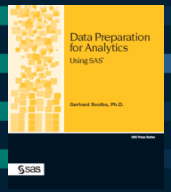
Hands-On Workshop

Machine Learning in Action: Feature Engineering, Modellierung und Modell Validierung in SAS Viya – Der Programmier-Ansatz



Gerhard Svolba, Data Scientist

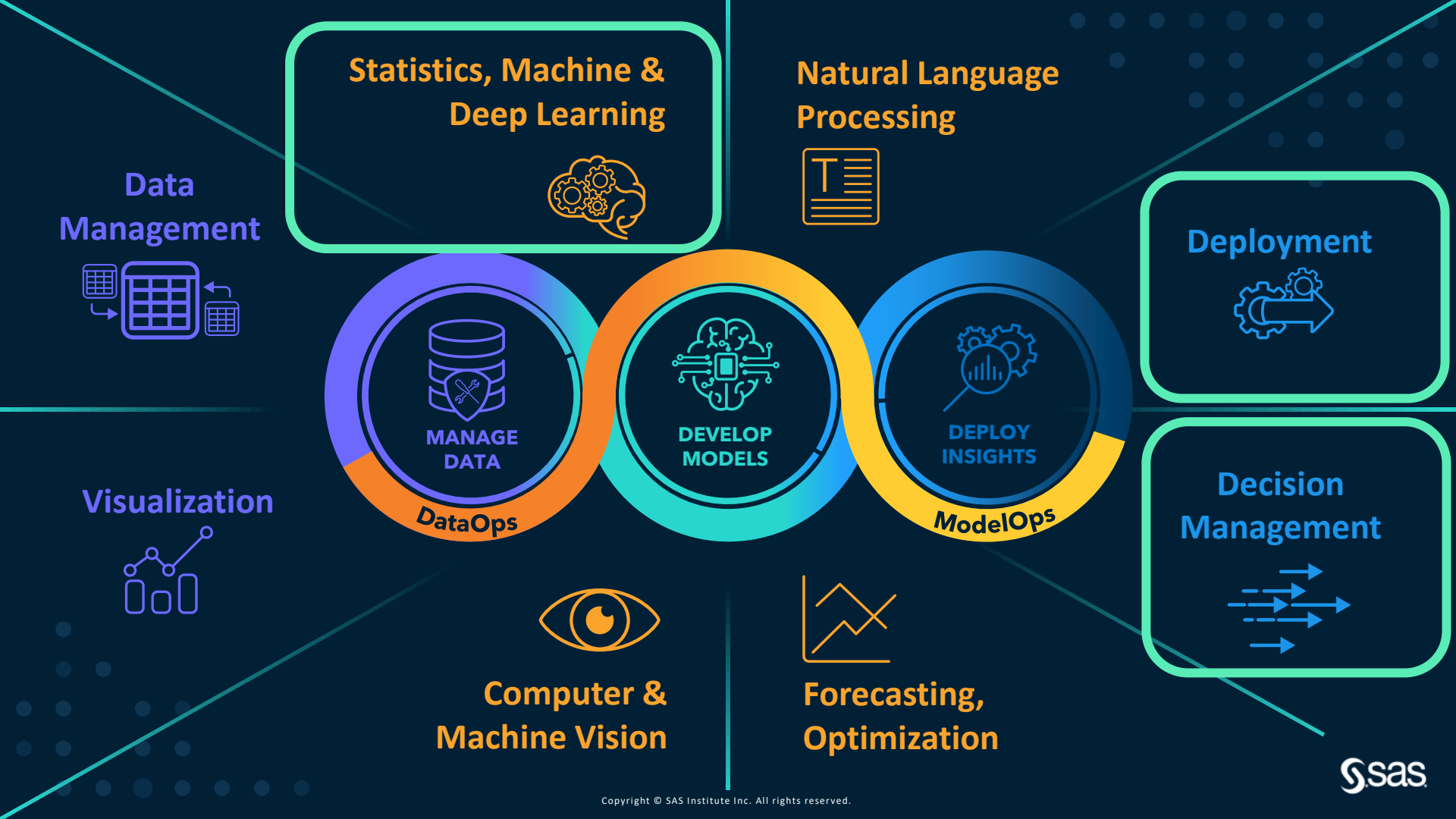
SAS Austria

Data Scientist @SAS - Medium LinkedIn Github SAS-Books SAS Articles
Youtube DataPreparation4DataScience Data Science Use Cases

Hinweis

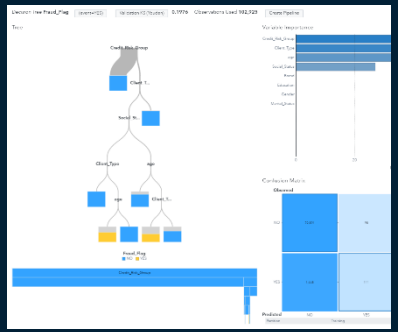
- Ein Beitrag mit Beispiel Code zum Thema dieses Vortrags ist in Vorbereitung. Der Link findet sich dann in dieser Sammlung
- Data Science and Data Preparation Article Overview by Gerhard
  - https://communities.sas.com/t5/SAS-Communities-Library/Data-Science-and-Data-Preparation-Article-Overview-by-Gerhard/ta-p/727875

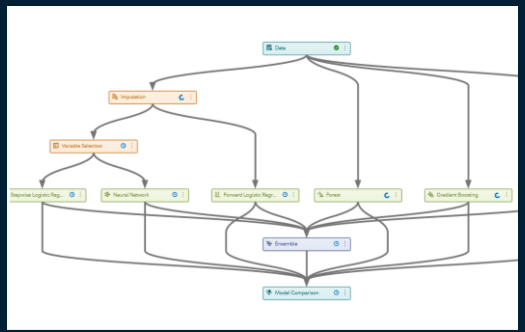
- Bei Fragen vorab kontaktieren sie gerne the Autor unter
  - Sastools.by.gerhard@gmx.net



