Imperial College London

Pyphi – A python open-source tool to perform MVA calculations

Centre for
Process Systems Engineering

Pyphi – download and installation

- How to download?

- GitHub - salvadorgarciamunoz/pyphi: pyPhi is a python package to perform multivariate analysis using PCA and PLS methods

- Alternative software to perform additional operations [not needed for the material covered in these modules]
  - Pyomo: Python-based, open-source optimization modeling language
  - Ipopt: GitHub - coin-or/Ipopt: COIN-OR Interior Point Optimizer IPOPT

Pyphi – download and installation

- How to know it's working?
- With the files provided there is a folder with examples, run the script **`Example_Script.py`**

- 25 plots should be produced and displayed in the default web browser.

- No errors in the console, just the output from modeling steps

Pyphi – A brief and simple guide

- This module will discuss how to build the models and how to produce all the diagnostic plots to be discussed throughout the topic.
- It is expected that you will consult this module at various times to refresh your memory as to how to use the tool to produce certain calculations as they are discussed in the course.

Pyphi – loading and importing

- Pyphi and Pyphi_plots

- pyphi.py: Contains all the numerical routines and calculations.
  - Uses numpy for all matrix operations

- Pyphi_plots: Routines to produce useful graphics with the models produced by pyphi and the data.
  - Uses Bokeh to produce all plots

- It is a good idea to import them both

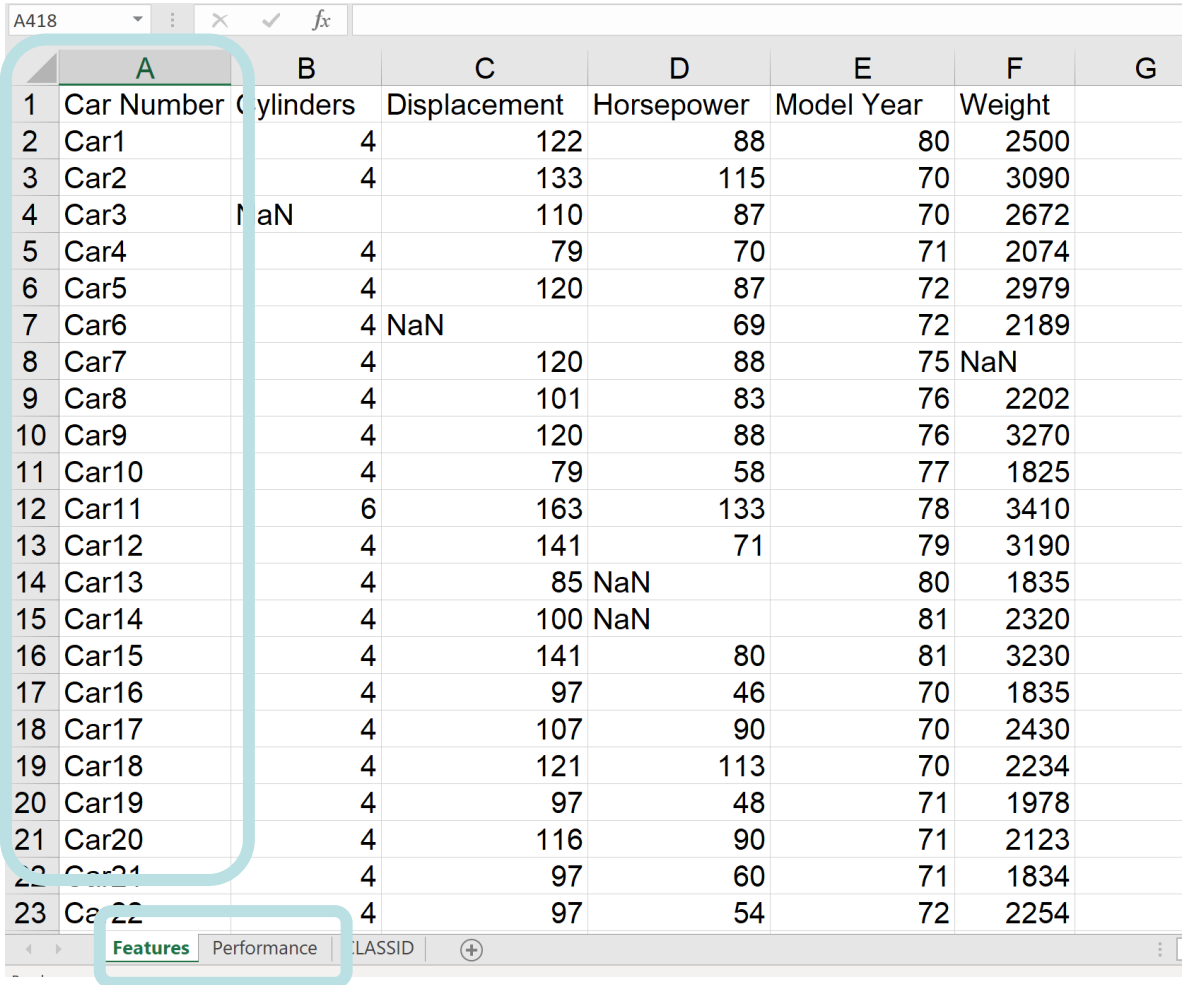
```
import pyphi as phi
import pyphi_plots as pp
```

Pyphi – Format of the data

- How to set up the Excel file

- Each sheet is read into a pandas DataFrame
- Alternatively, read one large table and separate columns in python.
- First column of DataFrame must be the observation name
  - This happens on its own when the Excel sheet is formatted like this
- Missing data to be read as **np.nan**
- **Data must be strictly numeric**

Column "A" are the observation names



	A	B	C	D	E	F	G
1	Car Number	Cylinders	Displacement	Horsepower	Model Year	Weight	
2	Car1	4	122	88	80	2500	
3	Car2	4	133	115	70	3090	
4	Car3	NaN	110	87	70	2672	
5	Car4	4	79	70	71	2074	
6	Car5	4	120	87	72	2979	
7	Car6	4	NaN	69	72	2189	
8	Car7	4	120	88	75	NaN	
9	Car8	4	101	83	76	2202	
10	Car9	4	120	88	76	3270	
11	Car10	4	79	58	77	1825	
12	Car11	6	163	133	78	3410	
13	Car12	4	141	71	79	3190	
14	Car13	4	85	NaN	80	1835	
15	Car14	4	100	NaN	81	2320	
16	Car15	4	141	80	81	3230	
17	Car16	4	97	46	70	1835	
18	Car17	4	107	90	70	2430	
19	Car18	4	121	113	70	2234	
20	Car19	4	97	48	71	1978	
21	Car20	4	116	90	71	2123	
22	Car21	4	97	60	71	1834	
23	Car22	4	97	54	72	2254	

