



End-to-end ML project



Lesmateriaal

- Via GitHub account:
- git clone https://github.com/oclaerbout/syntra_data_scientist.git
- © 2023 Olivier Claerbout, Tim Hellemans
- *Dit materiaal mag niet worden gekopieerd, verspreid, gepubliceerd of anderszins gereproduceerd zonder uitdrukkelijke schriftelijke toestemming van de auteur. Dit geldt eveneens voor de Jupyter Notebooks!*

Stappenplan

1. Grotere plaatje bekijken waarin de vraag zich bevindt.
2. Data verzamelen.
3. Exploratief onderzoek (met visualizaties) uitvoeren.
4. Beslis of ML de juiste aanpak is voor jouw probleem.
5. Data prepareren voor ML algoritmes.
6. Model selecteren en trainen.
7. Jouw model finetunen.
8. Presenteer je resultaten.
9. Oplossing lanceren, in het oog houden & blijven ondersteunen.

Dit kan je
al!

Dit zijn we
nu aan
het leren



California huizenprijzen

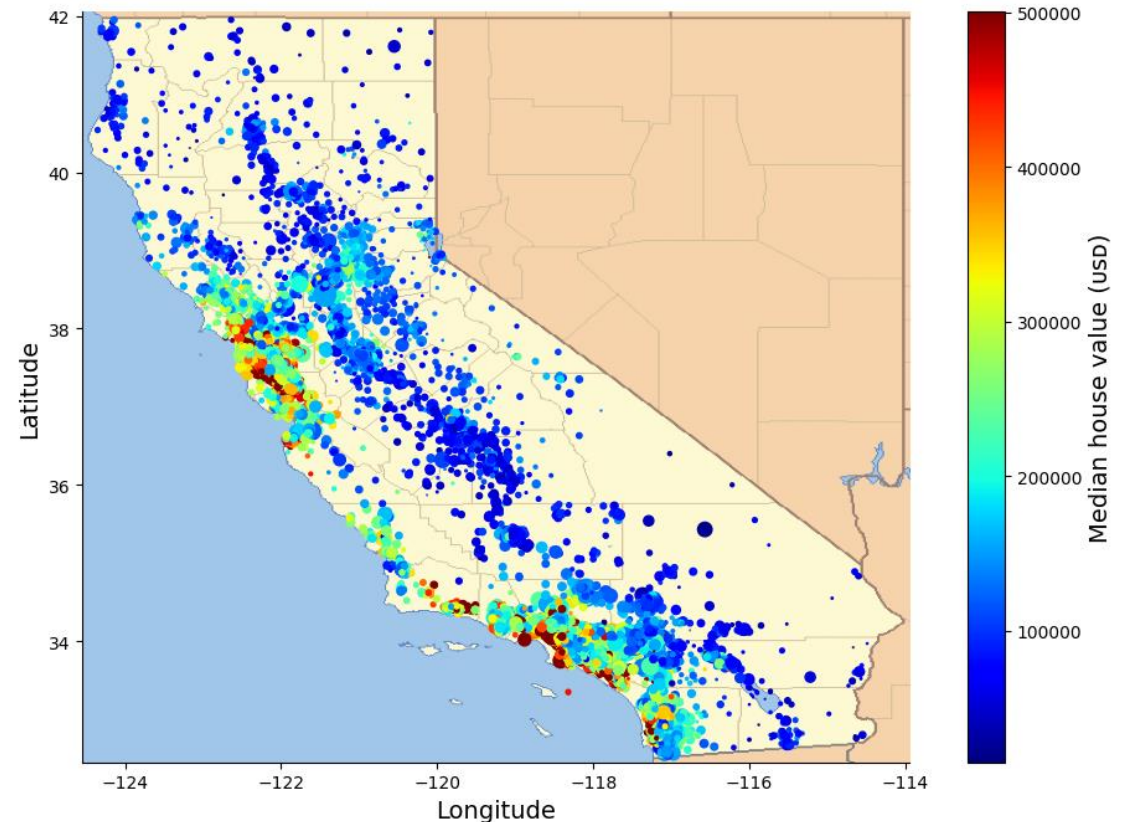
Bron: Dit slide deck vindt zijn inspiratie in het boek:

[Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems : Géron, Aurélien: Amazon.com.be: Books](#)

Huizenprijzen voorspellen

Overzicht:

- **Target:** (Mediane) huizenprijs voorspellen in districten.
- **Features:**
 - Locatie (longitude, latitude),
 - Aantal inwoners,
 - Mediaan loon,
 - ...
- **Granulariteit:** districten van 600-3000 mensen.



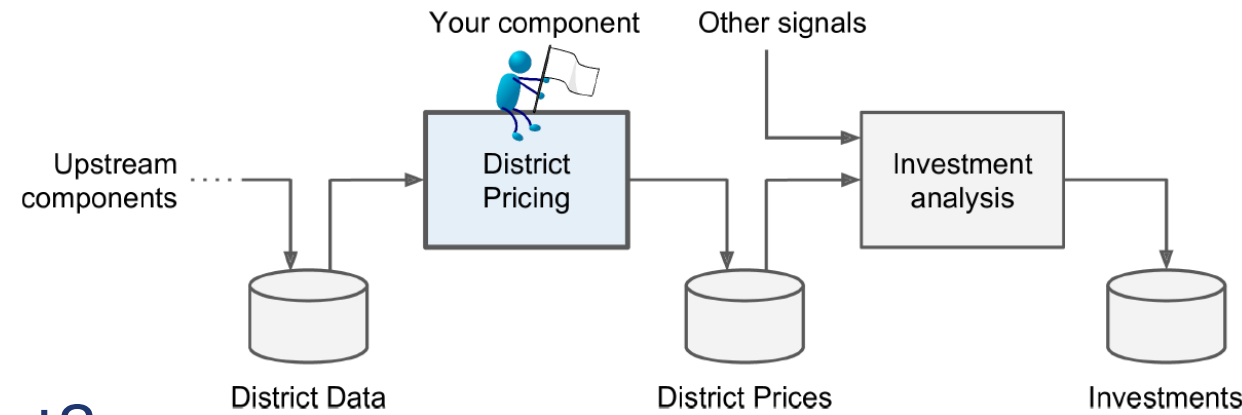
Groter plaatje

Frame het probleem



Frame het probleem

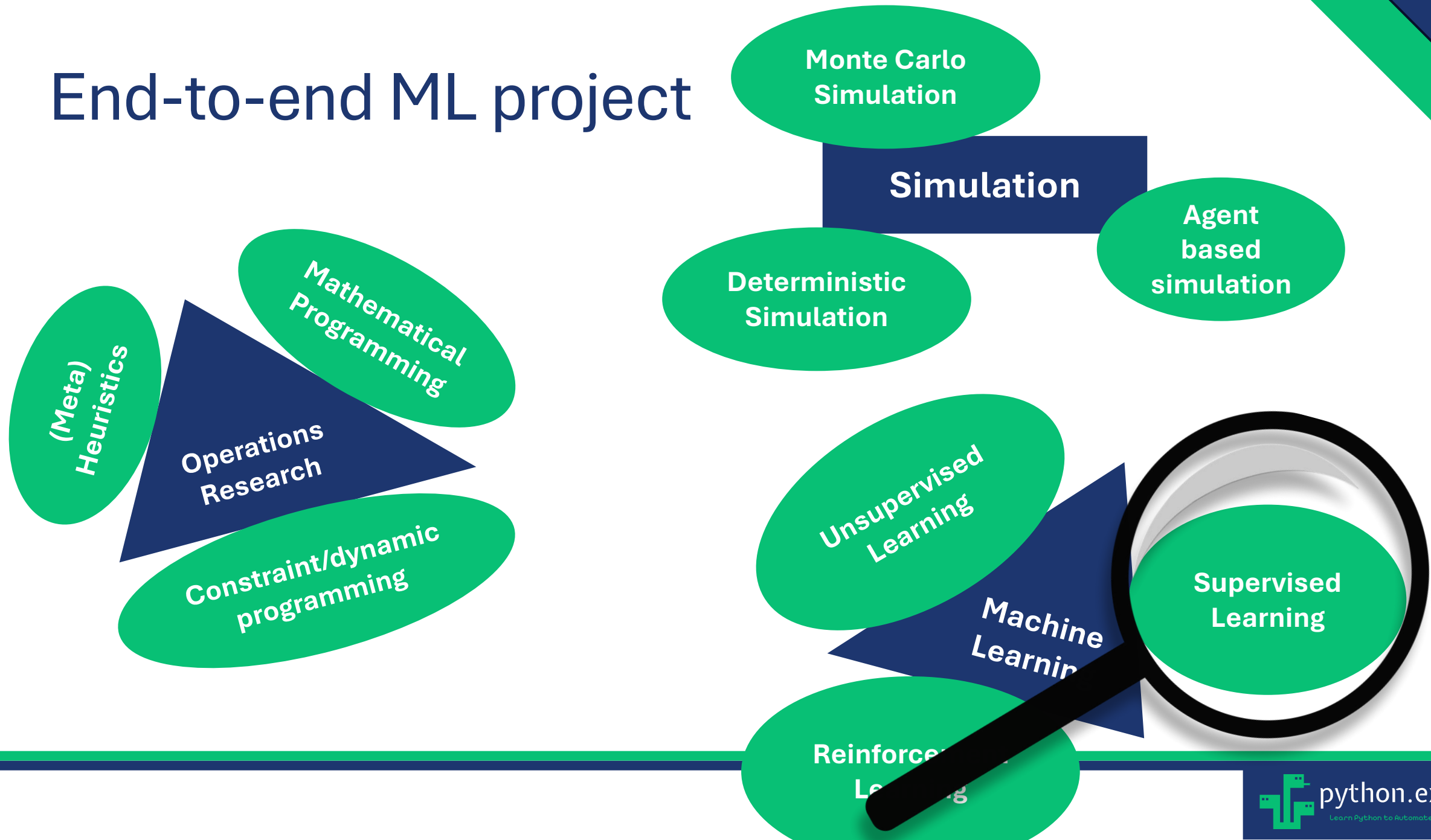
- **Q:** Wat ga je doen met het resultaat?
- **A:** Als input voor investeringsbeslissingen.
- **Q:** Hoe wordt dit nu gedaan?
- **A:** Manueel door een groep experts.
- **Q:** Waarom willen we dit wijzigen?
- **A:** Kost veel tijd & niet erg accuraat (estimates tot 30% afwijkend).



