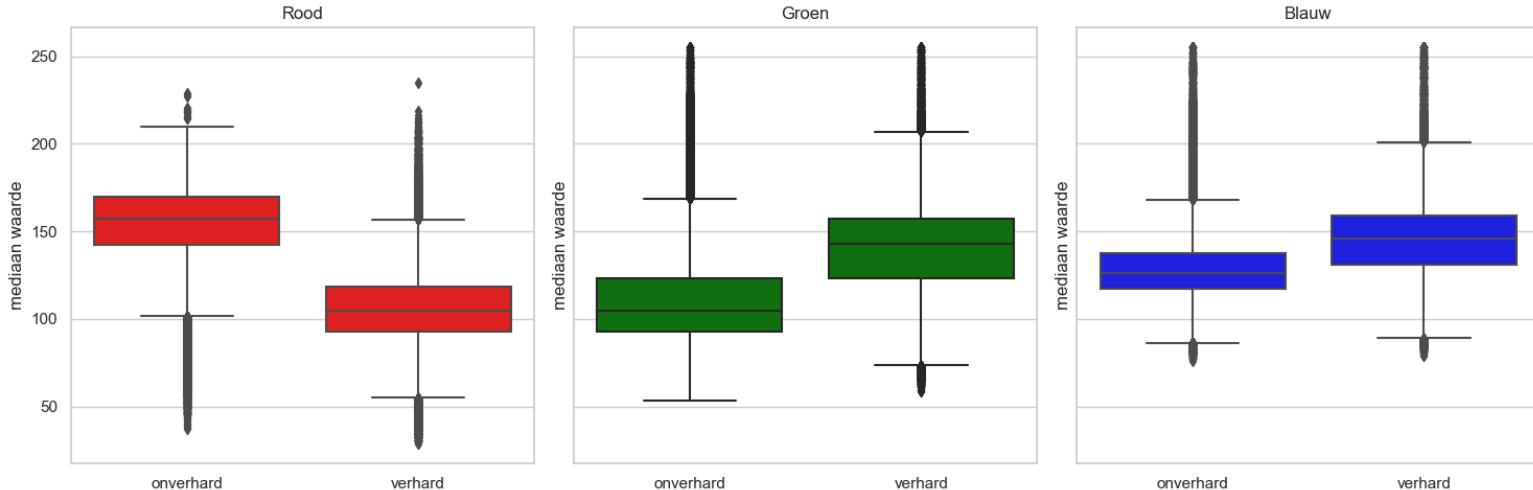
# Verdeling kleurwaarden

- Onverhard terrein heeft hier gemiddeld lager erod en blauw waarden en een kleinere range aan kleuren dan verhard terrein.
- Ons model kan hiervan leren welke kleurwaarden voorkomen bij welk type verharding.
- Er is gekozen om de mediaan waarde te gebruiken omdat deze hoge correlatie heeft met de verharding.
- Er is gekozen om alle waarden mee te nemen (niet alleen groen of rood) omdat dit in praktijk betere voorspellingen maakt voor witte oppervlaktes

Verdeling kleurwaarden Middelburg (ir)



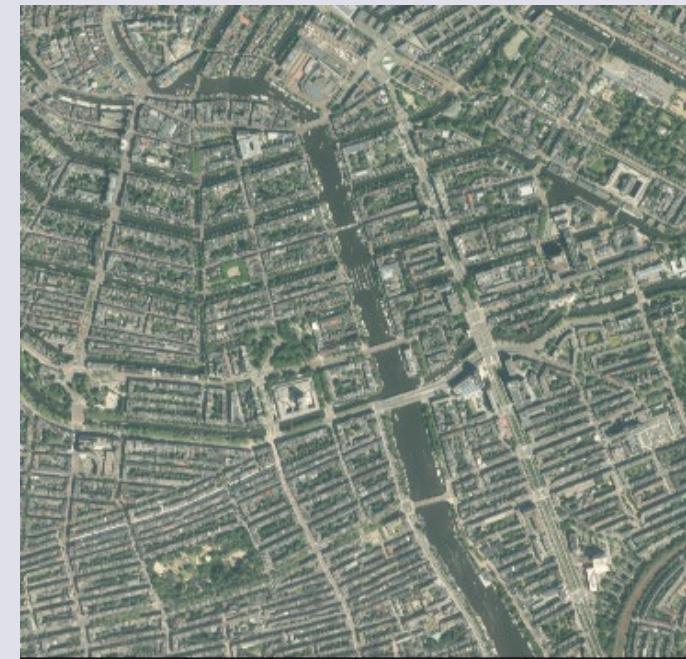
Verdeling kleurwaarden Middelburg (ir)



# Model

---

- Random Forest classifier.
  - We voorspellen de verharding van privé terrein (y) aan de hand van de zonal statistics (de kleuren in de luchtfoto's) (X).
  - Het model is getraind op Amsterdam omdat de kwaliteit van de data hier hoog is
  - De performance van het model is bepaald door de voorspelling voor openbaar terrein te vergelijken met de werkelijke waarde voor openbaar terrein.
  - Het model is redelijk nauwkeurig met een accuracy van 92%.
  - Het model presteert veel beter op het voorspellen van onverhard terrein (F1-score: 95%) dan op het voorspellen van verhard terrein (F1-score: 65%).
  - Precisie: wanneer het model een terrein voorspelt als onverhard is het 93% van de tijd correct. Een voorspelling voor verhard terrein is maar 79% van de tijd correct.
- 