Möglichkeiten der Interaktion mit der SAS Analytik Plattform

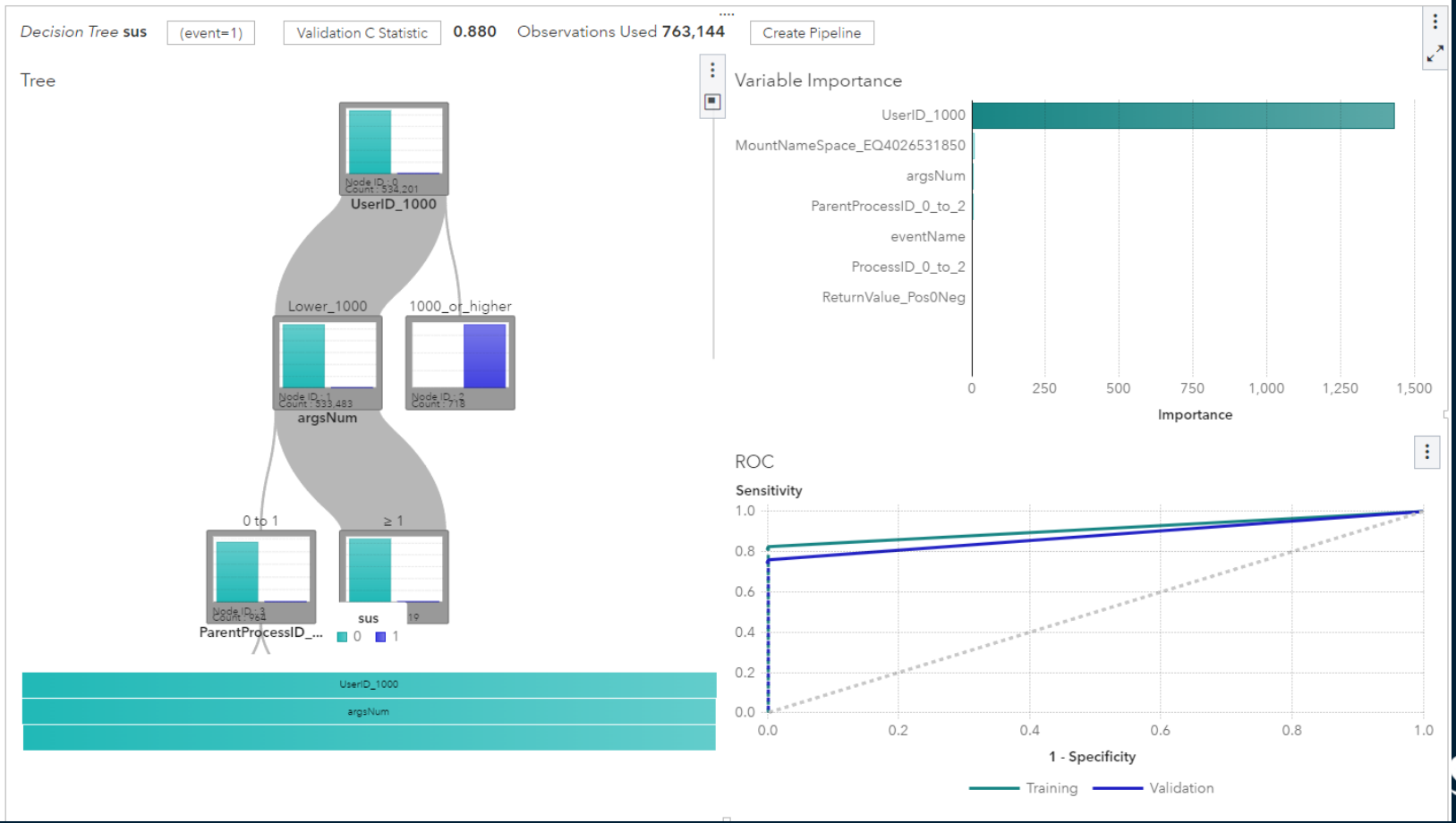
Graphische Benutzeroberfläche		Programmierung	
Visuelle Oberfläche	**Model Studio**	**SAS**	**Open Source Sprache**
Self-Service Analytik-Objekte Integration mit Model Studio & Model Manager	Pipelines und Knoten, Feature-Engineerung, Optionen, Tuning, Open Source Integration, Integration mit dem Model Manager	Volle Flexibilität bei der Programmierung in der SAS Language (Procedures, Actions, Functionen, …) Open Source Integration	Interaktion mit der SAS Analytik-Plattform aus dem Jupyter-Notebook oder R-Studio heraus





```
proc gradboost data=cas1.fc_review
     earlystop(tolerance=0 stagnation=5)
     numBin=20 binmethod=BUCKET
     maxdepth=6
     maxbranch=2
     minleafsize=5
     assignmissing=USEINSEARCH minuseinsearch=1
     seed=12345
     printtarget
;
partition rolevar=_partind_ (TRAIN='1' VALIDATE='0');
autotune useparameters=CUSTOM tuningparameters=(
     lasso(LB=0 UB=10 INIT=0)
     learningrate(LB=0.01 UB=1 INIT=0.1)
     ntrees(LB=20 UB=150 INIT=100)
     ridge(LB=0 UB=10 INIT=0)
     samplingrate(LB=0.1 UB=1 INIT=0.5)
     vars_to_try(LB=1 UB=7 INIT=7)
     )
     searchmethod=GA objective=KS maxtime=900
     maxevals=50 maxiters=5 popsize=10
     targetevent='1'
;
```

```
from swat.render import render_html
from swat import *
from pprint import pprint

import matplotlib.pyplot as plt
import pandas as pd
from pandas import*
import numpy as np

import seaborn as sns
get_ipython().magic('matplotlib inline')
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

# Provide Connection Information and Upload Data it not yet available

cashost='dach-viya33-smp.viya33.sas.com'
casport=5570
casauth='~/.authinfo'
indata_dir="/opt/demodata/sasdata"
indata='new_product_train'
table=indata_dir+"/"+indata+".sas7bdat"
shot_df=pd.read_sas(table)

# Create Demo Instance SAS Club and Load Actions Sets for Decision Trees
SASClub = CAS(cashost, casport, authinfo="~/.authinfo", caslib="casuser")

SASClub.loadactionset(actionset="decisionTree")

if not SASClub.table.tableExists(table=indata).exists:
    tbl = SASClub.upload_file(indata_dir+"/"+indata+".sas7bdat", casout={"name":indata})
NOTE: Added action set 'decisionTree'.
```


