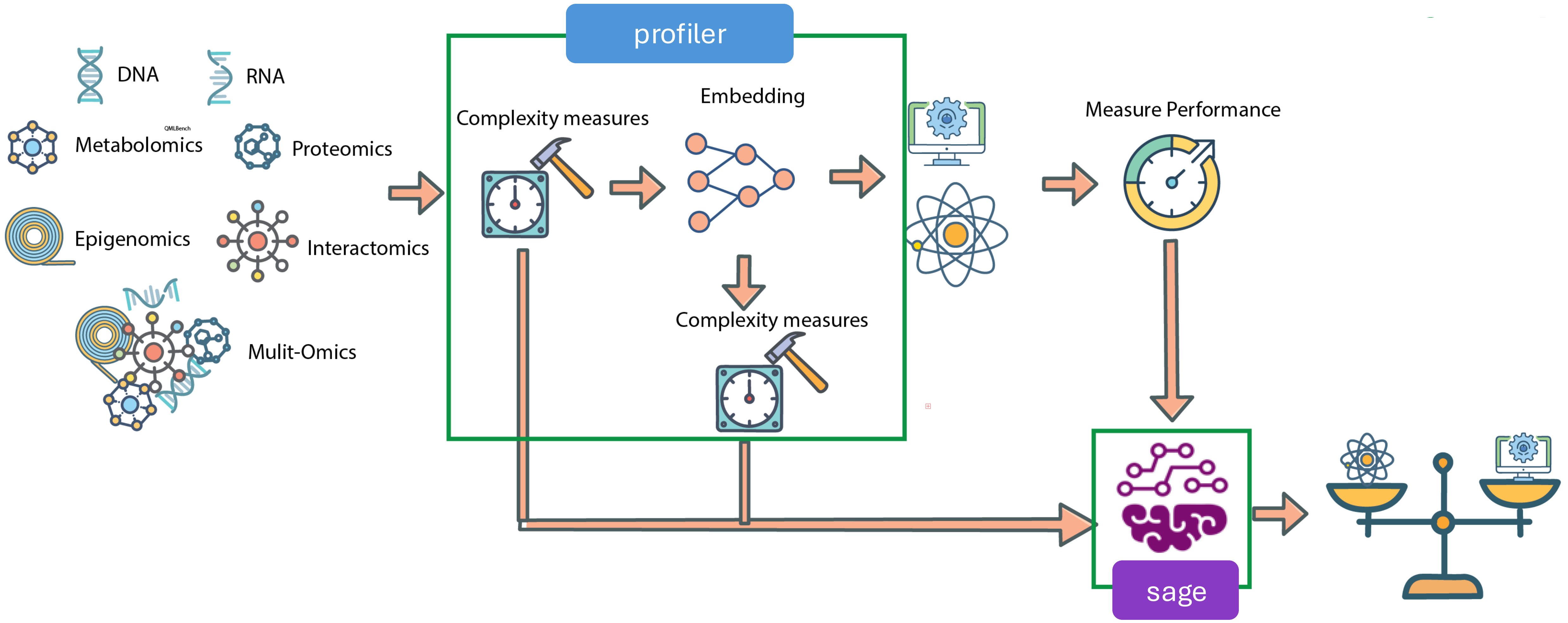
# qml4omics: A Quantum-Classical Machine Learning Benchmarking tool for multi-omics data

# qml4omics


DNA
RNA
QMLBench
Metabolomics
Proteomics
Epigenomics
Interactomics
Mulit-Omics
profiler
Complexity measures
Embedding
Complexity measures
Measure Performance
sage

# qml4omics: Data Complexities

## Dimensional

- Intrinsic Dimension (Rank)
- Manifold (Fractal Dimension)
- Volume
- Effective rank
- Eigenspectra

## Distributional

- Kurtosis & Skewness
- Mutual Information
- Sparsity
- Entropy
- Condition Number

## Geometric

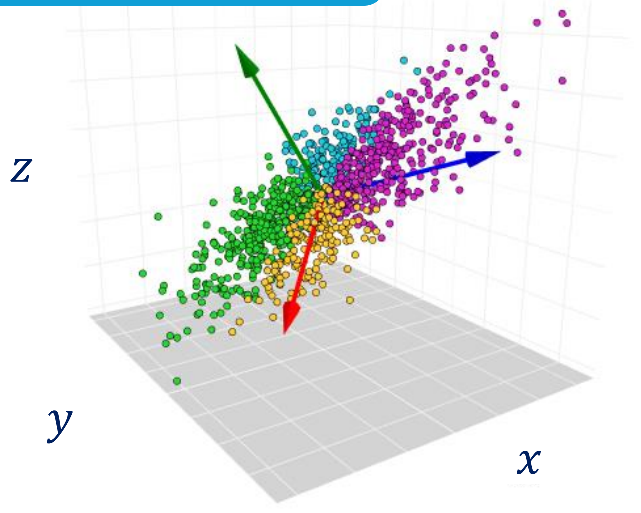
- Manifolds
- Clusters
- Density
- Topological Data Analysis
- Graph-based measures

## Sampling

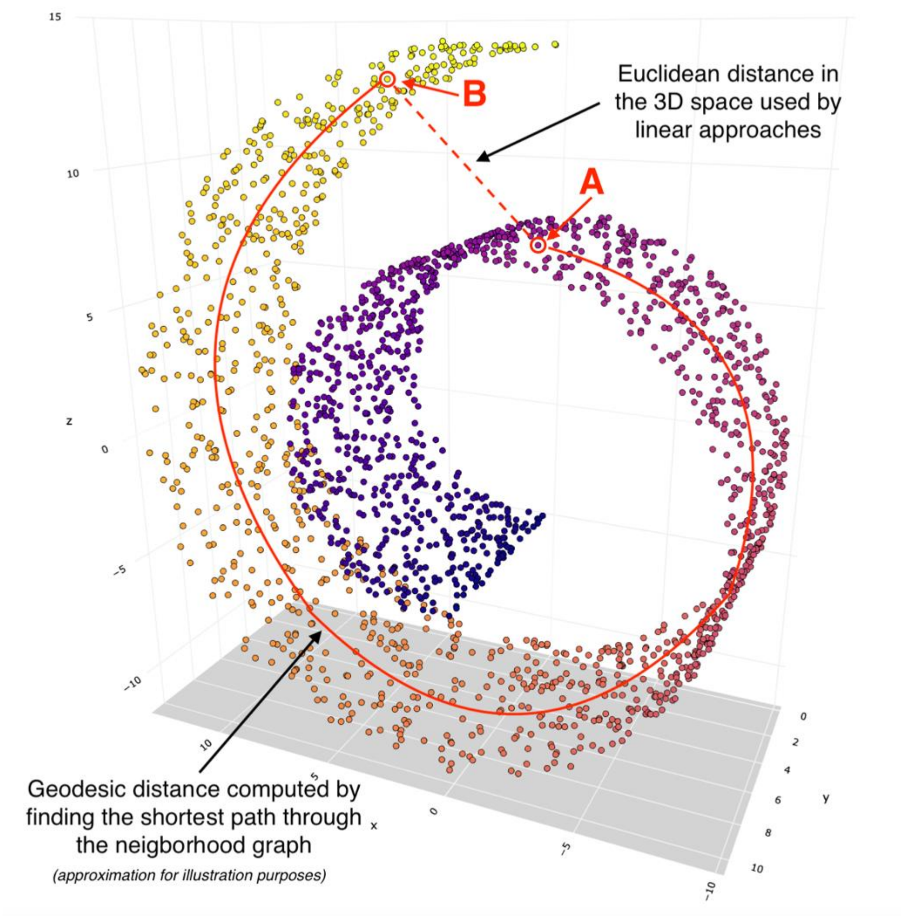
- Class imbalance ratio
- Class overlap measures
- Margin of separation between classes
- Sampling density variation

