QSPRPRED

A FLEXIBLE OPEN-SOURCE QUANTITATIVE STRUCTURE-PROPERTY RELATIONSHIP MODELLING TOOL

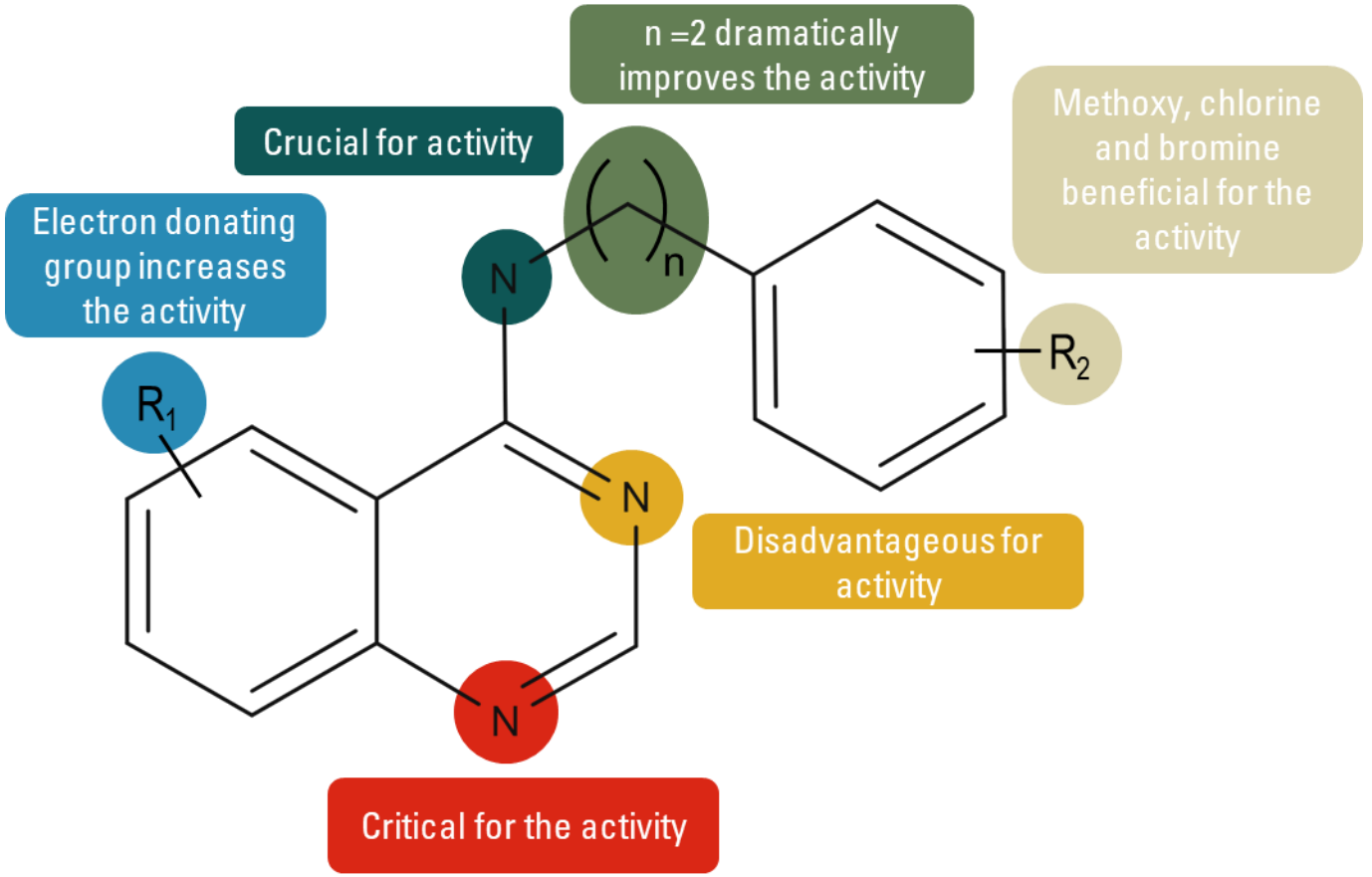
LACDR

Universiteit Leiden
The Netherlands

Helle W. van den Maagdenberg, Linde Schoenmaker, Martin Šícho, Olivier J. M. Béquignon, Sohvi Luukkonen, David Araripe, J.G. Coen van Hasselt, Piet H. van der Graaf, and Gerard J. P. van Westen

WHAT IS QSPR: QUANTITATIVE STRUCTURE-PROPERTY RELATIONSHIP MODELLING?

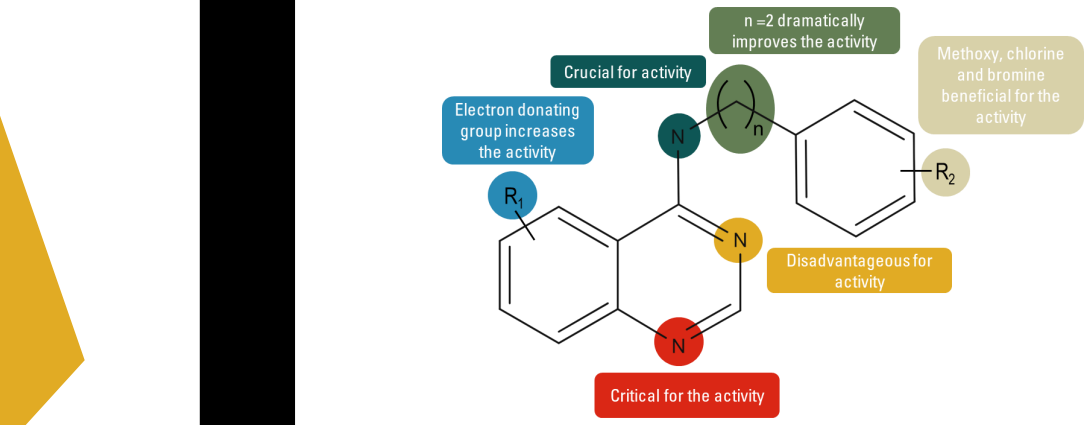
- Prediction of chemical bioactivity and physical properties from the molecular structure
- Data can be retrieved from online databases such as ChEMBL or in-house obtained data
- Using statistical, machine learning and artificial intelligence methods



 *Example SAR study of Spautin-1 for novel NEK4 inhibitors from Elsocht et al. *Int J Mol Sci*. 2020

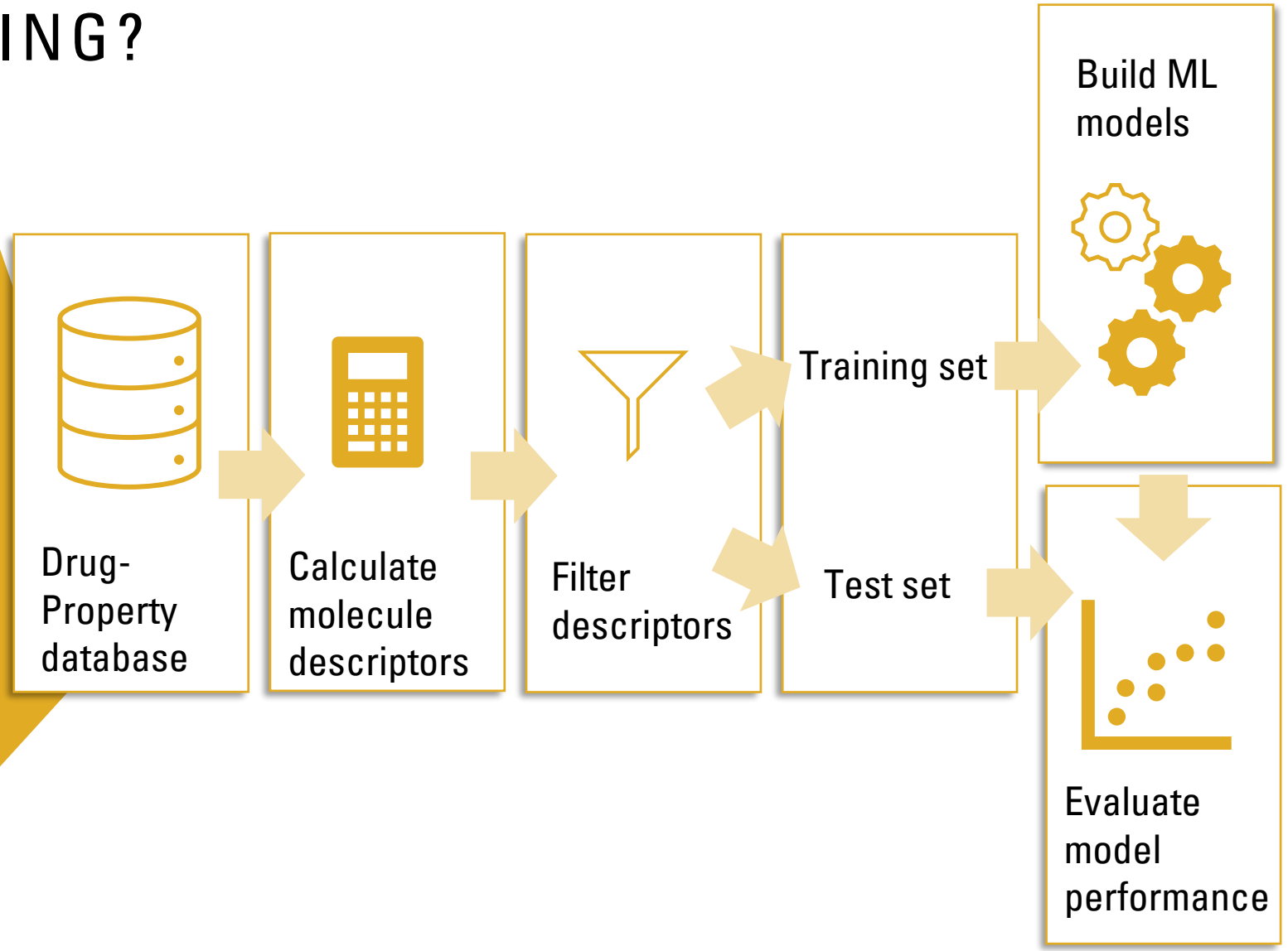
WHAT IS QSPR: QUANTITATIVE STRUCTURE-PROPERTY RELATIONSHIP MODELLING?

- Prediction of chemical bioactivity and physical properties from the molecular structure
- Data can be retrieved from online databases such as ChEMBL or in-house obtained data
- Using statistical, machine learning and artificial intelligence methods



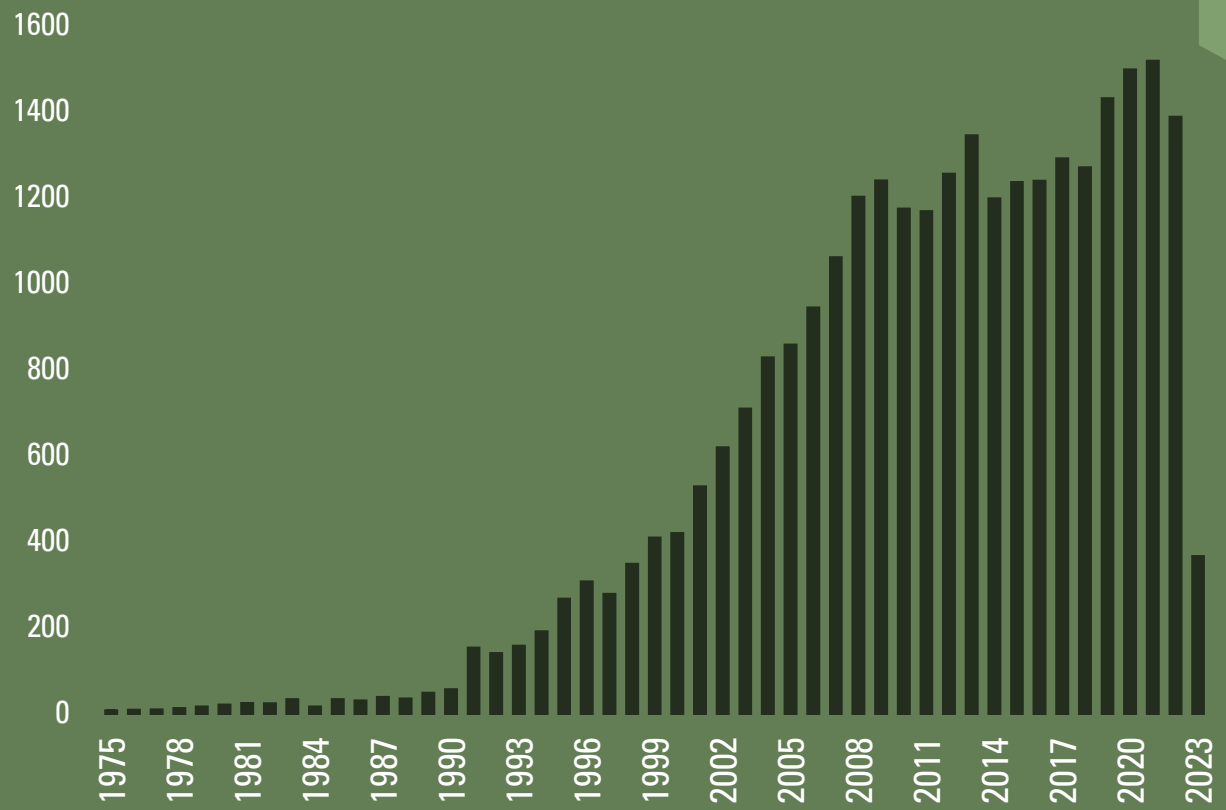
WHAT IS QSPR: QUANTITATIVE STRUCTURE-PROPERTY RELATIONSHIP MODELLING?

- Prediction of chemical bioactivity and physical properties from the molecular structure
- Data can be retrieved from online databases such as ChEMBL or in-house obtained data
- Using statistical, machine learning and artificial intelligence methods



WHY DO WE NEED A TOOL FOR QSPR MODELLING?

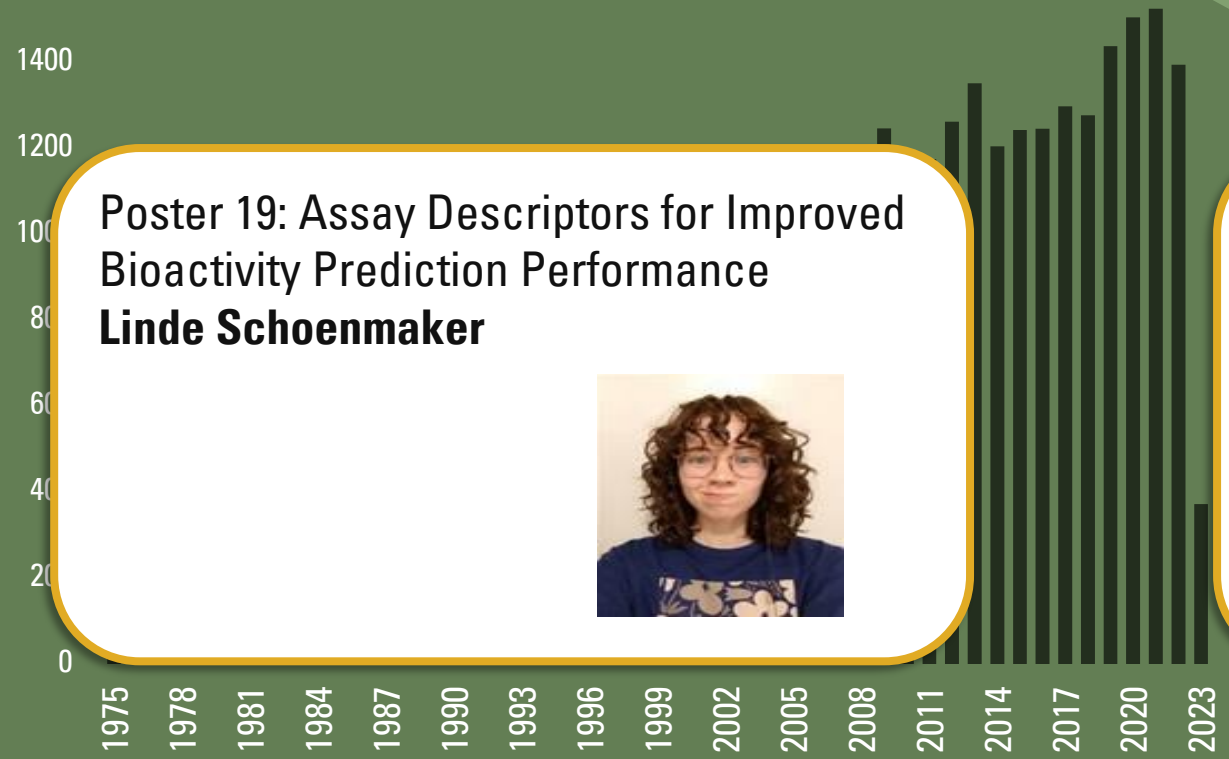
Number of papers on QSPR modelling from Web of Science



- QSPR modelling is used in both industry and academia.
- Development typically involves common steps and components.
- Cheminformaticians prefer the flexibility of Python over available tools.
- Experimenting with different models and workflows increases code complexity.