Features | Performance | CLASSID

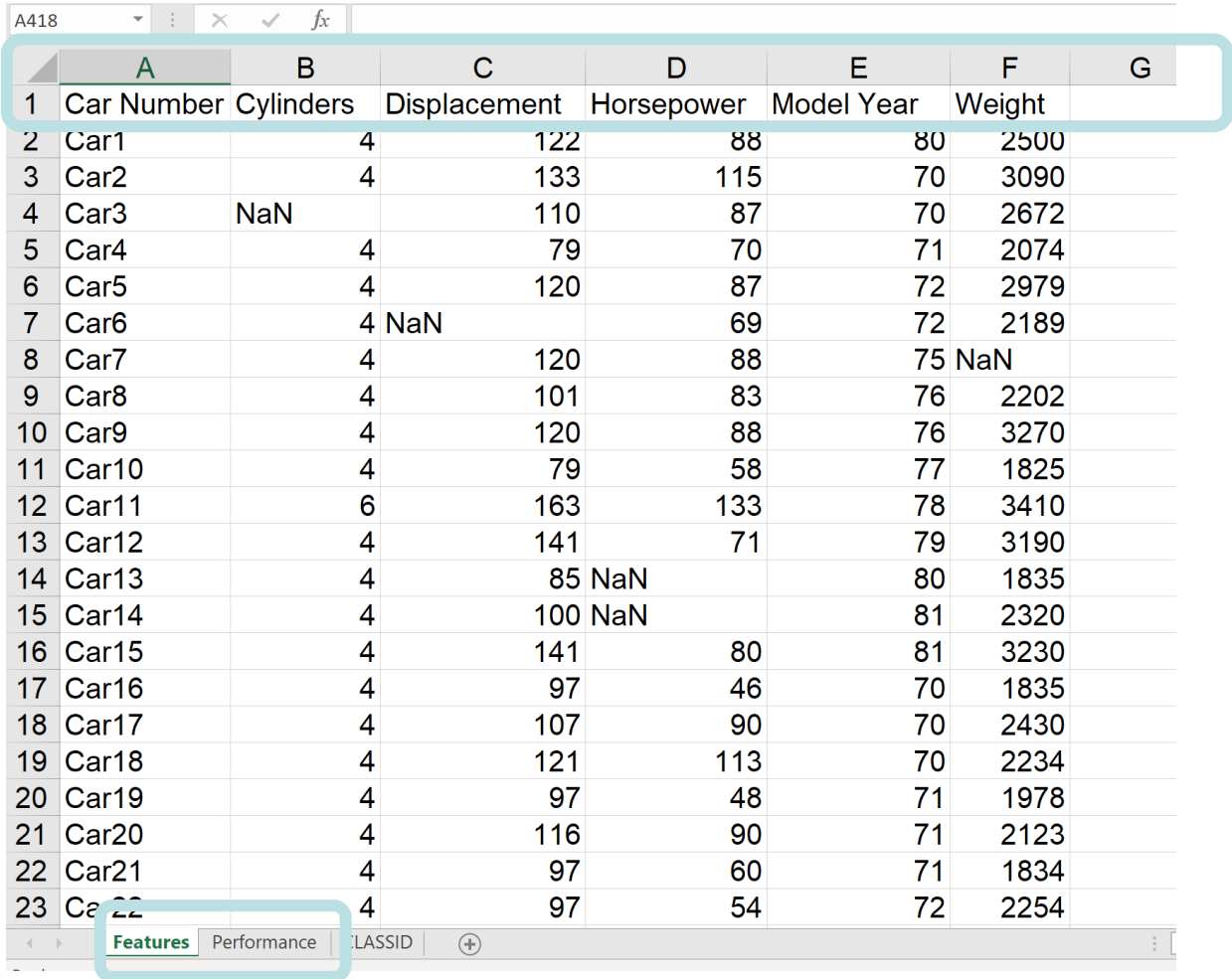
One file can contain multiple sheets

Pyphi – Format of the data

- How to set up the Excel file

- Each sheet is read into a pandas DataFrame
- Alternatively, read one large table and separate columns in python.
- First column of DataFrame must be the observation name
  - This happens on its own when the Excel sheet is formatted like this
- Missing data to be read as **np.nan**
- **Data must be strictly numeric**

Row 1 are variable names



	A	B	C	D	E	F	G
1	Car Number	Cylinders	Displacement	Horsepower	Model Year	Weight	
2	Car1	4	122	88	80	2500	
3	Car2	4	133	115	70	3090	
4	Car3	NaN	110	87	70	2672	
5	Car4	4	79	70	71	2074	
6	Car5	4	120	87	72	2979	
7	Car6	4	NaN	69	72	2189	
8	Car7	4	120	88	75	NaN	
9	Car8	4	101	83	76	2202	
10	Car9	4	120	88	76	3270	
11	Car10	4	79	58	77	1825	
12	Car11	6	163	133	78	3410	
13	Car12	4	141	71	79	3190	
14	Car13	4	85	NaN	80	1835	
15	Car14	4	100	NaN	81	2320	
16	Car15	4	141	80	81	3230	
17	Car16	4	97	46	70	1835	
18	Car17	4	107	90	70	2430	
19	Car18	4	121	113	70	2234	
20	Car19	4	97	48	71	1978	
21	Car20	4	116	90	71	2123	
22	Car21	4	97	60	71	1834	
23	Ca[illegible]	4	97	54	72	2254	

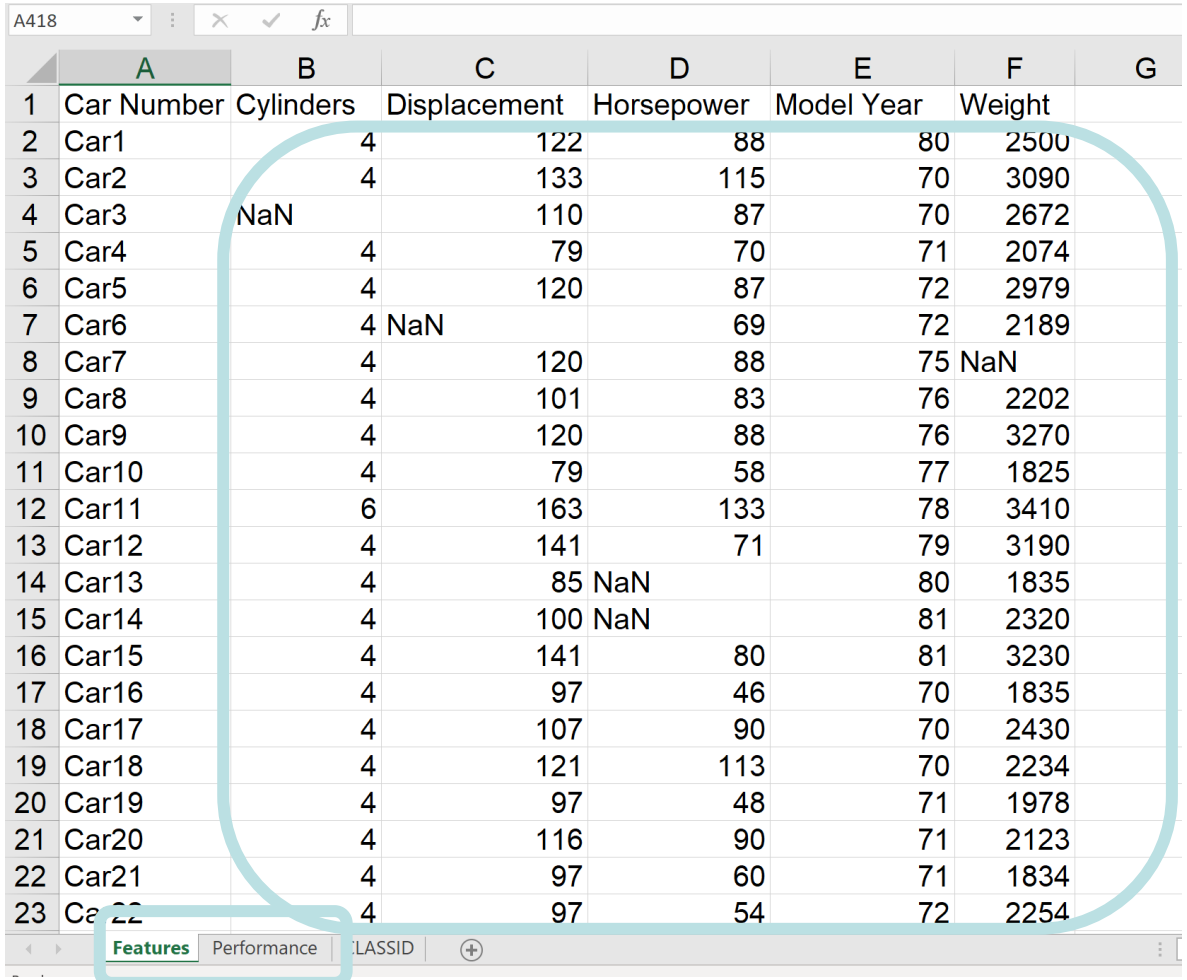
Features | Performance | CLASSID

One file can contain multiple sheets

Pyphi – Format of the data

- How to set up the Excel file

- Each sheet is read into a pandas DataFrame
- Alternatively, read one large table and separate columns in python.
- First column of DataFrame must be the observation name
  - This happens on its own when the Excel sheet is formatted like this
- Missing data to be read as **np.nan**
- **Data must be strictly numeric**



	A	B	C	D	E	F	G
1	Car Number	Cylinders	Displacement	Horsepower	Model Year	Weight	
2	Car1	4	122	88	80	2500	
3	Car2	4	133	115	70	3090	
4	Car3	NaN	110	87	70	2672	
5	Car4	4	79	70	71	2074	
6	Car5	4	120	87	72	2979	
7	Car6	4	NaN	69	72	2189	
8	Car7	4	120	88	75	NaN	
9	Car8	4	101	83	76	2202	
10	Car9	4	120	88	76	3270	
11	Car10	4	79	58	77	1825	
12	Car11	6	163	133	78	3410	
13	Car12	4	141	71	79	3190	
14	Car13	4	85	NaN	80	1835	
15	Car14	4	100	NaN	81	2320	
16	Car15	4	141	80	81	3230	
17	Car16	4	97	46	70	1835	
18	Car17	4	107	90	70	2430	
19	Car18	4	121	113	70	2234	
20	Car19	4	97	48	71	1978	
21	Car20	4	116	90	71	2123	
22	Car21	4	97	60	71	1834	
23	Ca[illegible]	4	97	54	72	2254	