# qml4omics:Embeddings

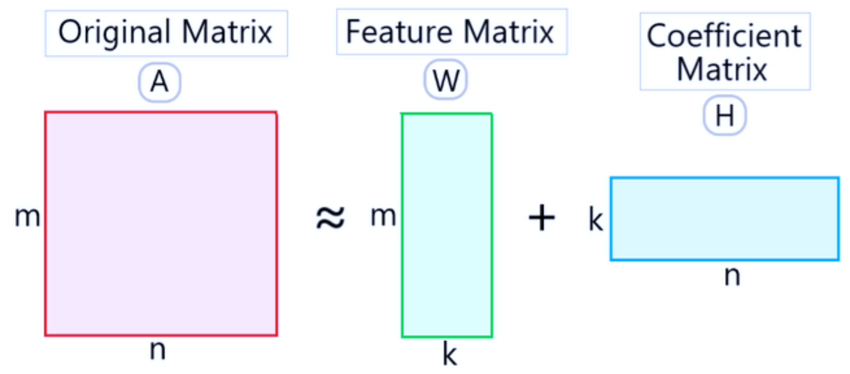
Principal Component Analysis (PCA)



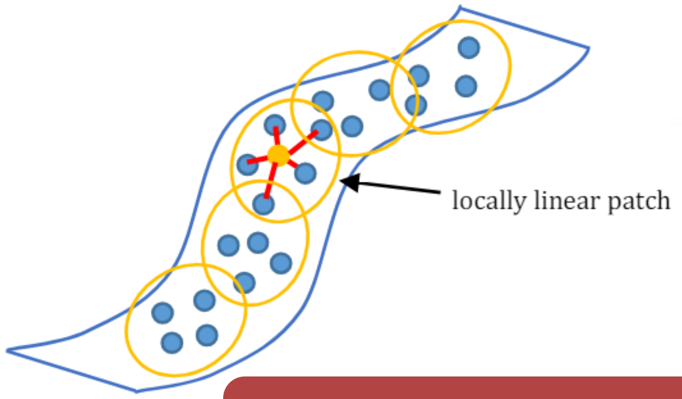
Spectral


B
A
Euclidean distance in the 3D space used by linear approaches
Geodesic distance computed by finding the shortest path through the neigborhood graph
(approximation for illustration purposes)
z
x
y

Isomap



Non-negative Matrix Factorization (NMF)



Locally Linear Embedding (LLE)

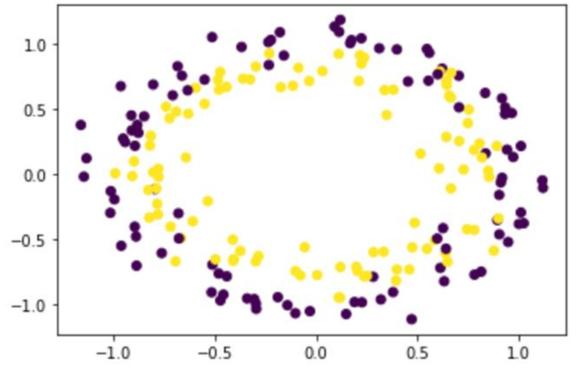
# qml4omics: Models

- Logistic Regression
- Support Vector Classifiers
- Naïve Bayes
- Random Forest
- XGBoost
- Multi-layer Perceptron

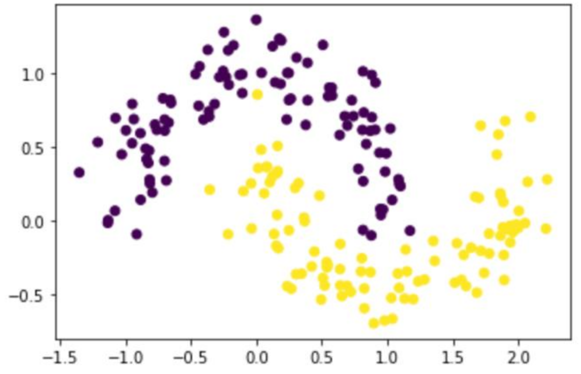
- Quantum Kernel Estimation
- Projected Quantum Kernel
- Quantum Support Vector Classifiers
- Variational Quantum Classifier / Quantum Neural Networks

# Artificial Data Generation

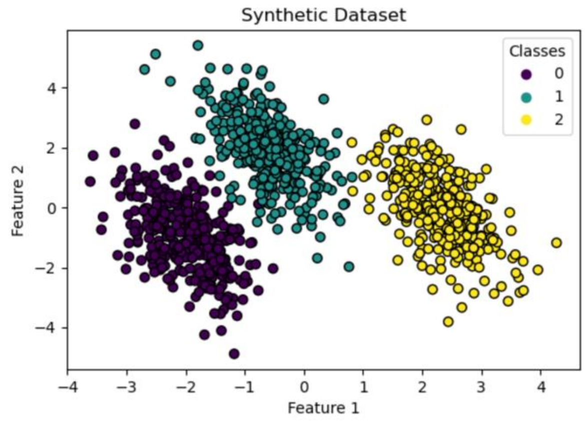
- To diversify datasets, we developed functions to generate artificial data blobs based on user-defined combinations of data characteristic.
- These modules generate blobs, moons, circles, spheres, spirals.
- Located inside /data/artificial_datasets/make_X



Circles



Half moons



Blobs

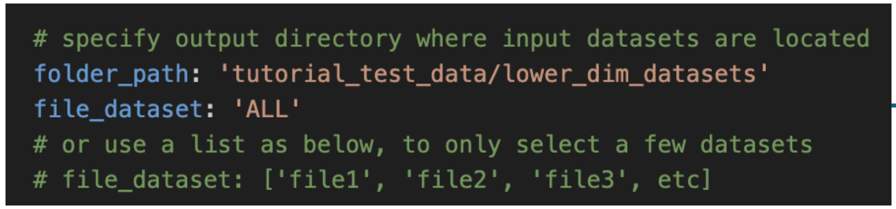
## How it works

# Understanding the `config.yaml`

- **It controls everything!**
  - Parameters passed as arguments throughout the code base

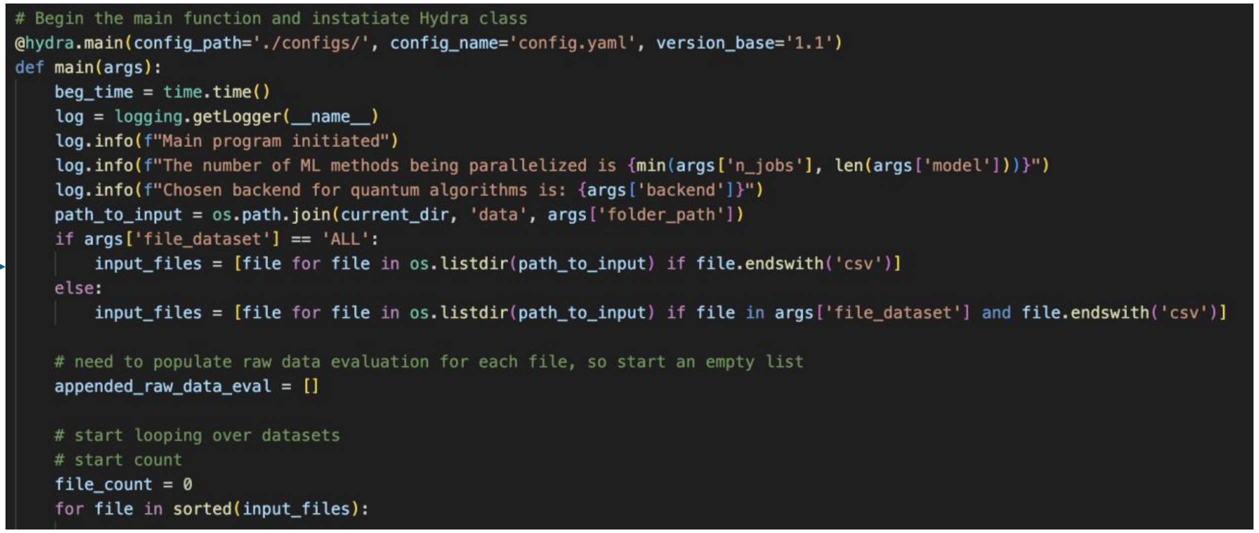
1) Input data sets

`config.yaml`



```yaml
# specify output directory where input datasets are located
folder_path: 'tutorial_test_data/lower_dim_datasets'
file_dataset: 'ALL'
# or use a list as below, to only select a few datasets
# file_dataset: ['file1', 'file2', 'file3', etc]
```