Poster 2: Generate What You Can Make: De Novo Generation of In-House Synthesizable Drug Candidates
Alan Kai Hassen

Poster 5: AlphaFold Meets Drug Design: A Novel Method for de novo Drug Discovery
Andrius Bernatavicius



Poster 19: Assay Descriptors for Improved Bioactivity Prediction Performance
Linde Schoenmaker

Poster 24: LED3Score in De Novo Design of Synthetically Accessible Inhibitors of Monoglyceride Lipase
Martin Šícho

TOOL
G?

oth industry

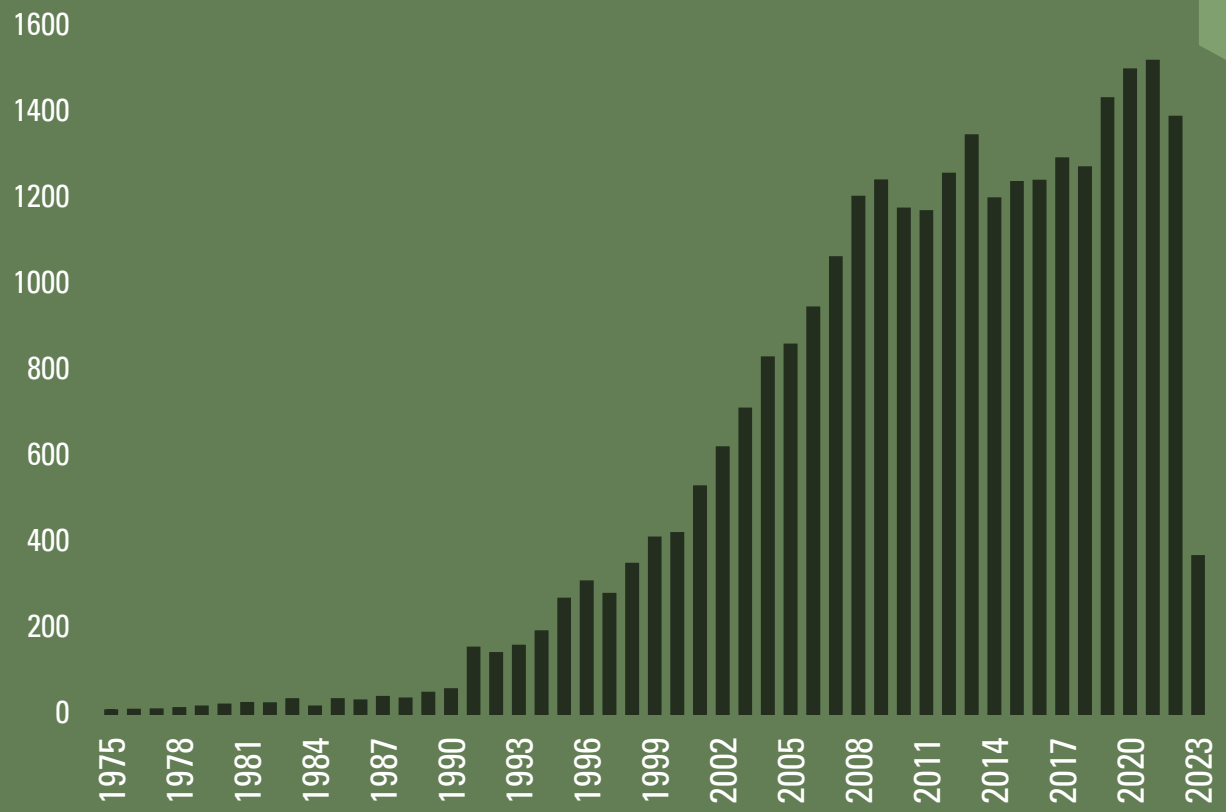
- Development typically involves common steps and components

flexibility of

models and
nplexity.

WHY DO WE NEED A TOOL FOR QSPR MODELLING?

Number of papers on QSPR modelling from Web of Science



- QSPR modelling is used in both industry and academia.
- Development typically involves common steps and components.
- Cheminformaticians prefer the flexibility of Python over available tools.
- Experimenting with different models and workflows increases code complexity.

A flexible and easy to use Quantitative Structure-Property Relationship Modelling Framework: QSPRpred

- Modular: simple to add new models, descriptors, etc.
- Easy to use: Includes Command Line Interface, Python API and tutorials.
- New features are regularly added
- Compatible with *de novo* drug design package: DrugEx*
- Code available on GitHub and pip installable

* Šícho M. *et al.*, *J. Chem. Inf. Model.* 2023

A flexible and easy to use Quantitative Structure-Property Relationship Modelling Framework: QSPRpred

- Modular: simple to add new models, descriptors, etc.
- Easy to use: Includes Command Line Interface, Python API and tutorials.
- New features are regularly added
- Compatible with *de novo* drug design package: DrugEx*
- Code available on GitHub and pip installable

QSPRpred

* Šícho M. *et al.*, *J. Chem. Inf. Model.* 2023

THE QSPRPRED WORKFLOW: FUNCTIONALITIES

Pre-processing
- SMILES sanitization & standardization
- Data transformation
- Exclusion by value
- Class creation

Descriptor calculation:
- Morgan Fingerprints
- RDKit descriptors
- Mordred descriptors
- Etc.
- **Protein descriptors**

Filter best descriptors:
- High correlation
- Low variability
- Boruta filter

Training set

Test set

Split types:
- Random
- Temporal
- Scaffold

Build ML models

Model types:
- Scikit-learn & PyTorch
- Regression
- Classification
- Multi-class
- Multi-task
- **Proteo-chemometrics**

Evaluate model performance

1. Hyperparameter optimization
   - Grid search
   - Bayes optimization
2. Cross-validation
3. Visualization

Data collection with Papyrus*
- Curated ligand-protein bioactivity data
- Multiple publicly available datasets such as ChEMBL
- Out-of-the-box usable with QSPRpred


*Béquignon, O.J.M., *et al. J. Cheminf. 2023*

HELPFUL TUTORIALS TO GET STARTED

Data Preparation

In this tutorial, you will learn how to prepare data sets with QSPRPred.

Data Representation (`PandasDataSet`)

The package basically uses wrapped `pandas.DataFrame` objects with some useful functions added on top to facilitate features relevant for QSPR modeling. The `PandasDataSet` class is the base class of all data sets in QSPRPred. Wrapping a `pandas.DataFrame` is easy:

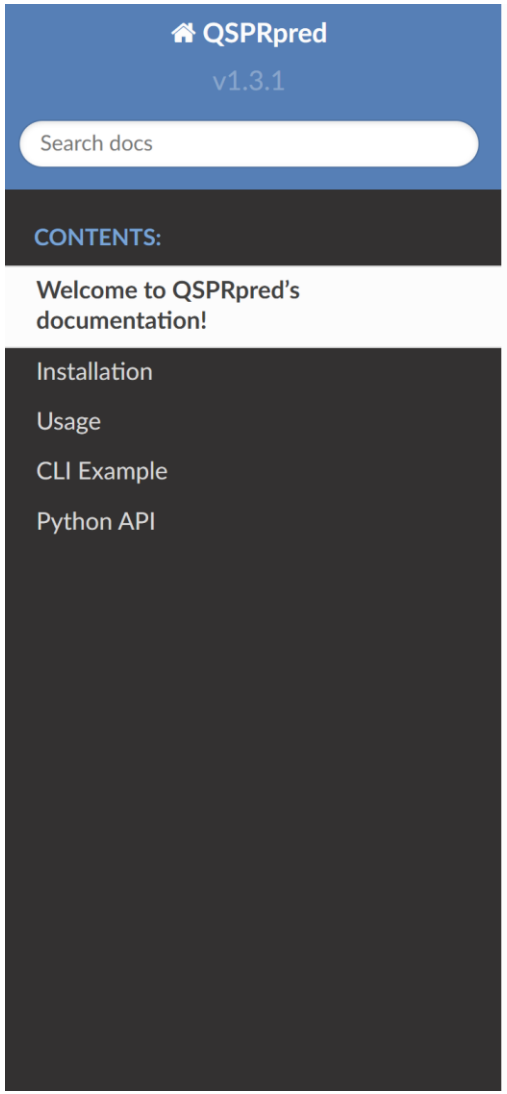
In [1]:
```
# load a sample data set on parkinson's disease
import pandas as pd

df = pd.read_table('data/parkinsons_pivot.tsv')
df
```

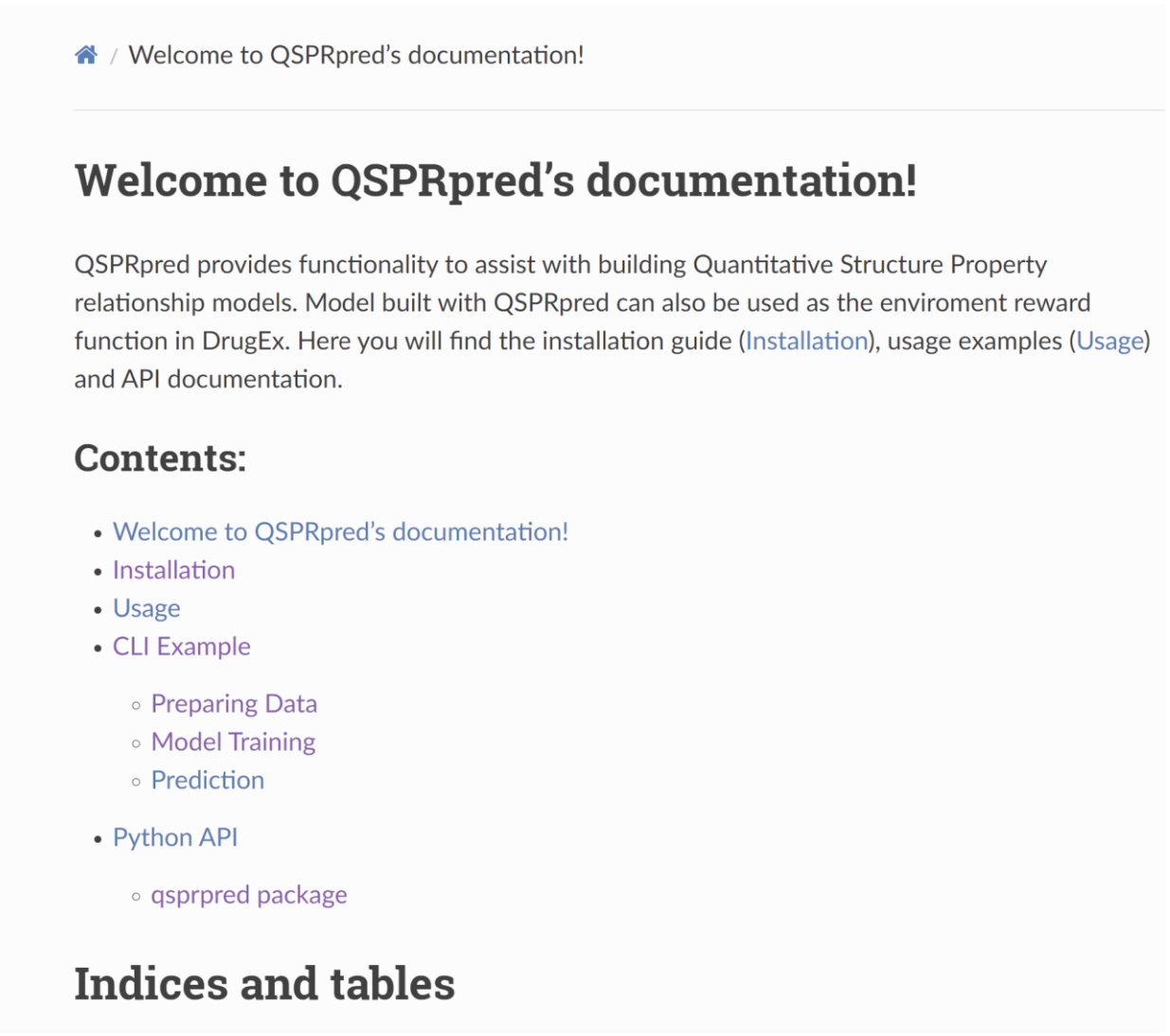
In [2]:
```
from qsprpred.data.data import PandasDataSet

ds = PandasDataSet(df=df, store_dir="data", name="parkinsons")
ds
```

Out [2]:
```
<qsprpred.data.data.PandasDataSet at 0x7f2eaf589240>
```

You can query this data set directly for simple information like the number of samples:

EXTENSIVE DOCUMENTATION





QSPRpred

EXAMPLE CASE STUDY: REIMPLEMENTING A CLASSIFICATION MODEL FOR CYTOCHROME P450 ENZYME SUBSTRATES

CYP ENZYMES: A CASE STUDY

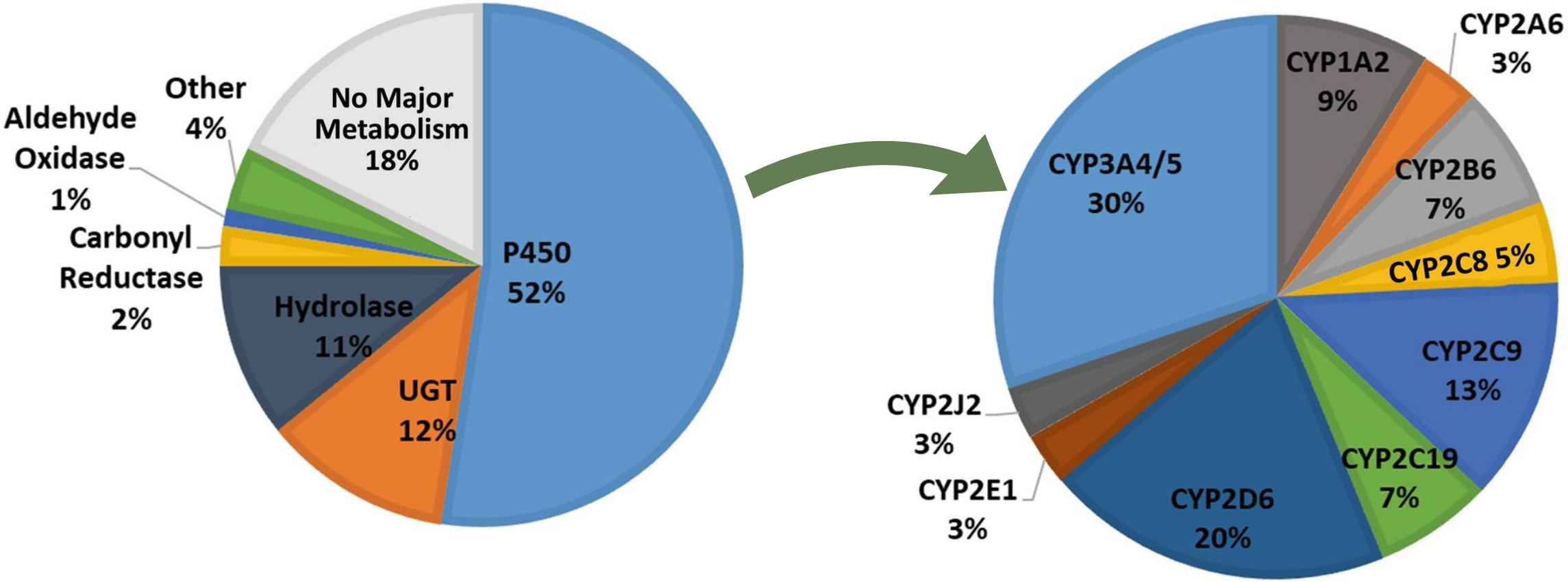
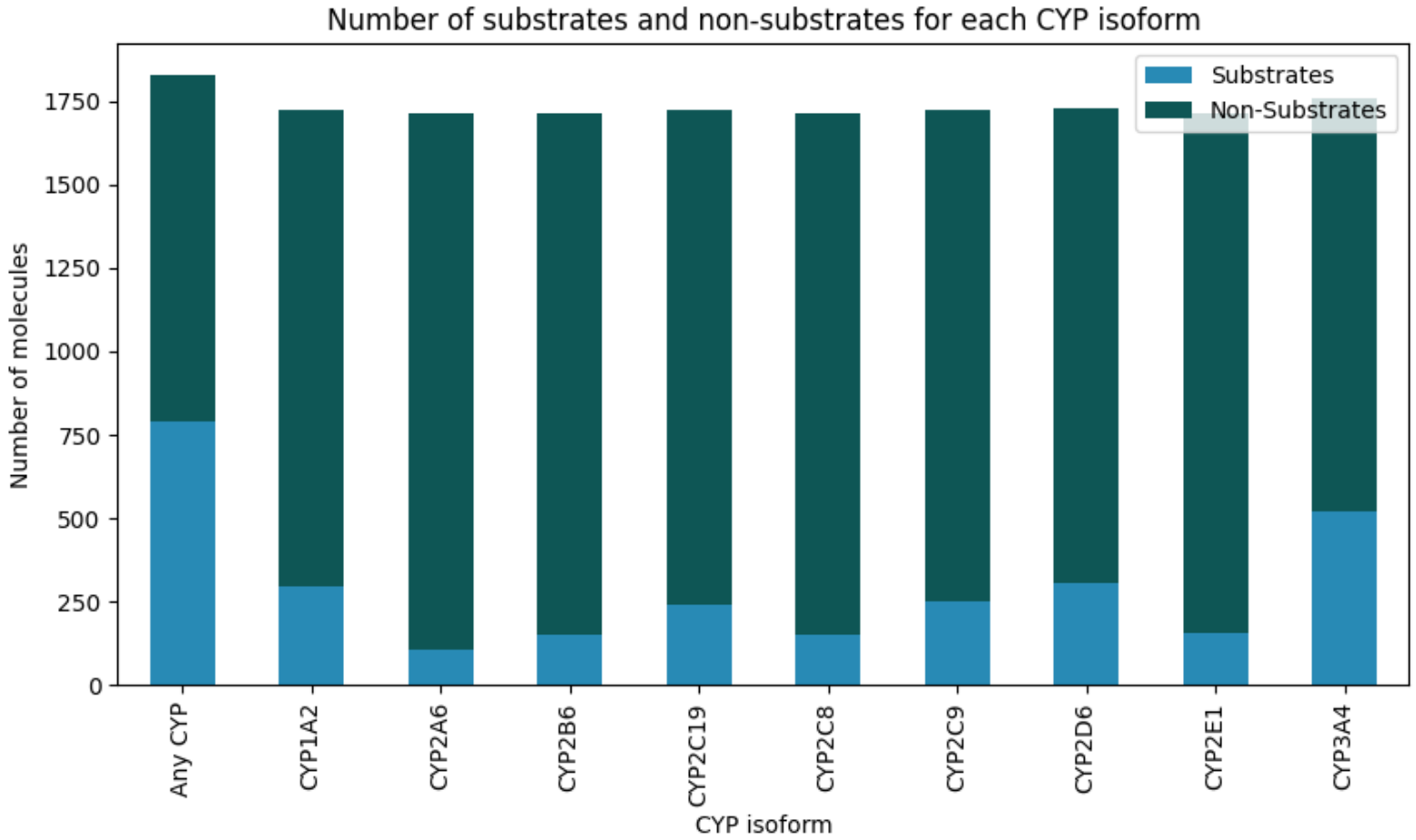