Classification Report:			precision	recall	f1-score	support
	0	1	0.93	0.98	0.95	340020
	1	0	0.79	0.55	0.65	55495
			accuracy		0.92	395515
			macro avg	0.86	0.76	395515
			weighted avg	0.91	0.92	395515

# Middelburg & Arnhem

- Kan ons model dat getraind is op Amsterdam ook goede voorspellingen doen voor Middelburg en Arnhem?
- Het model maakt alleen een goede voorspelling voor andere steden als de verhouding tussen verhard en onverhard terrein vergelijkbaar is.
- Het model was daarom erg slecht in het voorspellen van verhard terrein in Arnhem. Het model kon maar 4% van het verharde terrein correct identificeren.
- Daarom hebben we het model opnieuw getraind op een gebied van twee bij twee kilometer in Middelburg
- Dit nieuwe model gaf een veel beter resultaat: het model kan nu 83% van het verharde terrein correct identificeren voor Arnhem.
- Het nieuwe model voorspelt verhard en onverhard terrein ongeveer even goed



# Wat gaat er fout in het model?

---

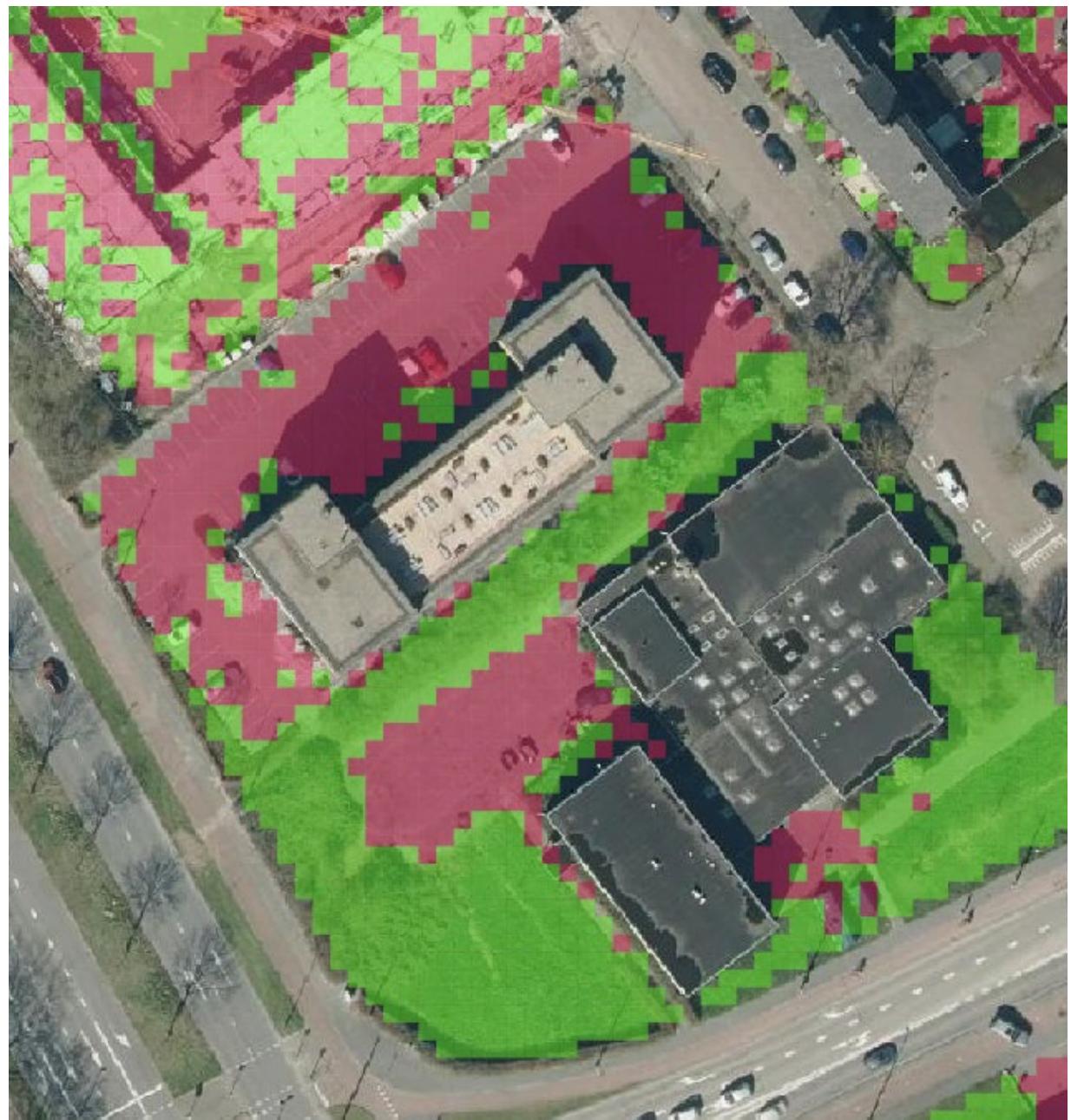
- Bomen worden voorspeld als onverhard, ook als die bijvoorbeeld midden in een parkeerplaats staat.
- De RGB luchtfoto is genomen in de winter, hierdoor heeft landbouw gebied vaak een grijze kleur. Hierdoor is het onterecht voorspeld als verhard terrein.
- Een deel van de data in de BGT is verkeerd gelabeld.
  - Bijvoorbeeld snelwegen die als onverhard zijn geclassificeerd. Hierdoor lijkt het alsof het model een fout maakt die er niet echt is.
  - Ook zijn er onverharde gebieden waar een verhard pad doorheen loopt die toch gelabeld zijn als geheel onverhard.