`qml4omics-profiler.py` (main function)

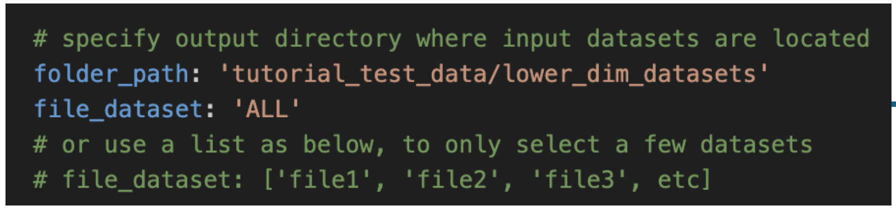


```python
# Begin the main function and instatiate Hydra class
@hydra.main(config_path='./configs/', config_name='config.yaml', version_base='1.1')
def main(args):
    beg_time = time.time()
    log = logging.getLogger(__name__)
    log.info(f"Main program initiated")
    log.info(f"The number of ML methods being parallelized is {min(args['n_jobs'], len(args['model']))}")
    log.info(f"Chosen backend for quantum algorithms is: {args['backend']}")
    path_to_input = os.path.join(current_dir, 'data', args['folder_path'])
    if args['file_dataset'] == 'ALL':
        input_files = [file for file in os.listdir(path_to_input) if file.endswith('csv')]
    else:
        input_files = [file for file in os.listdir(path_to_input) if file in args['file_dataset'] and file.endswith('csv')]

    # need to populate raw data evaluation for each file, so start an empty list
    appended_raw_data_eval = []

    # start looping over datasets
    # start count
    file_count = 0
    for file in sorted(input_files):
```

# Understanding the `config.yaml`

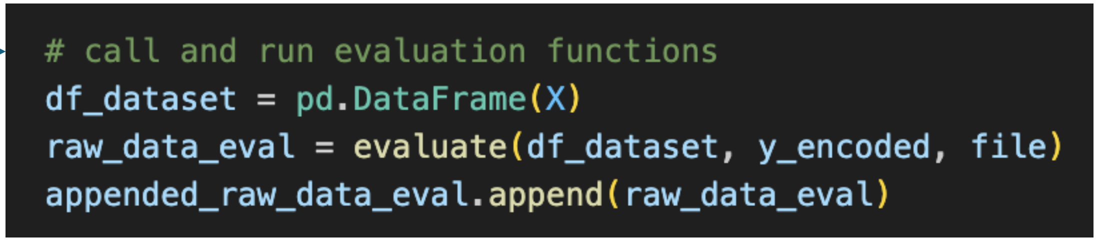
2) Complexity evaluation (on raw and embedded data)

`config.yaml`

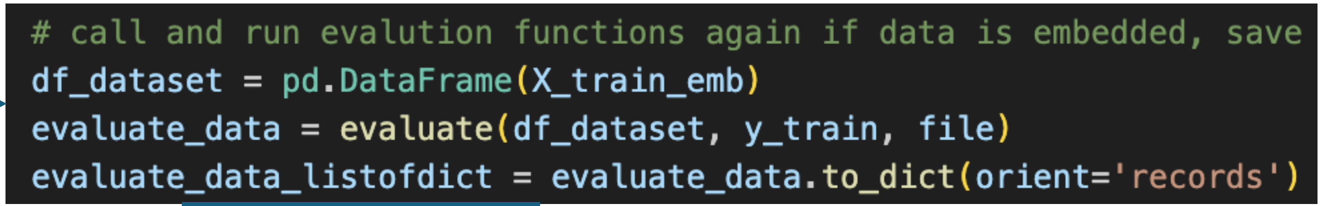


```yaml
# specify output directory where input datasets are located
folder_path: 'tutorial_test_data/lower_dim_datasets'
file_dataset: 'ALL'
# or use a list as below, to only select a few datasets
# file_dataset: ['file1', 'file2', 'file3', etc]
```

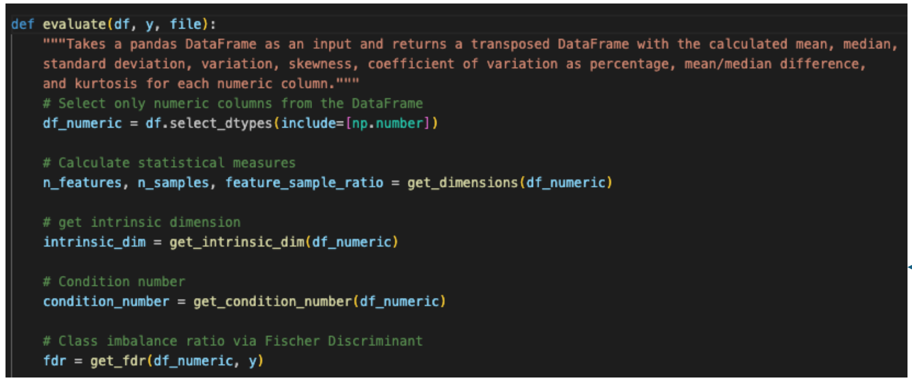
`qml4omics-profiler.py` (main function)



```python
# call and run evaluation functions
df_dataset = pd.DataFrame(X)
raw_data_eval = evaluate(df_dataset, y_encoded, file)
appended_raw_data_eval.append(raw_data_eval)
```



```python
# call and run evalution functions again if data is embedded, save
df_dataset = pd.DataFrame(X_train_emb)
evaluate_data = evaluate(df_dataset, y_train, file)
evaluate_data_listofdict = evaluate_data.to_dict(orient='records')
```



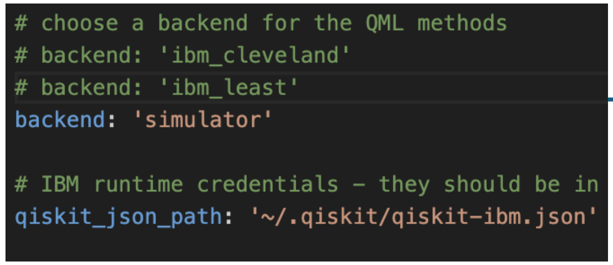
```python
def evaluate(df, y, file):
    """Takes a pandas DataFrame as an input and returns a transposed DataFrame with the calculated mean, median,
    standard deviation, variation, skewness, coefficient of variation as percentage, mean/median difference,
    and kurtosis for each numeric column."""
    # Select only numeric columns from the DataFrame
    df_numeric = df.select_dtypes(include=[np.number])

    # Calculate statistical measures
    n_features, n_samples, feature_sample_ratio = get_dimensions(df_numeric)

    # get intrinsic dimension
    intrinsic_dim = get_intrinsic_dim(df_numeric)

    # Condition number
    condition_number = get_condition_number(df_numeric)

    # Class imbalance ratio via Fischer Discriminant
    fdr = get_fdr(df_numeric, y)
```