*Image Johnson M. *et al.*, *Methods in Pharmacology and Toxicology*, 2021

CYPSTRATE

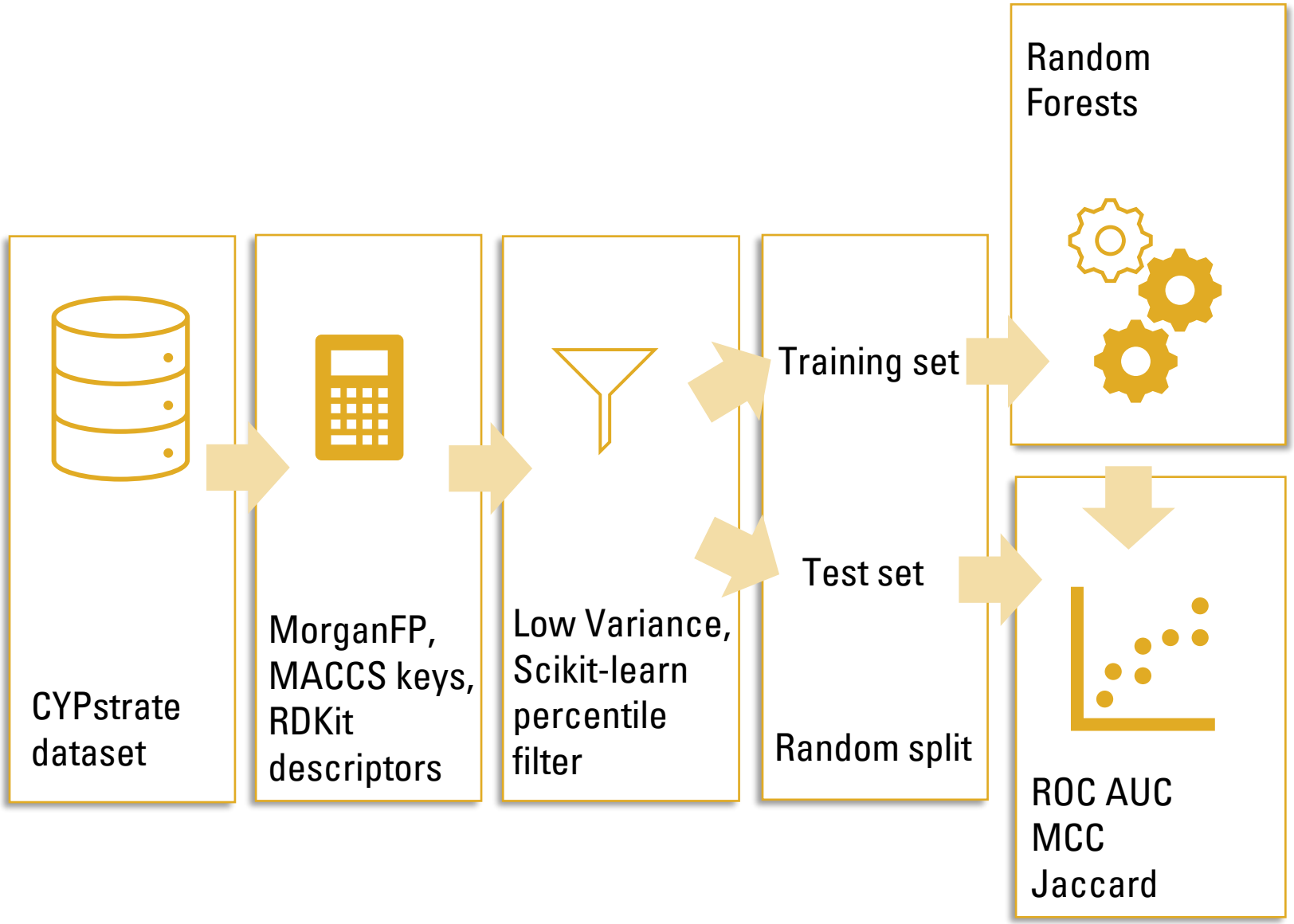
- Published QSPR models for CYP substrates by Holmer et al.*
- Dataset of ~1800 CYP substrates and non-substrates
- Single task classifiers and ensemble models
- Reimplemented random forest single task models with QSPRpred



*Holmer M. *et al.*, *Molecules.* 2021

CYPSTRATE

- Published QSPR models for CYP substrates by Holmer et al.*
- Dataset of ~1800 CYP substrates and non-substrates
- Single task classifiers and ensemble models
- Reimplemented random forest single task models with QSPRpred



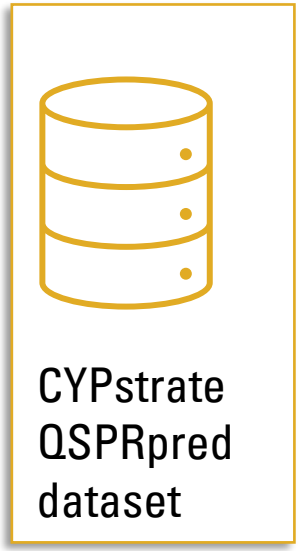
*Holmer M. *et al.*, *Molecules.* 2023

CREATING THE DATASET

SMILES	datasplit	CYP1A2	CYP2A6	CYP2B6	CYP2C19	CYP2C8	CYP2D6	CYP2E1	CYP3A4
BrCCBr	test	NaN	NaN	NaN	0.0	NaN	NaN	NaN	NaN
BrCCBr	train	0.0	1.0	1.0	NaN	0.0	0.0	1.0	0.0
C#CC#CC#CCCO	test	NaN	NaN	0.0	0.0	NaN	NaN	NaN	0.0
C#CC#CC#CCCO	train	0.0	0.0	NaN	NaN	0.0	0.0	0.0	NaN
…	…	…	…	…	…	…	…	…	…

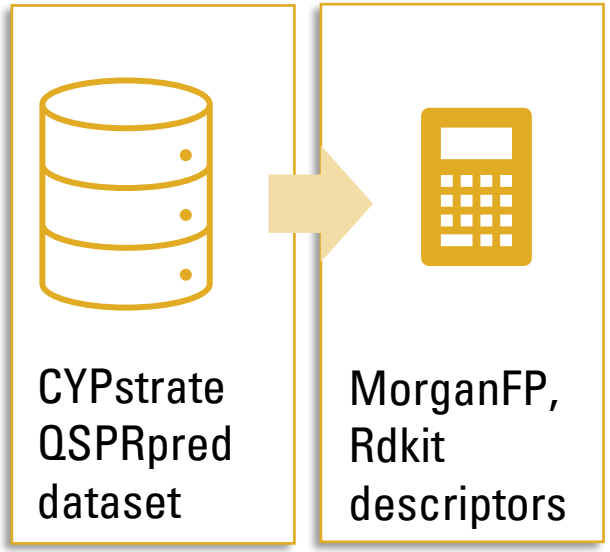
```
dataset = QSPRDataset(df=df,
                      store_dir=DATA_PATH},
                      name="CYP2C19",
                      target_props=[{"name": "CYP2C19",
                                     "task": TargetTasks.SINGLECLASS,
                                     "th"  :"precomputed"}],
                      n_jobs=8)
```

PREPARING THE DATASET



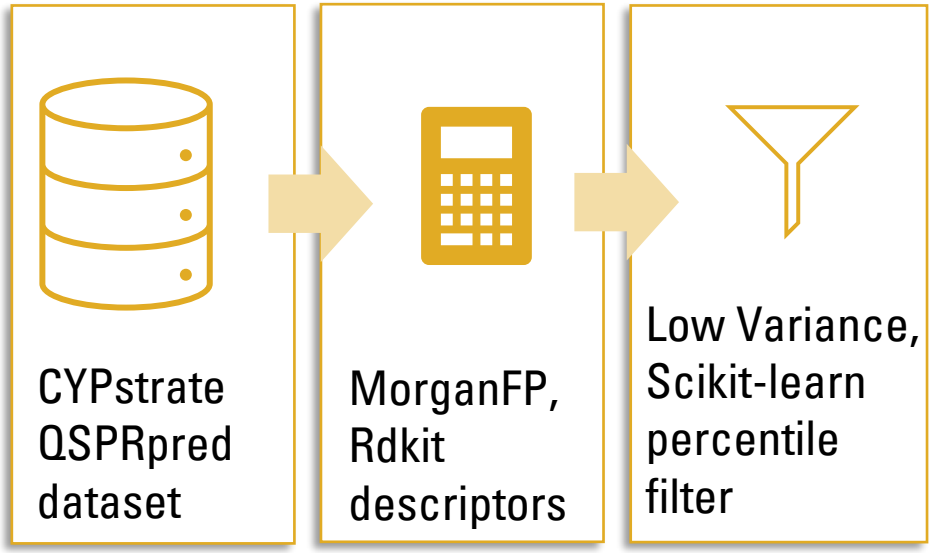

```
calculator = MoleculeDescriptorsCalculator([FingerprintSet(fingerprint_type="RDKitMACCSFP")])
filters = [lowVarianceFilter(th=0), SklearnSelectPercentile(percentile=percentile)]
split = ManualSplit(splitcol=dataset.getDF()["datasplit"], trainval="train", testval="test")

dataset.prepareDataset(smiles_standardizer = None,
            feature_calculators=calculator,
            split=split,
            feature_filters=filters)
```

PREPARING THE DATASET



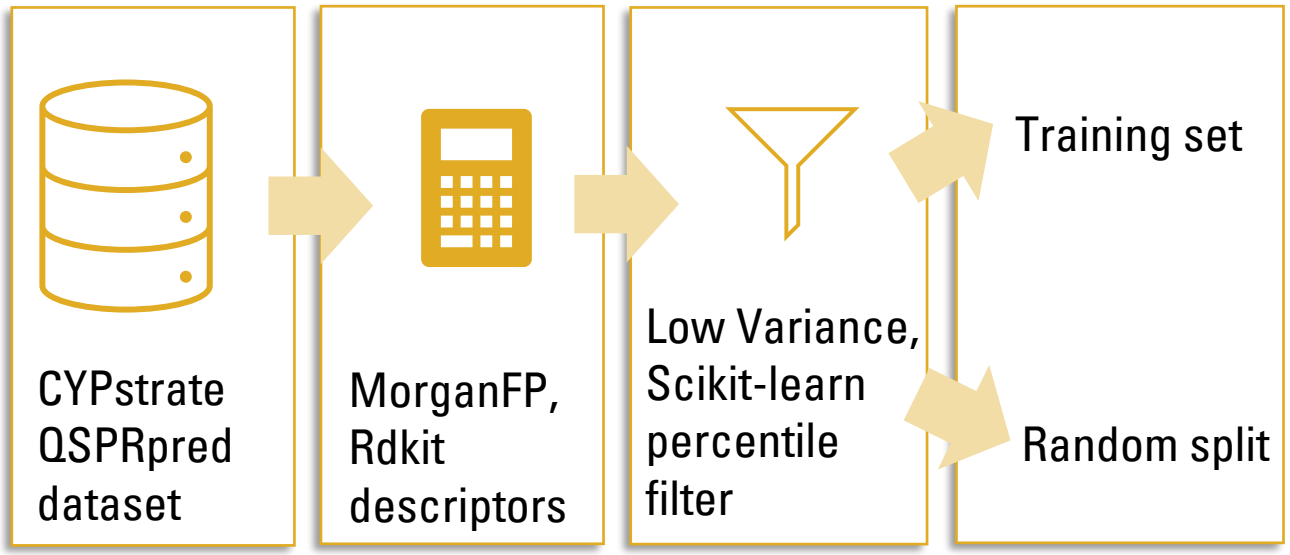

```
calculator = MoleculeDescriptorsCalculator([FingerprintSet(fingerprint_type="RDKitMACCSFP")])
filters = [lowVarianceFilter(th=0), SklearnSelectPercentile(percentile=percentile)]
split = ManualSplit(splitcol=dataset.getDF()["datasplit"], trainval="train", testval="test")

dataset.prepareDataset(smiles_standardizer = None,
            feature_calculators=calculator,
            split=split,
            feature_filters=filters)
```

PREPARING THE DATASET



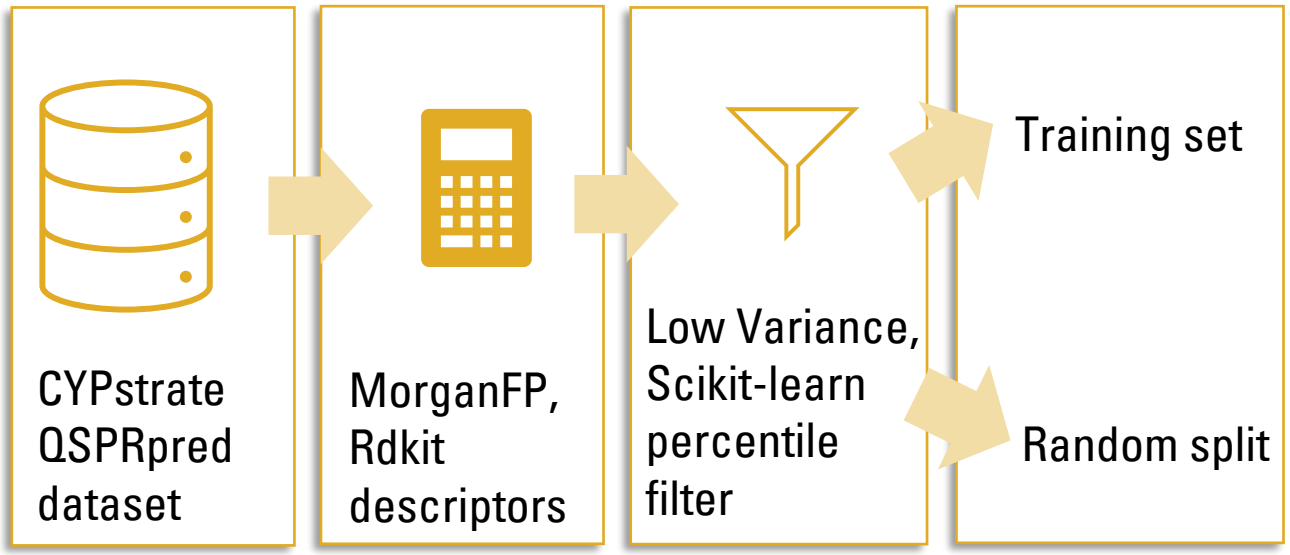

```
calculator = MoleculeDescriptorsCalculator([FingerprintSet(fingerprint_type="RDKitMACCSFP")])
filters = [lowVarianceFilter(th=0), SklearnSelectPercentile(percentile=percentile)]
split = ManualSplit(splitcol=dataset.getDF()["datasplit"], trainval="train", testval="test")

dataset.prepareDataset(smiles_standardizer = None,
            feature_calculators=calculator,
            split=split,
            feature_filters=filters)
```

PREPARING THE DATASET



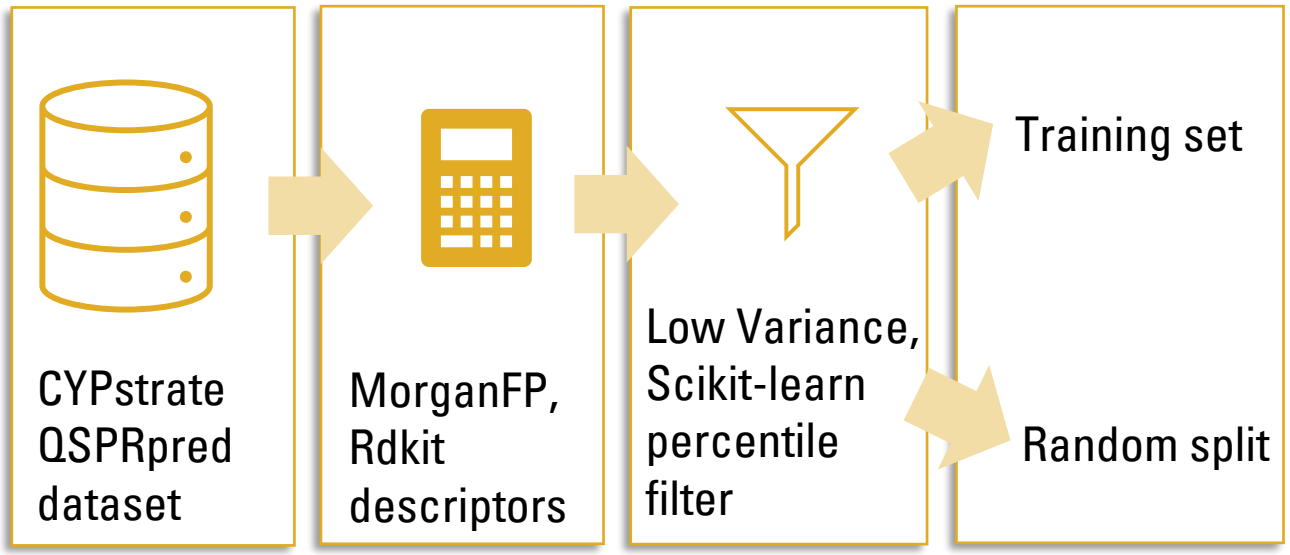

```
calculator = MoleculeDescriptorsCalculator([FingerprintSet(fingerprint_type="RDKitMACCSFP")])
filters = [lowVarianceFilter(th=0), SklearnSelectPercentile(percentile=percentile)]
split = ManualSplit(splitcol=dataset.getDF()["datasplit"], trainval="train", testval="test")

dataset.prepareDataset(smiles_standardizer = None,
            feature_calculators=calculator,
            split=split,
            feature_filters=filters)
```

PREPARING THE DATASET



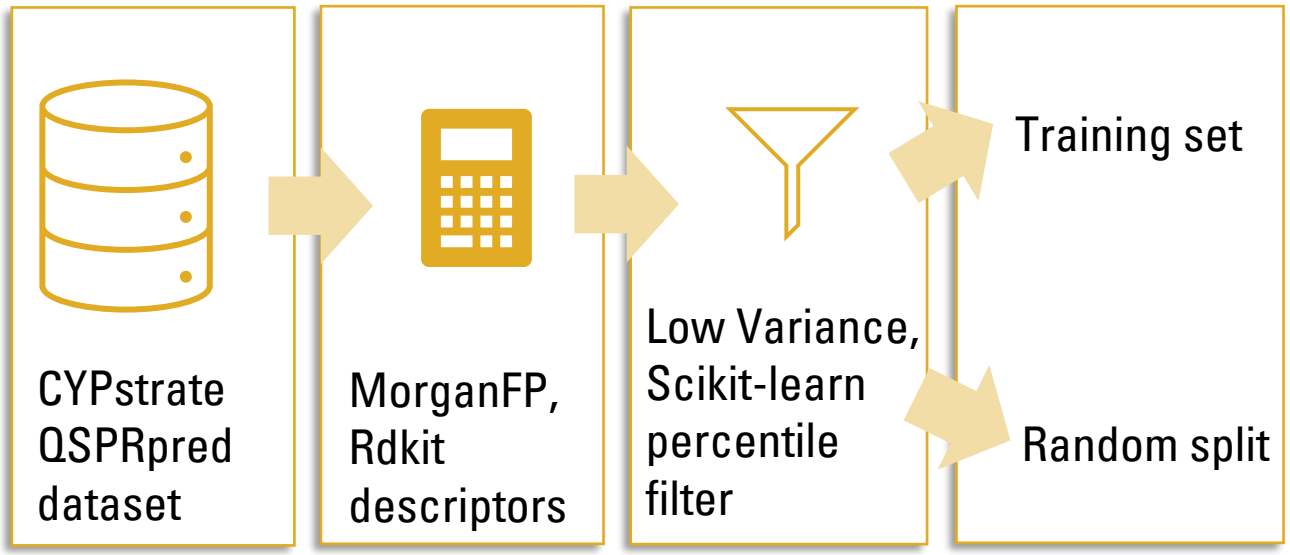

```
calculator = MoleculeDescriptorsCalculator([FingerprintSet(fingerprint_type="RDKitMACCSFP")])
filters = [lowVarianceFilter(th=0), SklearnSelectPercentile(percentile=percentile)]
split = ManualSplit(splitcol=dataset.getDF()["datasplit"], trainval="train", testval="test")

dataset.prepareDataset(smiles_standardizer = None,
            feature_calculators=calculator,
            split=split,
            feature_filters=filters)
```