Decision Tree sus
(event=1)
Validation C Statistic
0.880
Observations Used 763,144
Create Pipeline
Tree
Node ID : 0
Count : 534,201
UserID_1000
Lower_1000
1000_or_higher
Node ID : 1
Count : 533,483
argsNum
Node ID : 2
Count : 718
0 to 1
≥ 1
Node ID : 3
Count : 964
ParentProcessID_...
sus
19
0
1
UserID_1000
argsNum
Variable Importance
UserID_1000
MountNameSpace_EQ4026531850
argsNum
ParentProcessID_0_to_2
eventName
ProcessID_0_to_2
ReturnValue_Pos0Neg
0
250
500
750
1,000
1,250
1,500
Importance
ROC
Sensitivity
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
1 - Specificity
Training
Validation

CyberSecurity_BETH_TRN

Data | Pipelines | Pipeline Comparison | Insights

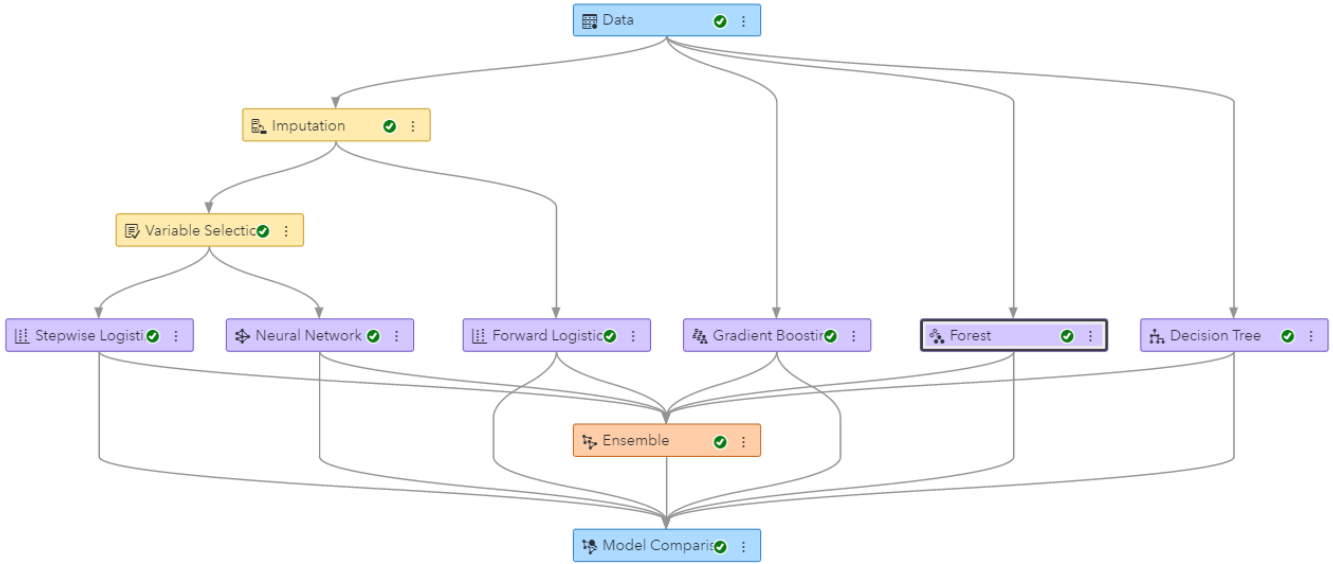
Nodes

Filter

- Data Mining Preprocessing
  - Anomaly Detection
  - Clustering
  - Feature Extraction
  - Feature Machine
  - Filtering
  - Imputation
  - Interactive Grouping
  - Manage Variables
  - PCA1
  - Reject Inference
  - Replacement
  - Text Mining
  - Transformations
  - Variable Clustering
  - Variable Selection
- Supervised Learning
  - Batch Code
  - Bayesian Network
  - Decision Tree
  - Forest
  - Forest_1
  - Forest_2
  - Forest_3

SVM from VisualAnalytics | AdvTemplate

Run Pipeline



Context: Cyber Security

- … anomalous data points and shifts in the data distribution are inevitable. From a cyber security perspective, these anomalies and dataset shifts are driven by both defensive and adversarial advancement.
- To withstand the cost of critical system failure, the development of robust models is therefore key to the performance, protection, and longevity of deployed defensive systems.
- We present the BPF-extended tracking honeypot (BETH) dataset as the first cybersecurity dataset for uncertainty and robustness benchmarking. Collected using a novel honeypot tracking system, our dataset has the following properties that make it attractive for the development of robust ML methods:
  1. At over eight million data points, this is one of the largest cyber security datasets available
  2. It contains modern host activity and attacks
  3. It is fully labelled
  4. It contains highly structured but heterogeneous features
  5. Each host contains benign activity and at most a single attack, which is ideal for behavioural analysis and other research tasks. In addition to the described dataset