`qml4omics/evaluation/dataset_evaluation.py`

# Understanding the `config.yaml`

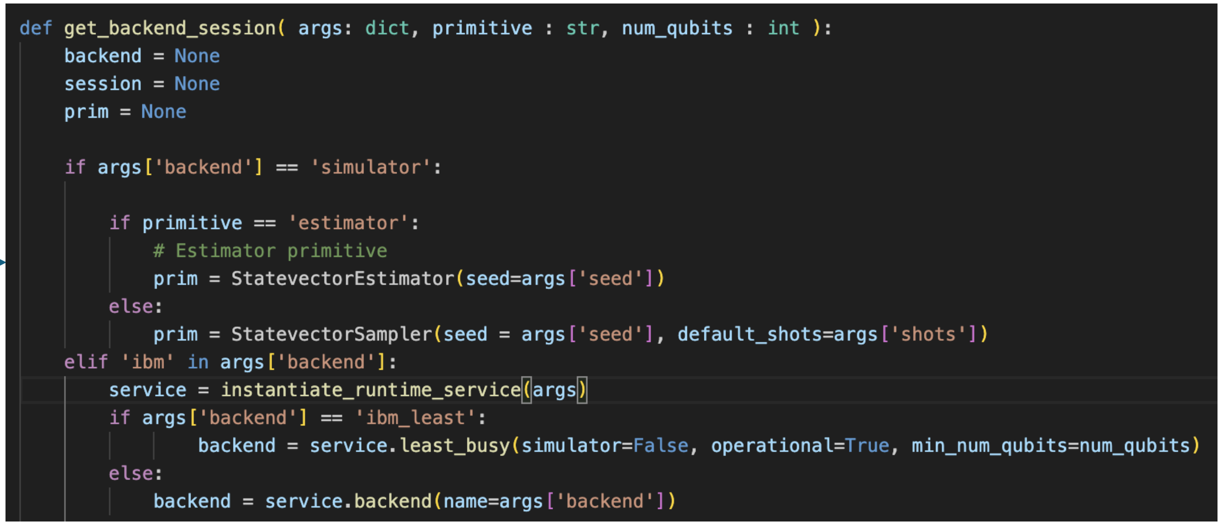
3) Quantum backend

config.yaml



```yaml
# choose a backend for the QML methods
# backend: 'ibm_cleveland'
# backend: 'ibm_least'
backend: 'simulator'

# IBM runtime credentials - they should be in
qiskit_json_path: '~/.qiskit/qiskit-ibm.json'
```

qml4omics/utils/qutils.py

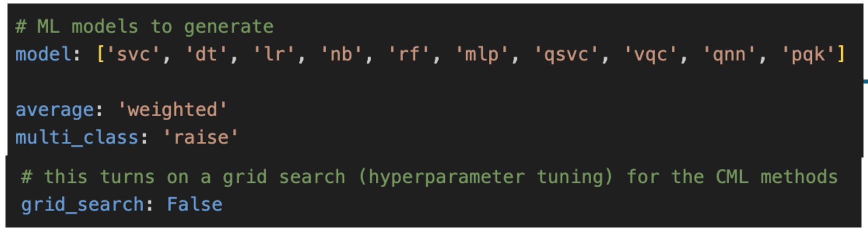


```python
def get_backend_session( args: dict, primitive : str, num_qubits : int ):
    backend = None
    session = None
    prim = None

    if args['backend'] == 'simulator':

        if primitive == 'estimator':
            # Estimator primitive
            prim = StatevectorEstimator(seed=args['seed'])
        else:
            prim = StatevectorSampler(seed = args['seed'], default_shots=args['shots'])
    elif 'ibm' in args['backend']:
        service = instantiate_runtime_service(args)
        if args['backend'] == 'ibm_least':
                backend = service.least_busy(simulator=False, operational=True, min_num_qubits=num_qubits)
        else:
            backend = service.backend(name=args['backend'])
```

# What does `config.yaml` do?

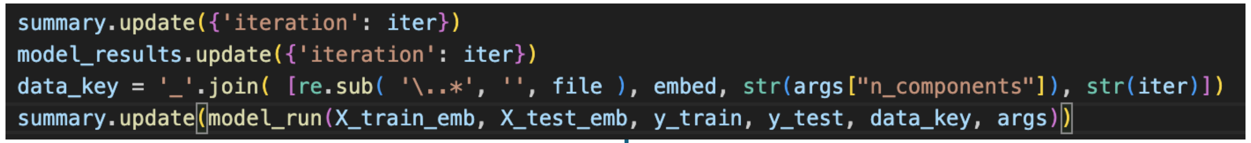
4) Run models (with and without grid search/hyperparameter tuning)

`config.yaml`

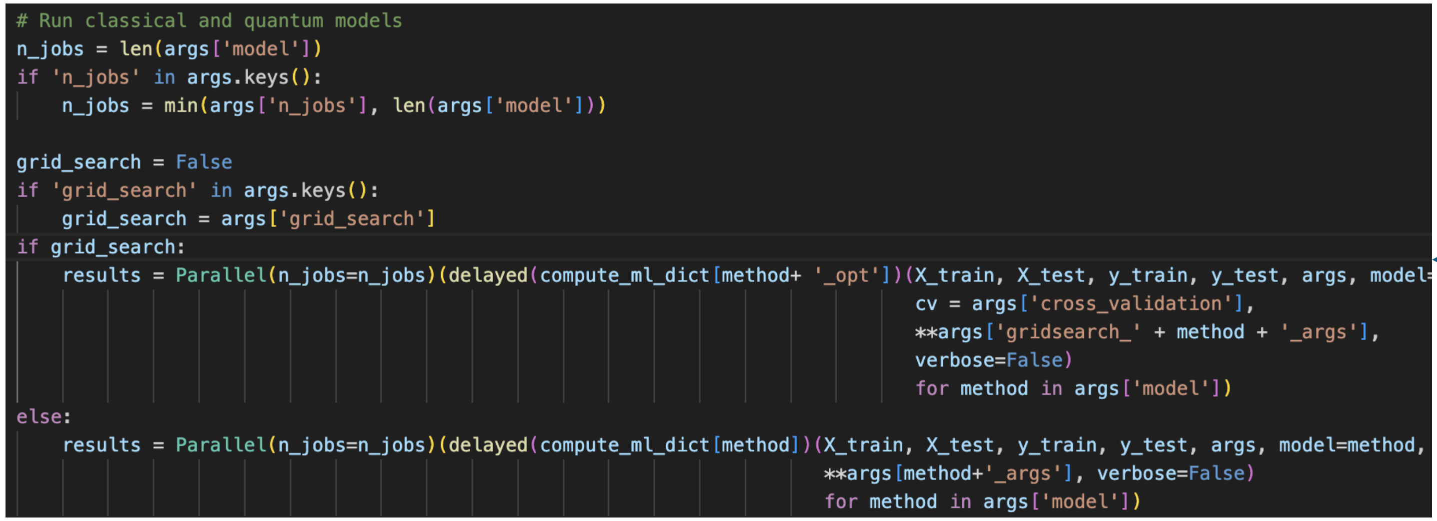


```yaml
# ML models to generate
model: ['svc', 'dt', 'lr', 'nb', 'rf', 'mlp', 'qsvc', 'vqc', 'qnn', 'pqk']

average: 'weighted'
multi_class: 'raise'
 # this turns on a grid search (hyperparameter tuning) for the CML methods
 grid_search: False
```

`qml4omics-profiler.py` (main function)



```python
summary.update({'iteration': iter})
model_results.update({'iteration': iter})
data_key = '_'.join( [re.sub( '\..*', '', file ), embed, str(args["n_components"]), str(iter)])
summary.update(model_run(X_train_emb, X_test_emb, y_train, y_test, data_key, args))
```