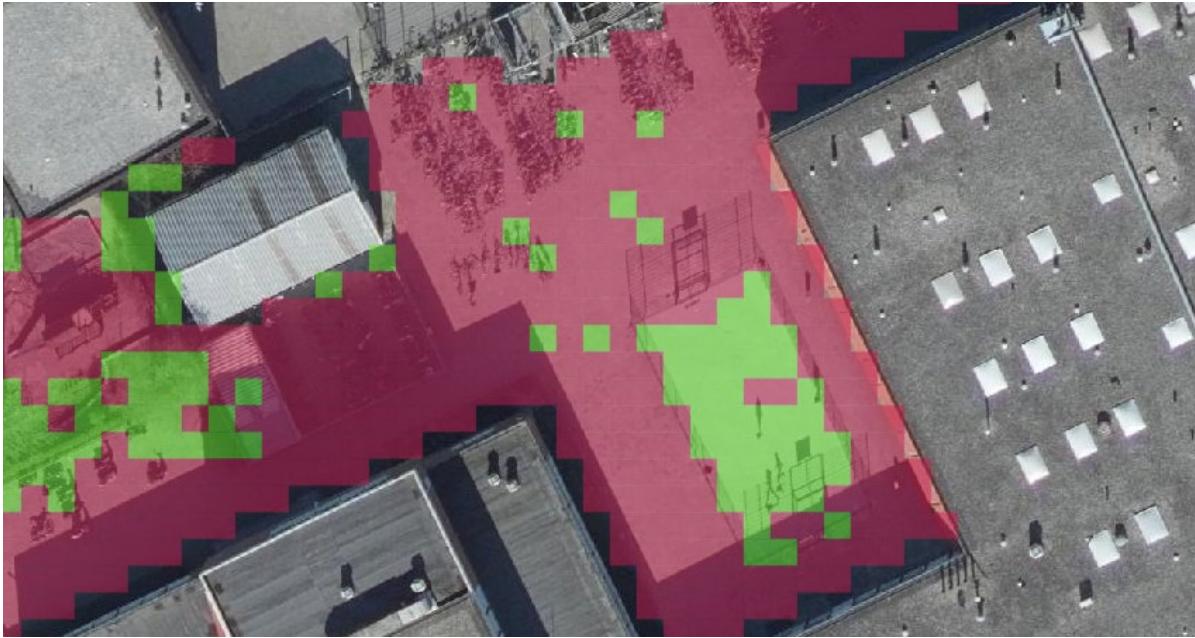
"fout" geclassificeerd als verhard, maar lijkt ook verhard te zijn



## Voorspelling voor privégebied van Arnhem



## Voorspelling voor privégebied van Middelburg



# Conclusies

---

Deze methode lijkt goed te werken voor het voorspellen van de verharding, maar er zijn wel limitaties:

- Het model is beter in het voorspellen van onverharde dan verharde ondergrond.
- Het model dat op Amsterdam getraind is presteert niet goed op een andere stad.
- Het model classificeert elke plek waar een boom staat als onverhard, maar we kunnen aan de hand van luchtfoto's niet zeker weten of de ondergrond onder de boom verhard is of niet.
- We hebben de voorspelling niet kunnen controleren met de echte verharding van het privéterrein omdat dit niet bekend is.

Ondanks de limitaties van het model is het wel goed genoeg om te gebruiken, vooral in het centrum van de stad werkt het model goed.

---



Bedankt voor jullie aandacht  
Zijn er nog vragen?



# Aanvullende slides

- 
- Gebruikte bounding boxes
  - Stappen inladen luchtfoto's en BGT data
  - Code snippets

# Bounding Boxes

---

Tijdens dit project hebben we gewerkt met deze bounding boxes voor de steden Amsterdam, Arnhem en Middelburg:

	wkt_geom	MINX	MINY	MAXX	MAXY
Amsterdam	Polygon ((121000,485000;122000,486000))	121000	485000	122000	486000
Arnhem	Polygon ((183540,438450;196450,454550))	183540	438450	196450	454550
Middelburg	Polygon ((29210,387980;41160,395540))	29210	387980	41160	395540