BETH Dataset: Real Cybersecurity Data for Anomaly Detection Research

Feature	Type	Description
timestamp	float	Seconds since system boot
processId*	int	Integer label for the process spawning this log
threadId	int	Integer label for the thread spawning this log
parentProcessId*	int	Parent's integer label for the process spawning this log
userId*	int	Login integer ID of user spawning this log
mountNamespace*	int (long)	Set mounting restrictions this process log works within
processName	String	String command executed
hostName	String	Name of host server
eventId*	int	ID for the event generating this log
eventName	String	Name of the event generating this log
argsNum*	int	Length of args
returnValue*	int	Value returned from this event log (usually 0)
stackAddresses	list of int	memory values relevant to the process
args	list of dictionaries	List of arguments passed to this process
sus	int (0 or 1)	Binary label as a suspicious event (1 is suspicious, 0 is not)
evil	int (0 or 1)	Binary as a known malicious event (0 is benign, 1 is not)

https://www.kaggle.com/datasets/katehighnam/beth-dataset

Pre-Processing / Feature Engineering

See Appendix A in the paper

processId: Process IDs 0, 1, and 2 are meaningful since these are always values used by the OS, but otherwise a random number is assigned to the process upon creation. We recommend replacing `processId` with a binary variable indicating whether or not `processID` is 0, 1, or 2.

threadId: While this value did not appear useful in our analysis, it might suggest how to link process calls if obfuscated in the system. No conversion is recommended at this time.

parentProcessId: Same as `processId`, the same mapping to a binary variable should suffice.

userId: The default in Linux systems is to assign OS activity to some number below 1000 (typically 0). As users login, they are assigned IDs starting at 1000, incrementally. This can be altered by a user, but none of the current logs gave evidence an attacker did this. We used a binary variable to indicate `userId` < 1000 or `userId` ≥ 1000. Alternatively, one could use an ordinal mapping that buckets all `userId` < 1000 at zero and then increment upwards for each new user. Also, no more than four logins were viewed per host in our current datasets.

BETH Dataset: Real Cybersecurity

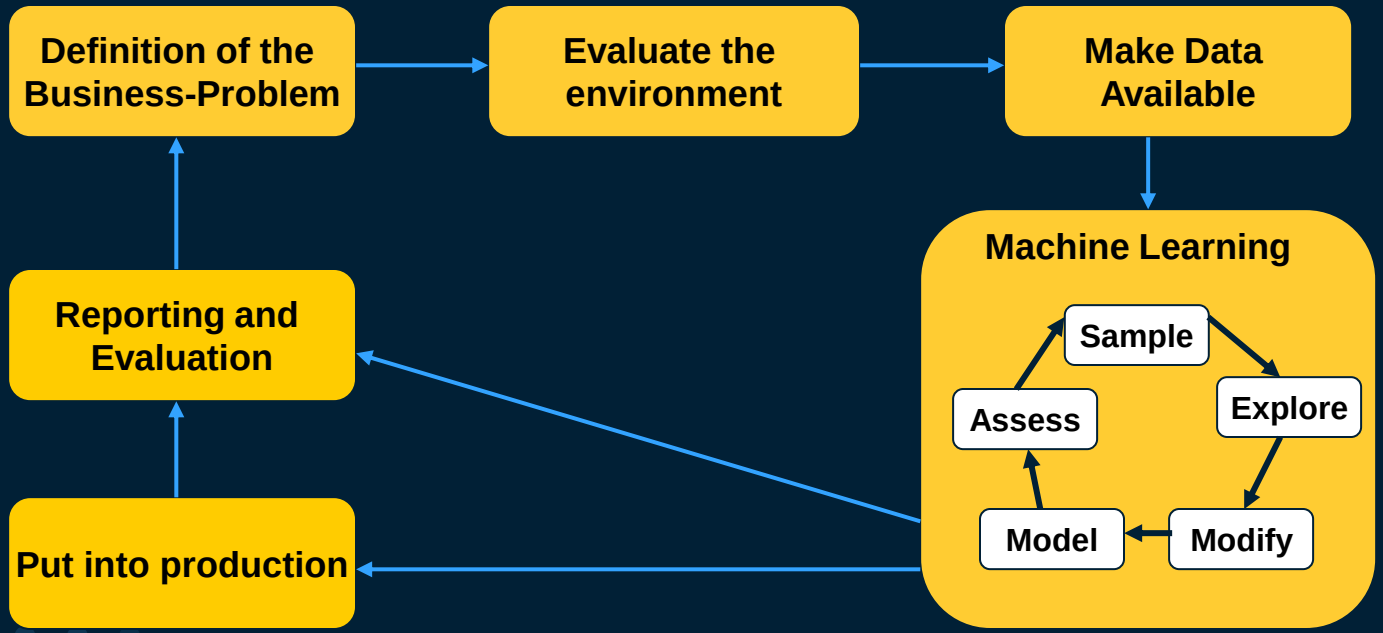
argsNum: This raw feature is included as-is, since, at this time, adequately parsing `args` requires either more sophisticated pre-processing or a more complex ML model.

returnValue: This is also called the exit status and can be used to determine whether a call completed successfully or not. Mappings for this can vary, as this value is decided between the parent and child process. We mapped `returnValue` into three values based on the common usage of the field: -1 when negative (bad errors), 0 when zero (success), and 1 when positive (success and signalling something to the parent process).

stackAddresses: It is difficult to clearly relate this feature during manual analysis and the large values within a variable size list make processing automatically difficult without encoding or learning an extra embedding. Thus this field was dropped from training our baselines.

args: There are many options in this variable list of dictionaries. For simplicity, we refrain from utilising any of these values. However, more features can and should be created for future work.