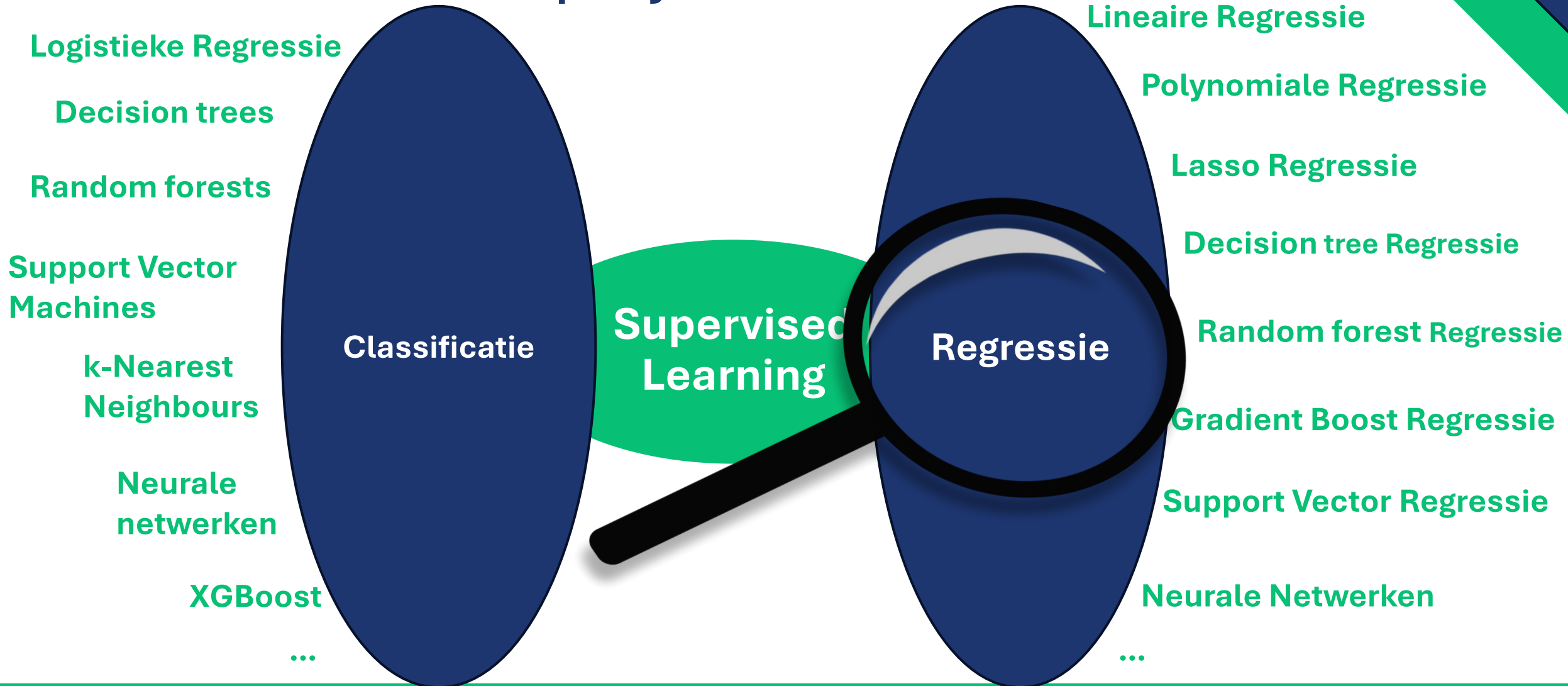
Een aantal keuzes maken

- ML, simulatie of operations research?
 - ML
- Supervised, unsupervised, reinforcement learning of semi-supervised?
 - Supervised
- Regressie of classificatie
 - Regressie
- Batch learning of online learning?
 - Batch learning

End-to-end ML project

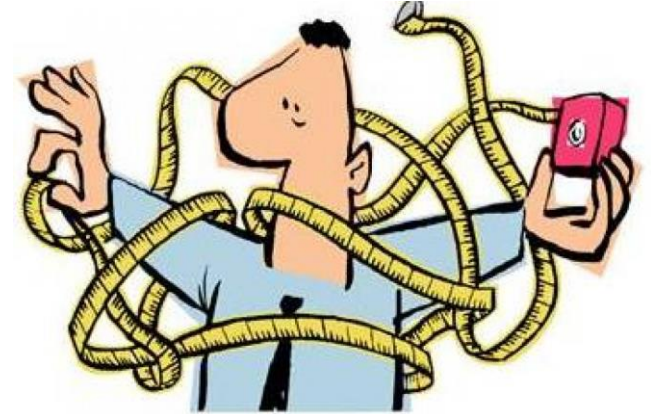


End-to-end ML project



Groter plaatje

Kies een performance measure



Error measures

- **MAPE:** Zeer intuïtieve measure, geeft je gemiddelde procentuele fout, ideaal voor rapportering.

Let wel op: deze werkt **niet** goed met erg kleine **y**-values.

- **MAE:** Goed alternatief op MAPE, maar moet je steeds vergelijken met \bar{x} .

Je kan bv. altijd het “aantal gemiddelden” dat je verwacht af te wijken: $\frac{MAE}{\bar{x}}$.

- **MSE:** Deze geeft meer gewicht aan grove fouten. Wordt vaak gebruikt voor training en dus goede measure voor gebruik tijdens modellering. **Niet gebruiken richting business.**
- **RMSE:** De MSE is kwadratisch tov \bar{x} , daarom trekken we de wortel om meer intuïtieve getallen te krijgen.

Error measures

Voor regressie taken is de “standaard keuze” de **MSE**.

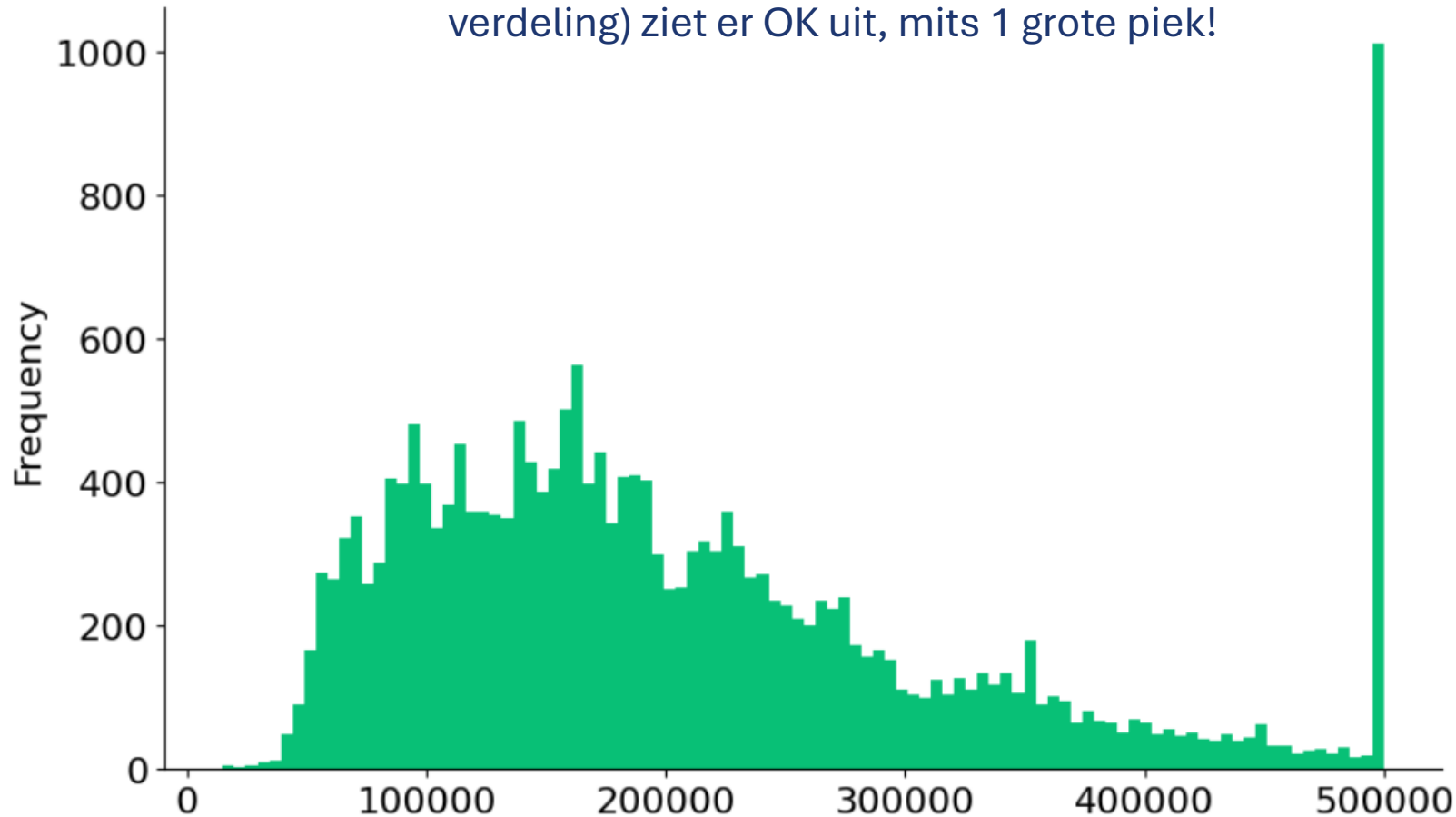
Deze geeft wel meer gewicht dus even controleren of het aantal outliers exponentieel afvalt. Dit betekent dat voor grote waarden van n :

$$P(X > n) \leq e^{-n}$$

Dit wil zeggen dat het aantal datapunten groter dan n exponentieel daalt.

Error measure

Exponentieel verval (zoals een normale verdeling) ziet er OK uit, mits 1 grote piek!



3 opties:

- Meer precieze prijzen zoeken voor deze huizen,
- datapunten uit de dataset halen of
- Klasse maken $> 500\ 000$.

De data bekijken

Exploratief



Data bekijken

`housing.sample(5)`

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
-120.66	40.41	52.0	2081.0	478.0	1051.0	419.0	2.2992	70200.0	INLAND
-116.57	35.43	8.0	9975.0	1743.0	6835.0	1439.0	2.7138	22500.0	INLAND
-117.07	34.24	21.0	4773.0	1047.0	337.0	130.0	3.9375	115000.0	INLAND
-122.57	37.98	49.0	2860.0	552.0	1178.0	522.0	4.6250	355000.0	NEAR BAY
-118.36	34.16	45.0	1755.0	335.0	822.0	342.0	5.1423	322900.0	<1H OCEAN

`housing.info()`

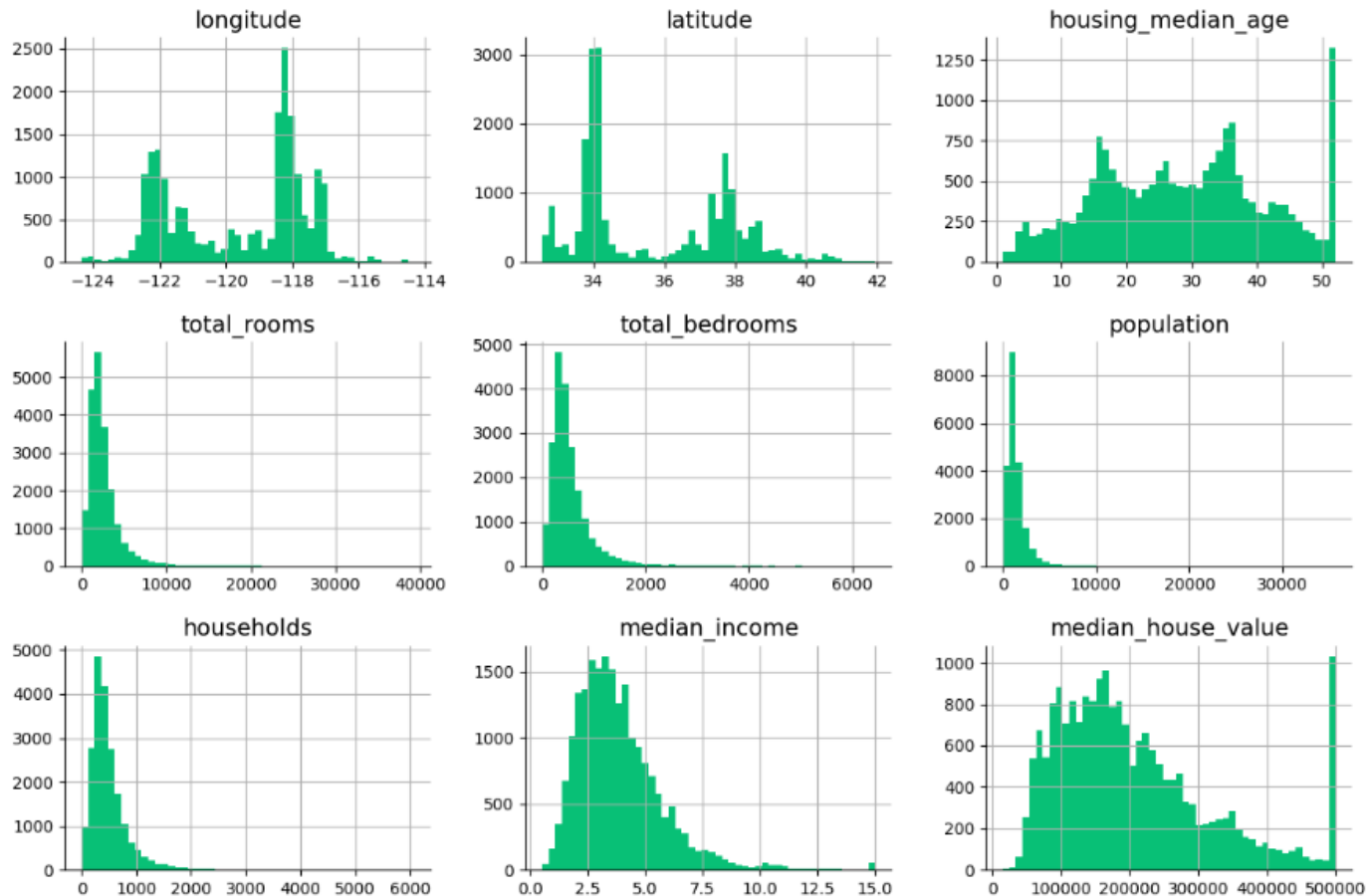
```
#   Column      Non-Null Count  Dtype
---  -
0   longitude    20640 non-null  float64
1   latitude     20640 non-null  float64
2   housing_median_age  20640 non-null  float64
3   total_rooms   20640 non-null  float64
4   total_bedrooms 20433 non-null  float64
5   population    20640 non-null  float64
6   households    20640 non-null  float64
7   median_income 20640 non-null  float64
8   median_house_value 20640 non-null  float64
9   ocean_proximity 20640 non-null  object
```

`housing.describe()`
voor meer info

`housing["ocean_proximity"].value_counts()`

Data bekijken

housing.hist(...)