Features | Performance | CLASSID

DATA

One file can contain multiple sheets

Pyphi – Format of the data

- How to set up the Excel file

- Categorical data can be loaded
- **Observation names must match.**
- Each category has a name
  - E.g. Origin, Cylinders
- Data is strictly text
- **This data can be used for color coding plots or for categorical regression.**

Column "A" are the observation names

	A	B	C	D	E
1	Car Number	Origin	Cylinders	Model_Year	CarID
2	Car1	England	4Cyl	post-1978	triumph tr7 coupe
3	Car2	France	4Cyl	pre-1978	citroen ds-21 pallas
4	Car3	France	4Cyl	pre-1978	peugeot 504
5	Car4	France	4Cyl	pre-1978	peugeot 304
6	Car5	France	4Cyl	pre-1978	peugeot 504 (sw)
7	Car6	France	4Cyl	pre-1978	renault 12 (sw)
8	Car7	France	4Cyl	pre-1978	peugeot 504
9	Car8	France	4Cyl	pre-1978	renault 12tl
10	Car9	France	4Cyl	pre-1978	peugeot 504
11	Car10	France	4Cyl	pre-1978	renault 5 gtl
12	Car11	France	6Cyl	post-1978	peugeot 604sl
13	Car12	France	4Cyl	post-1978	peugeot 504
14	Car13	France	4Cyl	post-1978	renault lecar deluxe
15	Car14	France	4Cyl	post-1978	renault 18i
16	Car15	France	4Cyl	post-1978	peugeot 505s turbo diesel
17	Car16	Germany	4Cyl	pre-1978	volkswagen 1131 deluxe sedan
18	Car17	Germany	4Cyl	pre-1978	audi 100 ls
19	Car18	Germany	4Cyl	pre-1978	bmw 2002
20	Car19	Germany	4Cyl	pre-1978	volkswagen super beetle 117
21	Car20	Germany	4Cyl	pre-1978	opel 1900
22	Car21	Germany	4Cyl	pre-1978	volkswagen model 111
23	Car22	Germany	4Cyl	pre-1978	volkswagen type 3
24	Car23	Germany	4Cyl	pre-1978	volkswagen 411 (sw)

Features | Performance | CLASSID

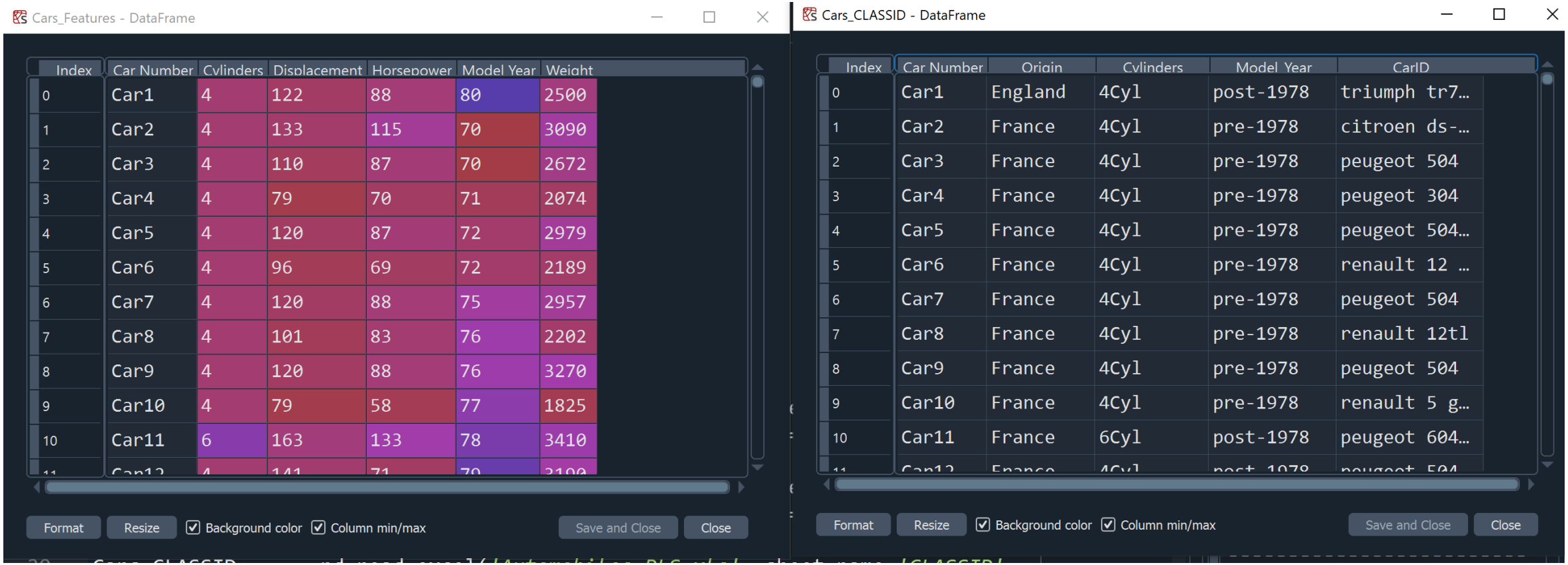
Pyphi – Format of the data

- How to set up the Excel file

```
7   import pandas as pd
8   import numpy as np
9   import pyphi as phi
10  import pyphi_plots as pp
11
12
13  # Load the data from Excel
14  Cars_Features    = pd.read_excel('Automobiles PLS.xls', sheet_name='Features',
15                                   index_col=None, na_values=np.nan)
16
17  Cars_Performance = pd.read_excel('Automobiles PLS.xls', sheet_name='Performance',
18                                   index_col=None, na_values=np.nan)
19
20  Cars_CLASSID     = pd.read_excel('Automobiles PLS.xls', sheet_name='CLASSID',
21                                   index_col=None, na_values=np.nan)
22
```

Pyphi – Format of the data

- The resulting DataFrame



Pyphi – Building models - PCA

- Principal Components Analysis

```
phi.pca(X, A, *, mcs=True, md_algorithm='nipals', force_nipals=False,
shush=False, cross_val=0)
```

Inputs:

X : Either a pandas dataframe, or a Numpy Matrix

A : Number of Principal Components to calculate

Pyphi – Building models - PCA

- Principal Components Analysis

```
phi.pca(X, A, *, mcs=True, md_algorithm='nipals', force_nipals=False,
shush=False, cross_val=0)
```

mcs: Pre-processing flag | 'True': Mean center + autoscale **default** | 'False' : No pre-processing | 'center' : Only center | 'autoscale' : Only autoscale

md_algorithm: Missing Data algorithm to use | 'nipals' : default | 'nlp' Uses non-linear programming approach by Lopez-Negrete et al. J. Chemometrics 2010; 24: 301–311 **and requires pyomo and ipopt**

force_nipals: Algorithm |'True' will use NIPALS. | 'False' if X is complete will use SVD. **default** if not sent. Will default to NIPALS if data has missing values.

shush: Consolor output flag | 'True': suppresses all printed output to console | 'False': **default** if not sent

cross_val: If sent a scalar between 0 and 100 to cross validate element-wise removing *cross_val*% of the data every round | if == 0: Bypass cross-validation **default** if not sent

Pyphi – Building models - PCA

- Principal Components Analysis

Example: Build a simple PCA model with the DataFrame "Cars_Features" using 3 principal components

```
23
24    # Build a PCA model with 3 PC's.
25    pcaobj=phi.pca(Cars_Features,3)
26
```

The calculation returns a dictionary where each field corresponds to an element of the model

pcaobj - Dictionary (14 elements)

Key	Type	Size	
mx	Array of float64	(1, 5)	[[5.4
obsidX	list	406	['Car1'
P	Array of float64	(5, 3)	[[0.48 [0.496
r2x	Array of float64	(3,)	[0.7768
r2xpv	Array of float64	(5, 3)	[[9.080 [9.563
speX	Array of float64	(406, 1)	[[0.004
speX_lim95	float64	1	0.588158
speX_lim99	float64	1	1.063766
sx	Array of float64	(1, 5)	[[1.71
T	Array of float64	(406, 3)	[[-1.514 [-0.131
T2	Array of float64	(406,)	[1.9411
T2_lim95	float64	1	7.858016

Pyphi – Building models - PCA

- Principal Components Analysis

Example: Build a simple PCA model with the DataFrame "Cars_Features" using 3 principal components

```
23
24  # Build a PCA model with 3 PC's.
25  pcaobj=phi.pca(Cars_Features,3)
26
```

Main diagnostics are also sent to the console

```
In [6]: pcaobj=phi.pca(Cars_Features,3)
phi.pca using NIPALS executed on: 2021-12-05 15:59:09.363982
# Iterations for PC #1:  10
# Iterations for PC #2:  8
# Iterations for PC #3:  35
-------------------------------------------------------------
PC #      Eig        R2X       sum(R2X)
PC #1:      3.872    0.777     0.777
PC #2:      0.818    0.164     0.941
PC #3:      0.159    0.032     0.973
-------------------------------------------------------------
```