PREPARING THE DATASET




```
calculator = MoleculeDescriptorsCalculator([FingerprintSet(fingerprint_type="RDKitMACCSFP")])
filters = [lowVarianceFilter(th=0), SklearnSelectPercentile(percentile=percentile)]
split = ManualSplit(splitcol=dataset.getDF()["datasplit"], trainval="train", testval="test")

dataset.prepareDataset(smiles_standardizer = None,
            feature_calculators=calculator,
            split=split,
            feature_filters=filters)
```

PREPARING THE DATASET




```
calculator = MoleculeDescriptorsCalculator([FingerprintSet(fingerprint_type="RDKitMACCSFP")])
filters = [lowVarianceFilter(th=0), SklearnSelectPercentile(percentile=percentile)]
split = ManualSplit(splitcol=dataset.getDF()["datasplit"], trainval="train", testval="test")

dataset.prepareDataset(smiles_standardizer = None,
            feature_calculators=calculator,
            split=split,
            feature_filters=filters
```

ADDING A NEW FEATURE FILTER

```python
from qsprpred.data.interfaces import featurefilter
from sklearn.feature_selection import SelectPercentile, f_classif

class SklearnSelectPercentile(featurefilter):
    """Select features according to a percentile of the highest scores using sklearn"""

    def __init__(self, score_func: callable=f_classif, percentile: int = 70) -> None:
        self.selector = SelectPercentile(score_func=score_func,percentile=percentile)
        self.selector.set_output(transform="pandas")

    def __call__(self, df: pd.DataFrame, y_col : pd.DataFrame = None) -> pd.DataFrame:

        df = self.selector.fit_transform(X=df, y=y_col.values.ravel())

        return df
```

ADDING A NEW FEATURE FILTER

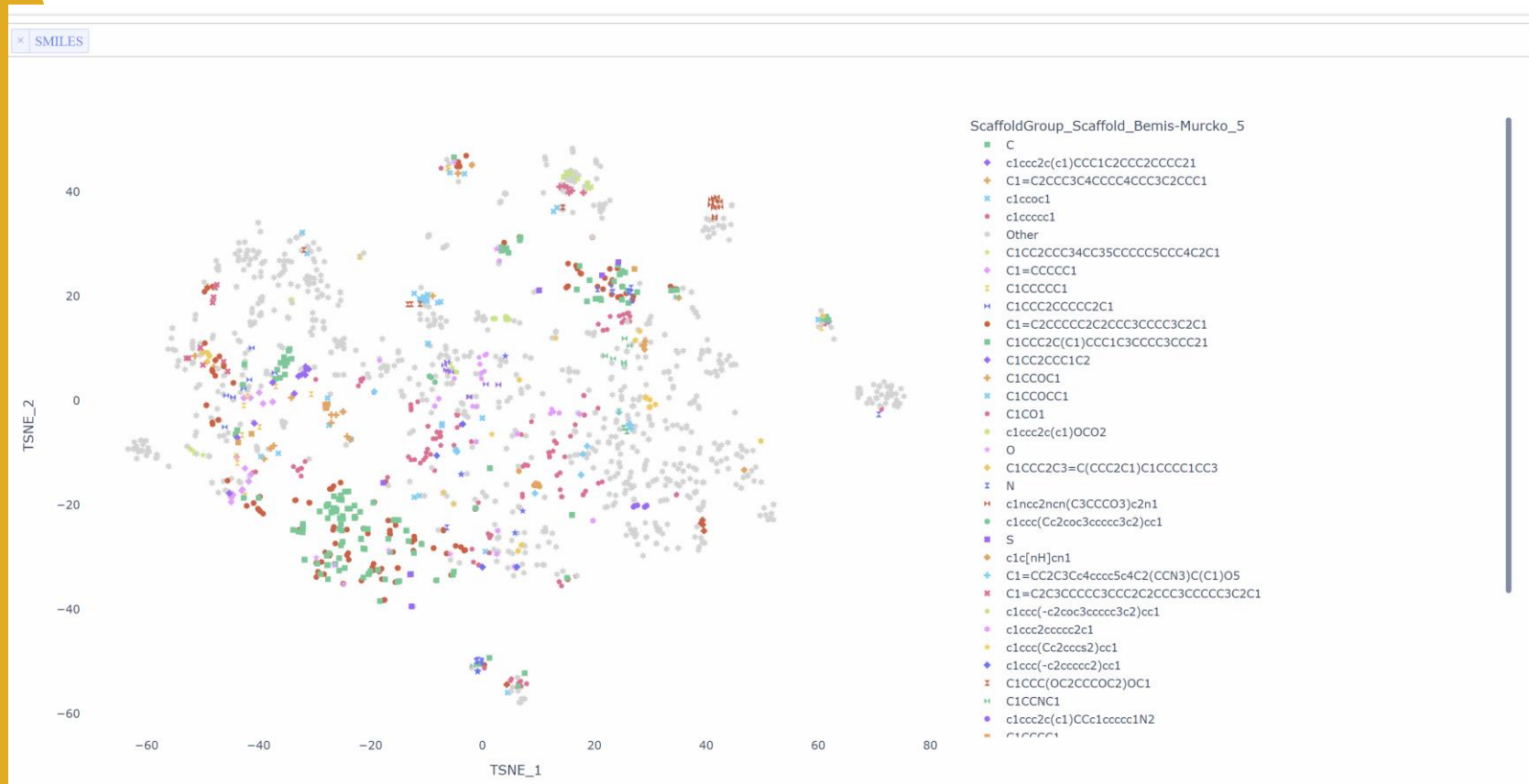
```
from qsprpred.data.interfaces import featurefilter
from sklearn.feature_selection import SelectPercentile, f_classif

class SklearnSelectPercentile(featurefilter):
    """Select features according to a percentile of the highest scores using sklearn"""

    def __init__(self, score_func: callable=f_classif, percentile: int = 70) -> None:
        self.selector = SelectPercentile(score_func=score_func,percentile=percentile)
        self.selector.set_output(transform="pandas")

    def __call__(self, df: pd.DataFrame, y_col : pd.DataFrame = None) -> pd.DataFrame:

        df = self.selector.fit_transform(X=df, y=y_col.values.ravel())

        return df
```

ADDING A NEW FEATURE FILTER

```python
from qsprpred.data.interfaces import featurefilter
from sklearn.feature_selection import SelectPercentile, f_classif

class SklearnSelectPercentile(featurefilter):
    """Select features according to a percentile of the highest scores using sklearn"""

    def __init__(self, score_func: callable=f_classif, percentile: int = 70) -> None:
        self.selector = SelectPercentile(score_func=score_func,percentile=percentile)
        self.selector.set_output(transform="pandas")

    def __call__(self, df: pd.DataFrame, y_col : pd.DataFrame = None) -> pd.DataFrame:

        df = self.selector.fit_transform(X=df, y=y_col.values.ravel())

        return df
```

DATASET VISUALIZATION: SCAFFVIZ

- Dataset visualization: papyrus-scaffold-visualizer
- Build on top off Molplotly library*
- Developed by Martin Šícho



*https://github.com/wjm41/molplotly

TRAINING A SINGLE TASK MODEL FOR CYP2C19

```
from sklearn.ensemble import RandomForestClassifier

model = QSPRsklearn(base_dir = DATA_PATH, data=dataset,
                    alg = RandomForestClassifier,
                    name='RF_CLS',
                    parameters={'n_estimators': 500, 'n_jobs': 8, 'class_weight': 'balanced'})

search_space_gs = {"min_samples_split": [2, 4, 8, 128, 16, 32, 64],
                   "max_features": [0.05, 0.1, 0.2, 0.4, 0.8, 'sqrt']}
optimizer = GridSearchOptimization(scoring='roc_auc_ovr', param_grid=search_space_gs)
best_params = optimizer.optimize(model)

model.evaluate()

model.save()
```

TRAINING A SINGLE TASK MODEL FOR CYP2C19

```
from sklearn.ensemble import RandomForestClassifier

model = QSPRsklearn(base_dir = DATA_PATH, data=dataset,
                    alg = RandomForestClassifier,
                    name='RF_CLS',
                    parameters={'n_estimators': 500, 'n_jobs': 8, 'class_weight': 'balanced'})
```

```
search_space_gs = {"min_samples_split": [2, 4, 8, 128, 16, 32, 64],
                   "max_features": [0.05, 0.1, 0.2, 0.4, 0.8, 'sqrt']}
optimizer = GridSearchOptimization(scoring='roc_auc_ovr', param_grid=search_space_gs)
best_params = optimizer.optimize(model)

model.evaluate()

model.save()
```

TRAINING A SINGLE TASK MODEL FOR CYP2C19

```
from sklearn.ensemble import RandomForestClassifier

model = QSPRsklearn(base_dir = DATA_PATH, data=dataset,
                    alg = RandomForestClassifier,
                    name='RF_CLS',
                    parameters={'n_estimators': 500, 'n_jobs': 8, 'class_weight': 'balanced'})

search_space_gs = {"min_samples_split": [2, 4, 8, 128, 16, 32, 64],
                   "max_features": [0.05, 0.1, 0.2, 0.4, 0.8, 'sqrt']}
optimizer = GridSearchOptimization(scoring='roc_auc_ovr', param_grid=search_space_gs)
best_params = optimizer.optimize(model)

model.evaluate()

model.save()
```

TRAINING A SINGLE TASK MODEL FOR CYP2C19

```
from sklearn.ensemble import RandomForestClassifier

model = QSPRsklearn(base_dir = DATA_PATH, data=dataset,
                    alg = RandomForestClassifier,
                    name='RF_CLS',
                    parameters={'n_estimators': 500, 'n_jobs': 8, 'class_weight': 'balanced'})

search_space_gs = {"min_samples_split": [2, 4, 8, 128, 16, 32, 64],
                   "max_features": [0.05, 0.1, 0.2, 0.4, 0.8, 'sqrt']}
optimizer = GridSearchOptimization(scoring='roc_auc_ovr', param_grid=search_space_gs)
best_params = optimizer.optimize(model)

model.evaluate()

model.save()
```

TRAINING A SINGLE TASK MODEL FOR CYP2C19

```
from sklearn.ensemble import RandomForestClassifier

model = QSPRsklearn(base_dir = DATA_PATH, data=dataset,
                    alg = RandomForestClassifier,
                    name='RF_CLS',
                    parameters={'n_estimators': 500, 'n_jobs': 8, 'class_weight': 'balanced'})

search_space_gs = {"min_samples_split": [2, 4, 8, 128, 16, 32, 64],
                   "max_features": [0.05, 0.1, 0.2, 0.4, 0.8, 'sqrt']}
optimizer = GridSearchOptimization(scoring='roc_auc_ovr', param_grid=search_space_gs)
best_params = optimizer.optimize(model)

model.evaluate()

model.save()
```

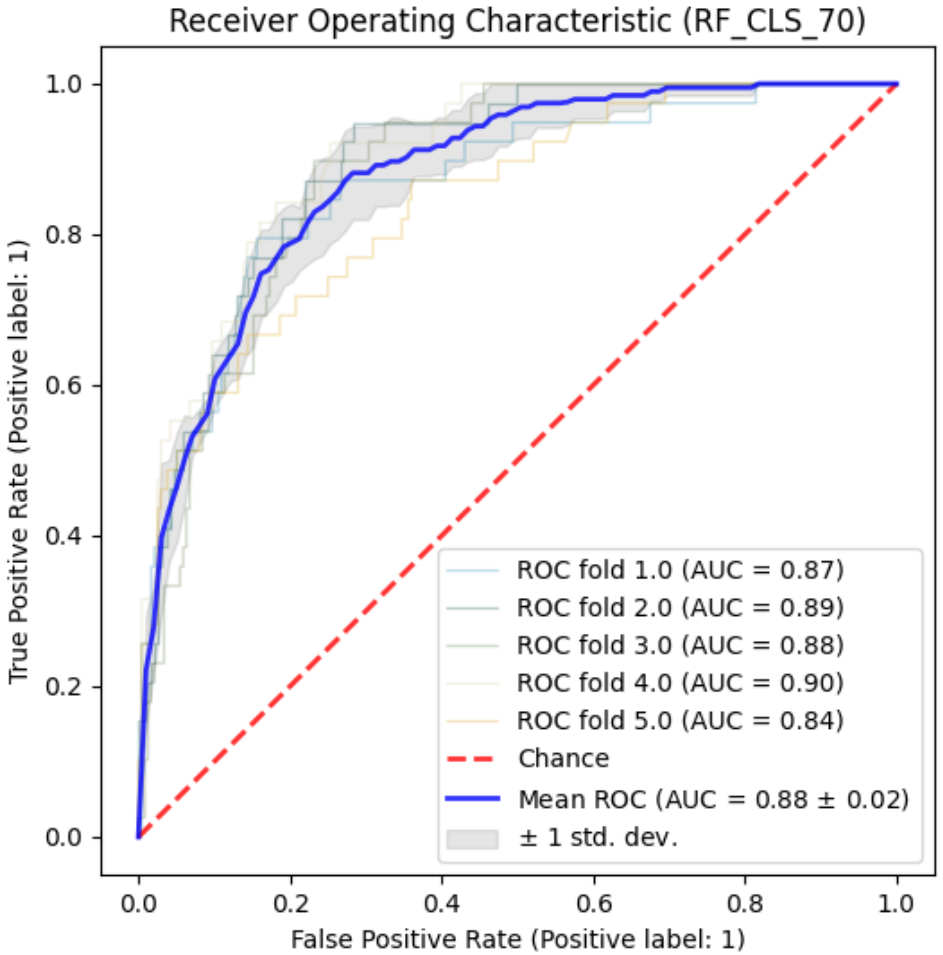
TRAINING A SINGLE TASK MODEL FOR CYP2C19

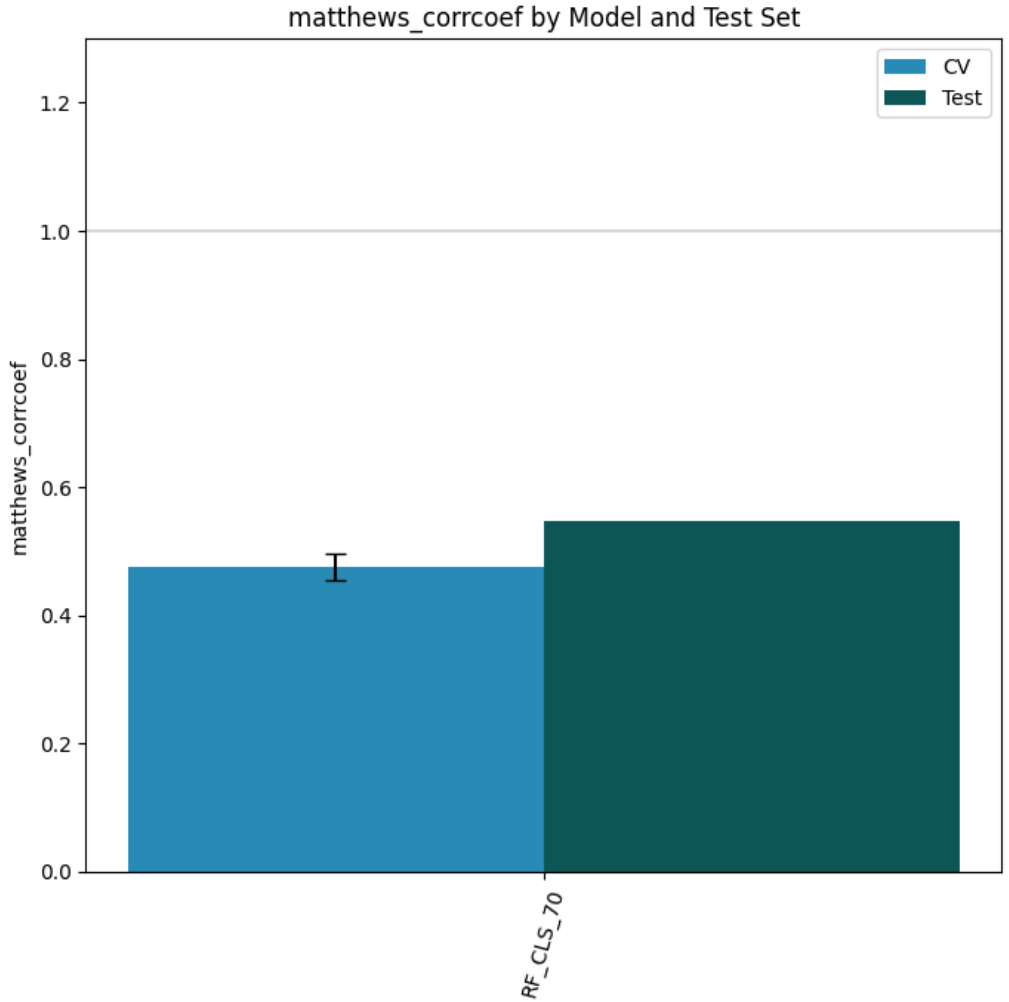
```
from sklearn.ensemble import RandomF

model = QSPRsklearn(base_dir = DA
                    alg = RandomF
                    name='RF_CLS'
                    parameters={'
                                                                    ed'})

search_space_gs = {"min_samples_s
                   "max_features"
optimizer = GridSearchOptimization
best_params = optimizer.optimize(model)

model.evaluate()

model.save()
```