Bevindingen:

- Ook de median_age wordt duidelijk afgekapt op 50.
- Het mediane inkomen wordt uitgedrukt in een vreemde (onbekende) eenheid.
- Verschillende features leven op volledig andere schalen.
- (Bijna) alle features zijn geskewed naar rechts.
- Er zijn een aantal features (total_rooms, total_bedrooms, population en households) die niet exponentieel afvallen.

Scikit-Learn



Wat is Scikit-Learn?

- Library die veel ML-algoritmes bevat & alles wat je nodig hebt om aan pre- & post-processing te doen.
- “estimator-api”, +/- zelfde code voor trainen/testen verschillende algoritmes.
- **Voordeel:** Eenvoudig vele algoritmes gebruiken.
- **Nadelen:** Algoritmes gebruiken zonder te weten wat je doet & weinig flexibiliteit.
- Focus van Scikit-Learn is snel kunnen **toepassen**, dus zeer snel resultaten krijgen. Ideaal voor in de bedrijfswereld.

De standaardcode

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

Algemeen

```
from sklearn.model_family import ModelAlgo
from sklearn.metrics import error_metric
mymodel = ModelAlgo(param1, param2)
mymodel.fit(X_train, y_train)
predictions = mymodel.predict(X_test)
performance = error_metric(y_test, predictions)
```

Lineaire regressie

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_percentage_error
model = LinearRegression()
model.fit(X_train, y_train)
test_predictions = model.predict(X_test)
MAPE = mean_absolute_percentage_error(y_test, test_predictions)
```

Tip: Bekijk zeker eens [API Reference — scikit-learn 1.5.1 documentation](https://scikit-learn.org/stable/documentation/1.5.1/)

Componenten van *sklearn*

Deze dingen hebben allemaal goede defaults → helpt snel starten!

Kan je allemaal steken in een **Pipeline** die start met estimators/transformers & eindigt met een predictor (typisch).

Estimators

fit om te leren, **eventueel** met een **transform**.

Predictors: hebben een `predict()` en een `score()` (of `predict_proba`)

Transformers

transform() om te transformeren, **eventueel** met een **fit**.

Inspectie:

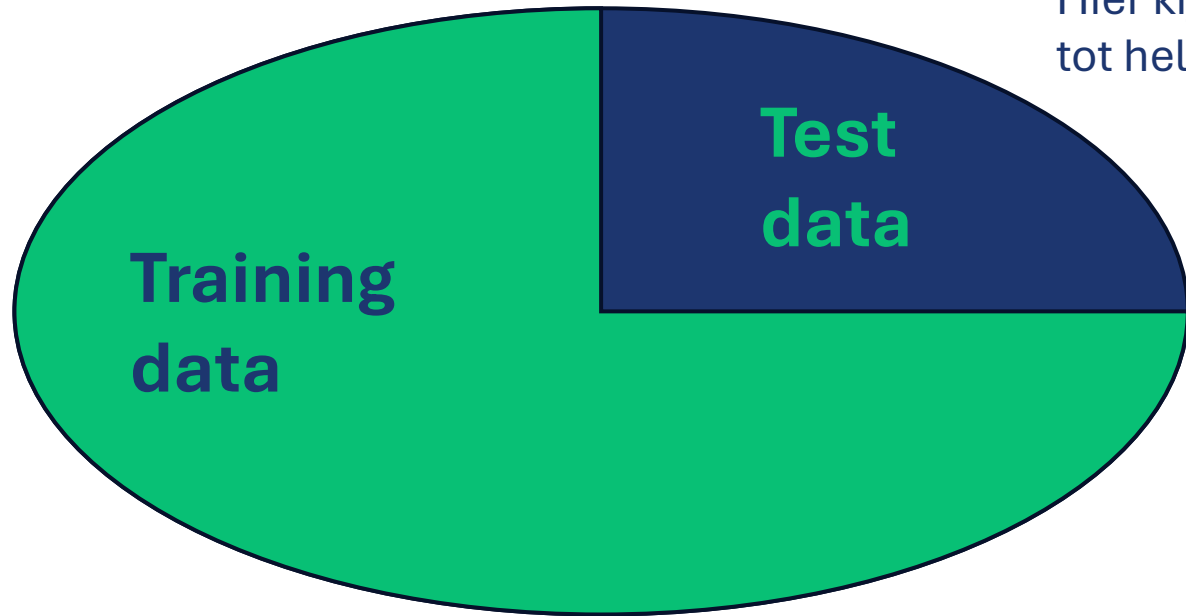
Parameters via `.strategy`, ...

Parameters na training met een `_`, bvb `.statistics_`

De data bekijken

Train/test split

Train/test split



Hier kijken we verder niet meer naar
tot helemaal op het einde!

20% is een standaard,
met grote datasets kan je
dit verkleinen.

Voor de
reproduceerbaarheid:
altijd dezelfde split
gebruiken!

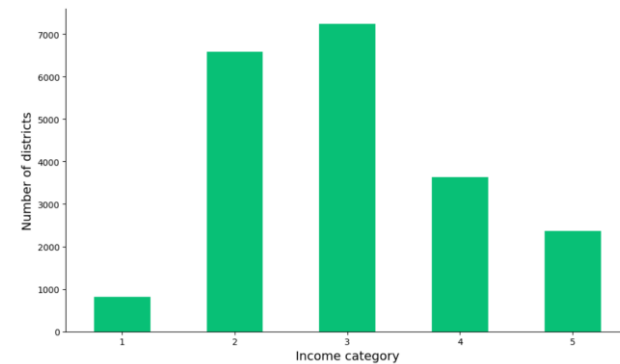
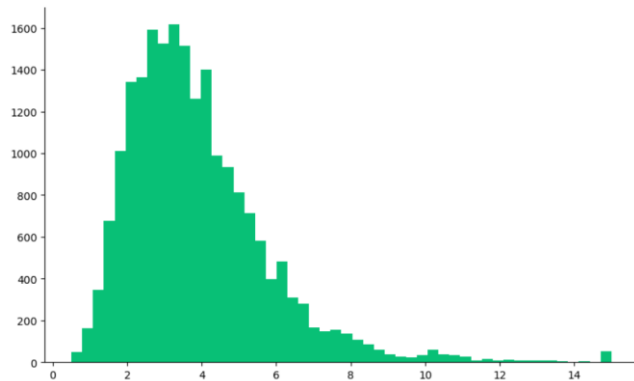
```
from sklearn.model_selection import train_test_split  
train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

Two blue arrows point from the code to the explanatory text above. One arrow points from the value `0.2` in `test_size=0.2` to the text '20% is een standaard...'. The other arrow points from the value `42` in `random_state=42` to the text 'Voor de reproduceerbaarheid: altijd dezelfde split gebruiken!'.

Stratified sampling – median income

Median income belangrijkste feature → we doen stratified sampling hierop!

```
pd.cut(housing["median_income"],  
       bins=[-np.inf] + percentiles + [np.inf],  
       labels=range(1, 100 + 1))
```



	Overall %	Stratified %	Random %	Strat. Error %	Rand. Error %
Income Category					
1	3.98	4.00	4.24	0.36	6.45
2	31.88	31.88	30.74	-0.02	-3.59
3	35.06	35.05	34.52	-0.01	-1.53
4	17.63	17.64	18.41	0.03	4.42
5	11.44	11.43	12.09	-0.08	5.63

```
strat_train_set, strat_test_set = train_test_split(  
    housing, test_size=0.2, stratify=housing["income_cat"], random_state=42)
```




python.exposed

Learn Python to Automate, Analyze, Accelerate.

AUSTRALIA IS SCOOPY DOO



Explorieren & visualisieren

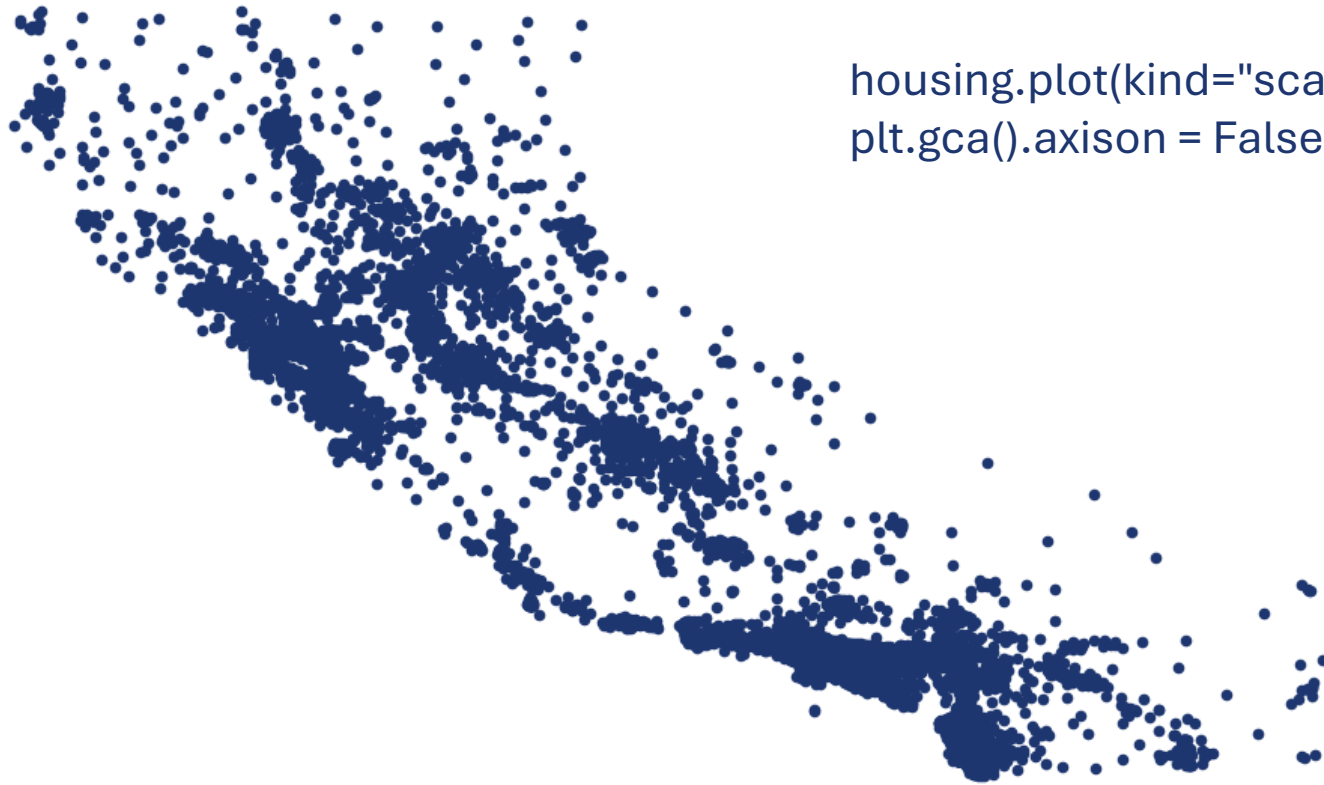
Geografische data visualisieren



python.exposed

Learn Python to Automate, Analyze, Accelerate.