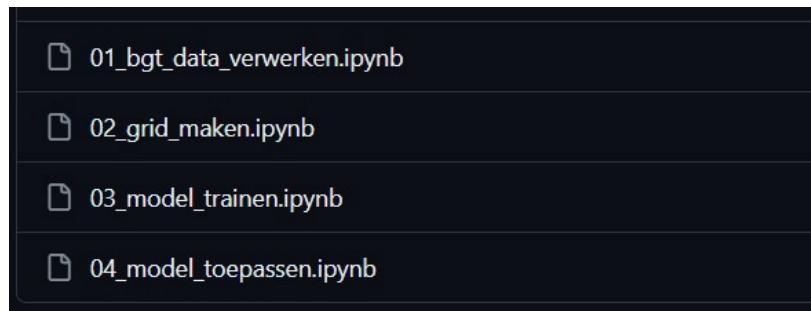
# Reproduceren van de methode

Voor verder onderzoek kan onze code gebruikt worden om het model uit te voeren voor andere gebieden. Hiervoor moet een pad naar de BGT data en luchtfoto's voor het gewenste gebied toegevoegd worden aan onze code, dat ziet er zo uit:

```
In [17]: Path_ir = r"C:\Users\tjibb\Documents\school\jaar_3\minor_2\hitte\bgt_data_2\luchtfotos\middelburg_ir_stad.tif"
Path_gdf = r"C:\Users\tjibb\Documents\school\jaar_3\minor_2\hitte\bgt_data_2\feather_bestanden\middelburg_stad.feather"
Path_rgb = r"C:\Users\tjibb\Documents\school\jaar_3\minor_2\hitte\bgt_data_2\luchtfotos\middelburg_rgb_stad.tif"

pad_naar_waar_ik_het_tijdelijk_wil_opslaan = r"C:\Users\tjibb\Documents\school\jaar_3\minor_2\hitte\bgt_data_2\feather_bestanden\
pad_naar_waar_ik_het_wil_opslaan = r"C:\Users\tjibb\Documents\school\jaar_3\minor_2\hitte\bgt_data_2\feather_bestanden\grid_middelburg_stad.feather"
```

Als vervolgens alle code uitgevoerd wordt is de output een feather bestand wat in QGIS geladen kan worden om het resultaat te zien.  
Deze code bestaat uit vier stappen:



In de volgende slides meer informatie over het ophalen van de BGT data en luchtfoto's met behulp van QGIS.

# Gebruikte bibliotheken

## 1. Pandas (``import pandas as pd``):

- Doel: Data-analyse en manipulatie.
- Belangrijkste functies: DataFrame-structuur voor het werken met tabulaire gegevens, gegevensselectie en -filtering.

## 2. NumPy (``import numpy as np``):

- Doel: Numerieke bewerkingen en wiskundige functies.
- Belangrijkste functies: Arrays voor efficiënte bewerkingen op numerieke gegevens, lineaire algebra, statistiek, enz.

## 3. GeoPandas (``import geopandas as gpd``):

- Doel: Geospatiale gegevensstructuren en -bewerkingen.
- Belangrijkste functies: Uitbreiding van Pandas voor het werken met geometrische gegevens, zoals punten, lijnen en polygonen.

## 4. Rasterio (``import rasterio``):

- Doel: Lezen en schrijven van rastergegevens (bijvoorbeeld afbeeldingen in geografische context).
- Belangrijkste functies: Functionaliteit voor het werken met georeferenteerde rastergegevens.

## 5. Shapely (``from shapely.geometry import Polygon``):

- Doel: Geometrische operaties op geometrische objecten.
- Belangrijkste functies: Ondersteunt de manipulatie van geometrische vormen zoals punten, lijnen en polygonen.

## 6. Rasterstats (``from rasterstats import zonal_stats``):

- Doel: Statistische berekeningen uitvoeren op rastergegevens binnen zones van geometrische objecten.
- Belangrijkste functies: Bereken statistieken (bijvoorbeeld gemiddelde, som) voor rasterwaarden binnen polygoonzones.

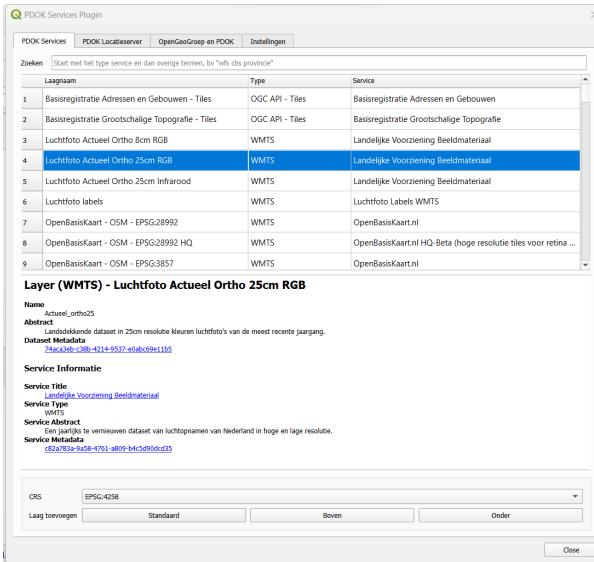
## 7. Scikit-learn (``from sklearn.ensemble import RandomForestClassifier``, ``from sklearn.metrics import accuracy_score, classification_report``):

- Doel: Machine learning en data-analyse.
- Belangrijkste functies: Implementeert verschillende machine learning-algoritmen, metrische functies voor evaluatie van modellen.