Pyphi – Building models - PCA

- Principal Components Analysis

Example: Build a simple PCA model with the DataFrame "Cars_Features" using 3 principal components, sending shush = True

```
24  # Build a PCA model with 3 PC's.
25  pcaobj=phi.pca(Cars_Features,3,shush=True)
26
```

Disables the output to console

```
In [10]: pcaobj=phi.pca(Cars_Features,3,shush=True)

In [11]:
```

Pyphi – Building models - PCA

- Principal Components Analysis

Example: Build a simple PCA model with the DataFrame "Cars_Features" using 3 principal components, and cross-validating the model removing 5% of elements each round of cross-validation:

```
28  # Build a PCA model with 3 PC's, cross validating
29  # by elements removing 5% of the data per round
30  pcaobj=phi.pca(Cars_Features,3,cross_val=5)
31
```

Cross validation diagnostics are sent to console

```
In [11]: pcaobj=phi.pca(Cars_Features,3,cross_val=5)
Cross validating PC #1
Cross validating PC #2
Cross validating PC #3
phi.pca using NIPALS and cross validation (5%) executed on: 2021-12-05
17:14:49.831920
------------------------------------------------------------
PC #          Eig      R2X     sum(R2X)      Q2X     sum(Q2X)
PC #1:      3.872    0.777     0.777       0.699     0.699
PC #2:      0.818    0.164     0.941       0.135     0.834
PC #3:      0.159    0.032     0.973       0.105     0.940
------------------------------------------------------------
```

Pyphi – Building models

- Partial Least Squares

```
phi.pls(X, Y, A, *, mcsX=True, mcsY=True, md_algorithm='nipals',
force_nipals=False, shush=False, cross_val=0, cross_val_X=False)
```

Inputs:

X,Y : Either a pandas dataframe, or a Numpy Matrix

A : Number of latent variables to calculate

Pyphi – Building models - PLS

- Partial Least Squares

pls(X, Y, A, *, mcsX=True, mcsY=True, md_algorithm='nipals', force_nipals=True, shush=False, cross_val=0, cross_val_X=False, cca=False)

```
Args:
      X,Y (DataFrame or Numpy) : Training Data
      A (int) : Number of Latent Variables to calculate

   Other Parameters:
     mcsX/mcsY:  'True'      : Will meancenter and autoscale the data *default if not sent*
                 'False'     : No pre-processing
                 'center'    : Will only center
                 'autoscale' : Will only autoscale
      md_algorithm: 'nipals' *default*
                  'nlp'    Uses  algorithm described in Journal of Chemometrics, 28(7), pp.575-584.
      force_nipals: If set to True and if X is complete, will use NIPALS.
                   Otherwise, if X is complete will use SVD.
      shush: If set to True supressess all printed output.
      cross_val: If sent a scalar between 0 and 100, will cross validate
                element wise removing cross_val% of the data every round
                if ==   0:  Bypass cross-validation  *default if not sent*
      cross_val_X: True : Calculates Q2 values for the X and Y matrices
                   False: Cross-validation strictly on Y matrix *default if not sent*
      cca:         True : Calculates covariable space of X with Y (analog to the predictive space in OPLS)
                     "Tcv" and "Pcv" and the covariant scores and loadings if more than one Y, then
                     there will be as many Tcv and Pcv vectors as columns in Y
   Returns:
      A dictionary with all PLS loadings, scores and other diagnostics.
```

Pyphi – Building models - PLS

- Partial Least Squares

- Example, build a simple PLS model between the matrices "Cars_Features' [X] and "Cars_Performance" [Y] using 3 Latent Variables.

```
In [15]: plsobj=phi.pls(Cars_Features,Cars_Performance,3)
phi.pls using NIPALS executed on: 2021-12-05 17:45:29.832840
# Iterations for LV #1:  5
# Iterations for LV #2:  30
# Iterations for LV #3:  7
-------------------------------------------------------------
LV #     Eig       R2X       sum(R2X)   R2Y       sum(R2Y)
LV #1:    3.850    0.776     0.776      0.541     0.541
LV #2:    0.143    0.042     0.818      0.136     0.677
LV #3:    0.750    0.151     0.969      0.023     0.700
-------------------------------------------------------------
```

Pyphi – Building models - PLS

- Partial Least Squares

- Model is a dictionary, every field contains a parameter or diagnostic of the model

```
In [18]: plsobj=phi.pls(Cars_Features,Cars_Performance,3)
phi.pls using NIPALS executed on: 2021-12-05 17:47:56.409799
# Iterations for LV #1:  5
# Iterations for LV #2:  30
# Iterations for LV #3:  7
-------------------------------------------------------------
LV #     Eig       R2X       sum(R2X)   R2Y       sum(R2Y)
LV #1:    3.850    0.776     0.776      0.541     0.541
LV #2:    0.143    0.042     0.818      0.136     0.677
LV #3:    0.750    0.151     0.969      0.023     0.700
-------------------------------------------------------------

In [19]: plsobj.keys()
Out[19]: dict_keys(['T', 'P', 'Q', 'W', 'Ws', 'U', 'r2x', 'r2xpv', 'mx', 'sx', 'r2y',
'r2ypv', 'my', 'sy', 'obsidX', 'varidX', 'obsidY', 'varidY', 'T2', 'T2_lim99', 'T2_lim95',
'speX', 'speX_lim99', 'speX_lim95', 'speY', 'speY_lim99', 'speY_lim95'])
```

Pyphi – Building models - PLS

- Partial Least Squares

- The model is returned in a dictionary object.
- Keys for PLS object are shown in table
  - Keys in PCA object are a subset of these

key	Description
T	T Score
P	P Loadings
Q	Q Loadings
W	W Loadings
Ws	W* Loadings
U	U Score
r2x	R2 global for X space
r2xpv	R2 per variabe for X space
mx	Mean of X matrix
sx	standard dev. of X matrix
r2y	R2 global for Y space
r2ypv	R2 per variabe for Y space
my	Mean of Y matrix
sy	standard dev. of Y matrix
obsidX	Observation's Identifiers X Space (should be == to Y)
varidX	Variable names for X space
obsidY	Observation's Identifiers Y Space (should be == to X)
varidY	Variable names for Y space
T2	Hotelling's T2 for each observation
T2_lim99	Hot. T2 99% Confidence Limit
T2_lim95	Hot. T2 95% Confidence Limit
speX	Squared Prediction Error for each observation - X space
speX_lim99	SPE 99% Confidence Limit - X Space
speX_lim95	SPE95% Confidence Limit - X Space
speY	Squared Prediction Error for each observation - Y space
speY_lim99	SPE 99% Confidence Limit -Y Space
speY_lim95	SPE95% Confidence Limit - Y Space

Pyphi – Building models - PLS

- Partial Least Squares

- Adding cross-validation of the Y space to the calculation (removing 5% of elements per round)

```
In [23]: plsobj=phi.pls(Cars_Features,Cars_Performance,3,cross_val=5)
Cross validating LV #1
Cross validating LV #2
Cross validating LV #3
phi.pls using NIPALS and cross-validation (5%) executed on: 2021-12-05 18:04:57.445941
-------------------------------------------------------------------------
PC #       Eig      R2X     sum(R2X)      R2Y     sum(R2Y)      Q2Y     sum(Q2Y)
PC #1:   3.850    0.776     0.776       0.541     0.541       0.539     0.539
PC #2:   0.143    0.042     0.818       0.136     0.677       0.107     0.646
PC #3:   0.750    0.151     0.969       0.023     0.700       0.053     0.699
-------------------------------------------------------------------------
```

Q2 is based on prediction of Y

Pyphi – Building models - PLS

- Partial Least Squares

- Adding cross-validation of the Y and X space to the calculation (removing 5% of elements per round)

```
In [24]: plsobj=phi.pls(Cars_Features,Cars_Performance,3,cross_val=5,cross_val_X=True)
Cross validating LV #1
Cross validating LV #2
Cross validating LV #3
phi.pls using NIPALS and cross-validation (5%) executed on: 2021-12-05 18:06:21.945074
--------------------------------------------------------------------------------
PC #       Eig      R2X     sum(R2X)      Q2X     sum(Q2X)      R2Y     sum(R2Y)      Q2Y     sum(Q2Y)
PC #1:   3.850    0.776     0.776       0.775     0.775       0.541     0.541       0.538     0.538
PC #2:   0.143    0.042     0.818       0.067     0.842       0.136     0.677       0.102     0.640
PC #3:   0.750    0.151     0.969       0.126     0.969       0.023     0.700       0.058     0.698
--------------------------------------------------------------------------------
```