Geografische data plotten



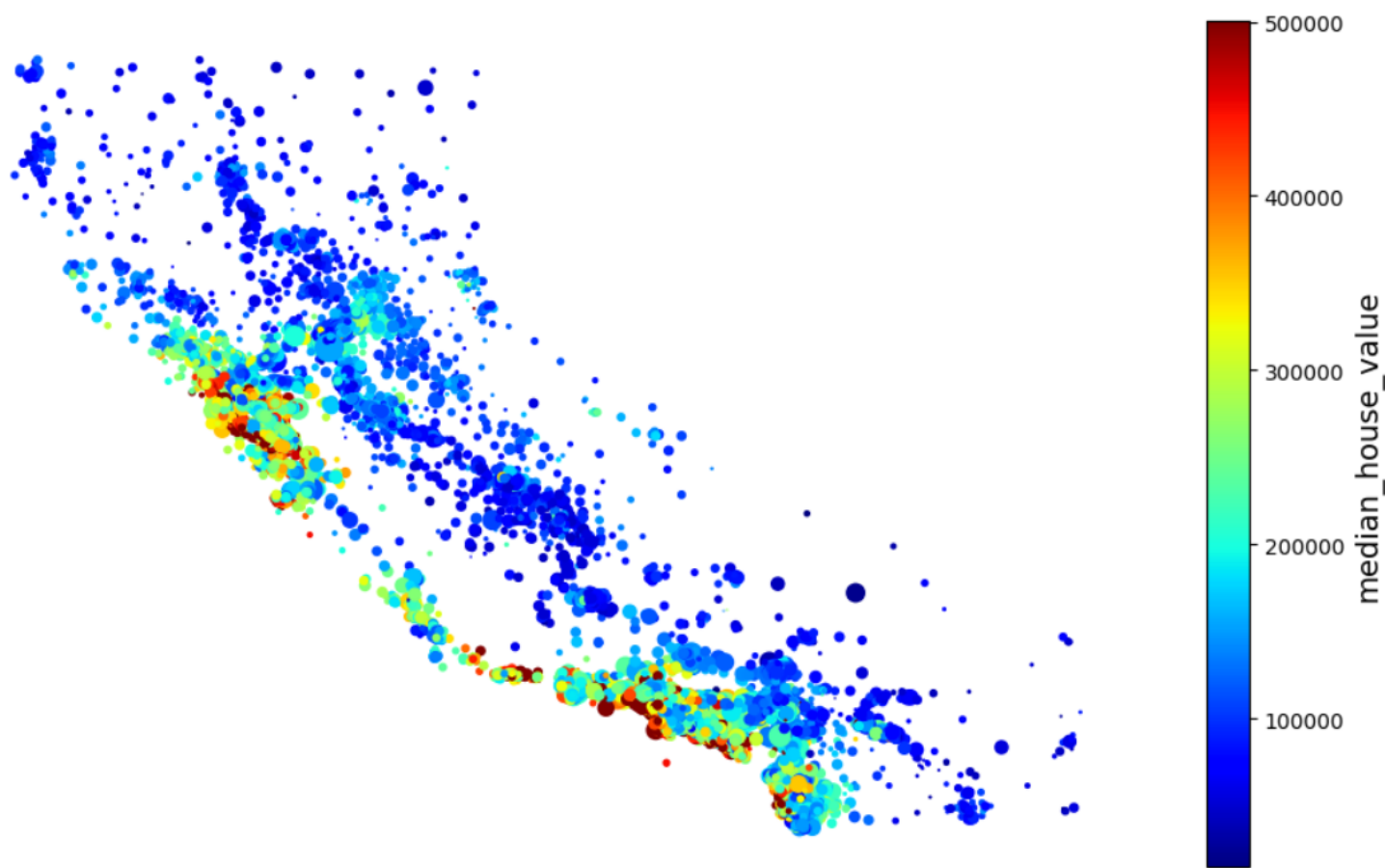
```
housing.plot(kind="scatter", x="longitude", y="latitude")  
plt.gca().axison = False
```

Geografische data plotten



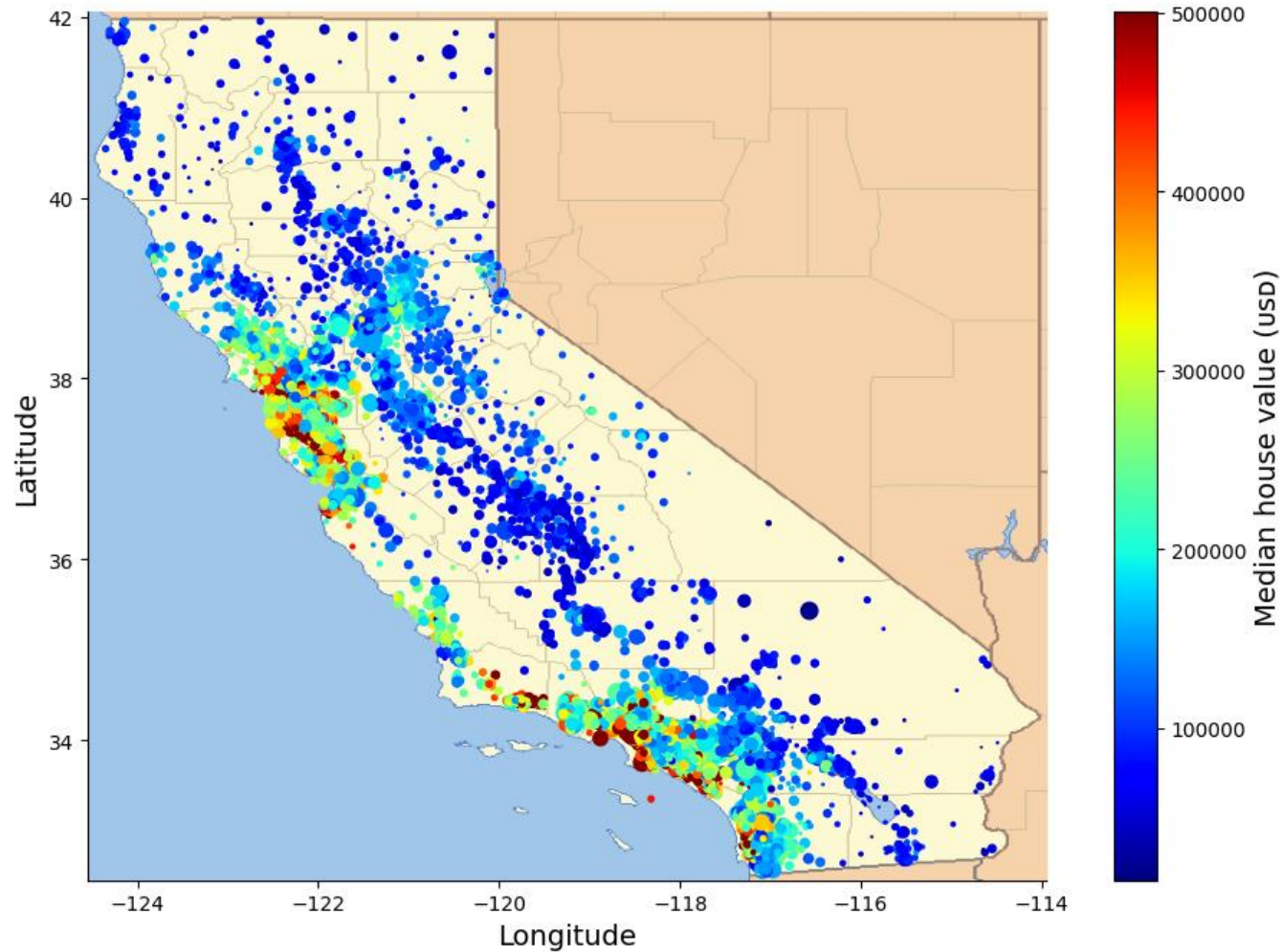
```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha = 0.2)  
plt.gca().axison = False
```

Geografische data plotten



```
housing.plot(kind="scatter",  
             x="longitude",  
             y="latitude",  
             s=housing["population"] / 100,  
             c="median_house_value",  
             cmap="jet",  
             colorbar=True,  
             legend=False)
```

Geografische data plotten



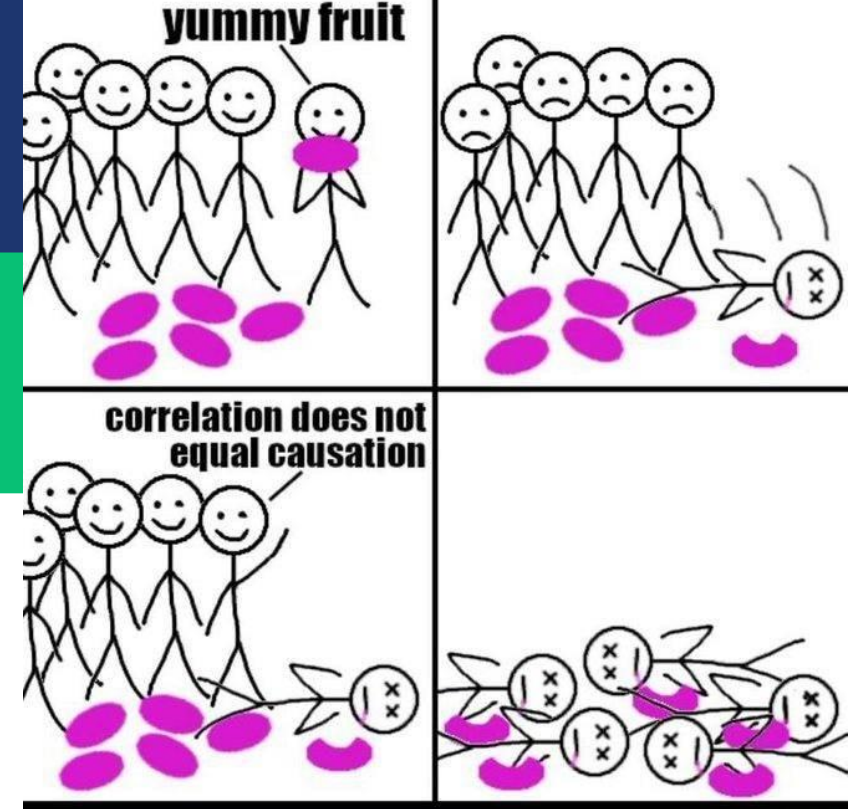
Dit soort plots kan je bv. bekomen vanuit:

`plotly.express.scatter_mapbox`



python.exposed

Learn Python to Automate, Analyze, Accelerate.