ml2json

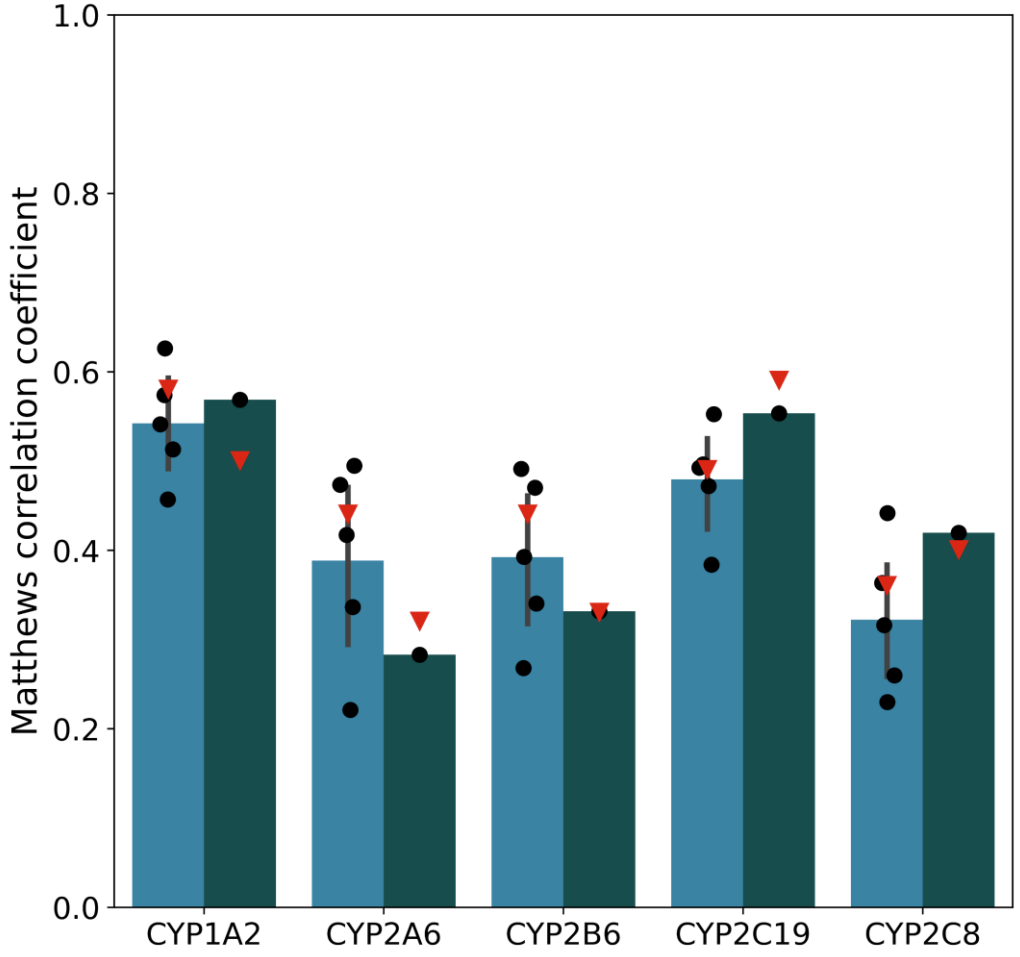
- Export scikit-learn model files to JSON
- Safe & transparent
- Developed by Olivier Béquignon

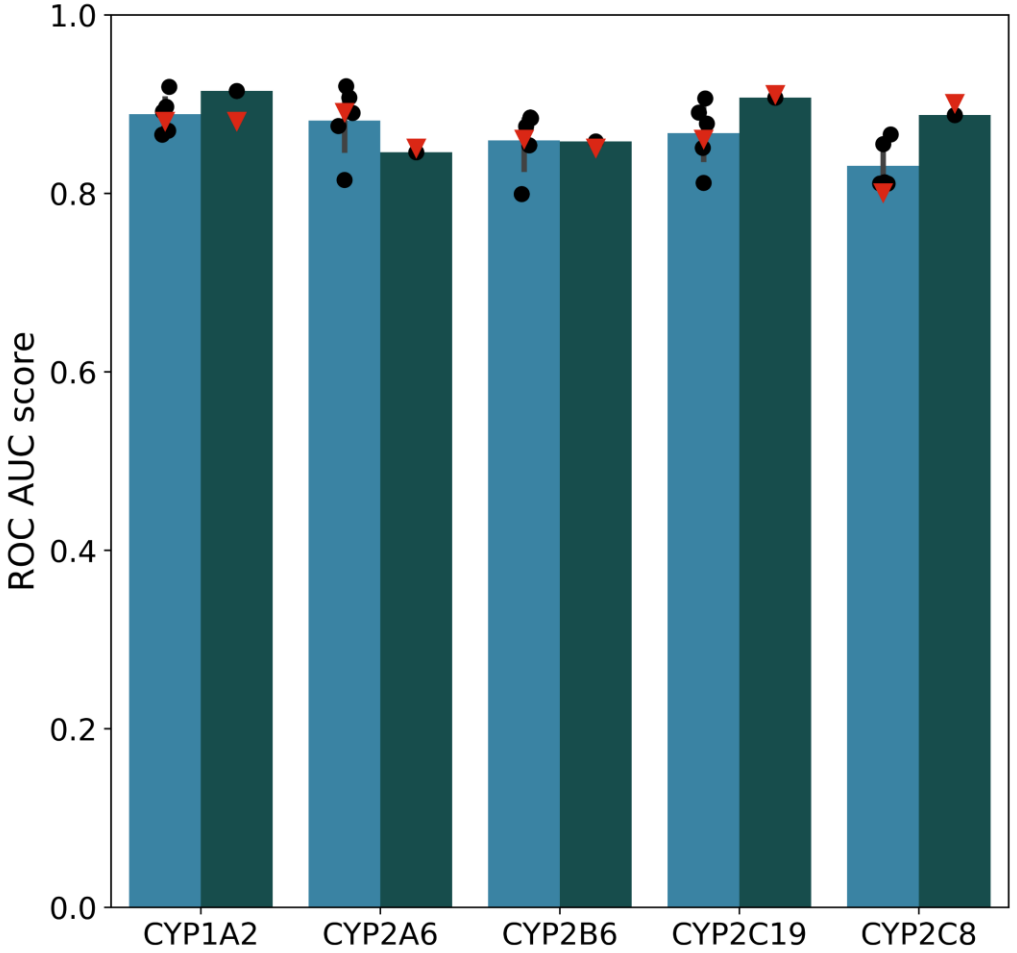
SINGLE TASK MODELS: CYP2C19





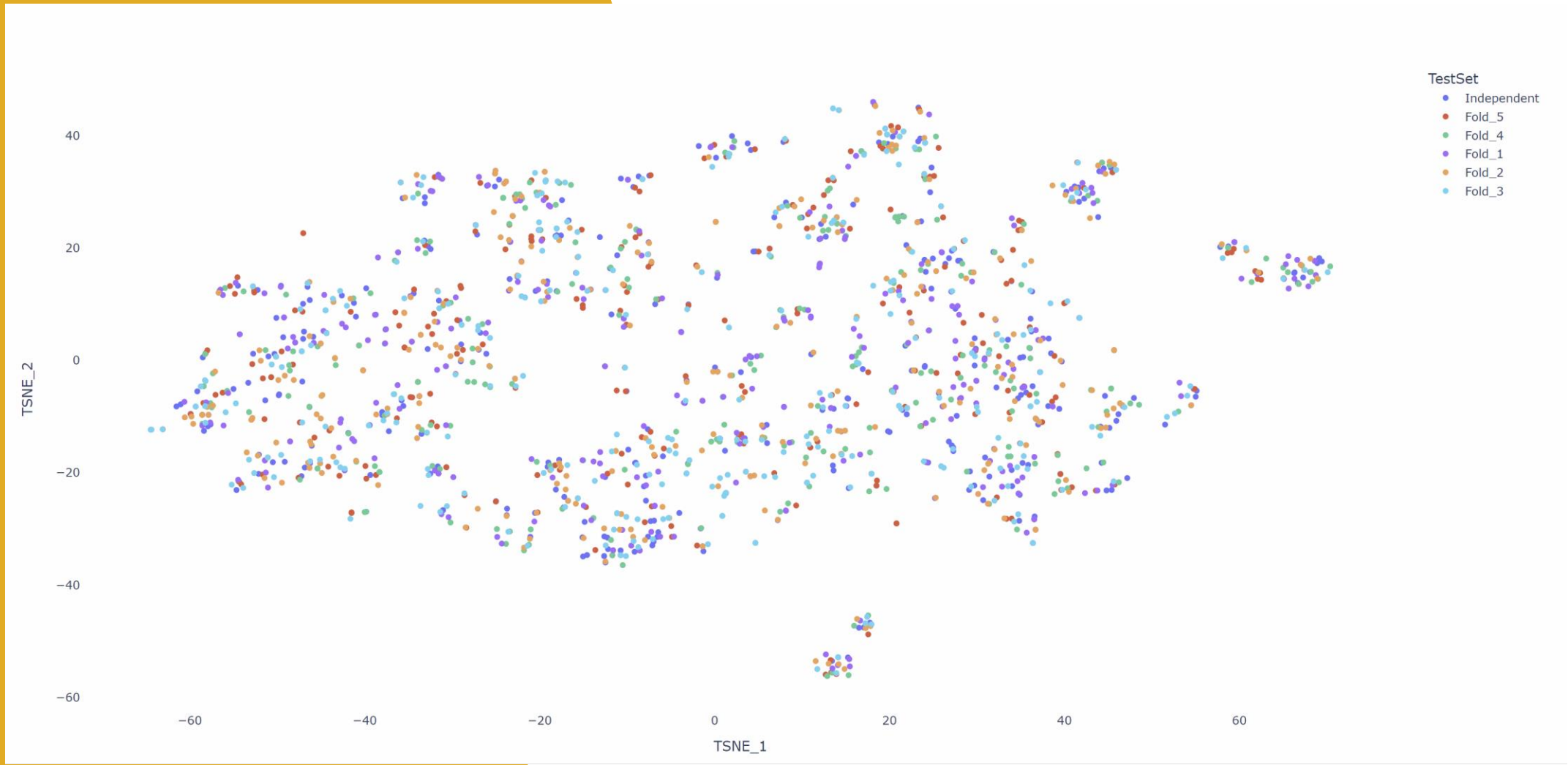
PERFORMANCE SIMILAR TO CYPSTRATE PAPER



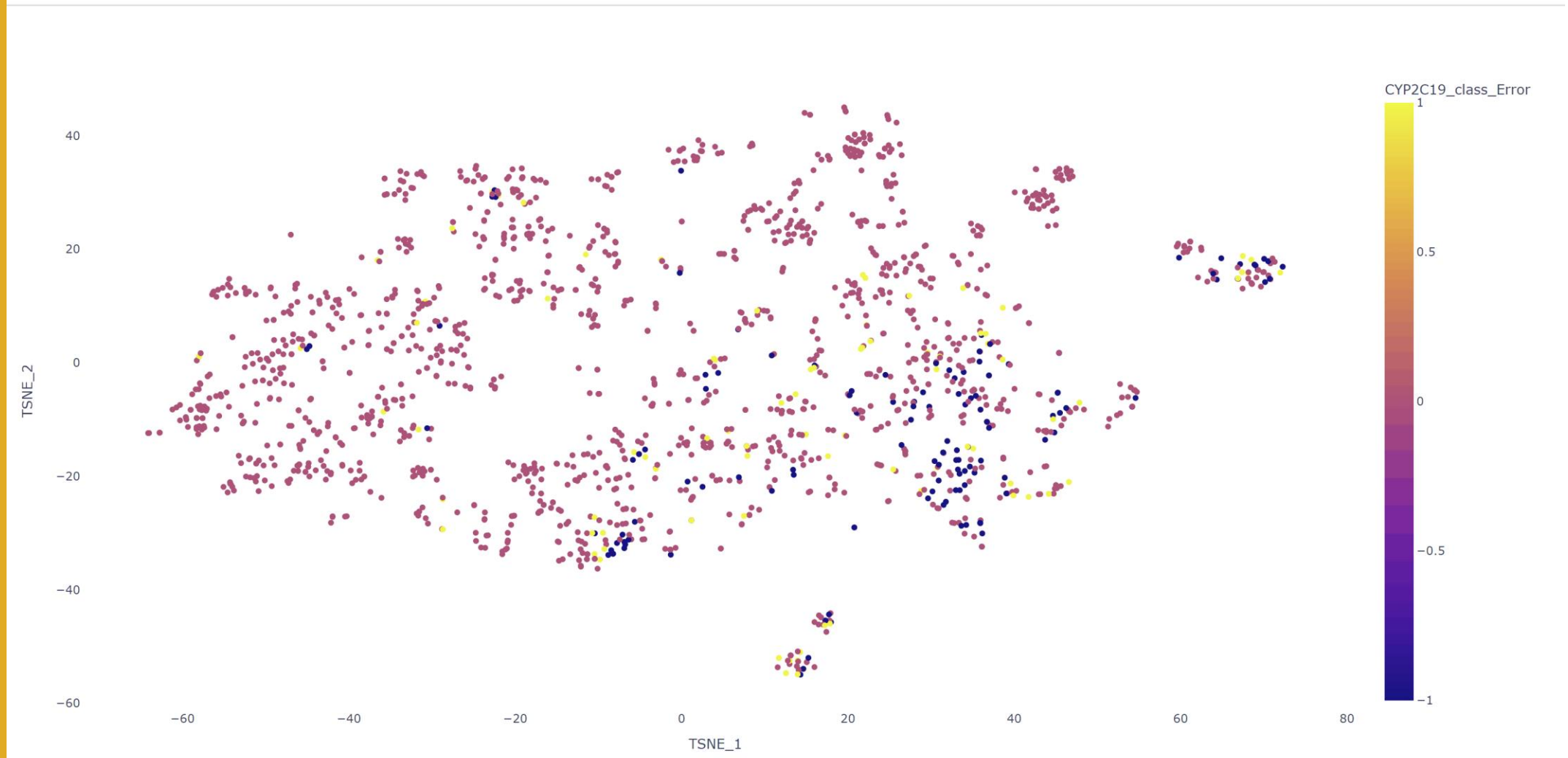


QSPRpred cross-validation
QSPRpred test set
QSPRpred performance
Median CYPSTRATE performance

DATASET VISUALIZATION: MODELLING RESULTS



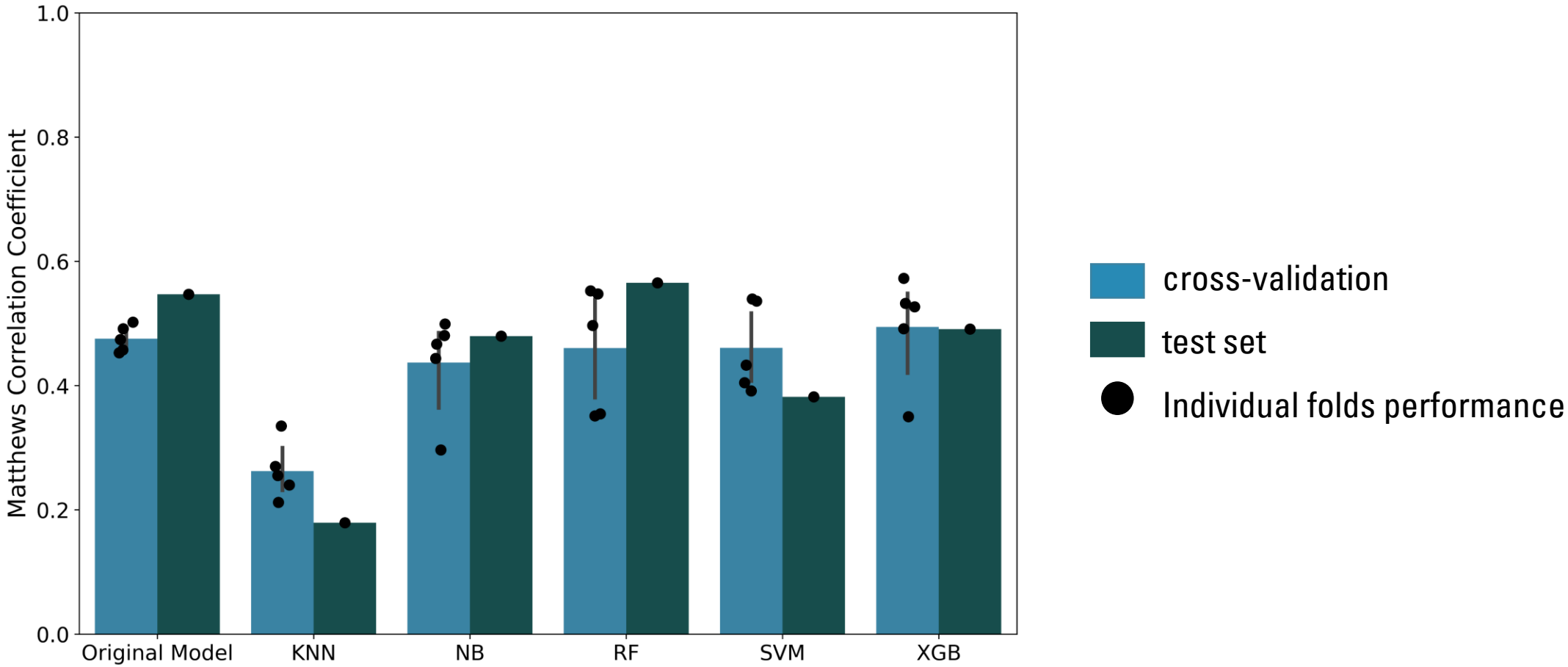
DATASET VISUALIZATION: MODELLING RESULTS