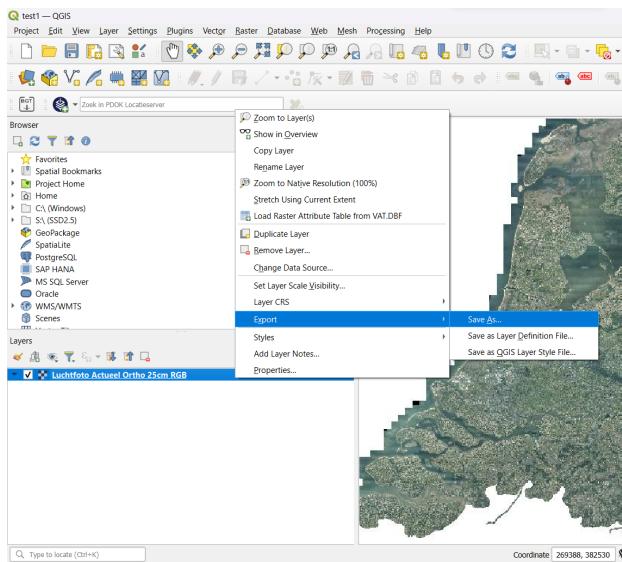
## 8. Joblib (``import joblib``):

- Doel: Parallelle uitvoering van taken en efficiënt opslaan/laden van Python-objecten.
- Belangrijkste functies: Parallelle uitvoering van functies, opslaan en laden van Python-objecten.

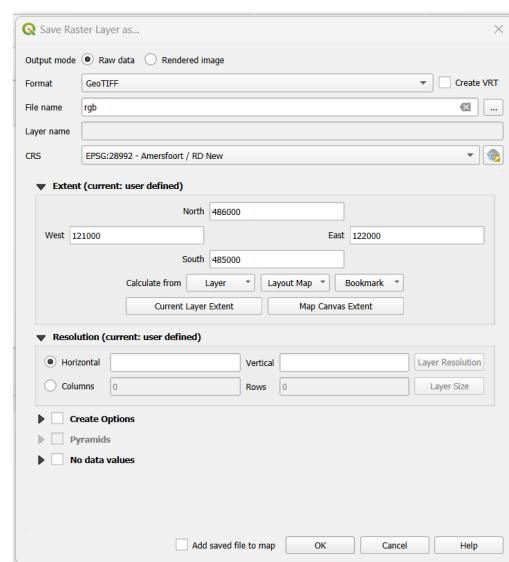
# Inladen Luchtfoto's (met QGIS)



1) Start met het inladen van de data in QGIS, bijvoorbeeld met behulp van de PODOK Services Plugin. Selecteer een actuele luchtfoto in RGB of IR.



2) Selecteer de laag en kies Exporteren.



3) Vul de bounding boxes van de gewenste locatie in en exporteer de laag als een GeoTIFF bestand.

```
import pandas as pd
import numpy as np
import geopandas as gpd
import rasterio
from shapely.geometry import Polygon
from rasterio import features

Path_ir = r'C:\Users\tjibb\Documents\school\jaar_3\minor_2\white\bgt_data_2\luchtfotos\middenburg_ir_stad.tif'
Path_gdf = r'C:\Users\tjibb\Documents\school\jaar_3\minor_2\white\bgt_data_2\feather_bestanden\middenburg_stad.feather'
Path_rgb = r'C:\Users\tjibb\Documents\school\jaar_3\minor_2\white\bgt_data_2\luchtfotos\middenburg_rgb_stad.tif'

pad_naar_waar_Ik_het_tijdelijk_wil_opslaan = r'C:\Users\tjibb\Documents\school\jaar_3\minor_2\white\bgt_data_2\feather_bestanden\middenburg_stad.feather'
pad_naar_waar_Ik_het_tijdelijk_wil_opslaan = r'C:\Users\tjibb\Documents\school\jaar_3\minor_2\white\bgt_data_2\feather_bestanden\middenburg_stad.feather'
pad_naar_waar_Ik_het_will_opslan = r'C:\Users\tjibb\Documents\school\jaar_3\minor_2\white\bgt_data_2\feather_bestanden\middenburg_stad.feather'

rgb = rasterio.open(Path_rgb)
ir = rasterio.open(Path_ir)

gdf = gpd.read_feather(Path_gdf)

transform = rgb.transform

width = rgb.width
height = rgb.height
x_t1, y_t1 = transform * (0, 0)
x_b1, y_b1 = transform * (width, height)

Groote_vierkant_x = 2
Groote_vierkant_y = 2

santal_vierkant_x = (x_b1-x_t1)/Groote_vierkant_x
santal_vierkant_y = (y_b1-y_t1)/Groote_vierkant_y

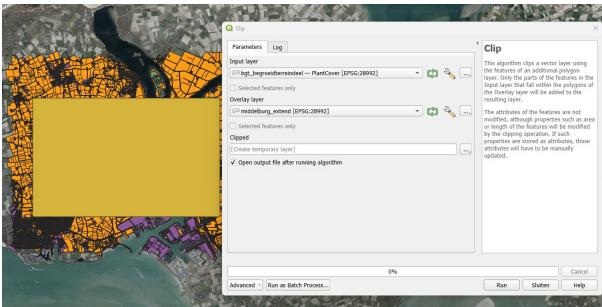
grid = []
for i in range(int(santal_vierkant_x)):
    for j in range(int(santal_vierkant_y)):
        polygon = Polygon([(x_t1+(Groote_vierkant_x*i), y_t1+Groote_vierkant_y*j),
                           (x_t1+(Groote_vierkant_x*(i+1)), y_t1+Groote_vierkant_y*j),
                           (x_t1+(Groote_vierkant_x*(i+1)), y_t1+(Groote_vierkant_y*(j+1))),
                           (x_t1+(Groote_vierkant_x*i), y_t1+(Groote_vierkant_y*(j+1)))])
```