Exploreren & visualizeren

Correlatie bekijken



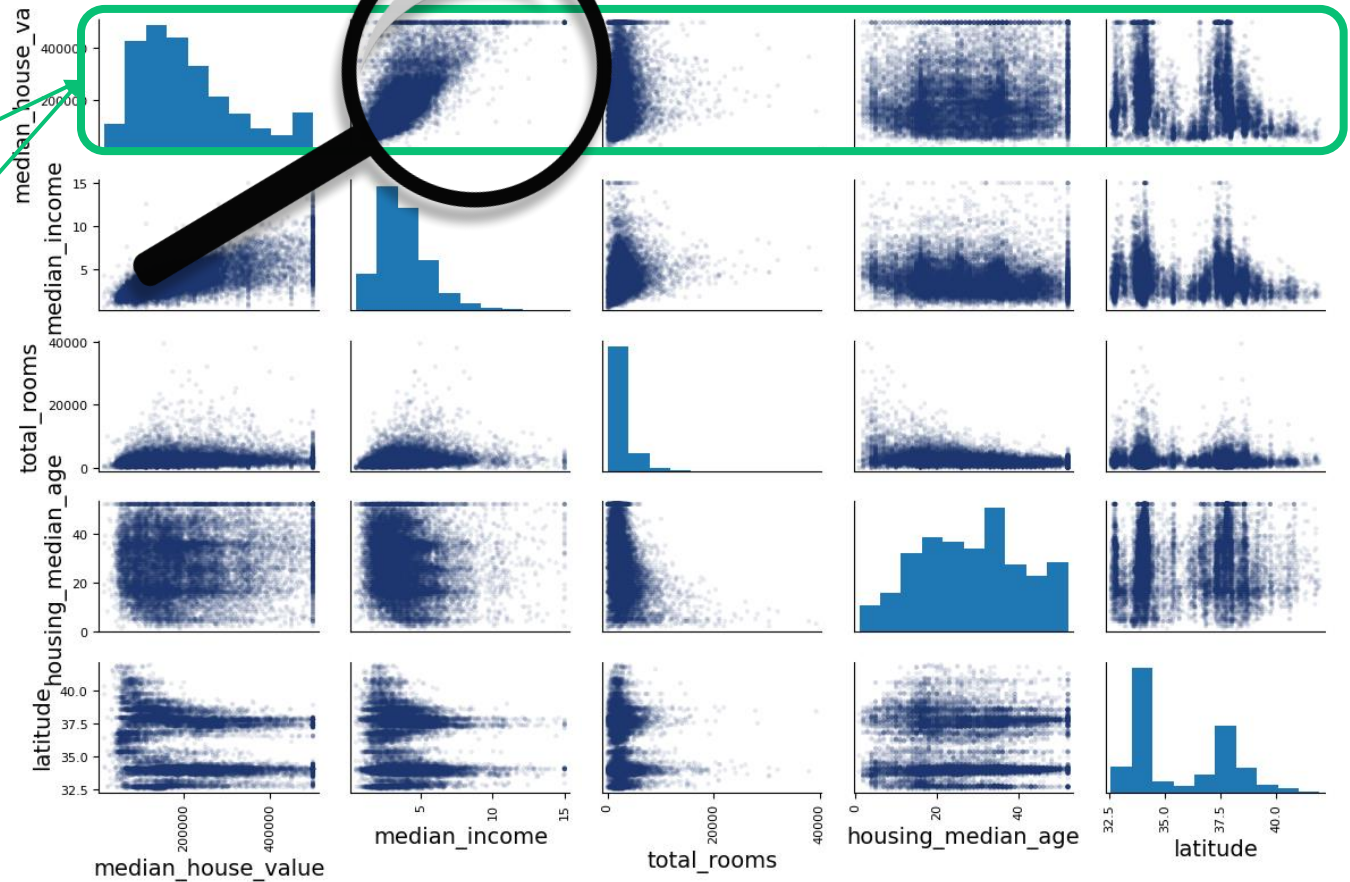
python.exposed

Learn Python to Automate, Analyze, Accelerate.

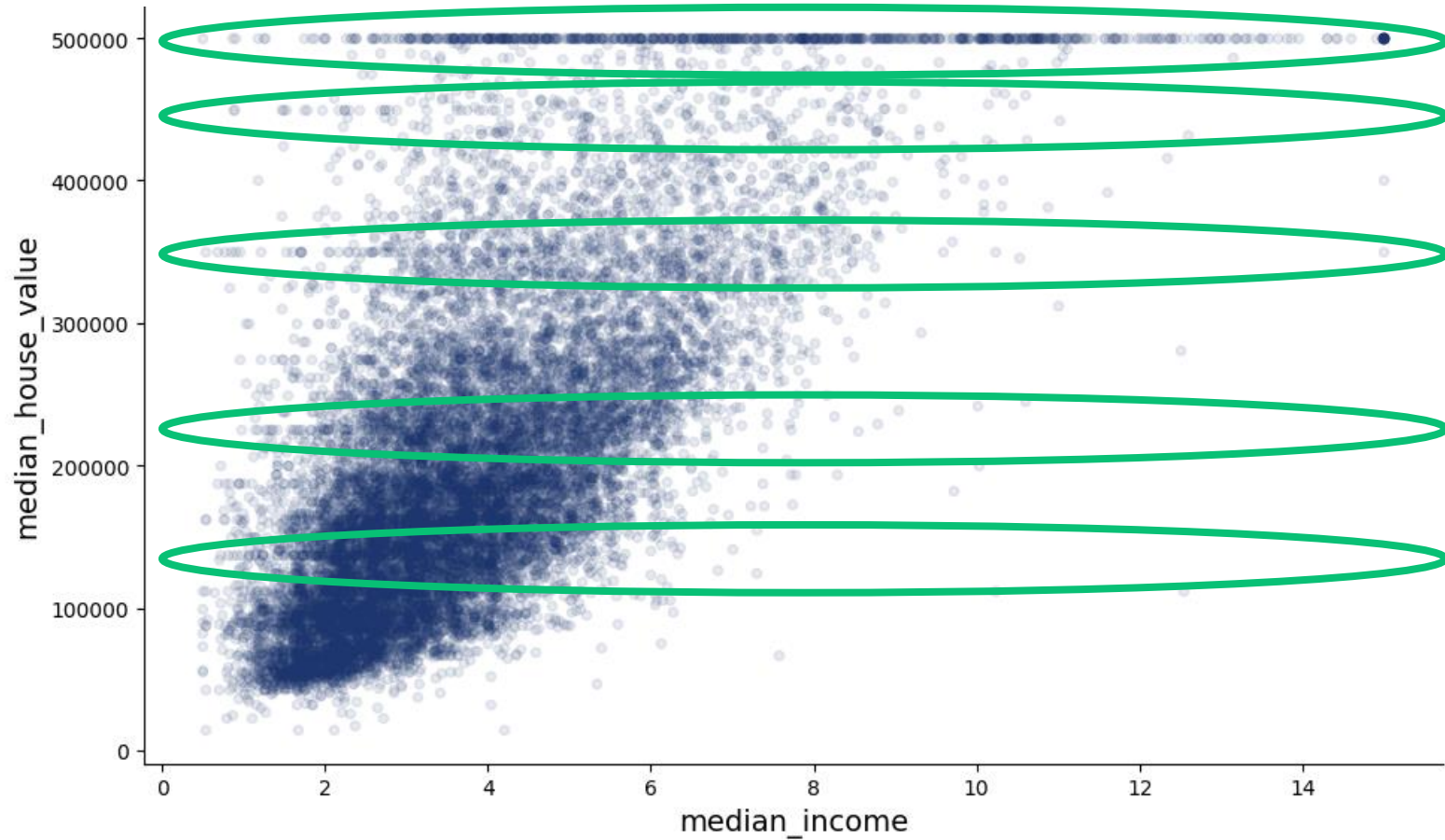
Correlatie met je target

```
from pandas.plotting import scatter_matrix
attributes = ["median_house_value", "median_income", "total_rooms",
             "housing_median_age", "latitude"]
scatter_matrix(housing[attributes], figsize=(12, 8), color = BLUE, alpha = 0.1)
```

median_house_value	1.000000
median_income	0.688380
total_rooms	0.137455
housing_median_age	0.102175
households	0.071426
total_bedrooms	0.054635
population	-0.020153
longitude	-0.050859
latitude	-0.139584



Inzoomen op de hoogste correlatie



We hebben een goed aantal horizontale lijnen, dit wijst sterk op “data quirks” en je kan overwegen deze datapunten eruit te halen.

Oefening: Hoe kan je deze datapunten vinden?



python.exposed

Learn Python to Automate, Analyze, Accelerate.



Exploreren & visualizeren

Nieuwe features bedenken



python.exposed

Learn Python to Automate, Analyze, Accelerate.

Nieuwe features bedenken

- Extra features?
 - Aantal kamers per huis (kamers/huizen)
→ Correlatie **0.14**
 - Percentage van de kamers dat een slaapkamer is (slaapkamers / kamers)
→ Correlatie **-0.25**
 - Aantal mensen per huis (populatie / huizen)
→ Correlatie **-0.038**

Data klaarmaken voor ML

Data cleanen



Data cleanen – missing values

```
housing = strat_train_set.drop("median_house_value", axis=1) # features
housing_labels = strat_train_set["median_house_value"].copy() # labels
```

- Missing values bij *total_bedrooms*, 3 opties:
 - Districten (=rijen) met NaN laten vallen.
 - De feature laten vallen.
 - Invullen met een geschatte waarde, *imputation*.
→ `from sklearn.impute import SimpleImputer`
Vraag: Geef een voordeel van SimpleImputer te gebruiken ipv `.fillna()`

Opmerking: `imputer.transform(df)` geeft een Numpy array terug!
Bekijk ook eens `KNNImputer` & `IterativeImputer`

Data cleanen - outliers

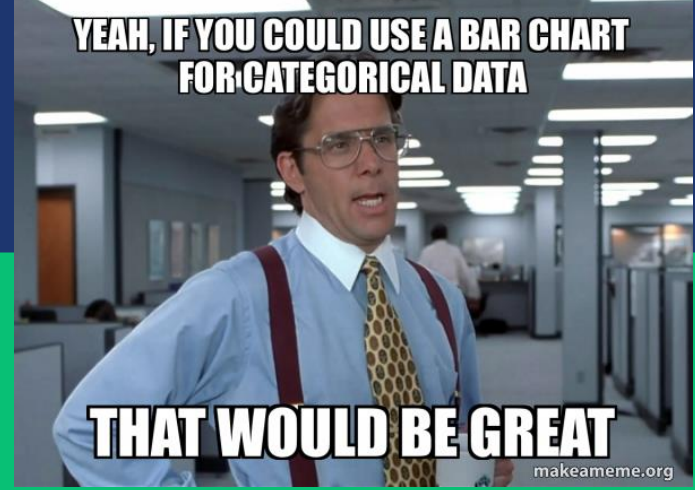
```
from sklearn.ensemble import IsolationForest  
isolation_forest = IsolationForest(random_state=42)  
outlier_pred = isolation_forest.fit_predict(X)
```

-1 als outlier, anders 1.



python.exposed

Learn Python to Automate, Analyze, Accelerate.



Data klaarmaken voor ML

Tekst & categorische data



python.exposed

Learn Python to Automate, Analyze, Accelerate.

Data cleanen – tekst & categorische data

housing[["ocean_proximity"]]

Ordinaal?

Ja

Nee

```
from sklearn.preprocessing import OrdinalEncoder
ordinal_encoder = OrdinalEncoder()
housing_cat_encoded =
ordinal_encoder.fit_transform(housing_cat)
```

```
from sklearn.preprocessing import OneHotEncoder
cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
```

Opmerking: Dit geeft je een **sparse** matrix terug.
Dit is een dictionary met enkel de niet-nulle elementen.