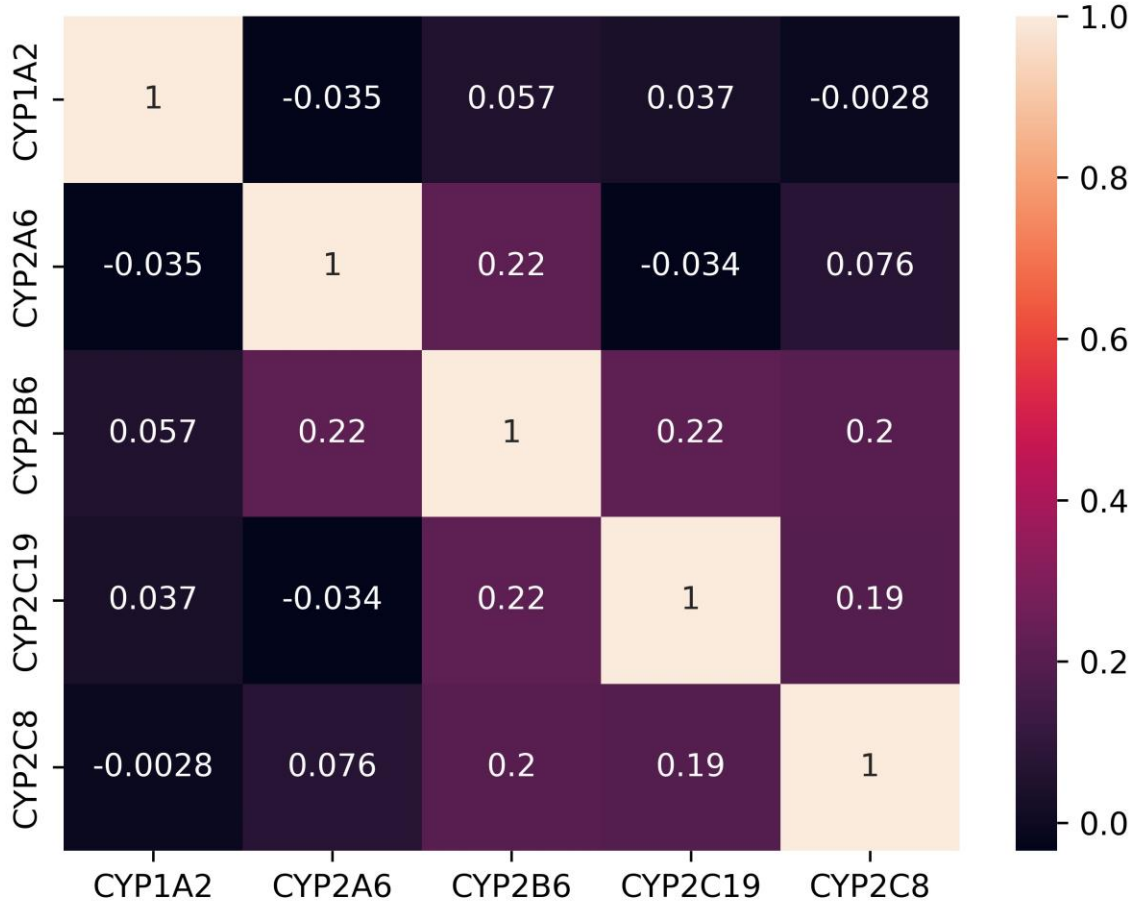
TRY OUT MANY DIFFERENT MODELS QUICKLY USING THE COMMAND LINE INTERFACE

```
python -m qsprpred.data_CLI -i cypstrate.tsv -pr CYP3A4 -r False -sp manual -fe RDkit Morgan Mordred
-lv 0.01 -hc 0.9 -bf -fv 0

python -m qsprpred.model_CLI -dp CYP3A4_SINGLECLASS -mt RF XGB SVM PLS NB KNN -sw -s -o bayes -nt 30
-me
```

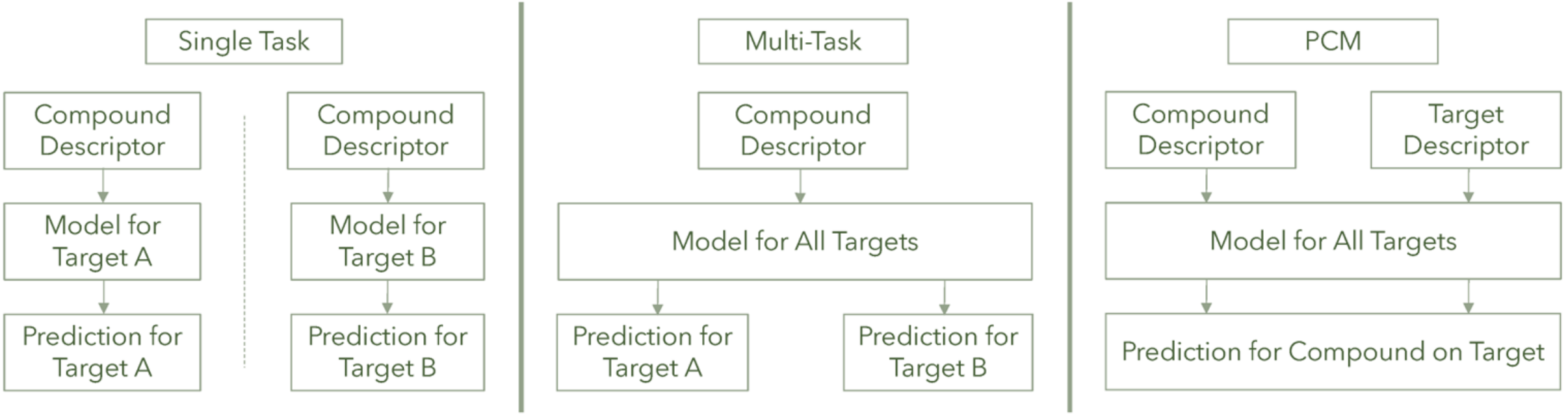
TRY OUT MANY DIFFERENT MODELS QUICKLY USING THE COMMAND LINE INTERFACE



CYP ENZYME ACTIVITIES SHOW SOME CORRELATION



MULTI-TASK AND PROTEOCHEMOMETRICS



CREATING THE DATASET

SMILES	cyp_isoform	cyp_substrate	accession
BrCCBr	CYP1A2	0.0	P05177
C#CC#CC#CCCO	CYP1A2	0.0	P05177
c1ccccc1	CYP3A4	0.0	P08684
c1ccoc1	CYP3A4	0.0	P08684
...	...	...	...

```
dataset = PCMDataset(df=df,
                     store_dir=f.",
                     name=f"pcm",
                     target_props=[{"name": "cyp_substrate", "task":
                                    TargetTasks.SINGLECLASS, "th":"precomputed"}],
                     n_jobs=8,
                     proteincol="accession",
                     proteinseqprovider=sequence_provider)
```

CREATING THE DATASET

SMILES	cyp_isoform	cyp_substrate	accession
BrCCBr	CYP1A2	0.0	P05177
C#CC#CC#CCCO	CYP1A2	0.0	P05177
c1ccccc1	CYP3A4	0.0	P08684
c1ccoc1	CYP3A4	0.0	P08684
…	…	…	…

```
dataset = PCMDataset(df=df,
                     store_dir=f.",
                     name=f"pcm",
                     target_props=[{"name": "cyp_substrate", "task":
                                    TargetTasks.SINGLECLASS, "th":"precomputed"}],
                     n_jobs=8,
                     proteincol="accession",
                     proteinseqprovider=sequence_provider)
```

PREPARING THE DATASET

```
calc_prot = ProteinDescriptorCalculator(
    descsets=[ProDecDescriptorSet(sets=["Zscale Hellberg"])],
    msa_provider=ClustalMSA(out_dir=dataset.storeDir)
)
calc_mol = MoleculeDescriptorsCalculator([FingerprintSet(fingerprint_type="MorganFP",
                                                         radius=2, nBits=2048)])

dataset.prepareDataset(
    smiles_standardizer = None,
    feature_calculators=[calc_prot, calc_mol],
    split=StratifiedPerTarget(splitter=randomsplit(0.2), dataset=dataset),
    feature_filters=[lowVarianceFilter(th=0), SklearnSelectPercentile(percentile=10)],
    feature_standardizer = StandardScaler(),
    )

dataset.save()
```

PREPARING THE DATASET

```
calc_prot = ProteinDescriptorCalculator(
    descsets=[ProDecDescriptorSet(sets=["Zscale Hellberg"])],
    msa_provider=ClustalMSA(out_dir=dataset.storeDir)
)
calc_mol = MoleculeDescriptorsCalculator([FingerprintSet(fingerprint_type="MorganFP",
                                                         radius=2, nBits=2048)])

dataset.prepareDataset(
    smiles_standardizer = None,
    feature_calculators=[calc_prot, calc_mol],
    split=StratifiedPerTarget(splitter=randomsplit(0.2), dataset=dataset),
    feature_filters=[lowVarianceFilter(th=0), SklearnSelectPercentile(percentile=10)],
    feature_standardizer = StandardScaler(),
    )

dataset.save()
```

PREPARING THE DATASET

```
calc_prot = ProteinDescriptorCalculator(
    descsets=[ProDecDescriptorSet(sets=["Zscale Hellberg"])],
    msa_provider=ClustalMSA(out_dir=dataset.storeDir)
)
calc_mol = MoleculeDescriptorsCalculator([FingerprintSet(fingerprint_type="MorganFP",


dataset.prepareDataset(
    smiles_standardizer = M
    feature_calculators=[ca
    split=StratifiedPerTarg
    feature_filters=[lowVar
    feature_standardizer =
    )

dataset.save()
```

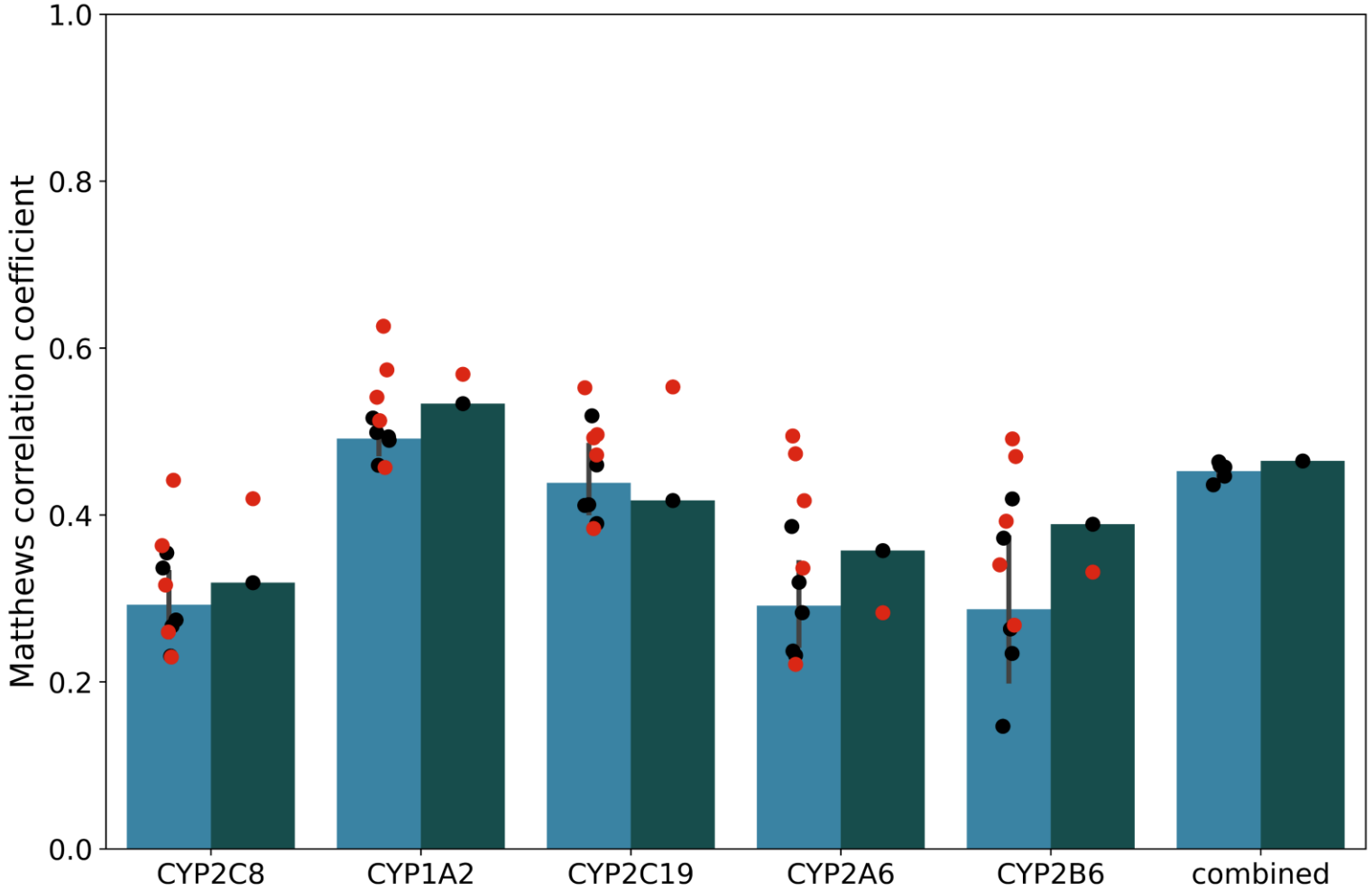
ProDEC

- easily calculate descriptors of protein sequences and their common transforms.
- Developed by Olivier Béquignon

PREPARING THE DATASET

```
calc_prot = ProteinDescriptorCalculator(
    descsets=[ProDecDescriptorSet(sets=["Zscale Hellberg"])],
    msa_provider=ClustalMSA(out_dir=dataset.storeDir)
)
calc_mol = MoleculeDescriptorsCalculator([FingerprintSet(fingerprint_type="MorganFP",
                                                         radius=2, nBits=2048)])

dataset.prepareDataset(
    smiles_standardizer = None,
    feature_calculators=[calc_prot, calc_mol],
    split=StratifiedPerTarget(splitter=randomsplit(0.2), dataset=dataset),
    feature_filters=[lowVarianceFilter(th=0), SklearnSelectPercentile(percentile=10)],
    feature_standardizer = StandardScaler(),
    feature_fill_value=0
    )

dataset.save()
```

PCM MODEL RESULTS



cross-validation PCM models

test set PCM model

Folds performance PCM

Folds performance Single Task

TAKE HOME MESSAGES

QSPRpred is a versatile tool for Quantitative Structure-Property Relationship modelling

Contains extensive data-preprocessing functionality.
Suitable for building single-task, multi-task and proteochemometric models.

QSPRpred is simple but flexible

Has a modular structure that allows for easily adding new functionalities.
Comprehensive tutorials available

QSPRpred is open-source

The code can be found on the Leiden Computational Drug Discovery group Github

ACKNOWLEDGEMENTS

- J.G. Coen van Hasselt, Piet H. van der Graaf, and Gerard J. P. van Westen
- The Quantitative Pharmacology group
- The Computational Drug Discovery group
- The QSPRpred dev team: Linde Schoenmaker, Martin Sicho, Olivier J. M. Béquignon, Sohvi Luukkonen, David Araripe, Marina Gorostiola González, Remco van den Broek, Andrius Bernatavicius