Q2 for X space

Q2 for Y space

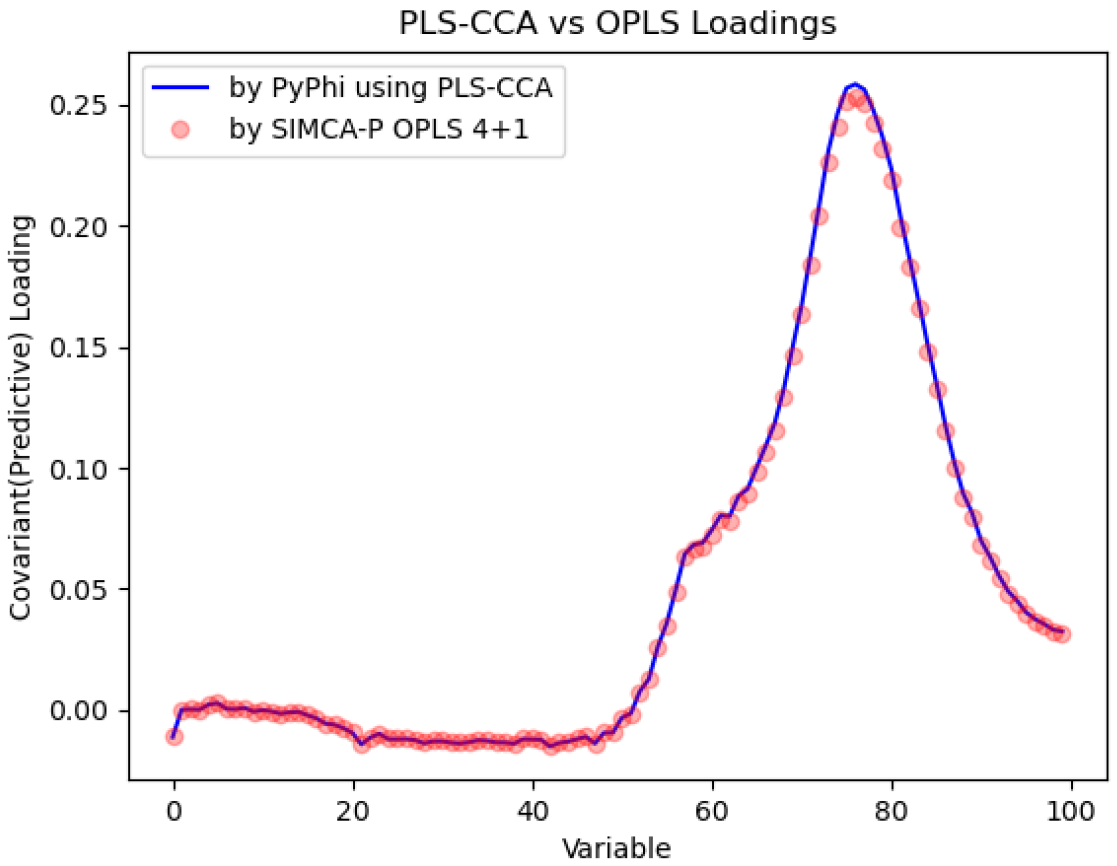
Pyphi – Building models – PLS – CCA (OPLS)

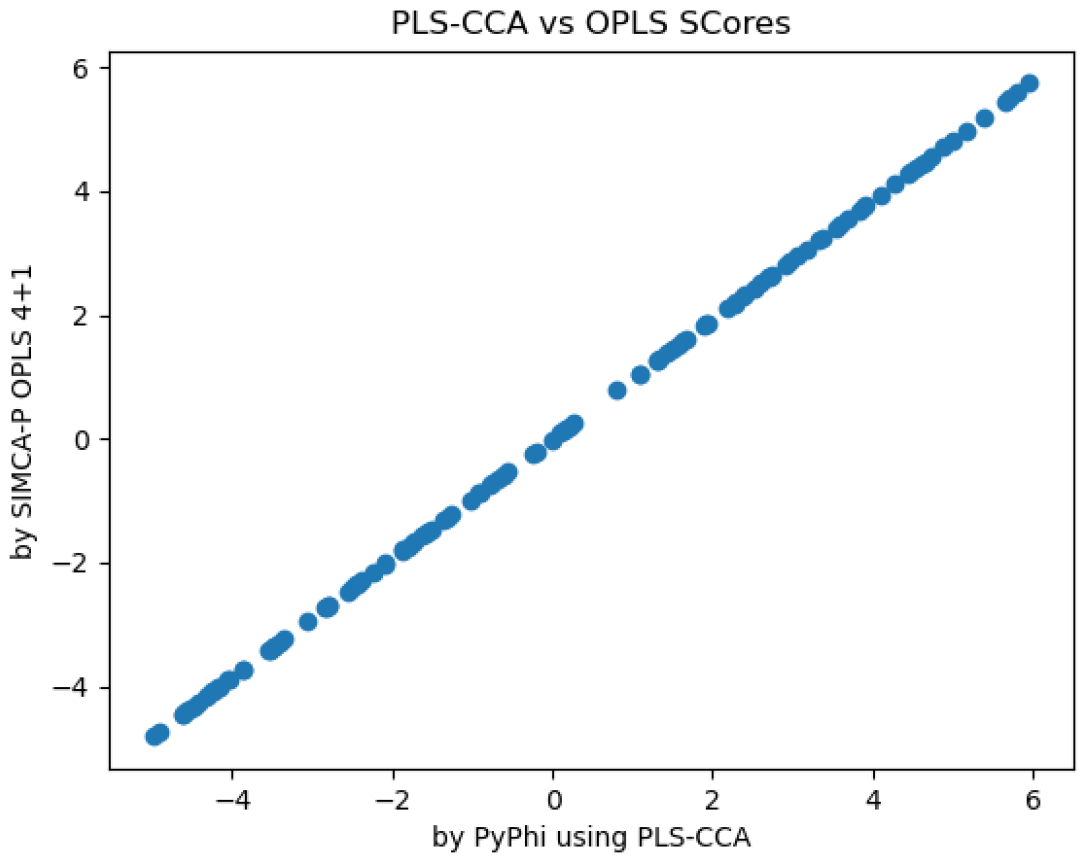
- Partial Least Squares – Canonical Correlation Analysis
  - Seeks the same space as OPLS
    - But instead of filtering the undesired features of X it extracts the space of X that is correlated with Y (disregarding the space of X that explains the X)
  - Use the flag: cca=True
  - The pls object now has three extra parameters:
    - Tcv: The covariate scores
    - Pcv: The covariate Loadings (equivalent to the predictive loadings in OPLS)
    - Wcv: The predictive loadings (used to obtain the covariate scores)

Yu, H. and MacGregor, J.F., 2004. Post processing methods (PLS–CCA): simple alternatives to preprocessing methods (OSC–PLS). *Chemometrics and intelligent laboratory systems*, *73*(2), pp.199-205.

Pyphi – Building models – PLS – CCA (OPLS)

- If the orthogonalization step in OPLS is done appropriately you get the same results as with PLS-CCA





Pyphi – Evaluating new data [Making predictions] -PCA

- Principal Components Analysys

`pca_pred(Xnew, pcaobj)`

Inputs:

Xnew: Matrix with data to be evaluated with model

pcaobj: Dictionary object of a pca model created by phi.pca

Output is a dictionary with fields:

Xhat: Reconstruction (prediction) of X

Tnew: Scores for new data

speX: Squared prediction error per observation

T2: Hotelling's T2 per observation

```
In [29]: pca_calcs = phi.pca_pred(Additional_Cars,pcaobj)

In [30]: pca_calcs.keys()
Out[30]: dict_keys(['Xhat', 'Tnew', 'speX', 'T2'])
```

Pyphi – Evaluating new data [Making predictions]-PLS

- Partial Least Squares

`pls_pred(Xnew, plsobj)`

Inputs:

Xnew: Matrix with data to be evaluated with model

plsobj: Dictionary object of a pls model created by phi.pls

```
In [33]: pls_pred=phi.pls_pred(Additional_Cars,plsobj)

In [34]: pls_pred.keys()
Out[34]: dict_keys(['Yhat', 'Xhat', 'Tnew', 'speX', 'T2'])
```

Output is a dictionary with fields:

Yhat: Predictions of the Y space

Xhat: Reconstruction (prediction) of X

Tnew: Scores for new data

speX: Squared prediction error per observation

T2: Hotelling's T2 per observation

Pyphi – Building Models – Multi-Block PLS

mbpls(XMB, YMB, A, *, mcsX=True, mcsY=True, md_algorithm_='nipals', force_nipals_=False, shush_=False, cross_val_=0, cross_val_X_=False)