4) Voeg het pad naar je bestand toe aan de variabelen 'Path\_rgb' of 'Path\_ir' in het document '02\_grid\_maken', en run de rest van de code.

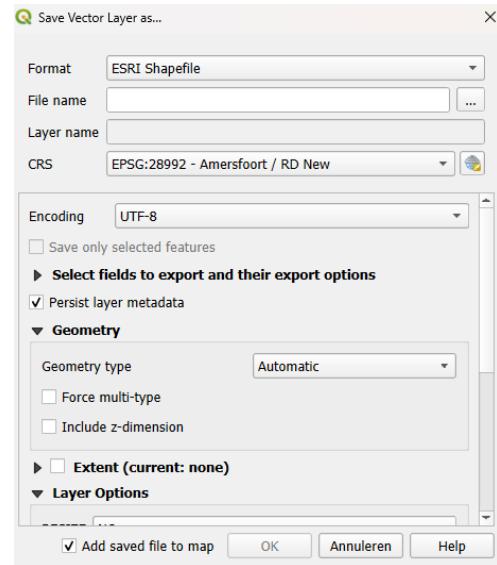
# Inladen BGT data (met QGIS)



1) Maak een selectie van het gewenste gebied in de PDOK BGT downloadviewer om de BGT data te downloaden: <https://app.pdok.nl/lv/bgt/download-viewer/>



2) Maak een clip met dezelfde bounding box als de luchtfoto



3) Exporteer de clip als een Shapefile. En geef het bestand een goede naam zodat het makkelijk terug te vinden is

```
import pandas as pd
import numpy as np
import geopandas as gpd
import rasterio
from shapely.geometry import Polygon
from rasterstats import zonal_stats

path_ir = "C:/Users/tjibb/Documents/school/jaar_3/minor_2/luchtfotos/middelburg_ir_stad.tif"
path_gdf = "C:/Users/tjibb/Documents/school/jaar_3/minor_2/bgt/bgt_data_2/feather_bestanden/middelburg_stad.feather"
path_rgb = "C:/Users/tjibb/Documents/school/jaar_3/minor_2/luchtfotos/middelburg_rgb_stad.tif"

pad_naar_waar_ik_het_tijdelijk_wil_opslaan = "C:/Users/tjibb/Documents/school/jaar_3/minor_2/bgt/bgt_data_2/feather/bestanden/pad_naar_waar_ik_het_tijdelijk_wil_opslaan"
pad_naar_waar_ik_het_tijdelijk_wil_opslaan = "C:/Users/tjibb/Documents/school/jaar_3/minor_2/luchtfotos/middelburg_rgb_stad.tif"

rgb = rasterio.open(path_rgb)
ir = rasterio.open(path_ir)

gdf = gpd.read_feather(path_gdf)

transform = rgb.transform

width = rgb.width
height = rgb.height
x_t1, y_t1 = transform * (0, 0)
x_b1, y_b1 = transform * (width, height)

Groote_vierkant_x = 2
Groote_vierkant_y = 2

santai_vierkant_x = (x_b1-x_t1)/Groote_vierkant_x
santai_vierkant_y = (y_b1-y_t1)/Groote_vierkant_y

grid = []
for i in range(int(santai_vierkant_x)):
    for j in range(int(santai_vierkant_y)):
        polygon = Polygon([(x_t1+Groote_vierkant_x*i), y_t1+Groote_vierkant_y*j])
        grid.append(polygon)
```

4) Voeg het pad naar je bestand toe aan de path variabelen (zoals 'pad\_naar\_wegdeel\_data') in het document '01\_bgt\_data\_verwerk en', en run de rest van de code.

# 01 - Verwerken BGT data

In [21]:

```
wegdeel = wegdeel[["surfaceMat", "geometry"]]
begroeidterreindeel = begroeidterreindeel[["class", "geometry"]]
onbegroeidterreindeel = onbegroeidterreindeel[["bgt-fysiek", "geometry"]]
ondersteunendwegdeel = ondersteunendwegdeel[["surfaceMat", "geometry"]]

wegdeel["verharding"] = wegdeel["surfaceMat"]
wegdeel = wegdeel[["verharding", "geometry"]]

begroeidterreindeel["verharding"] = begroeidterreindeel["class"]
begroeidterreindeel = begroeidterreindeel[["verharding", "geometry"]]

onbegroeidterreindeel["verharding"] = onbegroeidterreindeel["bgt-fysiek"]
onbegroeidterreindeel = onbegroeidterreindeel[["verharding", "geometry"]]

ondersteunendwegdeel["verharding"] = ondersteunendwegdeel["surfaceMat"]
ondersteunendwegdeel = ondersteunendwegdeel[["verharding", "geometry"]]

bgt = pd.concat([wegdeel, begroeidterreindeel, onbegroeidterreindeel, ondersteunendwegdeel], ignore_index=True)
bgt = bgt[bgt['verharding'] != "transitie"]
```