Data klaarmaken voor ML

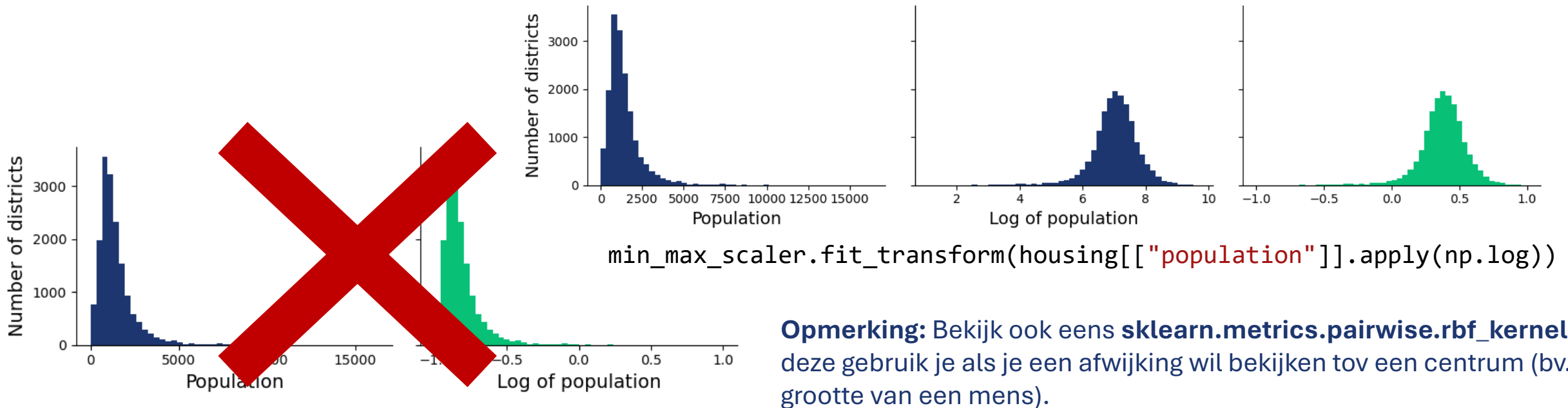
Schalen & transformeren

Data cleanen – Schalen

- **Probleem:** Verschillende features leven in andere grote orden van getallen. Bv. $nrooms \in [6, 39\ 320]$ & $median_income \in [0, 15]$, dit is niet handig voor meeste algoritmes, ze gaan dan bv. *median_income* negeren.
- **Oplossing:** feature scaling:
 - Min-max scaling, $x \rightarrow \frac{x - x_{min}}{x_{max} - x_{min}}$ (let op met outliers!)
 - Standaardisatie, $x \rightarrow \frac{x - \bar{x}}{s_x}$
 - Iets anders...
- **Vraag:** Welke datatypes moet je schalen?

Data cleanen – Schalen

- **Demo:** Bekijk eens wat er gebeurt met de *populatie* verdeling als je deze schaalt volgens min-max scaling.



Data cleanen - target

Optie 1 schalen, toepassen en terug schalen:

Schalen

```
target_scaler = StandardScaler()  
scaled_labels = target_scaler.fit_transform(housing_labels.to_frame())
```

Fitten

```
model = LinearRegression()  
model.fit(housing[["median_income"]], scaled_labels)  
some_new_data = housing[["median_income"]].iloc[:5] # pretend this is new data
```

Predicties

terug
schalen

```
scaled_predictions = model.predict(some_new_data)  
predictions = target_scaler.inverse_transform(scaled_predictions)
```

Optie 2, schaling in het model:

```
from sklearn.compose import TransformedTargetRegressor  
model = TransformedTargetRegressor(LinearRegression(), transformer=StandardScaler())  
model.fit(housing[["median_income"]], housing_labels)  
predictions = model.predict(some_new_data)
```

Data cleanen – DataFrames

DataFrames verliezen vaak hun kolommen bij transformaties.
Maar je kan ze terugwinnen 😊!

```
df_output = pd.DataFrame(cat_encoder.transform(df_test_unknown),  
                          columns=cat_encoder.get_feature_names_out(),  
                          index=df_test_unknown.index)
```



Data klaarmaken voor ML

Custom transformaties

Custom transformers – zonder training

```
from sklearn.preprocessing import FunctionTransformer
```

```
log_transformer = FunctionTransformer(np.log, inverse_func=np.exp)  
log_pop = log_transformer.transform(housing[["population"]])
```

De transformer `log_pop` kan je nu gewoon gebruiken zoals bijvoorbeeld `sklearn's StandardScaler`.



Custom transformers – met training

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.utils.validation import check_array, check_is_fitted

class StandardScalerClone(BaseEstimator, TransformerMixin):
    def __init__(self, with_mean=True): # no *args or **kwargs!
        self.with_mean = with_mean

    def fit(self, X, y=None): # y is required even though we don't use it
        X = check_array(X) # checks that X is an array with finite float values
        self.mean_ = X.mean(axis=0)
        self.scale_ = X.std(axis=0)
        self.n_features_in_ = X.shape[1] # every estimator stores this in fit()
        return self # always return self!

    def transform(self, X):
        check_is_fitted(self) # looks for learned attributes (with trailing _)
        X = check_array(X)
        assert self.n_features_in_ == X.shape[1]
        if self.with_mean:
            X = X - self.mean_
        return X / self.scale_
```



Custom transformers – met training

```
from sklearn.cluster import Kmeans
```

```
class ClusterSimilarity(BaseEstimator, TransformerMixin):
    def __init__(self, n_clusters=10, gamma=1.0, random_state=None):
        self.n_clusters = n_clusters
        self.gamma = gamma
        self.random_state = random_state

    def fit(self, X, y=None, sample_weight=None):
        self.kmeans_ = KMeans(self.n_clusters, n_init=10,
                               random_state=self.random_state)
        self.kmeans_.fit(X, sample_weight=sample_weight)
        return self # always return self!

    def transform(self, X):
        return rbf_kernel(X, self.kmeans_.cluster_centers_, gamma=self.gamma)

    def get_feature_names_out(self, names=None):
        return [f"Cluster {i} similarity" for i in range(self.n_clusters)]
```



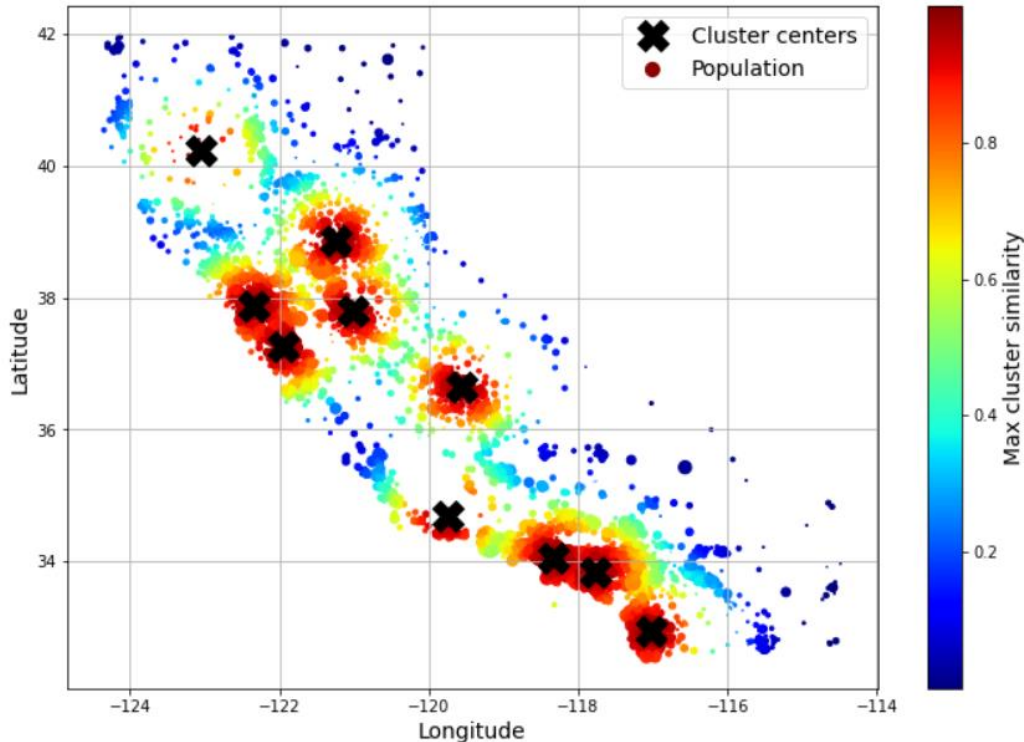

Custom transformers – met training

```
cluster_simil = ClusterSimilarity(n_clusters=10, gamma=1., random_state=42)
similarities = cluster_simil.fit_transform(housing[["latitude", "longitude"]],
sample_weight=housing_labels)
```

```
similarities[:3].round(2)
```

```
[0. , 0.14, 0. , 0. , 0. , 0.08, 0. , 0.99, 0. , 0.6 ]
[0.63, 0. , 0.99, 0. , 0. , 0. , 0.04, 0. , 0.11, 0. ]
[0. , 0.29, 0. , 0. , 0.01, 0.44, 0. , 0.7 , 0. , 0.3]
```

Afstand tot eerste cluster centrum.





python.exposed

Learn Python to Automate, Analyze, Accelerate.

DATA PIPELINE IS READY



LETS PARTY !!!

makeameme.org

Data klaarmaken voor ML

Pipelines



python.exposed

Learn Python to Automate, Analyze, Accelerate.

Pipeline maken

Tip: Je kan je pipeline steeds visualiseren:
`from sklearn import set_config`
`set_config(display='diagram')`
`num_pipeline`

```
from sklearn.pipeline import Pipeline
```

```
num_pipeline = Pipeline([
```

```
    ("impute", SimpleImputer(strategy="median")),  
    ("standardize", StandardScaler()),
```

```
])
```

Naampjes die je er zelf aan geeft.

Willekeurig aantal transformers

Evt estimator die enkel een fit moet hebben.

Tip: ChatGPT kent de syntax voor het maken van pipelines zeer goed, vraag hem gedetailleerd wat je pipeline zou moeten doen & inspecteer zijn resultaat. Je moet wel zelf opletten dat je pipeline logisch is & steeds inspecteren wat je terugkrijgt.

Pipeline maken – kolom specifiek

```
from sklearn.compose import ColumnTransformer
```

```
num_attribs = ["longitude", "latitude", "housing_median_age", "total_rooms",  
               "total_bedrooms", "population", "households", "median_income"]
```

```
cat_attribs = ["ocean_proximity"]  
cat_pipeline = make_pipeline(  
    SimpleImputer(strategy="most_frequent"),  
    OneHotEncoder(handle_unknown="ignore"))
```

```
preprocessing = ColumnTransformer([  
    ("num", num_pipeline, num_attribs),  
    ("cat", cat_pipeline, cat_attribs),  
])
```

naampje Transformer Lijst kolommen

Opmerking: De categorische pipeline geeft een sparse matrix en de numerieke een dense matrix.



python.exposed

Learn Python to Automate, Analyze, Accelerate.



Selecteer & train een model

Train & evalueer op trainingsset



python.exposed

Learn Python to Automate, Analyze, Accelerate.

Evaluëren op de trainingdata – train error

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

lin_reg = make_pipeline(preprocessing, LinearRegression())
lin_reg.fit(housing, housing_labels)
housing_predictions = lin_reg.predict(housing)
lin_rmse = mean_squared_error(housing_labels, housing_predictions, squared=False)
lin_rmse
```

69k error, tov waarden die rond de 120k-265k liggen, dus een grote error! Dus... Underfitting, model moet complexer zijn!

Error van 0, dus... het perfecte model is gevonden!
The holy grail van modellen.

Fout: Dit is overfitting.