Multi-block PLS model using the approach by Westerhuis, J. Chemometrics, 12, 301–321 (1998)

Inputs

XMB : Dictionary or PandasDataFrame

{'BlockName1':block_1_data_pd,

'BlockName2':block_2_data_pd}

YMB : Dictionary or PandasDataFrame

{'BlockName1':block_1_data_pd,

'BlockName2':block_2_data_pd}

All other flags are same as in PLS (slide 19)

Pyphi – Building Models – Multi-Block PLS

- Example on how to prepare the data for a multi-block model:

X is multi-block and Y is not in this case (but it could be)

```
13  x1_data=pd.read_excel('MBDataset.xlsx','X1')
14  x2_data=pd.read_excel('MBDataset.xlsx','X2')
15  x3_data=pd.read_excel('MBDataset.xlsx','X3')
16  x4_data=pd.read_excel('MBDataset.xlsx','X4')
17  x5_data=pd.read_excel('MBDataset.xlsx','X5')
18  x6_data=pd.read_excel('MBDataset.xlsx','X6')
19  y_data=pd.read_excel('MBDataset.xlsx','Y')
20
21  mbdata={'X1':x1_data,
22          'X2':x2_data,
23          'X3':x3_data,
24          'X4':x4_data,
25          'X5':x5_data,
26          'X6':x6_data
27         }
28
29  mbpls_obj=phi.mbpls(mbdata,y_data,2)
```

- Prediction are made with the same routine for pls (pls_pred)

```
preds=phi.pls_pred(mbdata, mbpls_obj)
```

Pyphi – Building Models – LWPLS

lwpls(xnew, loc_par, mvmobj, X, Y, *, shush=False)

LWPLS algorithm as in: International Journal of Pharmaceutics 421 (2011) 269– 274

xnew : Regressor vector to make prediction

loc_par: Localization parameter

mvmobj : PLS model between X and Y

X,Y : Training set used for mvmobj (PLS model) numpy arrays or pandas dataframes

shush : 'True' will silent output

'False' will display output *default if not sent*

Pyphi – Building Models - LPLS

lpls(X, R, Y, A, *, shush=False)

LPLS Algorithm per Muteki et. al Chemom.Intell.Lab.Syst.85(2007) 186 – 194

X = [m x p] Phys. Prop. DataFrame of [materials x mat. properties]

R = [b x m] Blending ratios DataFrame of [blends x materials]

Y = [b x n] Product characteristics DataFrame of [blends x prod. Properties]

First column of all DataFrames are observation identifier

A = Number of components to fit

```
import pandas as pd
import pyphi as phi
import pyphi_plots as pp


#This is how to fit a basic LPLS model and do some plots
X=pd.read_excel('lpls_dataset.xlsx',sheet_name='X')
R=pd.read_excel('lpls_dataset.xlsx',sheet_name='R')
Y=pd.read_excel('lpls_dataset.xlsx',sheet_name='Y')

lpls_obj = phi.lpls(X,R,Y,4)
```

Pyphi – Building Models – JRPLS

jrpls(Xi, Ri, Y, A, *, shush=False)

JRPLS Algorithm per Garcia-Munoz Chemom.Intel.Lab.Syst., 133, pp.49-62.

X = Phys. Prop. dictionary of Dataframes of materials_i x mat. properties

X = {'MatA':df_with_props_for_mat_A (one row per lot of MatA, one col per property),

'MatB':df_with_props_for_mat_B (one row per lot of MatB, one col per property)}

R = Blending ratios dictionary of Dataframes of blends x materials_i

R = {'MatA': df_with_ratios_of_lots_of_A_used_per_blend,

'MatB': df_with_ratios_of_lots_of_B_used_per_blend,

}

Rows of X[i] must correspond to Columns of R[i] (use routine reconcile_rows_to_columns)

Y = [b x n] Product characteristics dataframe of blends x prod. properties

First column of all dataframes is the observation identifier all observation ids in R and Y must match

use routine reconcile_rows

Refer to example script jrpls_tpls_example.py

Pyphi – Building Models – TPLS

tpls(Xi, Ri, Z, Y, A, *, shush=False)

TPLS Algorithm per Garcia-Munoz Chemom.Intel.Lab.Syst., 133, pp.49-62.

X = Phys. Prop. dictionary of DataFrames of materials_i x mat. properties

X = {'MatA':df_with_props_for_mat_A (one row per lot of MatA, one col per property),
'MatB':df_with_props_for_mat_B (one row per lot of MatB, one col per property)}

R = Blending ratios dictionary of Dataframes of blends x materials_i

R = {'MatA': df_with_ratios_of_lots_of_A_used_per_blend,
'MatB': df_with_ratios_of_lots_of_B_used_per_blend,
}

Rows of X[i] must correspond to Columns of R[i]

Y = [b x n] Product characteristics DataFrame of blends x prod. properties

Z = [b x p] Process conditions DataFrame of blends x process variables

first column of all dataframes is the observation identifier

Refer to example script jrpls_tpls_example.py

Pyphi – Building Models – JRPLS and TPLS preparing data

parse_materials(filename, sheetname)

Routine to parse out compositions from linear table

This reads an excel file with four columns:

'Finished Product Lot' 'Material Lot' 'Ratio or Quantity' 'Material'

where the usage per batch of finished product is recorded. e.g.

'Finished Product Lot'	'Material Lot'	'Ratio or Quantity'	'Material'
A001	A	0.75	Drug
A001	B	0.25	Drug
A001	Z	1.0	Excipient

Returns:

JR = Joint R matrix of material consumption, list of dataframes

materials_used = Names of materials

Refer to example script jrpls_tpls_example.py

Pyphi-Making predictions – JRPLS and TPLS

- Information about material, lot to consider and blending ratio is given through a dictionary, e.g.:

```
rnew={
    'MAT1':      [('A0129',0.557949425 ),('A0130',0.442050575 )],
    'MAT2':   [('Lac0003',1)],
    'MAT3':      [('TLC018', 1) ],
    'MAT4':      [('M0012',  1)  ],
    'MAT5':[('CS0017', 1) ]
    }
```

jrpreds=**phi.jrpls_pred(rnew,jrplsobj)**

tpreds=**phi.tpls_pred(rnew,znew,tplsobj)** znew is a numpy array with the process conditions

- rnew={
- 'API': [('A0129',0.557949425),('A0130',0.442050575)],
- 'Lactose': [('Lac0003',1)],
- 'Talc': [('TLC018', 1)],
- 'MgSt': [('M0012', 1)],
- 'Conf. Sugar':[('CS0017', 1)]
- }
- znew=process[process['LotID']=='L001']
- znew=znew.values.reshape(-1)[1:].astype(float)

Pyphi – Additional useful routines

- Data cleaning, preprocessing

phi.clean_low_variances – routine to remove columns of small variance in a matrix [recommended]
phi.clean_empty_rows – routine to remove rows of missinmg data (np.nan) in a matrix
phi.snv – Preprocess spectra using Standard Normal Variate
phi.savgol- Preprocess spectra using savitzy golay derivatives
phi.hott2 – Numerical calculation of the Hotelling's T2
phi.spe – Numerical calculation of SPE
phi.cat_2_matrix - - Convert categorical data into binary matrices for regression

Help can be obtained for any of the above using the command 'help(module)'

phi.mbpls – Routine to build Multi-Block PLS models
phi.lwpls – Routine to build predictions using the LWPLS approac

e.g.

```
In [77]: help (phi.clean_low_variances)
Help on function clean_low_variances in module pyphi:

clean_low_variances(X, *, shush=False)
     Input:
         X: Matrix to be cleaned for columns of low variance
         shush: 'True' disables output to console
    Returns:
    X_clean:  Matrix without low variance columns
    cols_removed:  Columns removed
```