In [22]:

```
# Define the conditions
conditions = ['half verhard', 'gesloten verharding', 'open verharding']

# Create a new column 'verhard' with values 1 if 'verharding' is in the specified conditions, else 0
bgt['verhard'] = np.where(bgt['verharding'].isin(conditions), 1, 0)

# Define a function to apply the conditions
def is_verhard(row):
    return 1 if row['verharding'] in conditions else 0
```

In [23]:

```
# Create a new column 'verhard' by applying the function to each row
bgt['verhard'] = bgt.apply(is_verhard, axis=1)
```

In [24]:

```
bgt
```

Out[24]:

	verharding	geometry	verhard
0	half verhard	POLYGON ((31465.709 390499.999, 31482.923 3905...	1
1	half verhard	POLYGON ((31465.709 390499.999, 31482.923 3905...	1
2	half verhard	POLYGON ((31465.709 390499.999, 31482.923 3905...	1

# 02 - Grid maken

```
In [8]: width = rgb.width
height = rgb.height
x_tl, y_tl = transform * (0, 0)
x_br, y_br = transform * (width, height)

In [9]: Groote_vierkant_x = 2
Groote_vierkant_y = 2

In [10]: aantal_vierkant_x = (x_br-x_tl)/Groote_vierkant_x
aantal_vierkant_y = (y_tl-y_br)/Groote_vierkant_y

In [11]: grid = []
for i in range(int(aantal_vierkant_x)):
    for j in range(int(aantal_vierkant_y)):
        polygon = Polygon([(x_tl+Groote_vierkant_x*i), y_tl-Groote_vierkant_y*j),
                           (x_tl+Groote_vierkant_x*(i+1)), y_tl-Groote_vierkant_y*j),
                           (x_tl+Groote_vierkant_x*(i+1)), y_tl-Groote_vierkant_y*(j+1)),
                           (x_tl+Groote_vierkant_x*i), y_tl-Groote_vierkant_y*(j+1))]
        grid.append(polygon)

In [12]: gdf_grid = gpd.GeoDataFrame(geometry=grid, crs = "EPSG:28992")

In [13]: gdf_grid
```

	geometry
0	POLYGON ((29850.000 393700.000, 29852.000 3937...
1	POLYGON ((29850.000 393698.000, 29852.000 3936...
2	POLYGON ((29850.000 393696.000, 29852.000 3936...
3	POLYGON ((29850.000 393694.000, 29852.000 3936...
4	POLYGON ((29850.000 393692.000, 29852.000 3936...

```
In [ ]: joined = gpd.sjoin(gdf[["geometry",'verharding','verhard']],gdf_grid, predicate='contains', how ='right')

In [ ]: gdf_cleaned = joined.dropna()

In [ ]: len(gdf_cleaned[gdf_cleaned['verharding']!= 'erf'])

Out[ ]: 10575695

In [ ]: gdf_Openbaar = gdf_cleaned[gdf_cleaned['verharding']!='erf']

In [ ]: gdf_Openbaar
```

	index_left	verharding	verhard	geometry
0	39653.0	grasland agrarisch	0.0	POLYGON ((29850.000 393700.000, 29852.000 3937...
0	39657.0	grasland agrarisch	0.0	POLYGON ((29850.000 393700.000, 29852.000 3937...
0	39658.0	grasland agrarisch	0.0	POLYGON ((29850.000 393700.000, 29852.000 3937...
0	39659.0	grasland agrarisch	0.0	POLYGON ((29850.000 393700.000, 29852.000 3937...
0	39696.0	grasland agrarisch	0.0	POLYGON ((29850.000 393700.000, 29852.000 3937...

```
In [ ]: with rasterio.open(Path_rgb) as rds_rgb:
    x_rgb = rds_rgb.read()
    print(rds_rgb.meta)

{'driver': 'GTiff', 'dtype': 'uint8', 'nodata': None, 'width': 20400, 'height': 18600, 'count': 4, 'crs': CRS.from_epsg(2899
2), 'transform': Affine(0.25, 0.0, 29850.0,
                      0.0, -0.25, 393700.0)}

In [ ]: # Open the infrared raster dataset and read the pixel values
with rasterio.open(Path_ir) as rds_infrared:
    x_infrared = rds_infrared.read()
    print(rds_infrared.meta)

{'driver': 'GTiff', 'dtype': 'uint8', 'nodata': None, 'width': 20400, 'height': 18600, 'count': 4, 'crs': CRS.from_epsg(2899
2), 'transform': Affine(0.25, 0.0, 29850.0,
                      0.0, -0.25, 393700.0)}

In [ ]: def calculate_stats(raster_path, gdf, column_prefix):
    with rasterio.open(raster_path) as raster:
        rgb_data = raster.read()
        rgb_band_names = ['r', 'g', 'b']
        stats = [ 'median']

        # Calculate the average for each RGB band
        for band_num, band_name in enumerate(rgb_band_names, start=1):
            band_data = rgb_data[band_num - 1, :, :]
            result = zonal_stats(gdf, band_data, affine=raster.transform, stats=stats, nodata=raster.nodata)

            # Add the new columns using a Loop
            for stat in stats:
                column_name = f'{column_prefix}_{band_name}_{stat}'
                gdf[column_name] = [feature[stat] for feature in result]
```

```
In [ ]: calculate_stats(Path_rgb, gdf_grid, 'rgb')
```