QSPRPRED DEV TEAM







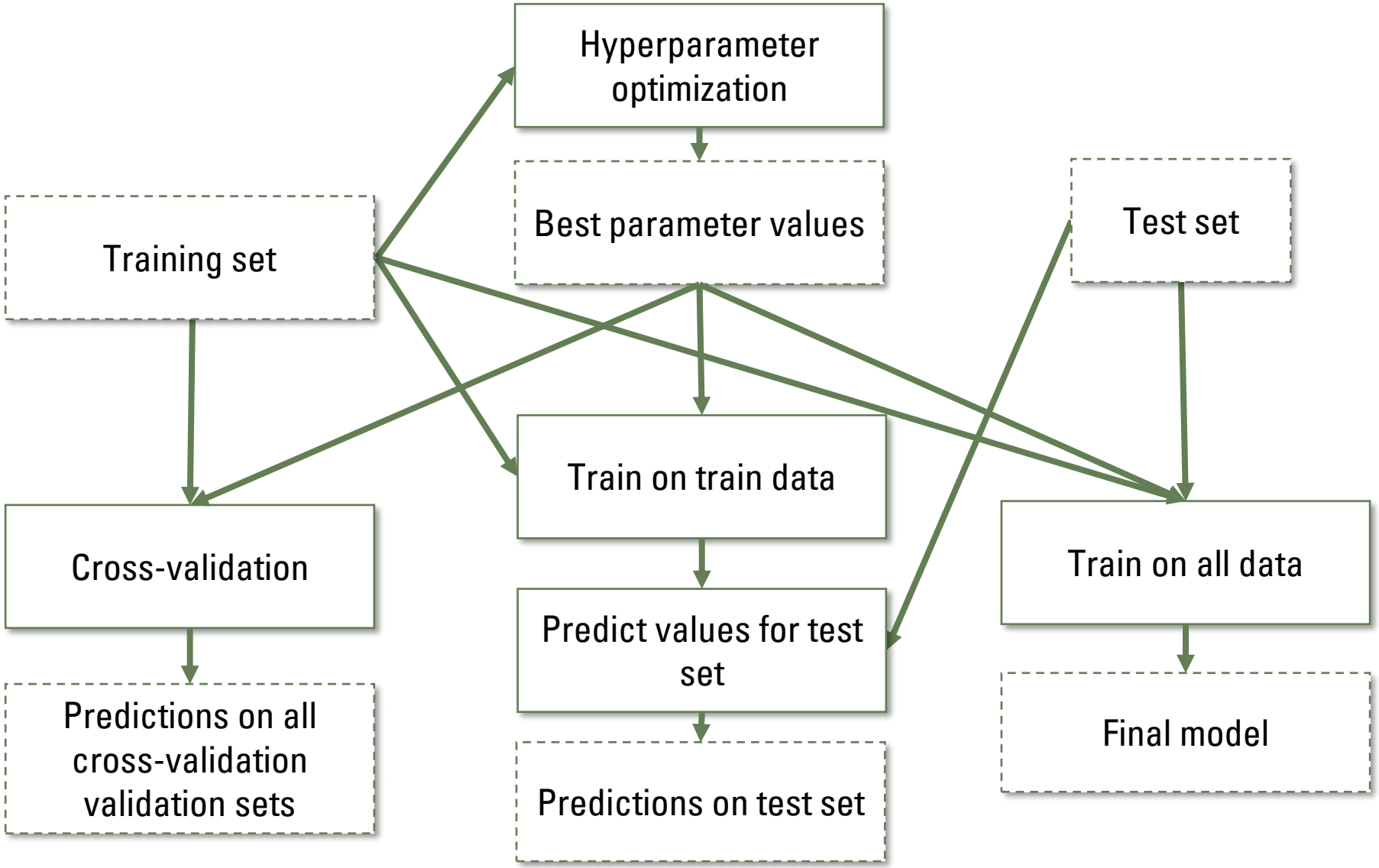


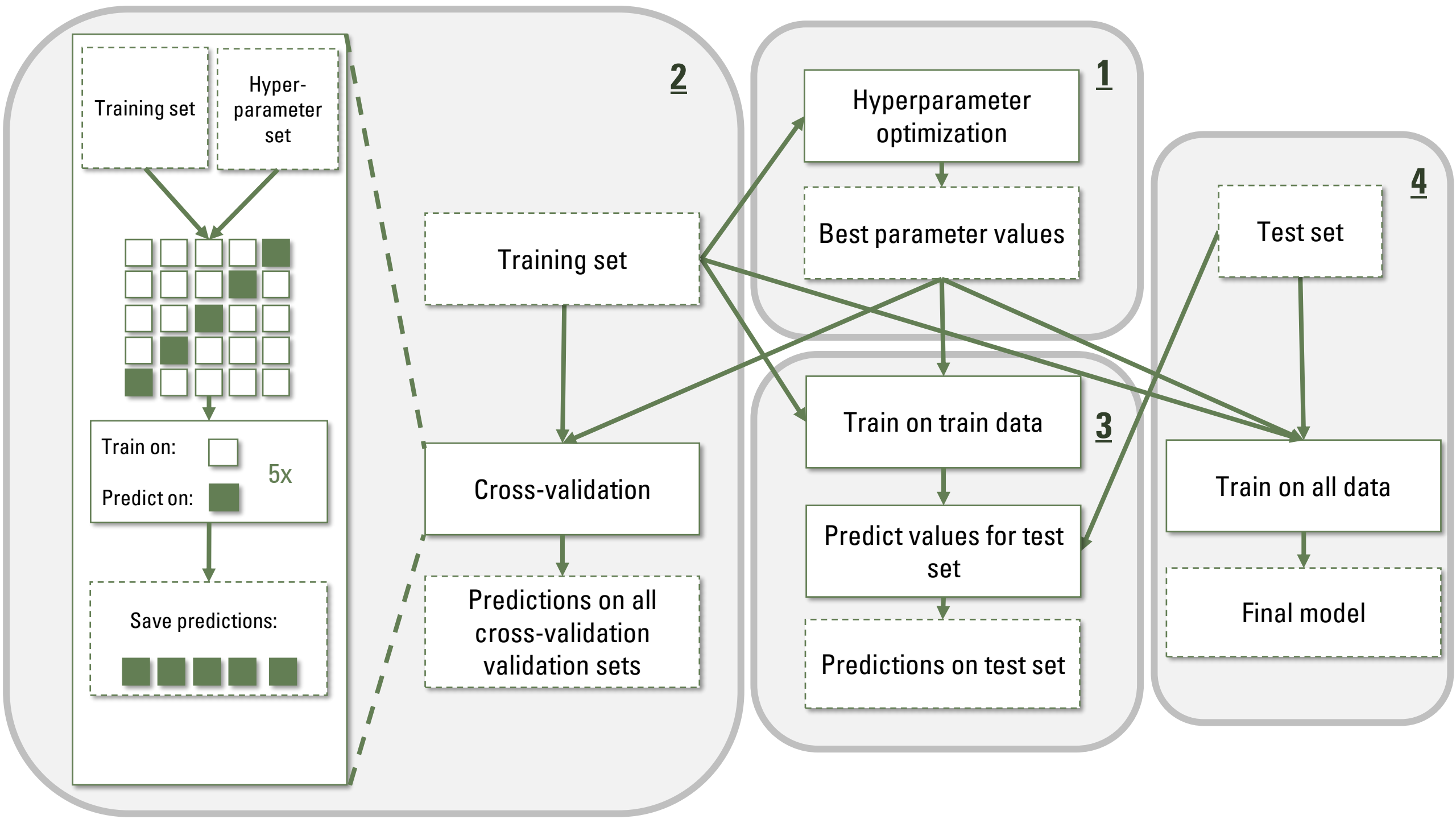

QUESTIONS?

QSPRpred

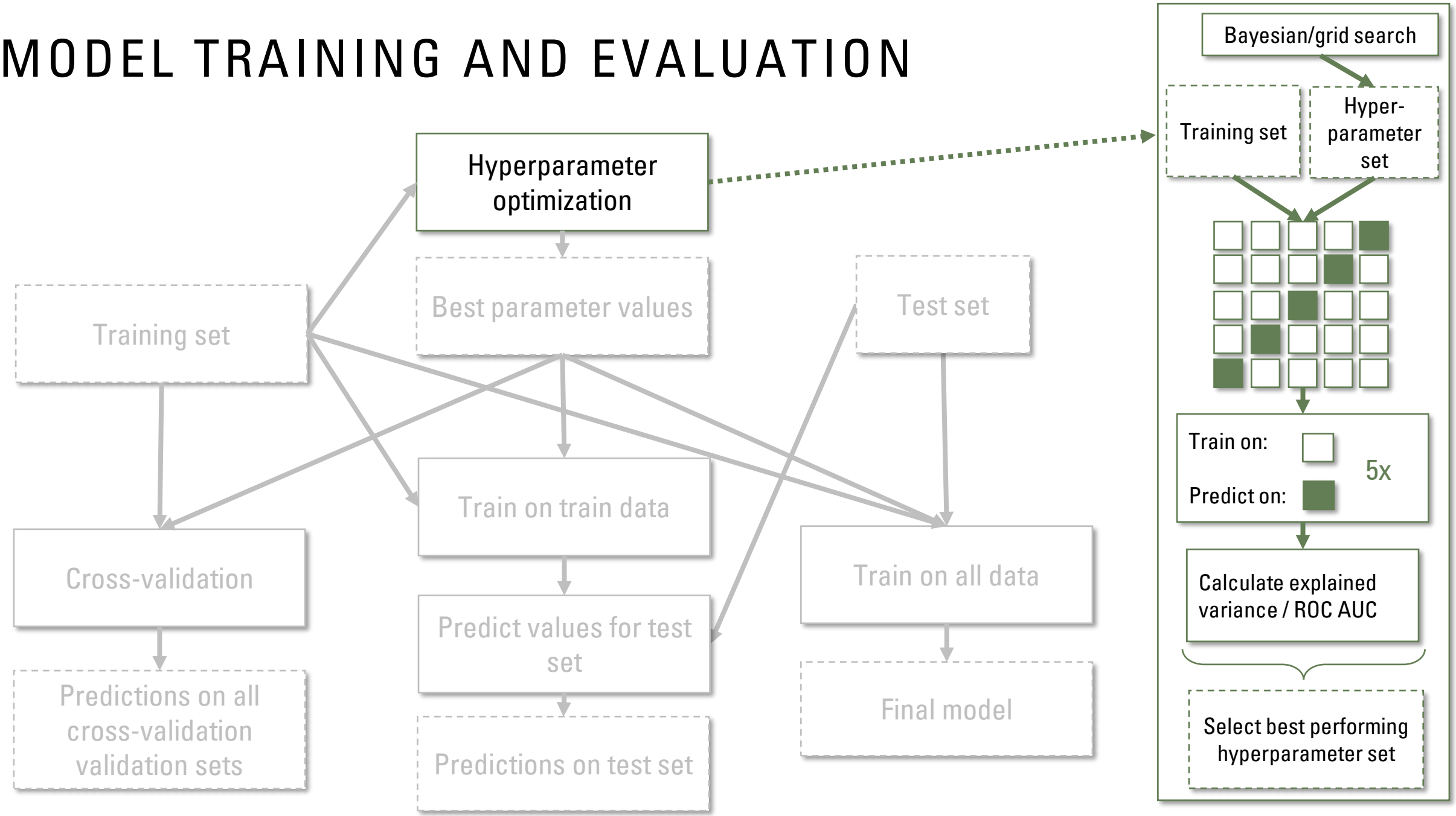
BACK-UP SLIDES

MODEL TRAINING AND EVALUATION

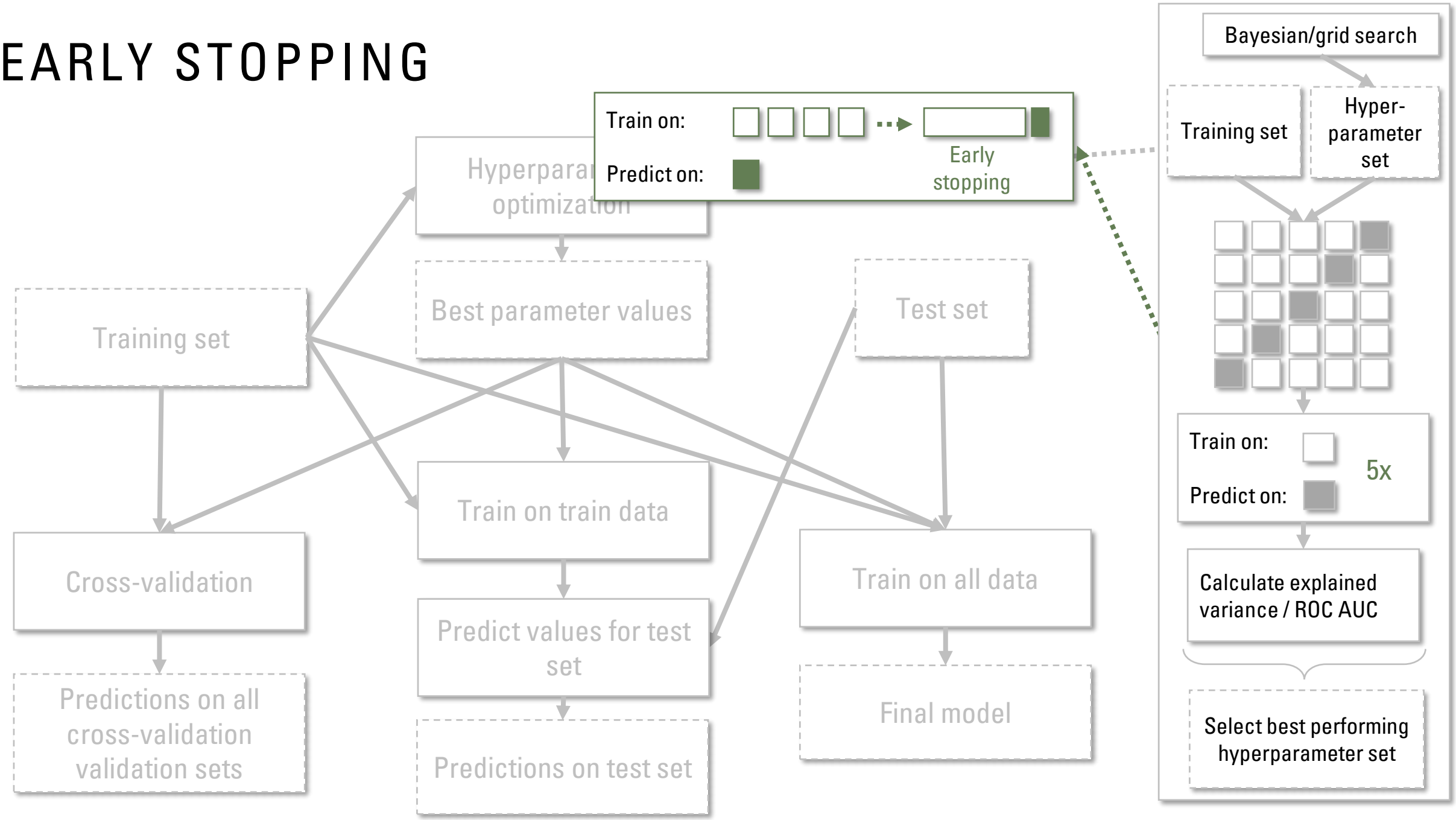



Training set
Hyper-parameter set
Train on:
Predict on:
5x
Save predictions:
2
Training set
Cross-validation
Predictions on all cross-validation validation sets
1
Hyperparameter optimization
Best parameter values
3
Train on train data
Predict values for test set
Predictions on test set
4
Test set
Train on all data
Final model