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = make_pipeline(preprocessing, DecisionTreeRegressor(random_state=42))
tree_reg.fit(housing, housing_labels)
housing_predictions = tree_reg.predict(housing)
tree_rmse = mean_squared_error(housing_labels, housing_predictions,
                              squared=False)

tree_rmse
```



Selecteer & train een model

Cross-validation

Cross-validation – tree_reg

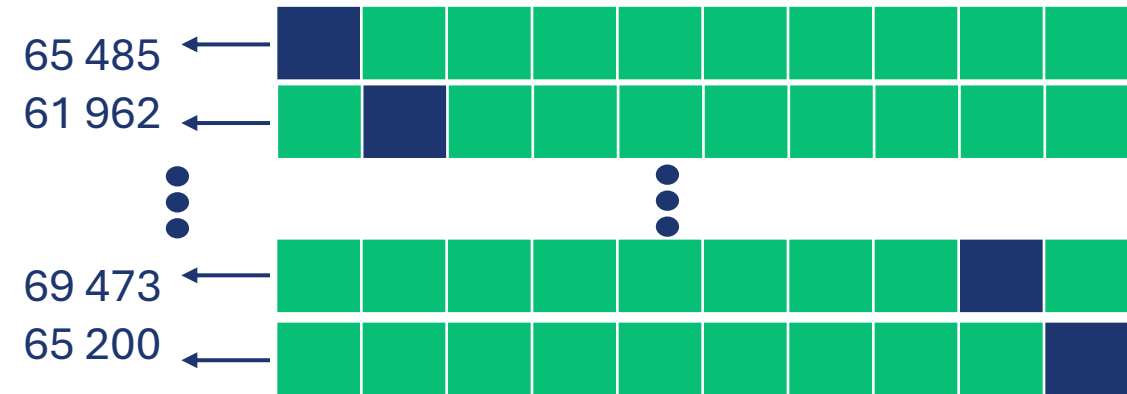
```
from sklearn.model_selection import cross_val_score
```

```
tree_rmse = -cross_val_score(tree_reg, housing, housing_labels,  
                             scoring="neg_root_mean_squared_error", cv=10)
```

```
pd.Series(tree_rmse).describe()
```

count	10
mean	66664
std	2334
min	61962
25%	65546
50%	66529
75%	67902
max	70054

Scoring functie, hoe hoger
de score, hoe beter.



Cross-validation – lin_reg

```
lin_rmse = -cross_val_score(lin_reg, housing, housing_labels,  
                             scoring="neg_root_mean_squared_error", cv=10)  
pd.Series(lin_rmse).describe()
```

count	10
mean	69809
std	4006
min	65700
25%	68098
50%	68680
75%	69704
max	80478

Conclusie:

Beide modellen doen het ongeveer even slecht;

- Lineaire regressie is sterk aan het underfitten.
- Regression trees zijn sterk aan het overfitten!

Cross-validation – RandomForestRegressor

```
from sklearn.ensemble import RandomForestRegressor
forest_reg = make_pipeline(preprocessing, RandomForestRegressor(random_state=42))
# Train error: 17 474
forest_reg.fit(housing, housing_labels)
housing_predictions = forest_reg.predict(housing)
forest_rmse = mean_squared_error(housing_labels, housing_predictions, squared=False)
# Test error:
forest_rmses = -cross_val_score(forest_reg, housing, housing_labels,
                                scoring="neg_root_mean_squared_error", cv=10)
```

→ Preprocessing pipeline die we al hadden gemaakt.

min	45 690
25%	46 556
50%	47 081
75%	47 353
max	49 183

Vraag: Wat concludeer je hieruit?

Modellen opslaan & laden

```
import joblib

model = LinearRegression()
model.fit(X_train, y_train)

# Je model opslaan
joblib.dump(model, 'linear_regression_model.joblib')

# Je model terug inladen
loaded_model = joblib.load('linear_regression_model.joblib')
```

Handig omdat:

- **Tijd:** Modellen trainen kan lang duren, maar toepassen gaat heel snel.
- **Reproduceerbaarheid:** Door iets andere data, split, ... is het vaak moeilijk om resultaten te reproduceren als je je model niet opslaat.

Algemene aanpak – model selecteren

Stap 1: Kies een model waarvan je denkt dat het evt kan werken.

Stap 2: Bereken de train error (TE) & cross-validation error (CVE) voor je model.

Stap 3: Als $TE \ll CVE$, ben je aan het overfitten & moet je een eenvoudiger model hebben, als $CVE \approx TE$, maar hoger dan wat je wenst, moet je een complexer model hebben of een ander soort model.

Stap 4: Sla je model op & ga terug naar **Stap 1** met de info uit **Stap 3**.

De volgende stap is het fine-tunen van het goede model die je hier vond!

Componenten van *sklearn*

Deze dingen hebben allemaal goede defaults → helpt snel starten!

Kan je allemaal steken in een **Pipeline** die start met estimators/transformers & eindigt met een predictor (typisch).

Estimators

fit om te leren, **eventueel** met een **transform**.

Predictors: hebben een **predict()** en een **score()** (of **predict_proba()**)

Transformers

transform() om te transformeren, **eventueel** met een **fit**.

Inspectie:

Parameters via **.strategy**, ...

Parameters na training met een **_**, bvb **.statistics_**



python.exposed

Learn Python to Automate, Analyze, Accelerate.



Fine tune model

Grid Search



python.exposed

Learn Python to Automate, Analyze, Accelerate.

Grid Search - uitvoeren

```
from sklearn.model_selection import GridSearchCV
```

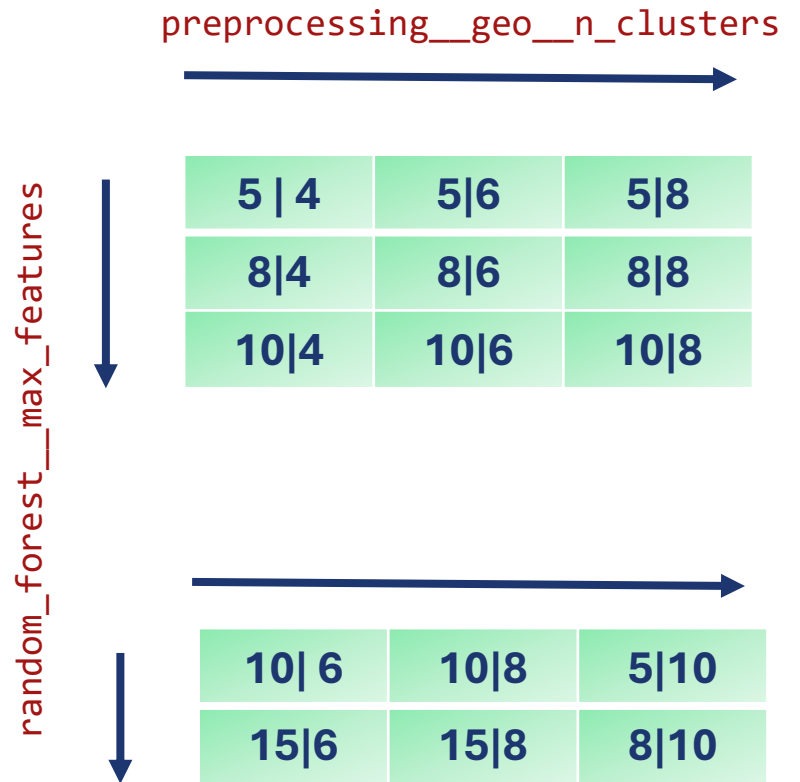
```
full_pipeline = Pipeline([
    ("preprocessing", preprocessing),
    ("random_forest", RandomForestRegressor(random_state=42)),
])
```

```
param_grid = [
    {'preprocessing__geo__n_clusters': [5, 8, 10],
     'random_forest__max_features': [4, 6, 8]},
    {'preprocessing__geo__n_clusters': [10, 15],
     'random_forest__max_features': [6, 8, 10]},
]
```

Eerste grid

Tweede grid

```
grid_search = GridSearchCV(full_pipeline, param_grid, cv=3,
                           scoring='neg_root_mean_squared_error')
grid_search.fit(housing, housing_labels)
```



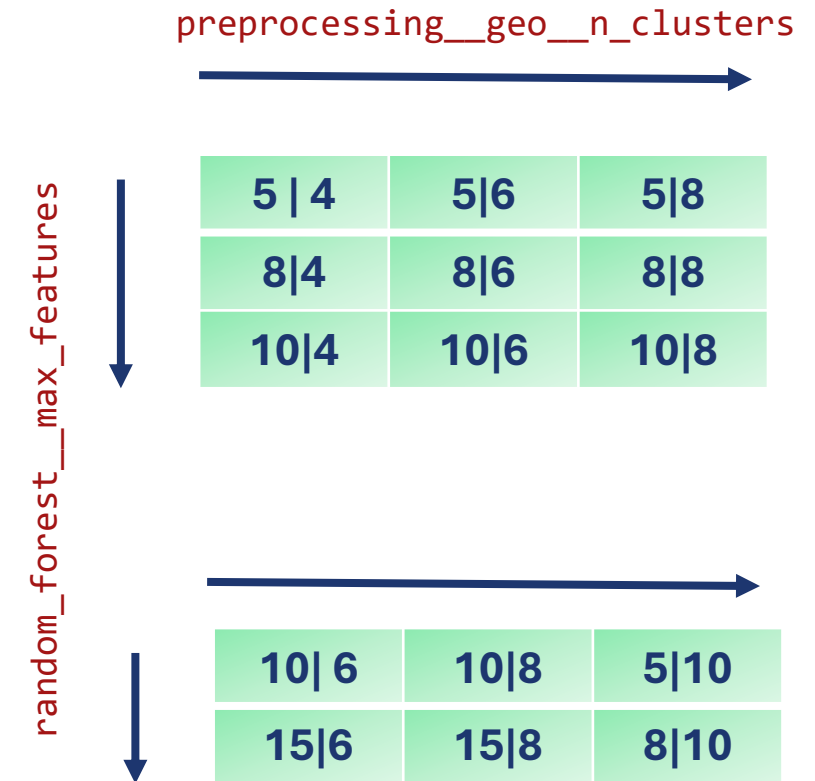
Vraag: Hoeveel fit/evaluatie cycli hebben we hier?

Grid Search - uitvoeren

Wat doet:

```
grid_search.fit(housing,  
housing_labels)
```

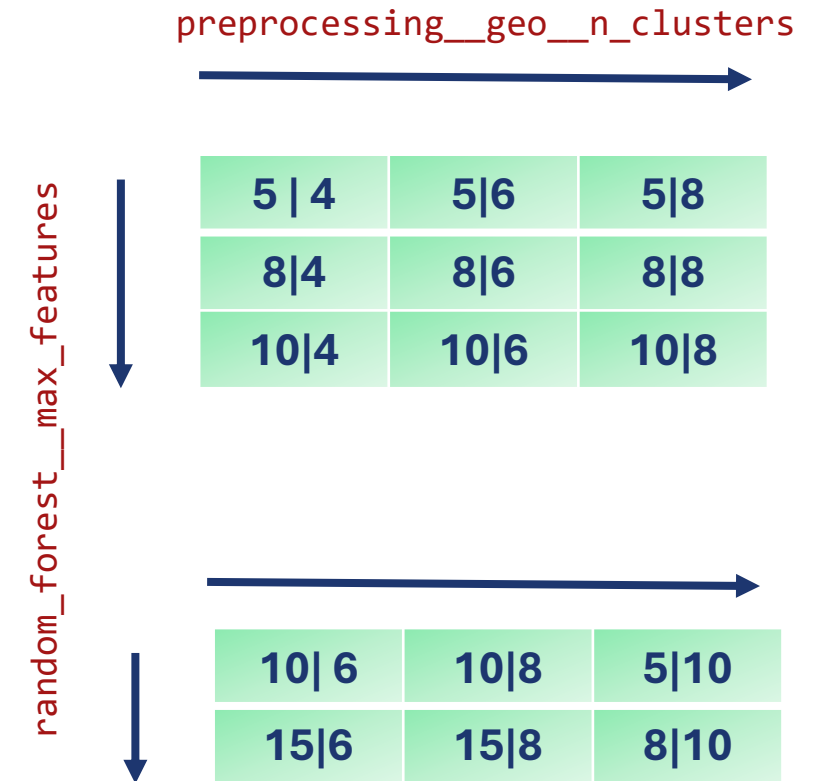
- **Stap 1:** Alle 12 + 6 combinaties afgaan, cross-validation gebruiken voor elke combinatie.
- **Stap 2:** Voor de beste combinatie nog eens een training doen op de volledige trainingsdata.



Grid Search – resultaten bekijken