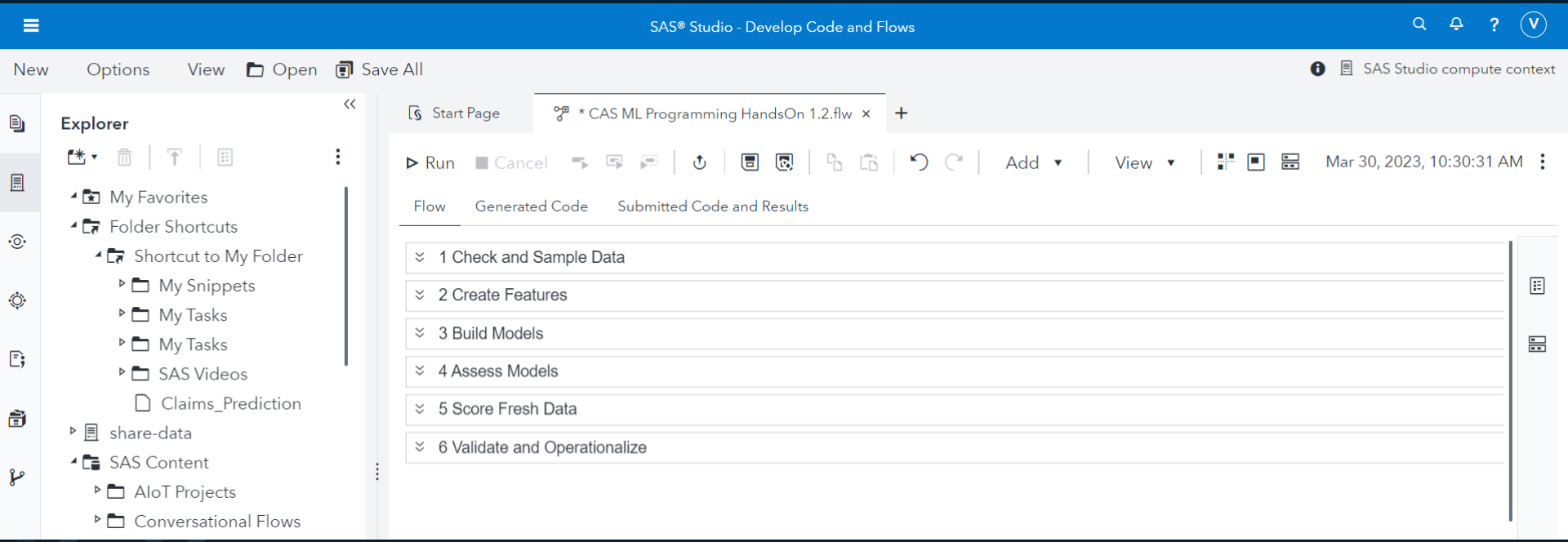
Machine Learning as part of a „Closed Loop“ Environment



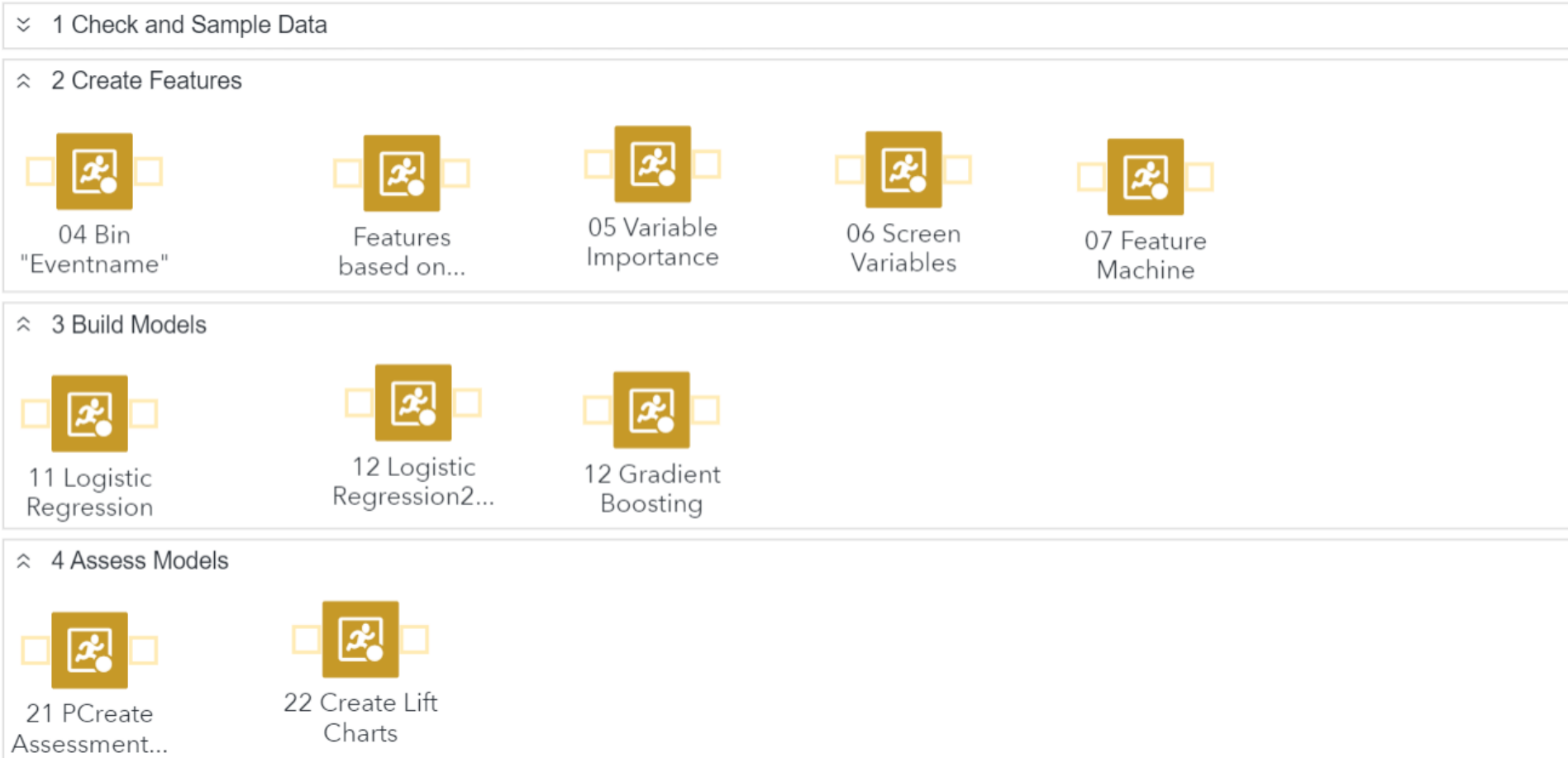
SAS Institute METHODOLOGY Data Mining Projects

sas

Flow SAS Studio (Viya4)