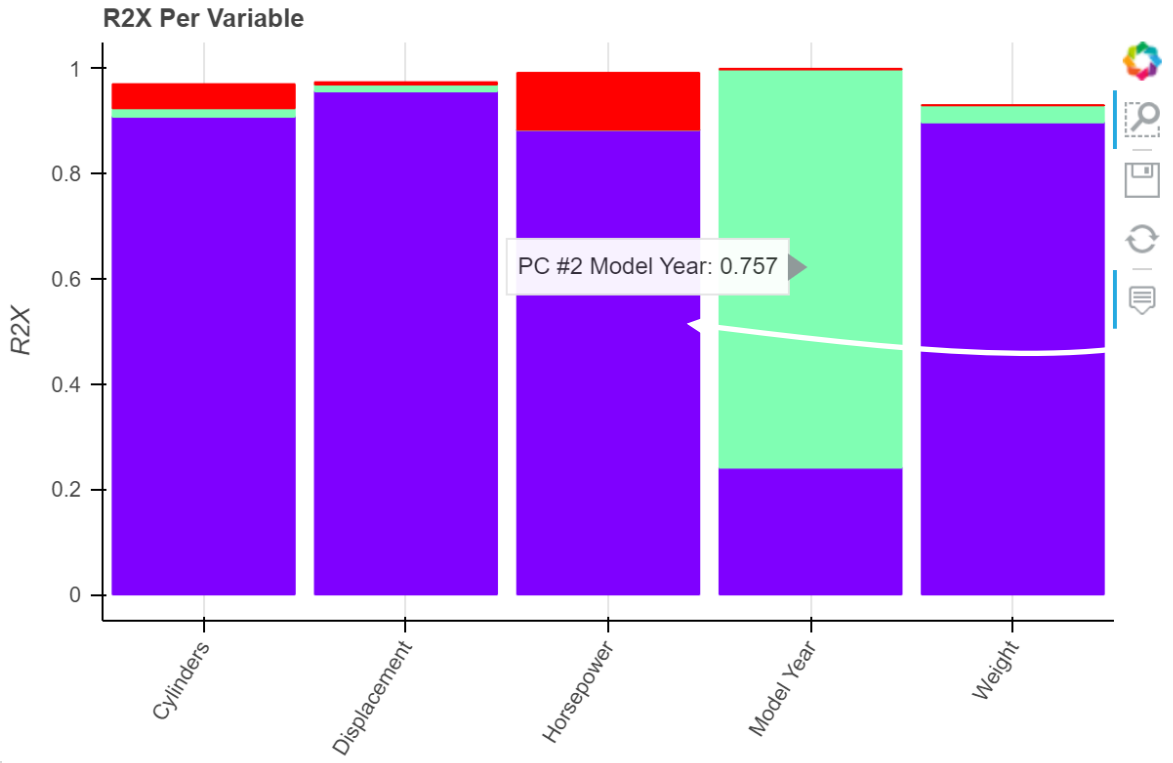
Pyphi_plot – Plotting diagnostics – The R2 per variable

- The R2 diagnostic per variable per dimension in the model

```
r2pv(mvmobj, *, plotwidth=600, plotheight=400)
```

Hoover mouse pointer for detail

```
In [39]: pp.r2pv(pcaobj)
```



Plot is shown in the default web-browser

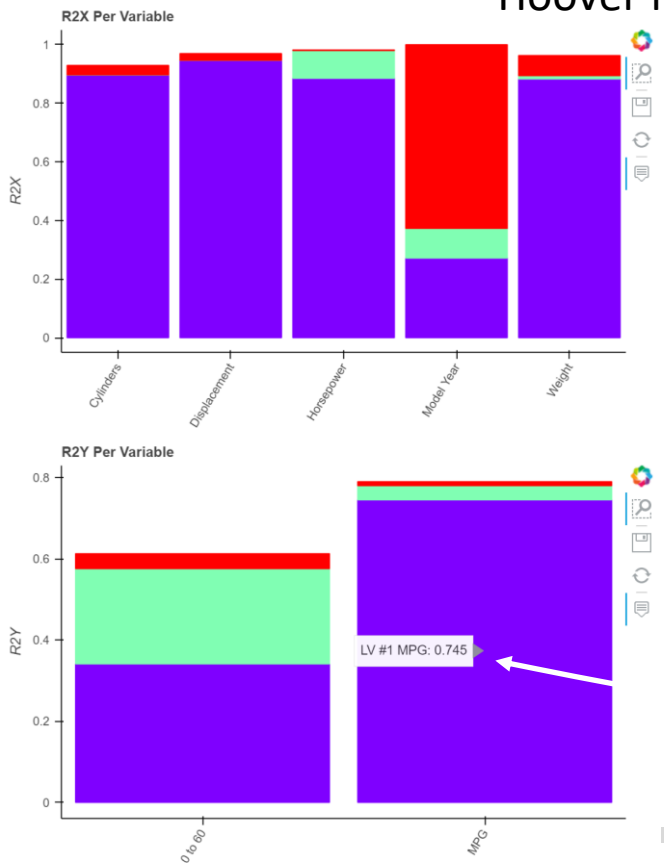
Pyphi_plot – Plotting diagnostics – The R2 per variable

- The R2 diagnostic per variable per dimension in the model

```
r2pv(mvmobj, *, plotwidth=600, plotheight=400)
```

```
In [40]: pp.r2pv(plsobj)
```

Hoover mouse pointer for detail



Plot is shown in the default web-browser

Pyphi_plot – Plotting diagnostics – score lines

- Scores

```
score_line(mvmobj, dim, *, CLASSID=False, colorby=False, Xnew=False, add_ci=False,
add_labels=False, add_legend=True, plotline=True, plotwidth=600, plotheight=600)
```

mvmobj: Either a PCA or a PLS dictionary created wth phi.pca or phi.pls

dim: Dimension to plot (# of PC or LV)

Simplest use, for example to make a line plot of the values for the scores corresponding to the 1st principal component in model object `pcaobj`

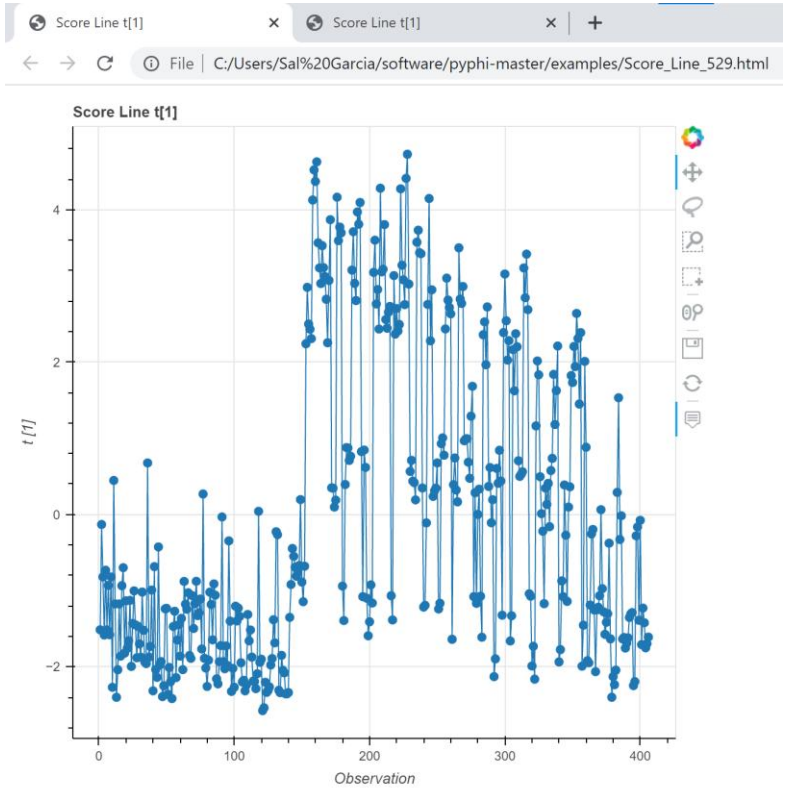
```
In [35]: pp.score_line(pcaobj,1)
```

Plot is shown in the default web-browser

Pyphi_plot – Plotting diagnostics – score lines

- Scores

```
score_line(mvmobj, dim, *, CLASSID=False, colorby=False, Xnew=False, add_ci=False,
add_labels=False, add_legend=True, plotline=True, plotwidth=600, plotheight=600)
```



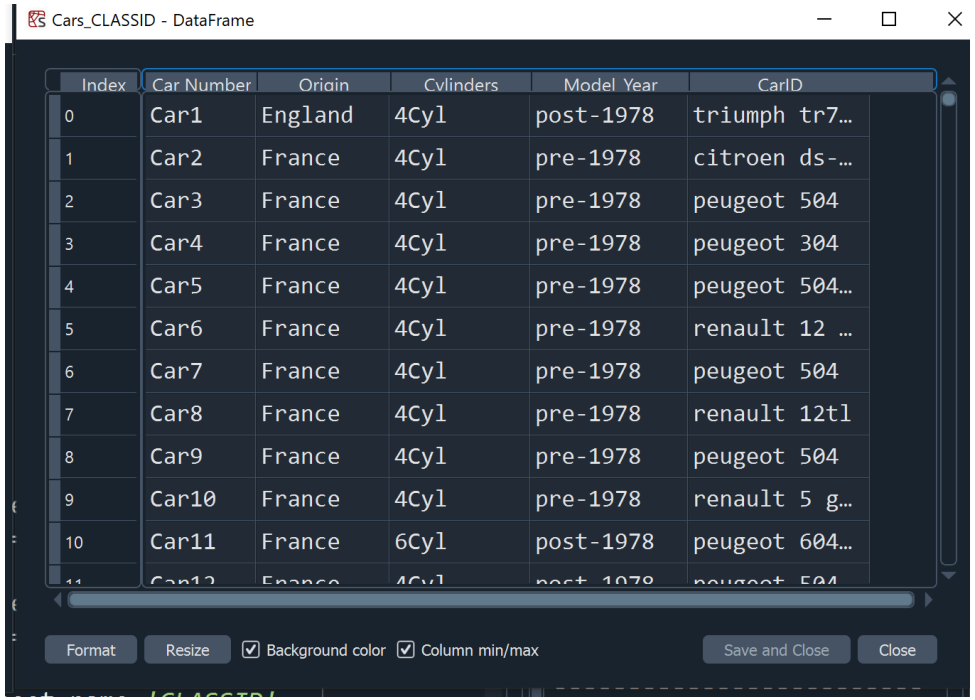
Pyphi_plot – Color coding with categories