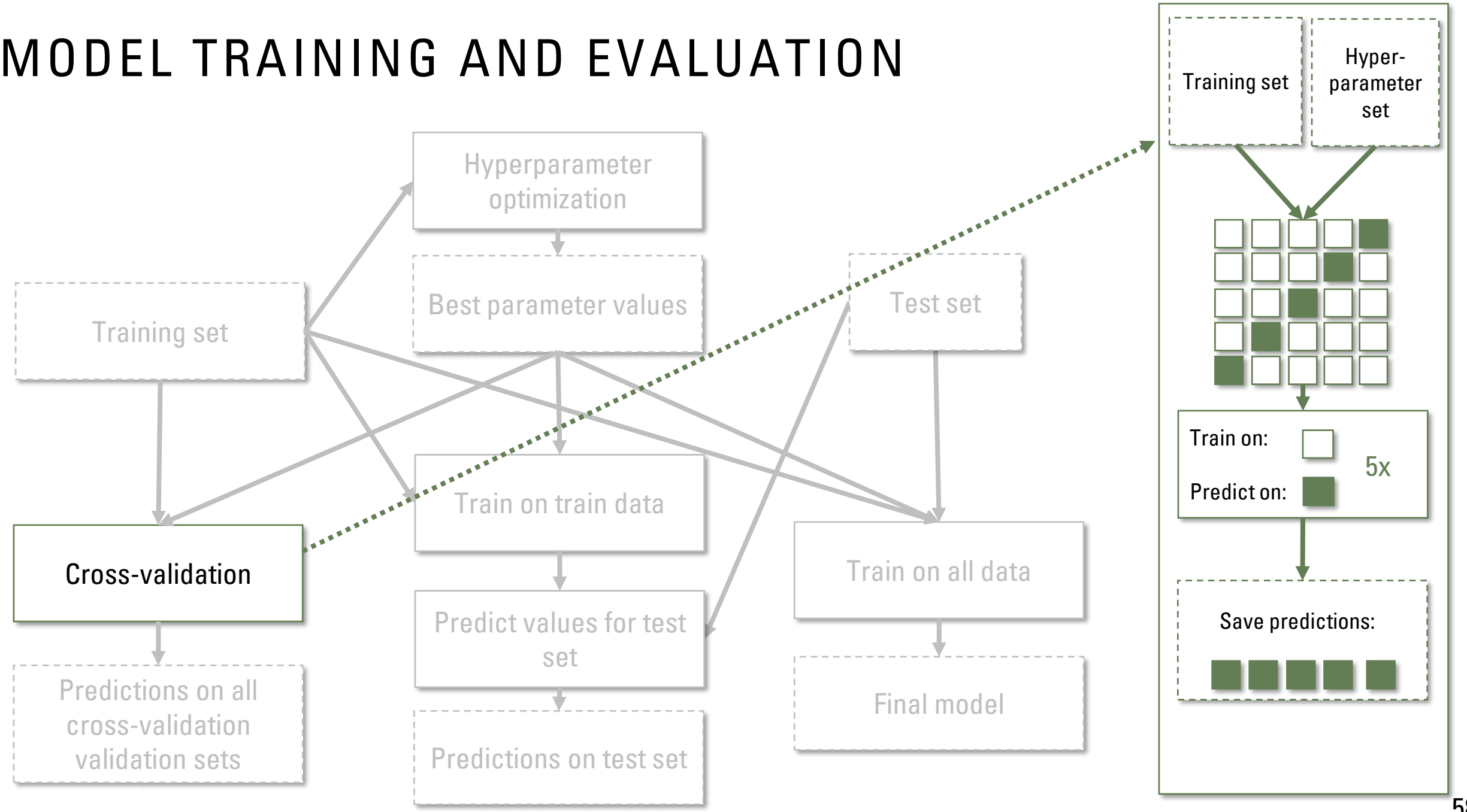
MODEL TRAINING AND EVALUATION



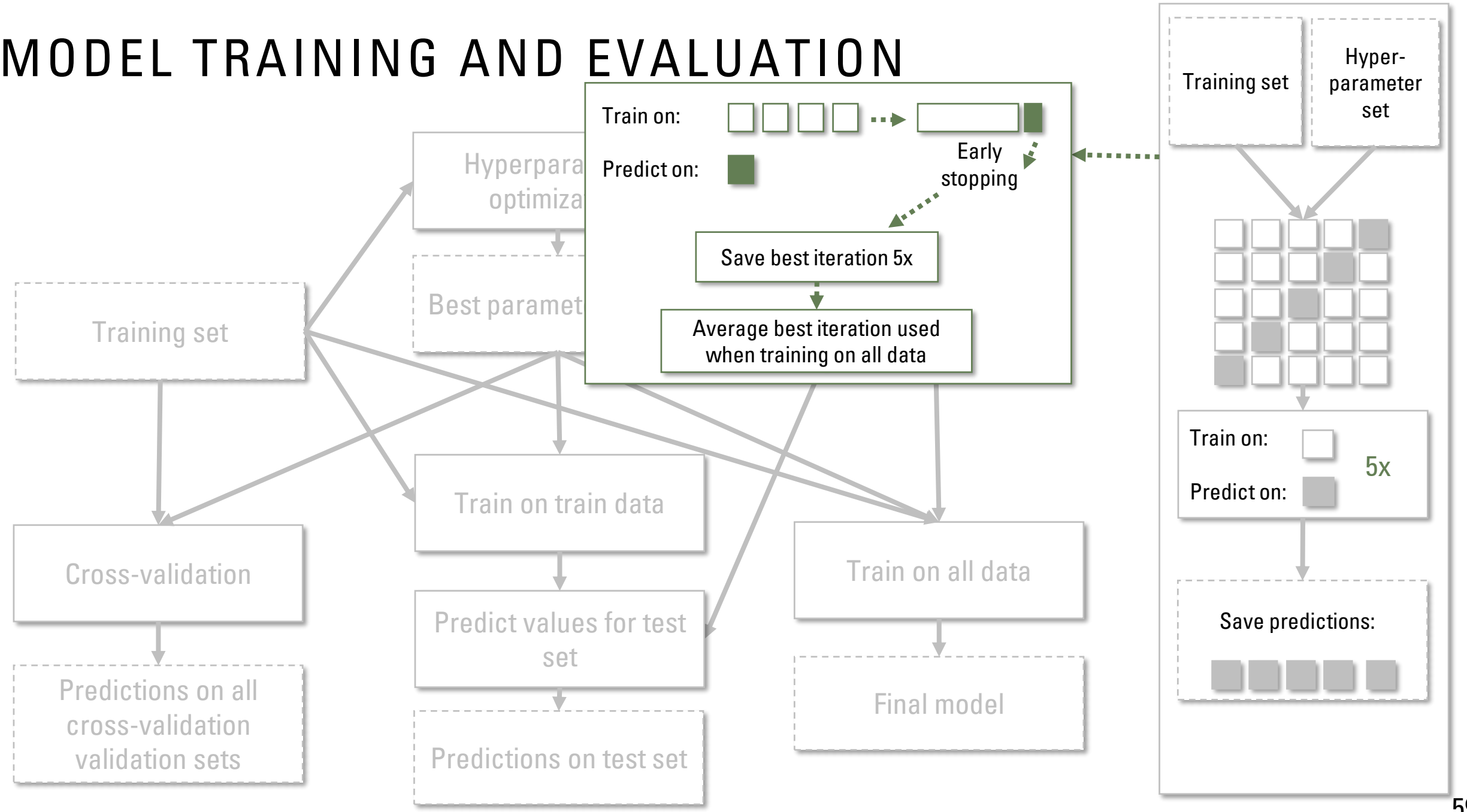
EARLY STOPPING



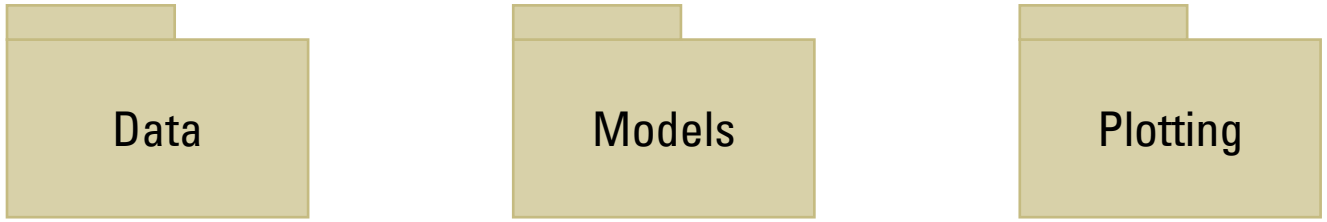
MODEL TRAINING AND EVALUATION



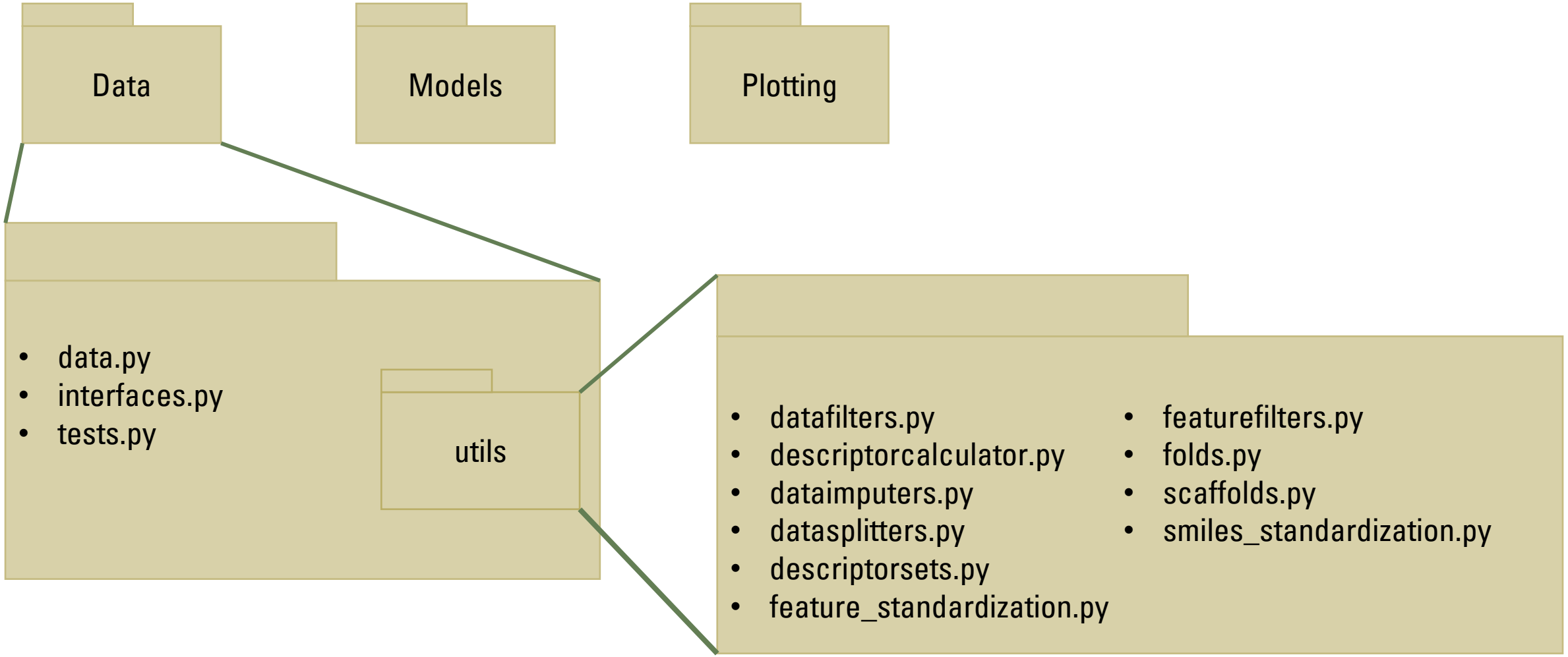
MODEL TRAINING AND EVALUATION



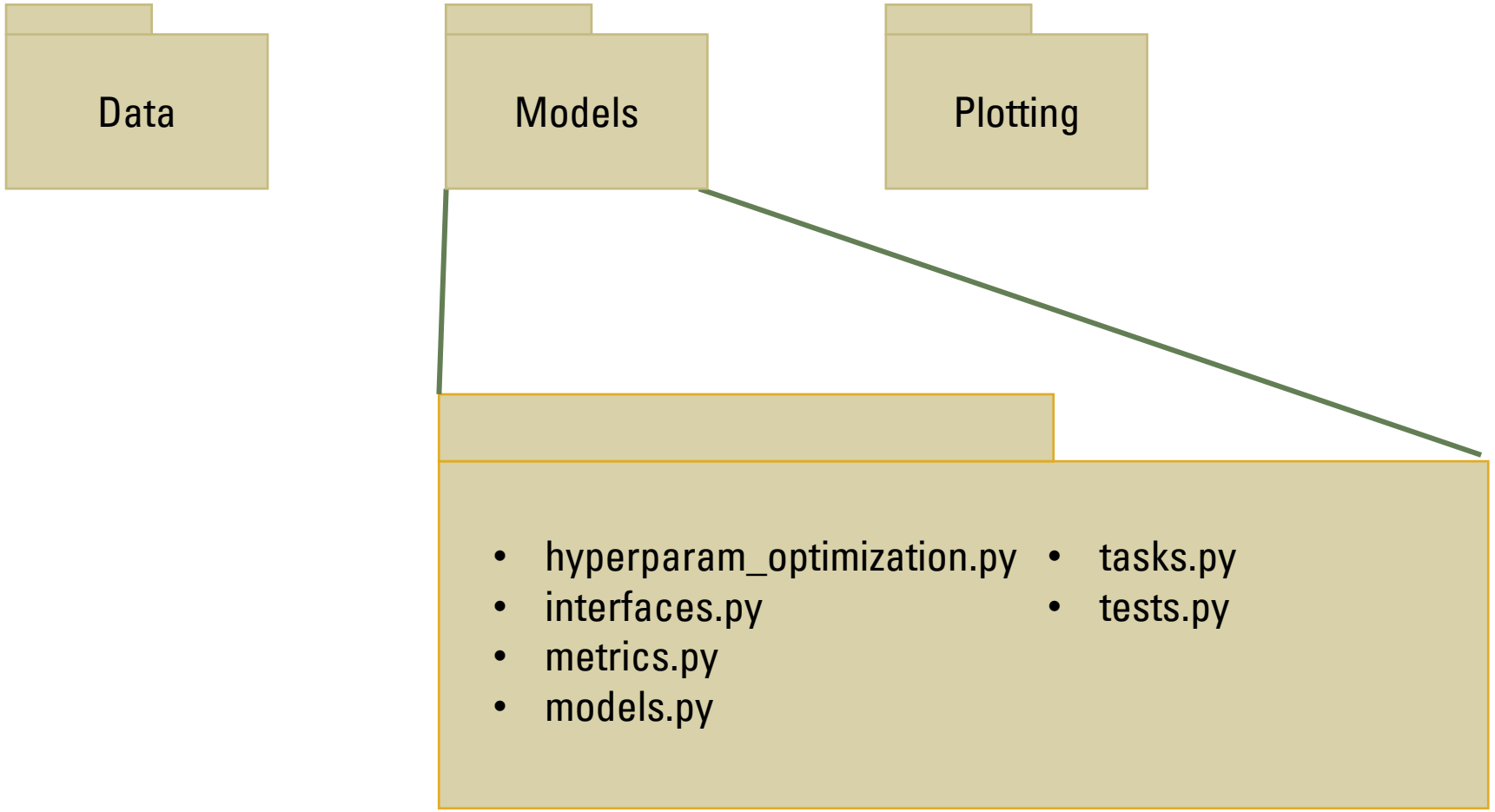
THE PACKAGE STRUCTURE: BASE



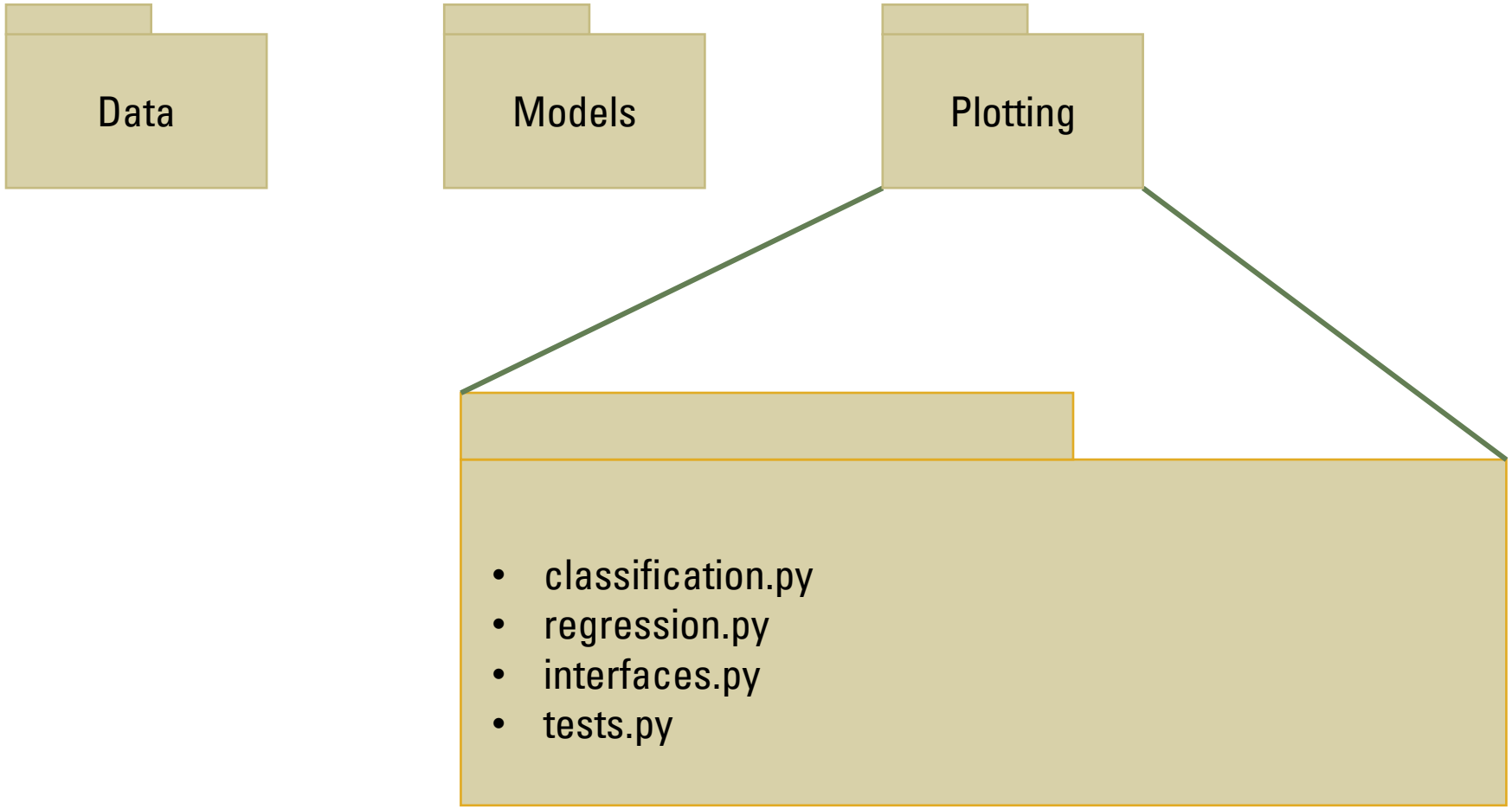
THE PACKAGE STRUCTURE: BASE



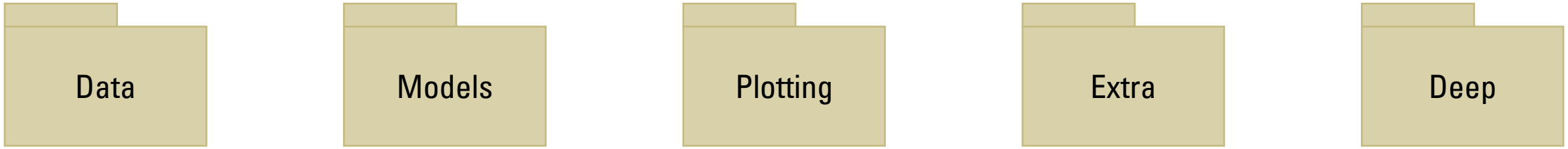
THE PACKAGE STRUCTURE: BASE



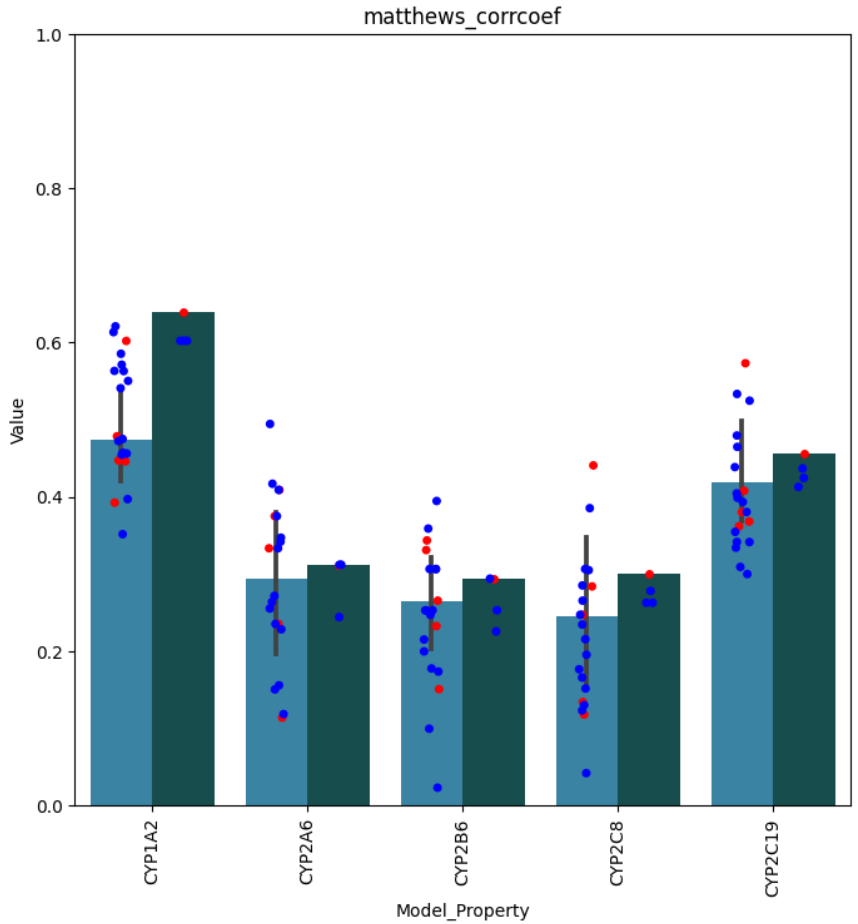
THE PACKAGE STRUCTURE

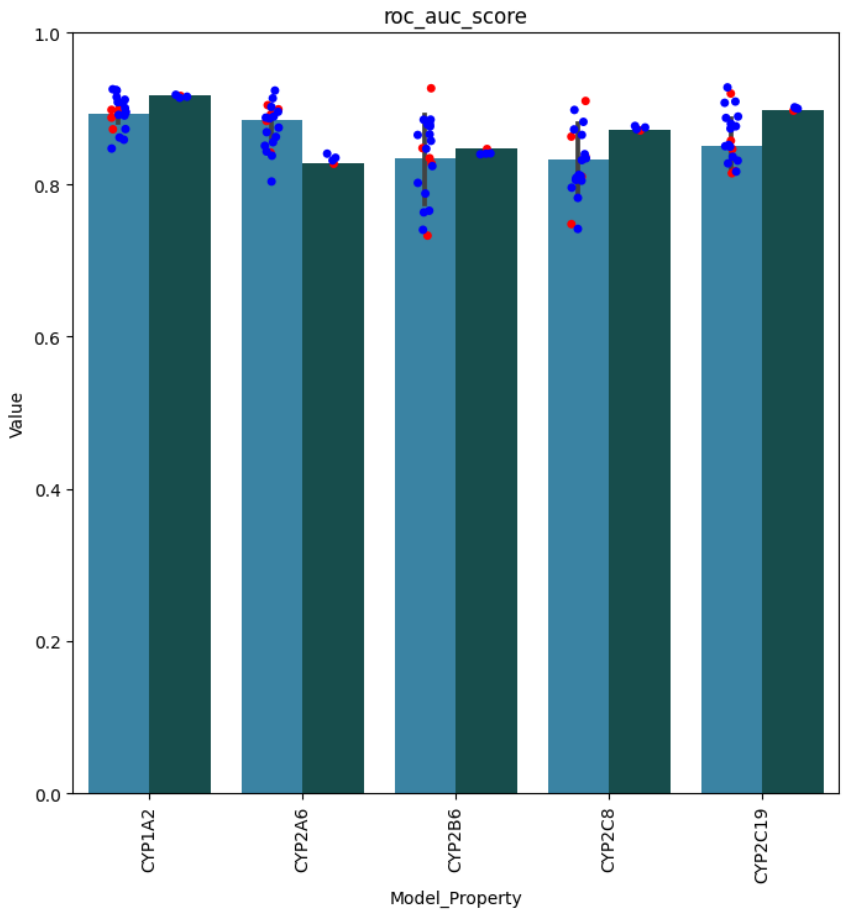


THE PACKAGE STRUCTURE: EXTENSIONS



REPRODUCTION OF RESULTS WITHOUT QSPRPRED





QSPRpred cross-validation
QSPRpred test set
QSPRpred performance
NO-QSPRpred performance