# 03 - Model trainen

```
In [35]: bottom_50_percent = int(0.5 * len(gdf_model_A))

# Select the bottom 50% of the DataFrame by index
X_test_A = gdf_model_A.iloc[-bottom_50_percent:][['rgb_r_median', 'rgb_g_median', 'rgb_b_median','ir_r_median', 'ir_g_median']]
y_test_A = gdf_model_A.iloc[-bottom_50_percent:]['verhard']

top_50_percent = int(0.5 * len(gdf_model_A))

# Select the top 50% of the DataFrame by index
X_train_A = gdf_model_A.iloc[:top_50_percent][['rgb_r_median', 'rgb_g_median', 'rgb_b_median','ir_r_median', 'ir_g_median']]
y_train_A = gdf_model_A.iloc[:top_50_percent]['verhard']

In [36]: bottom_50_percent = int(0.5 * len(gdf_model_M))

# Select the bottom 50% of the DataFrame by index
X_test_M = gdf_model_M.iloc[-bottom_50_percent:][['rgb_r_median', 'rgb_g_median', 'rgb_b_median','ir_r_median', 'ir_g_median']]
y_test_M = gdf_model_M.iloc[-bottom_50_percent:]['verhard']

top_50_percent = int(0.5 * len(gdf_model_M))

# Select the top 50% of the DataFrame by index
X_train_M = gdf_model_M.iloc[:top_50_percent][['rgb_r_median', 'rgb_g_median', 'rgb_b_median','ir_r_median', 'ir_g_median']]
y_train_M = gdf_model_M.iloc[:top_50_percent]['verhard']

In [37]: X_test = pd.concat([X_test_M,X_test_A])
y_test = pd.concat([y_test_M,y_test_A])
X_train = pd.concat([X_train_M,X_train_A])
y_train = pd.concat([y_train_M,y_train_A])

In [43]: # Create a Random Forest classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model on the training set
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)

# Display results
print(f'Accuracy: {accuracy:.2f}')
print('\nClassification Report:')
print(classification_report_str)

In [44]: X = pd.concat([gdf_model_M[['rgb_r_median', 'rgb_g_median','rgb_b_median','ir_r_median', 'ir_g_median', 'ir_b_median']]]
y = pd.concat([gdf_model_M['verhard'],gdf_model_A['verhard']])

In [45]: #nu het model trainen op de gehele dataset
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model on the training set
model.fit(X, y)

Out[45]: RandomForestClassifier(random_state=42)
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [46]: print(f'Accuracy: {accuracy:.2f}')
print('\nClassification Report:')
print(classification_report_str)

Accuracy: 0.92
Classification Report:
precision    recall   f1-score   support
          0       0.93      0.98      0.95     340020
          1       0.79      0.55      0.65     55495

   accuracy                           0.92     395515
    macro avg       0.86      0.76      0.80     395515
weighted avg       0.91      0.92      0.91     395515

In [51]: joblib.dump(model, pad_naar_waar_het_model_moet_worden_opgeslagen)

Out[51]: ['C:\\\\Users\\\\tjibb\\\\Documents\\\\school1\\\\jaar_3\\\\minor_2\\\\hitte\\\\bgt_data_2\\\\model\\\\varnhem_en_middelburg_model.joblib']
```

# 04 - Model toepassen

data inladen en verwerken en laten zien

```
In [11]: bgt = gpd.read_feather(pad_naar_bgt_data)
```

```
In [6]: gdf = gpd.read_feather(pad_naar_grids_bestand)
```

```
In [12]: joined = gpd.sjoin(bgt[["geometry", "verharding", "verhard"]], gdf, predicate='contains', how ='right')
```

```
In [29]: gdf_cleaned = joined.dropna()
gdf_cleaned = gdf_cleaned.drop_duplicates(subset=['geometry'])
```

```
In [34]: gdf_cleaned.duplicated(subset = ['geometry']).sum()
```

```
Out[34]: 0
```

```
In [33]: gdf_prive = gdf_cleaned[gdf_cleaned['verharding']=='erf']
```

```
In [35]: gdf_prive = gdf_prive.drop(columns='verhard')
```

```
In [37]: model = joblib.load(pad_naar_model)
```

model toepassen

```
In [39]: voorspelling = model.predict(gdf_prive[['rgb_r_median', 'rgb_g_median', 'rgb_b_median', 'ir_r_median', 'ir_g_median', 'ir_b_
```

```
In [40]: gdf_prive['verhard'] = voorspelling
```

```
In [42]: gdf_prive.to_feather(pad_naar_waar_het_resultaat_moet_komen)
```