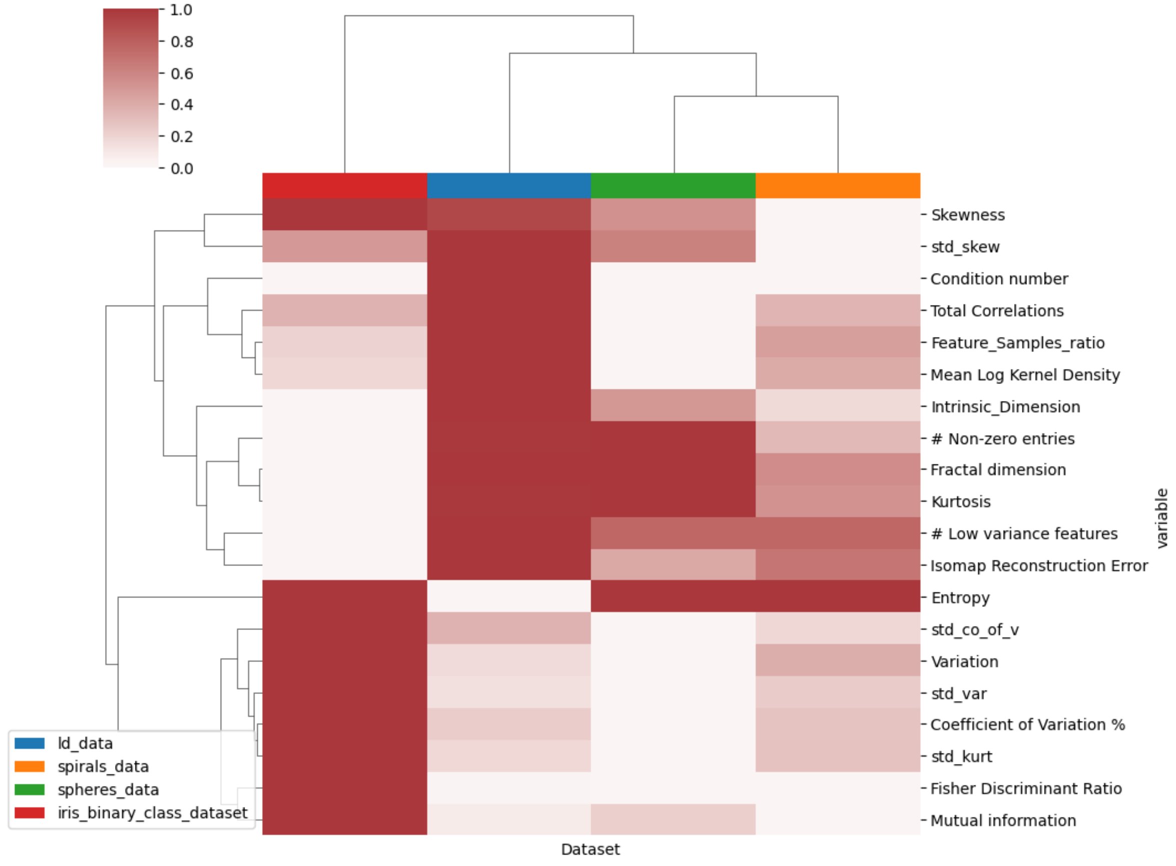
```python
# Run classical and quantum models
n_jobs = len(args['model'])
if 'n_jobs' in args.keys():
    n_jobs = min(args['n_jobs'], len(args['model']))

grid_search = False
if 'grid_search' in args.keys():
    grid_search = args['grid_search']
if grid_search:
    results = Parallel(n_jobs=n_jobs)(delayed(compute_ml_dict[method+ '_opt'])(X_train, X_test, y_train, y_test, args, model=
                                                                    cv = args['cross_validation'],
                                                                    **args['gridsearch_' + method + '_args'],
                                                                    verbose=False)
                                                                    for method in args['model'])
else:
    results = Parallel(n_jobs=n_jobs)(delayed(compute_ml_dict[method])(X_train, X_test, y_train, y_test, args, model=method,
                                                               **args[method+'_args'], verbose=False)
                                                               for method in args['model'])
```

`qml4omics/evaluation/model_run.py`

# Understanding the analyses

## Hierarchical clustering heat maps

- What it’s doing here:
  - Complexity measure range is normalized between 0 and 1.
  - Euclidean distance is calculated between columns and rows, clustering together those with the shortest distance → similar intensities for complexity measures.
  - The dendrogram branches create a pairing hierarchy.
  - Outlier has longest branch.
  - Helps answer:
    *Is there some structure or pattern in my data?*



# Understanding the analyses

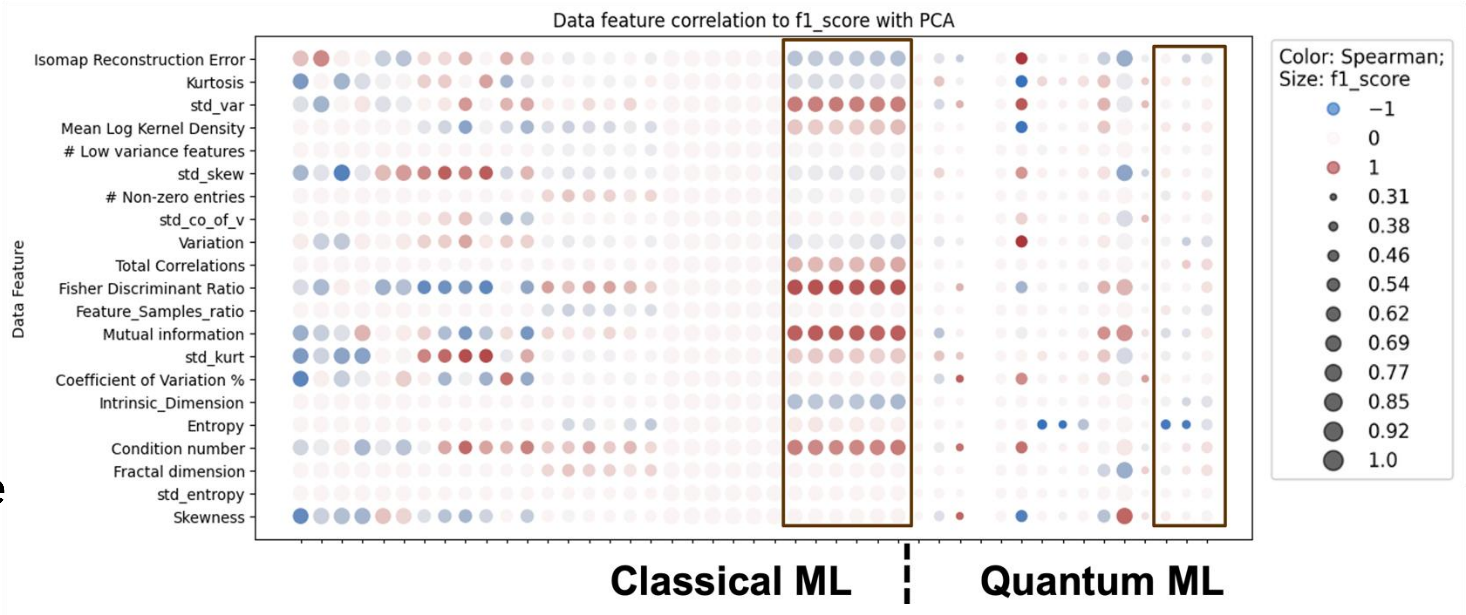
## Spearman Rank Correlations

- What it's doing here:
  - Correlates data complexity measure to model performance (F1-score)