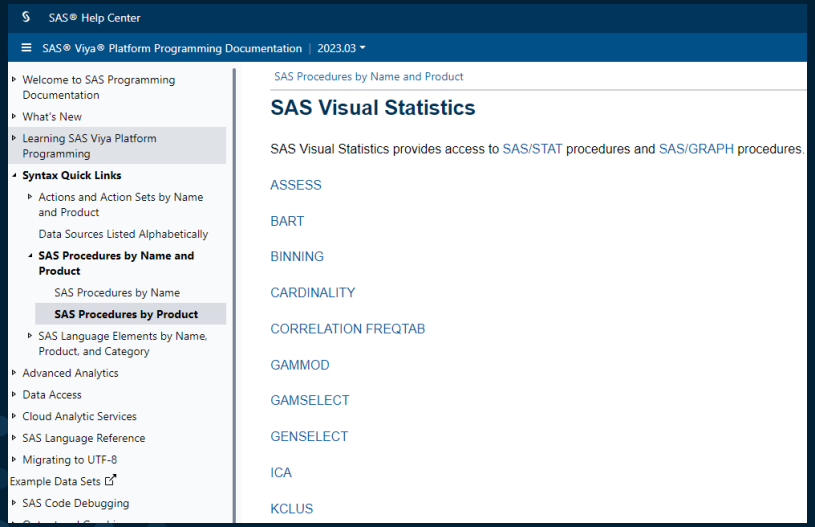


Flow SAS Studio (Viya4)