```
grid_search.best_params_  
    {'preprocessing__geo__n_clusters': 15,  
     'random_forest__max_features': 6}  
grid_search.best_estimator_  
    Pipeline(...)  
pd.DataFrame(grid_search.cv_results_)
```

Probleem: Bij een redelijke search space, explodeert het aantal stappen van grid search!





Fine tune model

Randomized Search

Randomized search

- Stel je bv. voor:
 - Hyperparameter 1, value in 1, 2, ..., 137
 - Hyperparameter 2, value in 84, ..., 528
 - Hyperparameter 3, value in 100, ..., 1057
- In totaal een grid van 58 miljoen mogelijke combinaties

Oplossing: Randomized Search gaat hier willekeurig in zoeken, bv. 1000 iteraties met 100 verschillende combinaties.

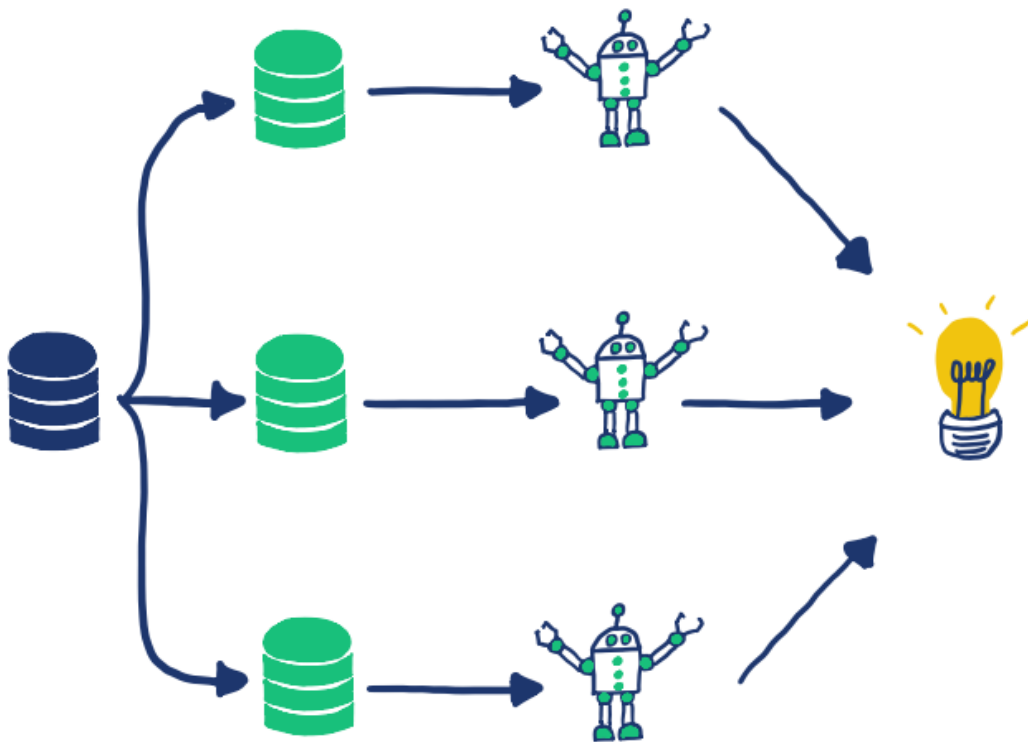
- Je kan willekeurig verspreide combinaties ontdekken.
- Expliciete controle over hoeveelheid rekenwerk.

Fine tune model

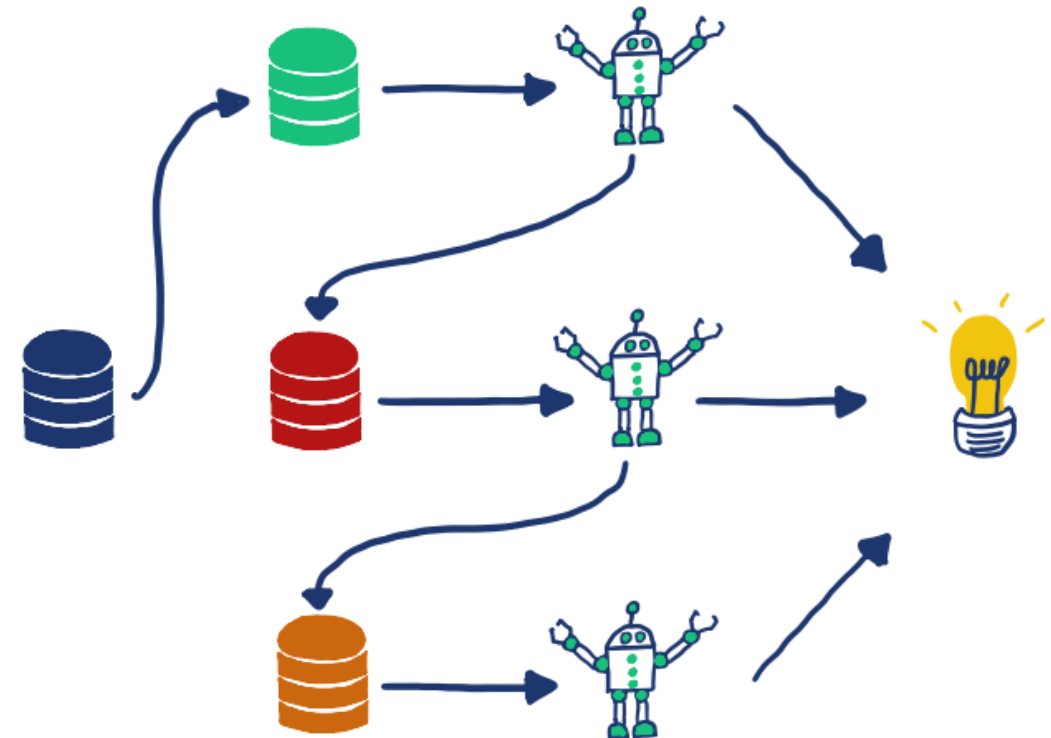
Ensemble methodes

Ensemble methodes

Bagging



Boosting



Opmerking: Komen we later nog in de praktijk tegen



Fine tune model

Beste model bekijken

Feature importance

```
final_model = rnd_search.best_estimator_ # includes preprocessing
feature_importances = final_model["random_forest"].feature_importances_
print(sorted(zip(feature_importances,
                  final_model["preprocessing"].get_feature_names_out()), reverse=True))
```

Te veel features is vaak niet handig voor modellen

→ Je kan de weinig belangrijke features proberen weg te gooien!

Opmerking: eenzelfde feature kan belangrijk zijn voor *model_1* maar niet voor *model_2*! Betekent mogelijk dat *model_2* hier niet correct mee rekent.

Specifieke fouten bekijken

Finaal kan je voor je model kijken welke specifieke fouten het maakt.

Hoe ze repareren, denk aan:

- Extra features,
- Slechte features verwijderen,
- Outliers opkuisen,
- Wat data cleaning gemist,
- ...

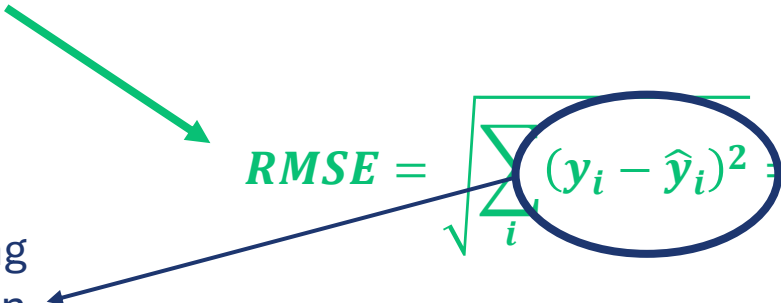
Fine tune model
Evaluate on test set

**ME AT MY ANNUAL
WORK EVALUATION!**



Performance op test set

```
X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()
final_predictions = final_model.predict(X_test)
final_rmse = mean_squared_error(y_test, final_predictions,
                                squared=False)
```


$$RMSE = \sqrt{\sum_i (y_i - \hat{y}_i)^2} = 47\,730$$

$\{(y_i - \hat{y}_i)^2\}$ is nu een verzameling errors die je statistisch kan onderzoeken (bv. met `scipy.stats`).

Top! Deze performance ligt in lijn met wat we verwachten uit voorgaand onderzoek. Hoe zeker zijn we van dit getal?



python.exposed

Learn Python to Automate, Analyze, Accelerate.



Launch, monitor & maintain



python.exposed

Learn Python to Automate, Analyze, Accelerate.

To-do's

- Getrainde model ergens plaatsen en klaarmaken om nieuwe data te verwerken.
- Monitoring output (met **echte** mensen):
 - Plotse falingen &
 - Langzaam verval van performance.
- Monitoring input
 - Fouten in de input, slechte input, ... vangen.
- Eenvoudig maken om te hertrainen:
 - Batch learning: bv. goede notebook die een getraind model geeft.
 - Online learning: volledig automatisch met reguliere snapshots

Allerbelangrijkste

DOCUMENTATIE

- Code setup informatie. → typisch in de *readme.md*
 - Input (met locaties).
 - Informatie model.
 - Details deployment.
- Typisch op confluence pages, sharepoint, ...

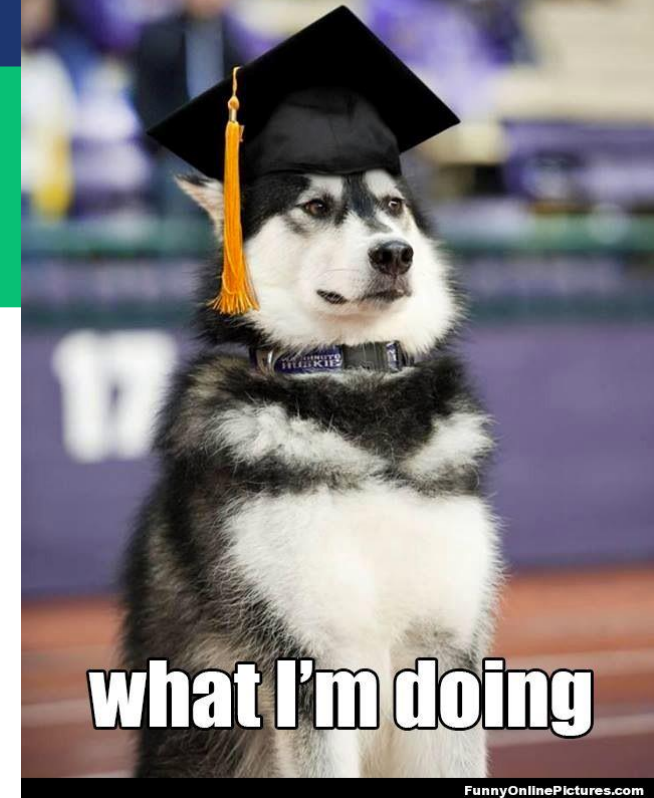
En nu....

Pas deze dingen toe in jouw bedrijfs-/leefwereld!

Bij gebrek aan inspiratie of nood aan goede voorbeelden, gebruik een website zoals **Kaggle**.

Assignment

I still have no idea



what I'm doing

FunnyOnlinePictures.com

Opdracht

Ga naar de notebook

`slides_end_to_end_machine_learning_project.ipynb`

en voeg dingen samen zodat je een volledige end-to-end pipeline maakt die je ook zou kunnen toepassen op ongeziene data. Hiervoor maak je:

- Eerst een split training/test data
- Fine-tune je model (incl cross-validation of andere methodes) op de train set.

Het doel van de opdracht is een eerste ervaring met de ML flow op te doen. Ik verwacht **niet** dat je:

- Geavanceerde technieken toepast &
- volledig begrijpt wat elke functie doet.