  - Red = positive correlated
  - Blue = anti-correlated

  - Size of sphere = magnitude of F1-score

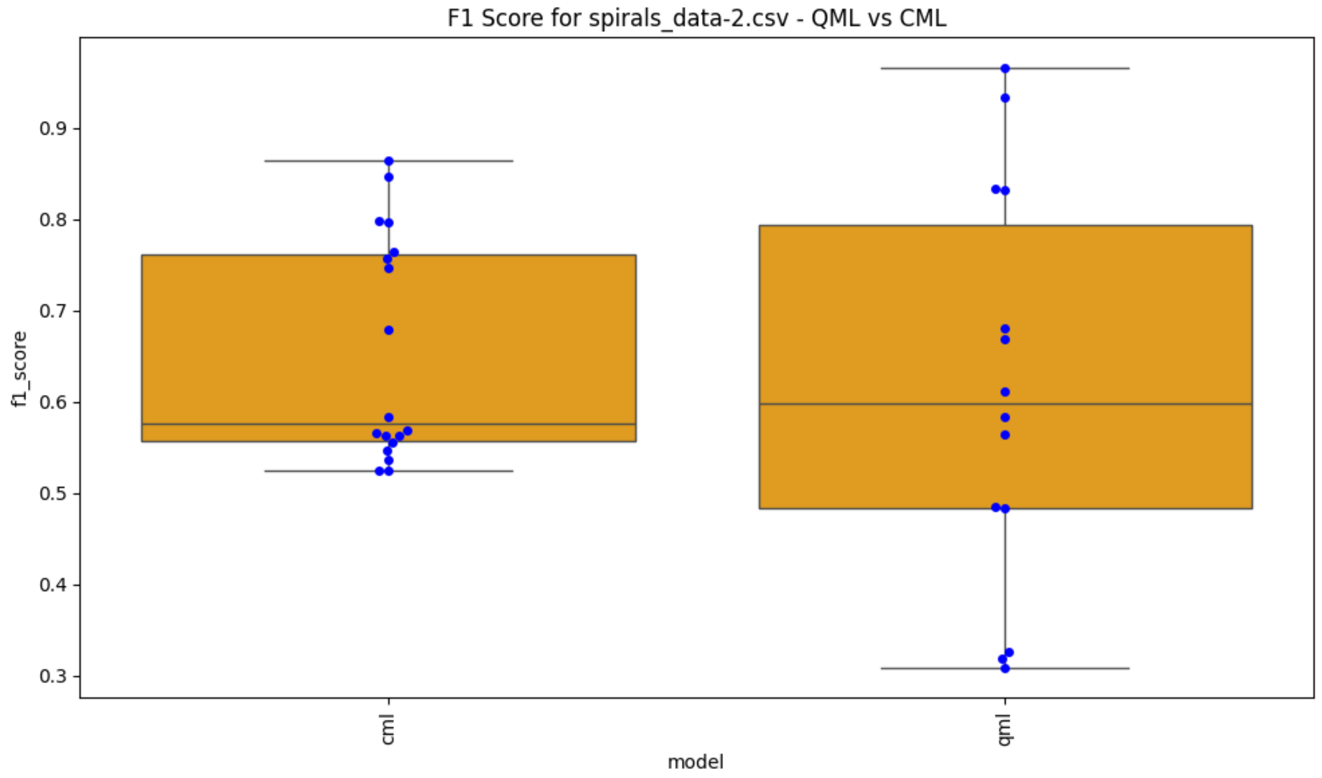
  - Helps answer:
    *What complexity measures influence your model score the most?*



# Understanding the analyses

## Box-and-whisker plots

- What it's doing here:
  - Plots distribution of median F1-scores per datasets, across all splits of data, per model
  - Top and bottom of box = upper and lower quartiles (Q3 and Q1)
  - Whiskers denote range in F1- scores
  - Helps answer:
    *What is the locality, spread, and skewness groups in my data (F1-scores) based on their quartiles?*



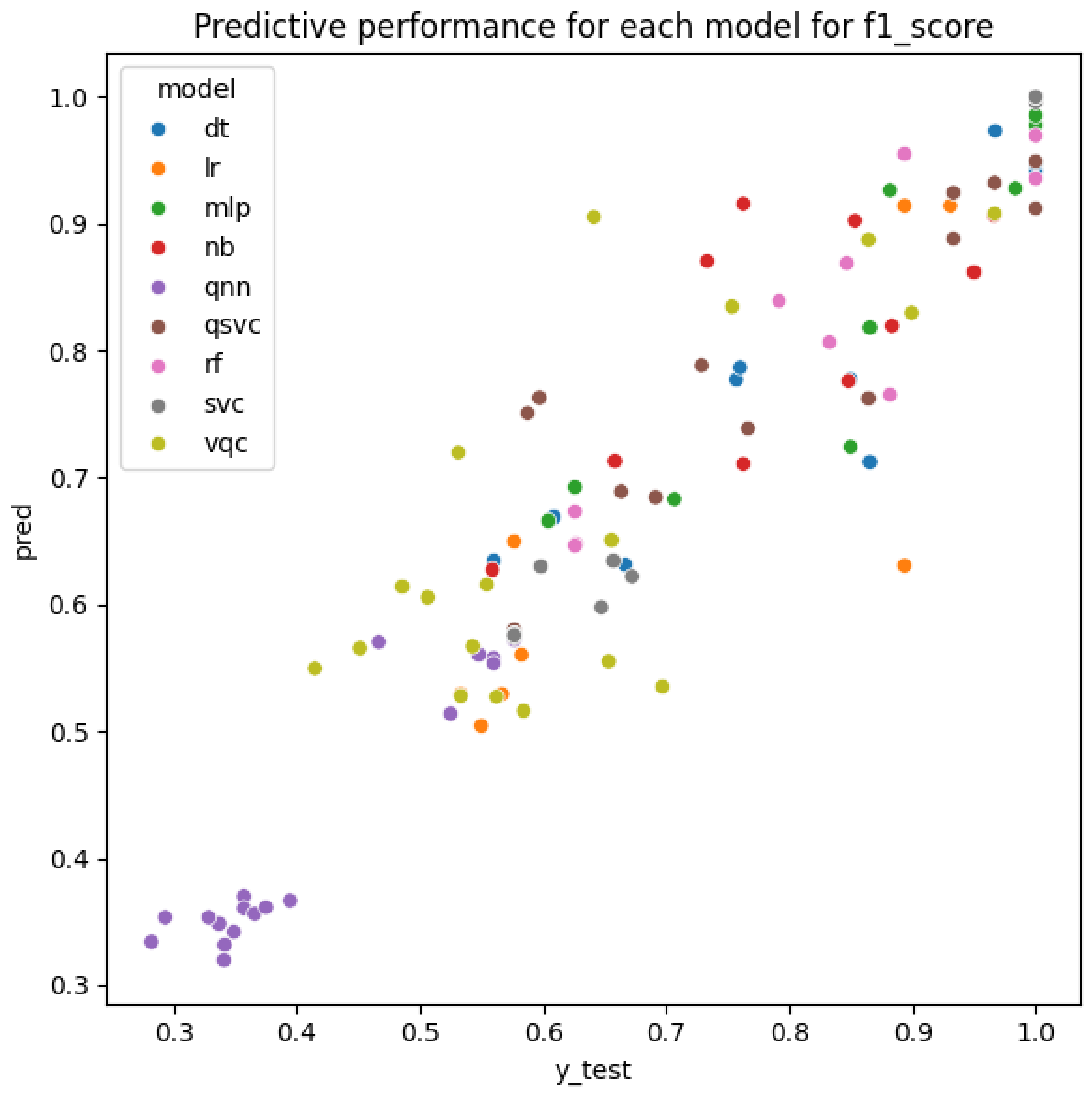
# Understanding QSage

So, what is the big picture?

What do we do with all this stuff?

- *What if I were to tell you, what ML method to use, just by looking at your data?*
- We trained a new model on all of these correlations --> QSage

Predicts F1, AUC, and accuracy beforehand --> no need to run all model!

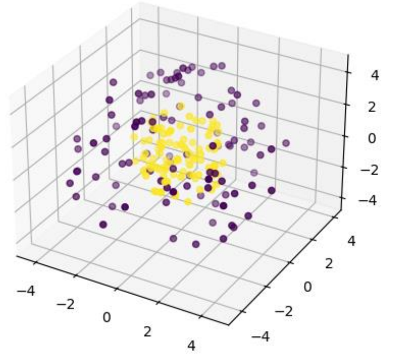


## Examples

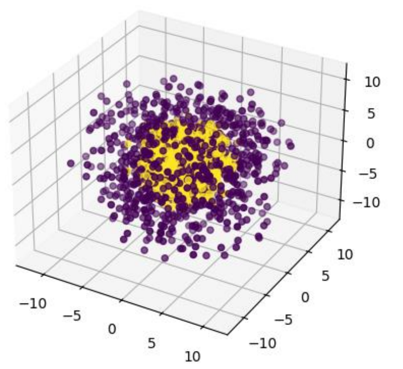
# Examples: geometric shapes

- Let's look at higher dimensional artificial, geometric data sets (3D and beyond).
- Task – generate QML and CML models for these and compare performance.
- This data is periodic – can QML do well with these