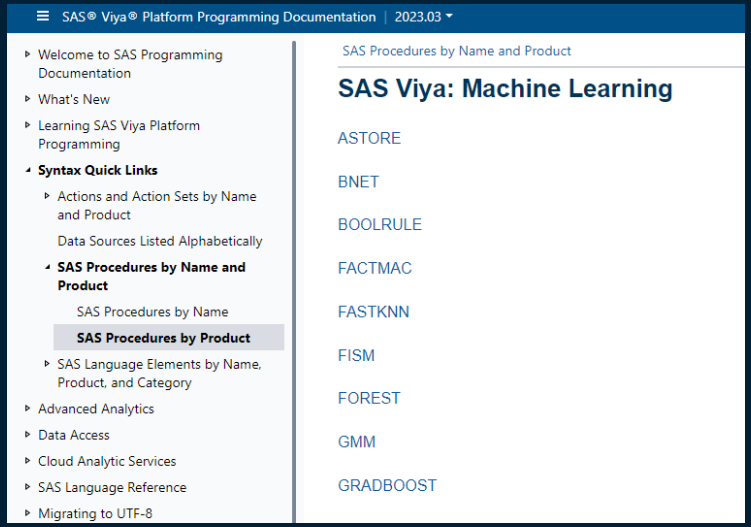


Ausgewählte SAS Procedurs in SAS Viya

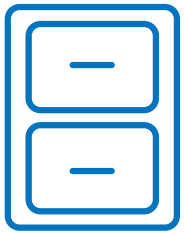
- SAS Visual Statistics



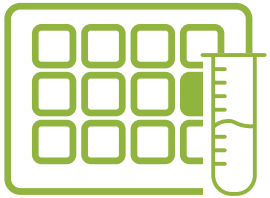
- SAS Viya: Machine Learning



Data Preparation

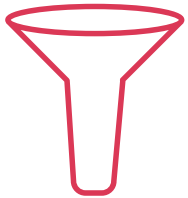


binning



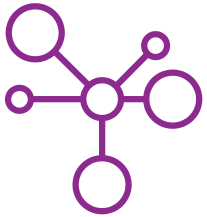
sample

transformation

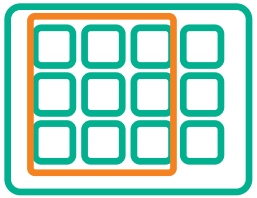


filter

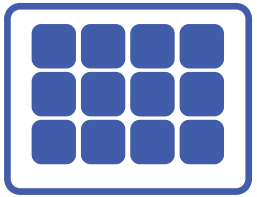
re-scaling



variable clustering