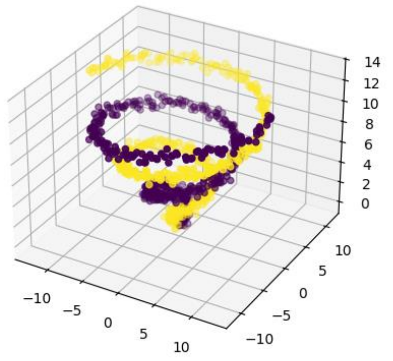
5 qubits/features



10 qubits/features

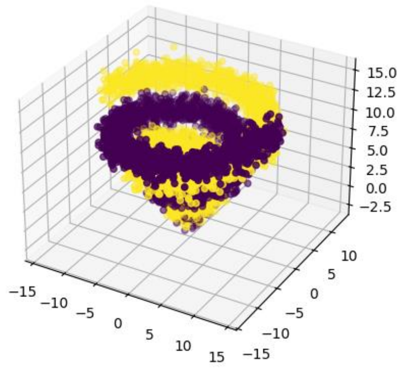


Low dimension and # of samples



3 qubits/features

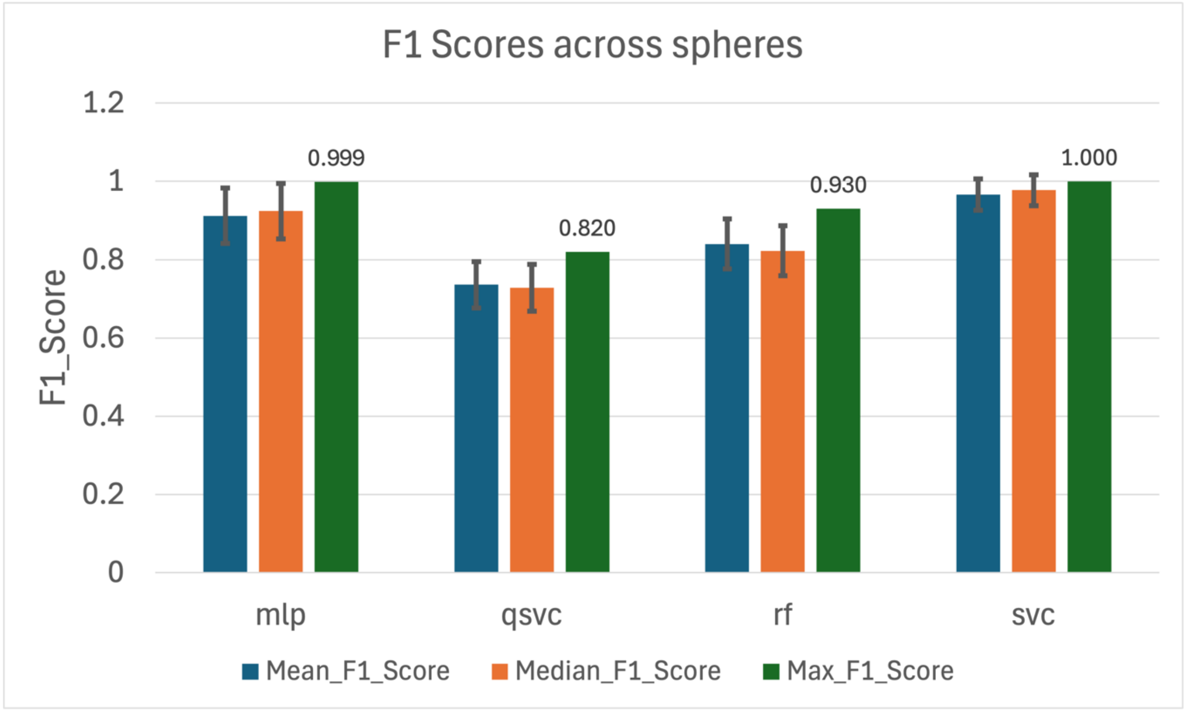
High dimension and # of samples

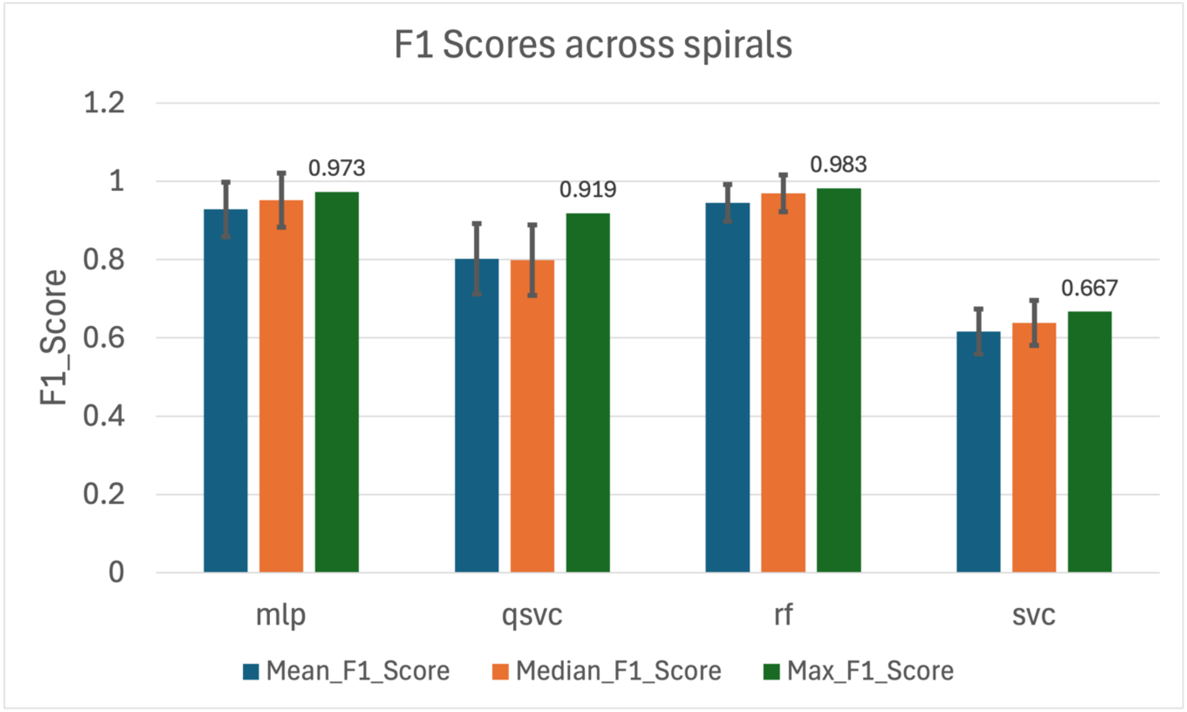


12 qubits/features

# Examples: spheres and spirals

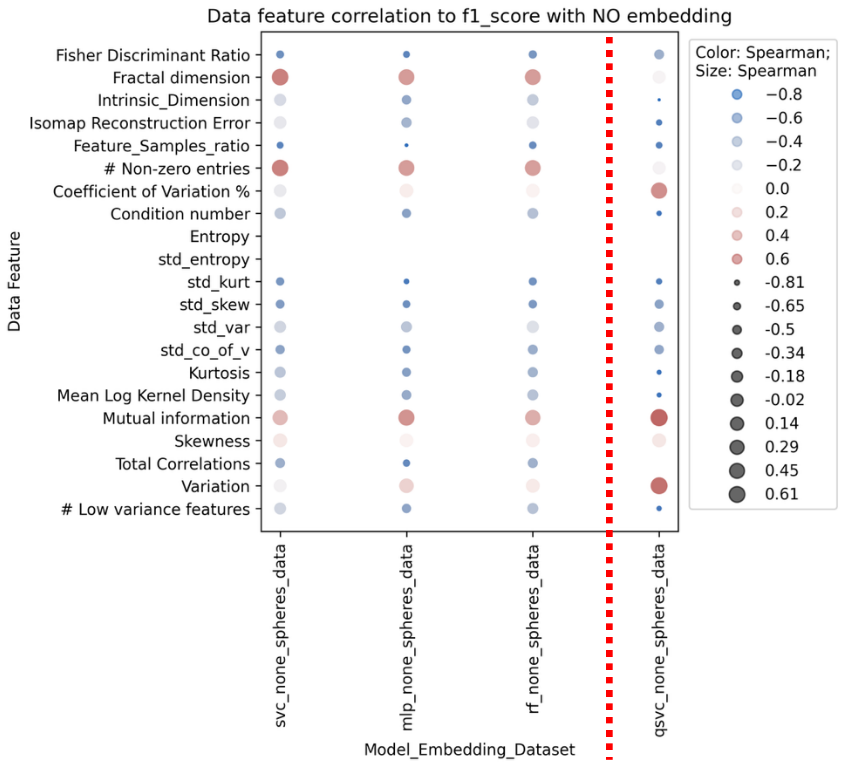
- Spirals seem QML friendly.
- SVC>QSVC with spheres, but it flips to QSVC>SVC with spirals
- RF improves with spheres, MLP is consistent across both