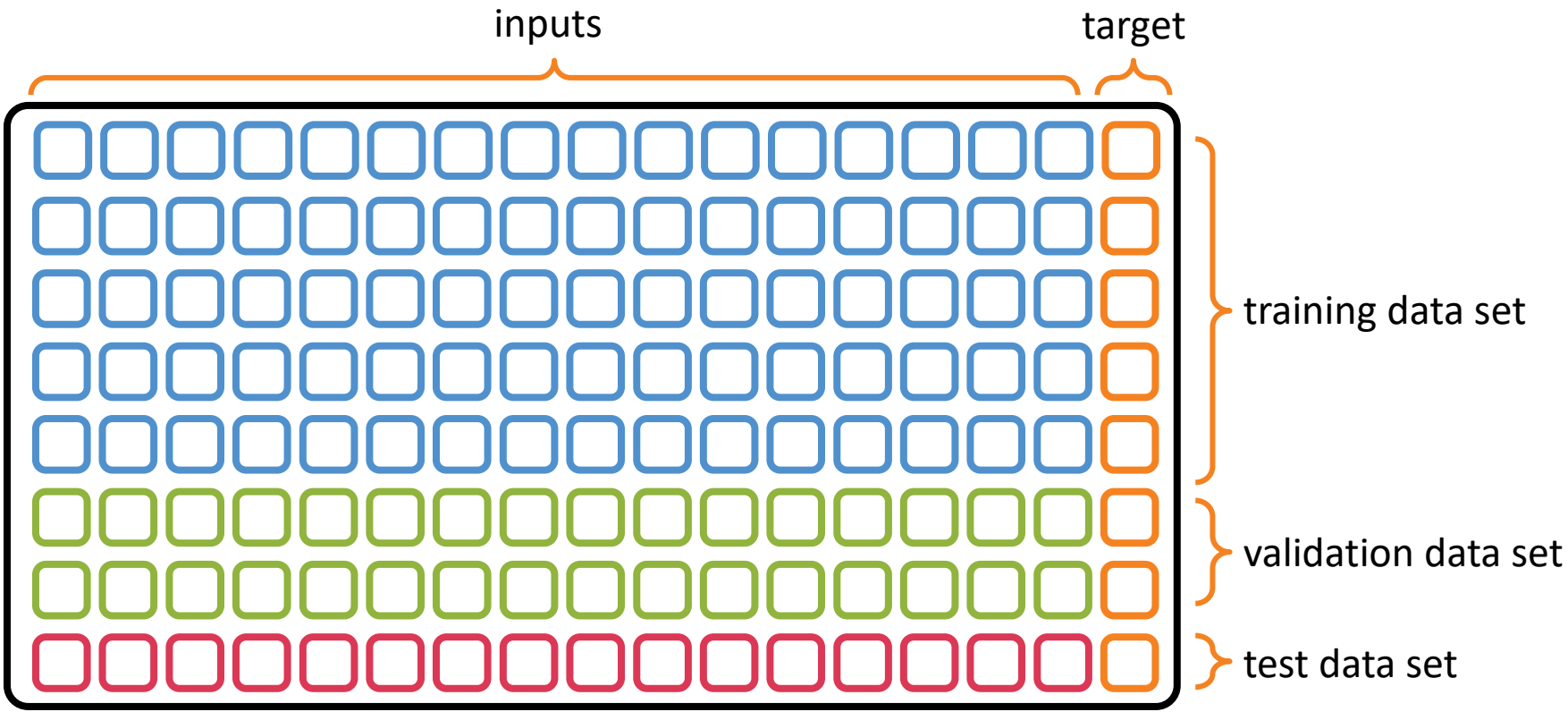
custom subset selection



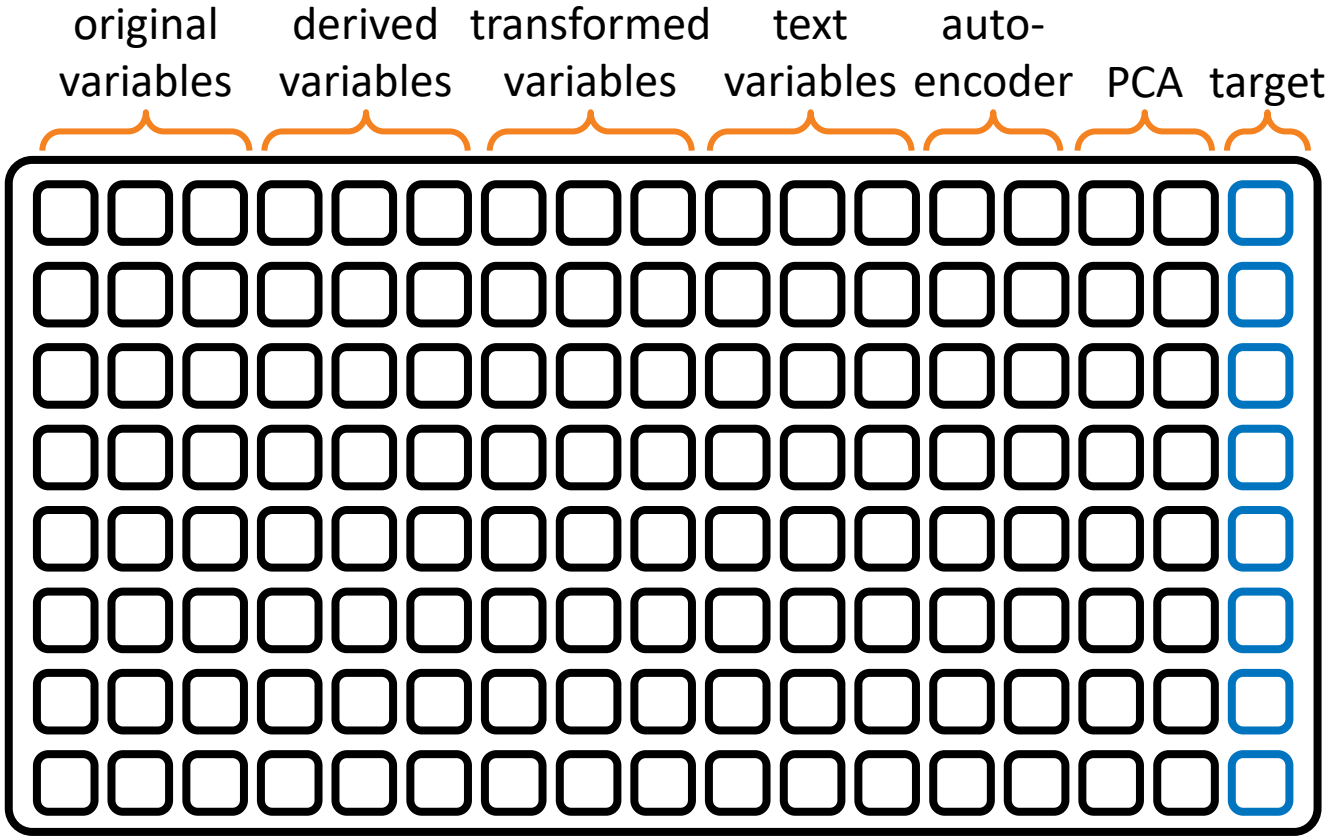
imputation

feature extraction

Partitioning

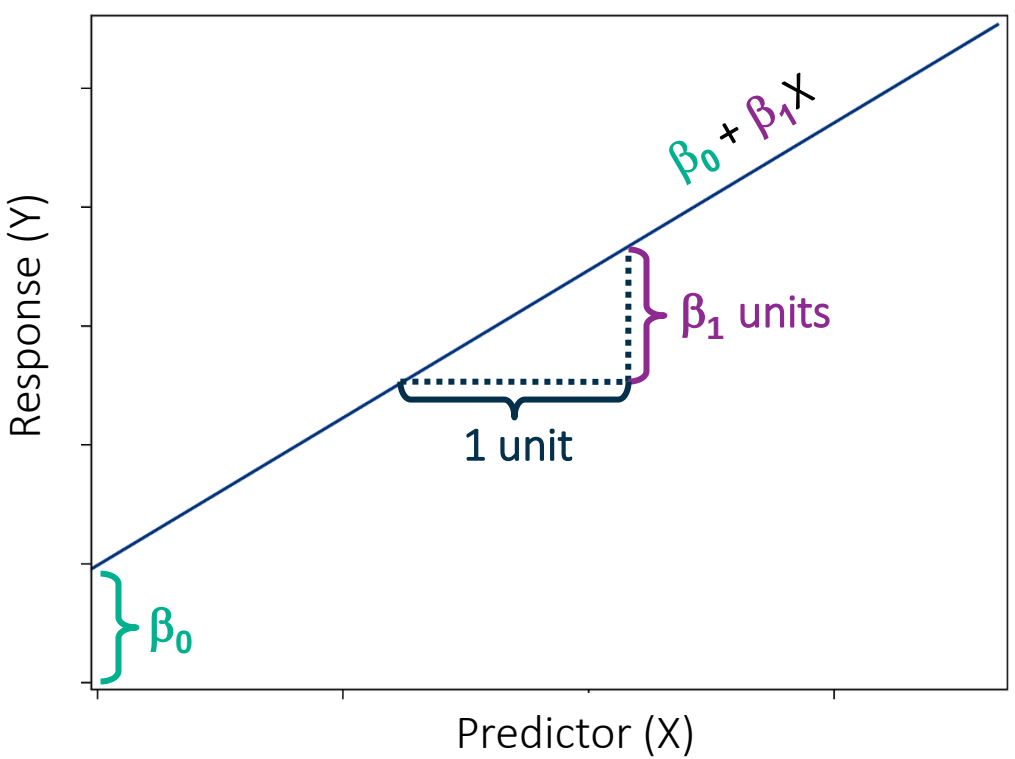


Feature Extraction

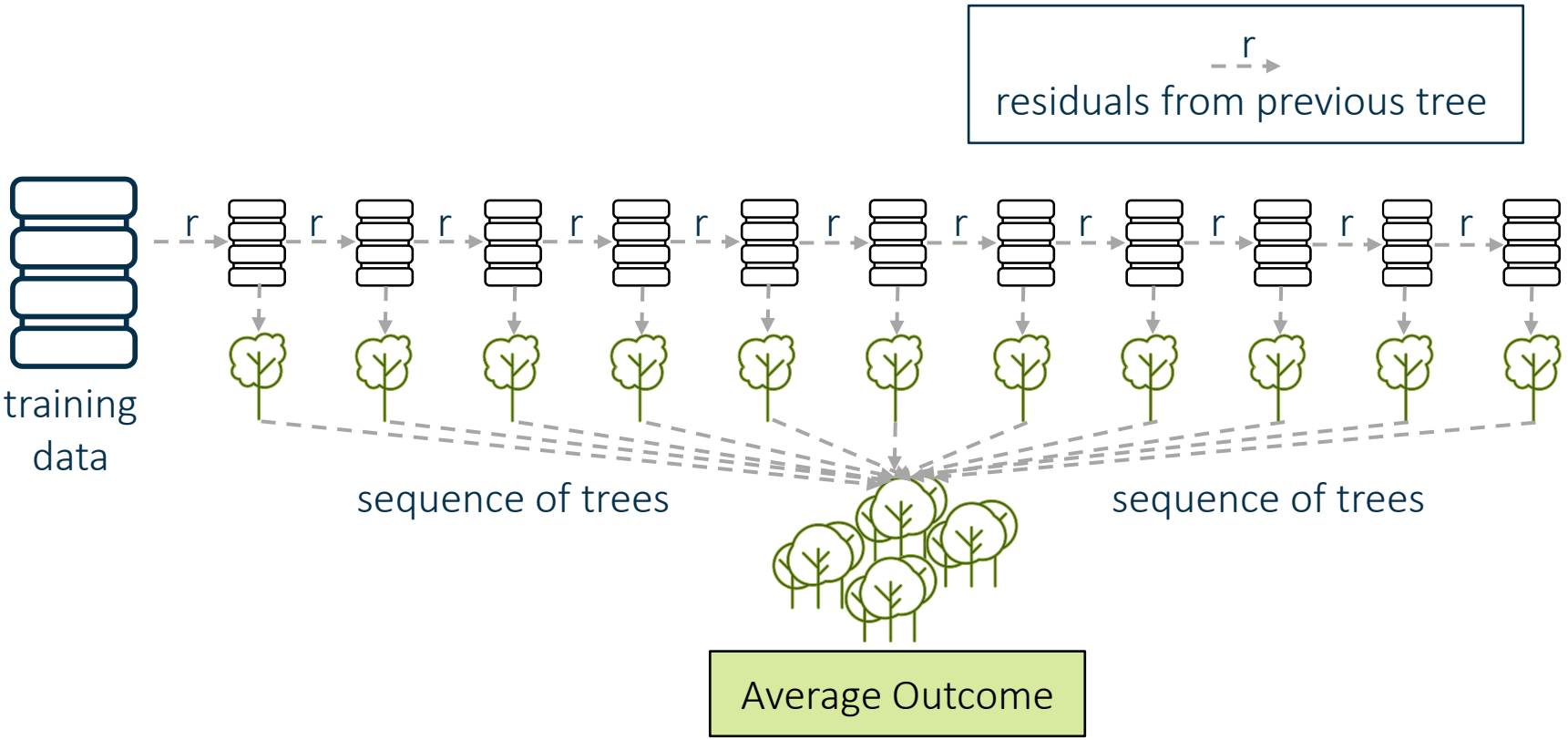


Ordinary Least Squares (OLS) Regression Model

$$y_i=\beta_0+\beta_1 x_i+\varepsilon_i$$



Ensemble of Trees: Gradient Boosting



Lift Value

- Multiplicative Factor how your predictive model is better than a random selection.