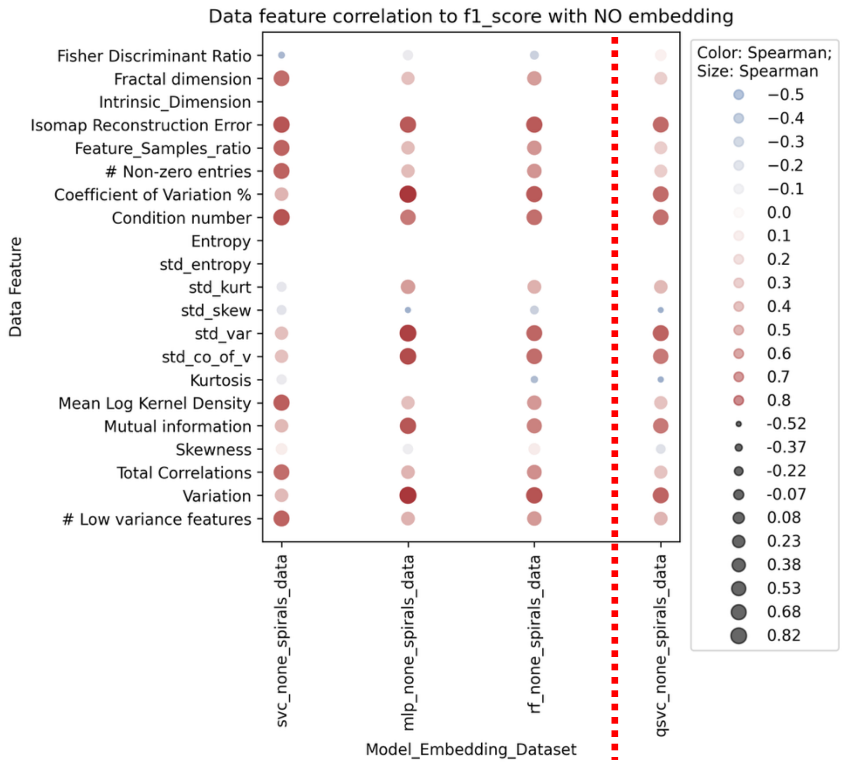


# Examples: spheres and spirals

- Spheres: clear switch with Intrinsic dimension, Coeff of variance, and total correlations
- Spirals: correlation type switches (red vs blue) for Fischer Discrimination Ratio (measures imbalance) between CML and QSVC



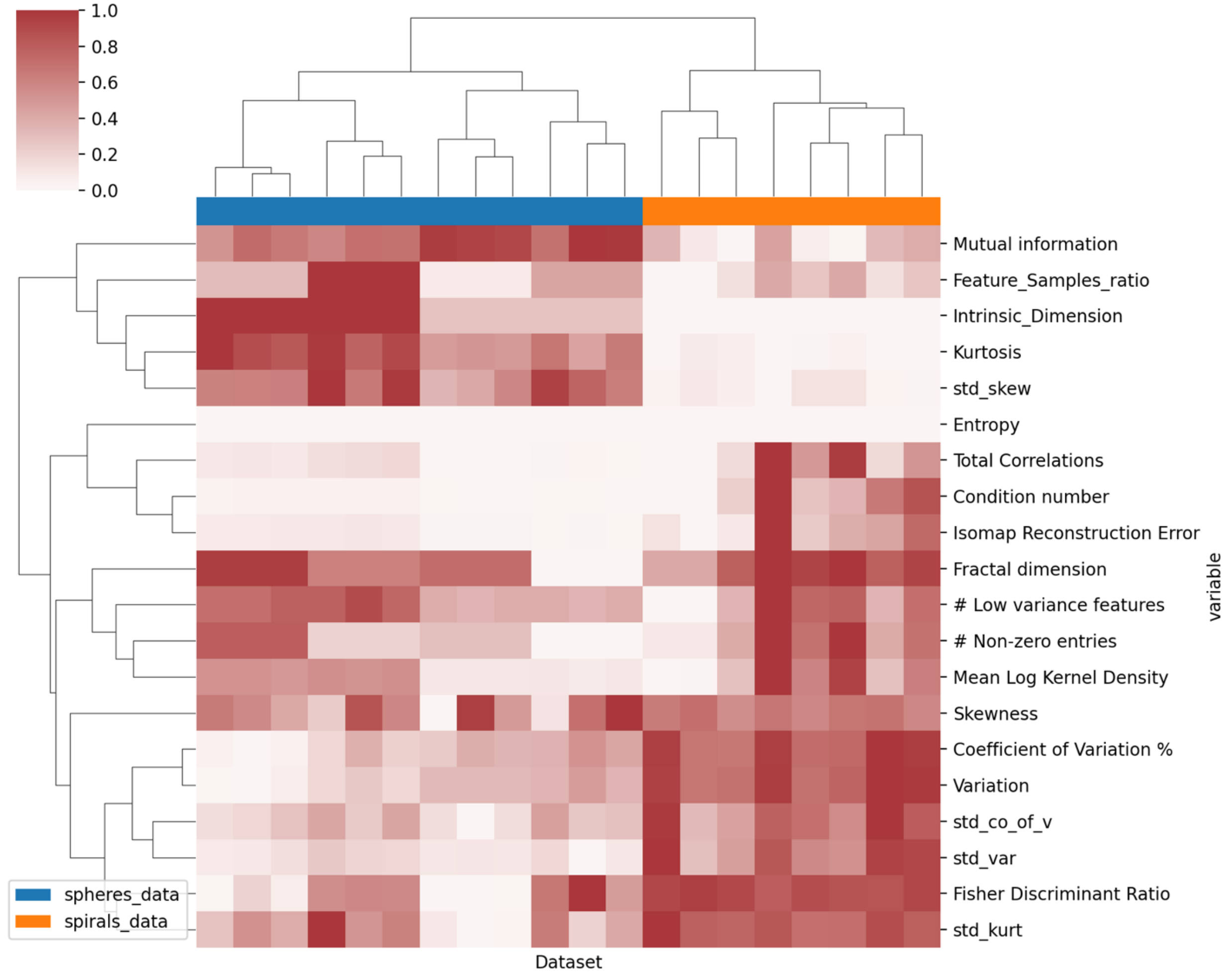
Spheres



Spirals

# Examples: spheres and spirals

- Remember: on average QSVC>SVC with spirals.
- So, what is it about the spirals?
- There is a rather obvious disparity in a few areas



## Future Directions

# Future Directions

Finalize current simulations and analyses and publish!

Develop open-source package distribution via pypi and miniforge.

Turn this into a Galaxy Tool! No coding, all done with UI.

Help you all get going with QML!

Help QSage learn more!

- Set up live server where you can upload results, and *continously train QSage with HCLS data*

## Let’s take it for a ride

# So, let's try it out!

1) Go over the `config.yaml` file and learn how to change parameters

2) Activate your environment
   - If you haven't done so, set up your environment now following the instructions in the README.md

3) Run the main code.  In your terminal, type:

```
python qml4omics-profiler.py --config-name=config.yaml
```

4) Wait and watch the progress outputs being printed out.

5) We'll analyze the results and run your own data in the afternoon 😎.