- Scores

```
score_line(mvmobj, dim, *, CLASSID=False, colorby=False, Xnew=False, add_ci=False,
add_labels=False, add_legend=True, plotline=True, plotwidth=600, plotheight=600)
```

CLASSID: A matrix with categorical data, e.g.

colorby: Criteria to plot by (e.g. 'Origin')



Pyphi_plot – Color coding with categories

- Scores

```
score_line(mvmobj, dim, *, CLASSID=False, colorby=False, Xnew=False,
add_ci=False, add_labels=False, add_legend=True, plotline=True, plotwidth=600,
plotheight=600)
```

CLASSID: A matrix with categorical data, e.g.

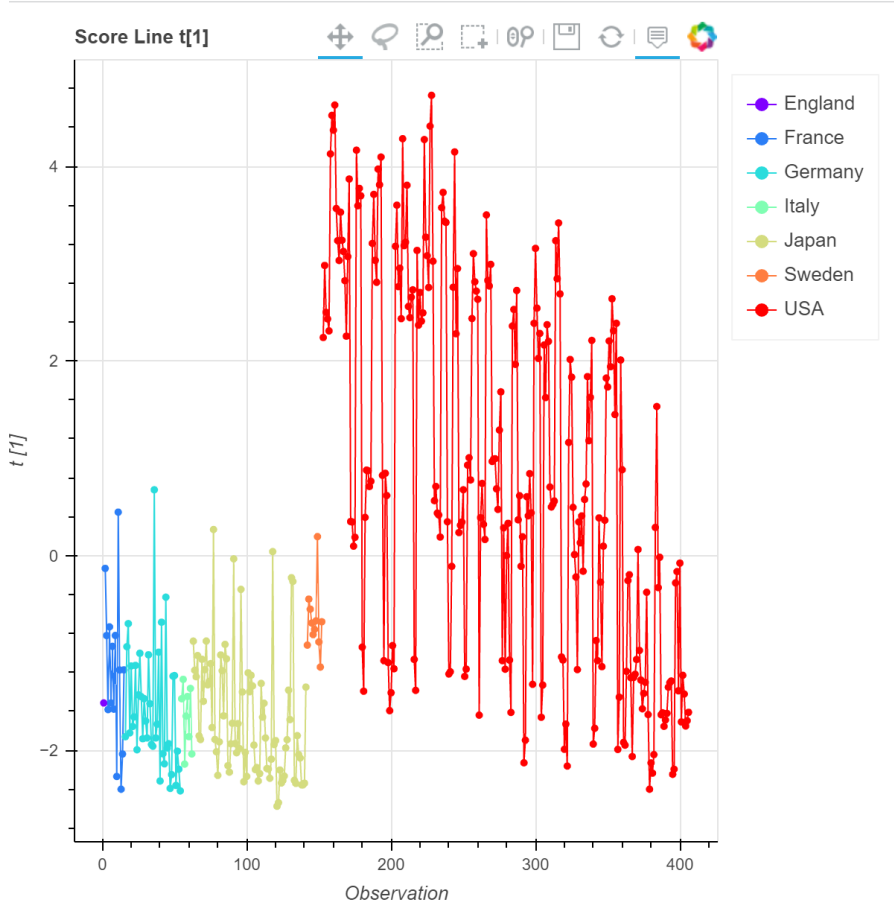
colorby: Criteria to plot by (e.g. 'Origin')

```
In [36]: pp.score_line(pcaobj,1,CLASSID=Cars_CLASSID,colorby='Origin')
```

Pyphi_plot – Color coding with categories

- Scores

```
In [36]: pp.score_line(pcaobj,1,CLASSID=Cars_CLASSID,colorby='Origin')
```



Pyphi_plot – Plotting diagnostics – score lines

- Scores

```
score_line(mvmobj, dim, *, CLASSID=False, colorby=False, Xnew=False,
add_ci=False, add_labels=False, add_legend=True, plotline=True,
plotwidth=600, plotheight=600)
```

CLASSID: A matrix with categorical data, e.g.

colorby: Criteria to plot by (e.g. 'Origin')

Xnew: Matrix of dataset different from training set, routine will first evaluate the scores for new data and then will plot it.

add_ci: "True" will add 95% and 99% confidence intervals to the plot

add_labels: "True" add a lablel with the observation IDs to evey point

add_legend: Adds the categorical legend

plotline: "False" will only user markers with no line

Plotwidth/plotheight: dimensions of the plot in pixels

Pyphi_plot – Plotting diagnostics – score scatters

- Scores

```
score_scatter(mvmobj, xydim, *, CLASSID=False, colorby=False, Xnew=False,
add_ci=False, add_labels=False, add_legend=True, plotwidth=600, plotheight=600)
```

Arguments are the same as in score_line, with one exeption:

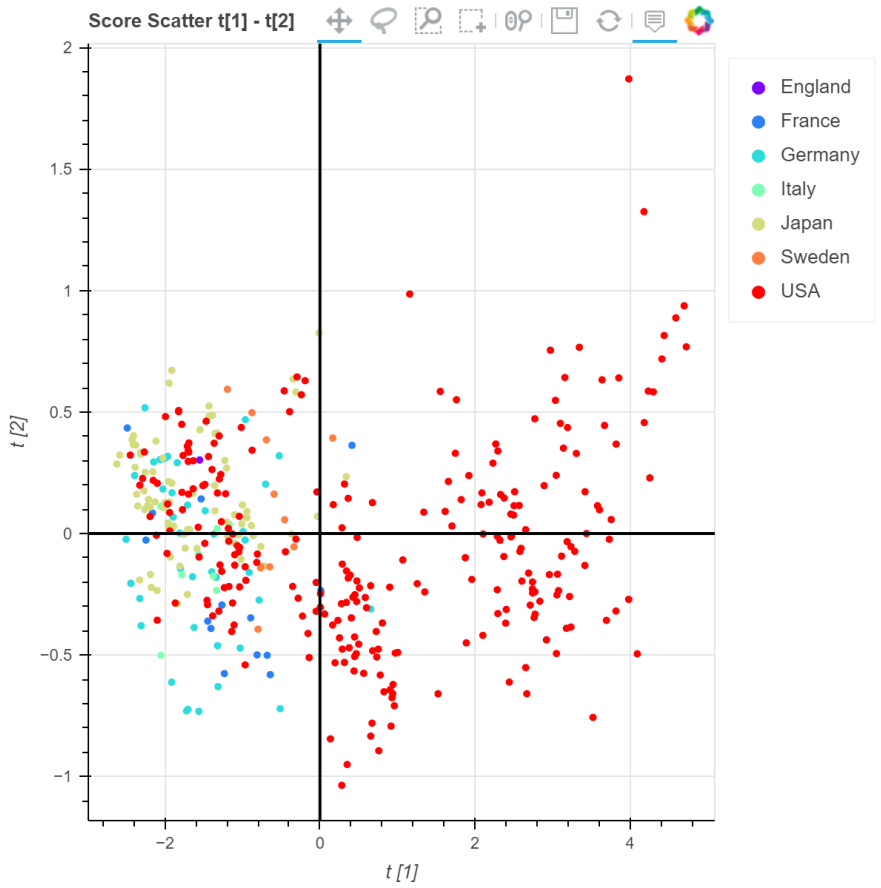
xydim: List of dimensions to be plotted in XY plot, e.g. to plot the values of the scores for the 1st LV against the values of the scores for the 2nd LV in PLS model the argument would be [1,2]

```
In [38]: pp.score_scatter(plsobj,[1,2],CLASSID=Cars_CLASSID,colorby='Origin')
```

Pyphi_plot – Plotting diagnostics – score scatters

- Scores

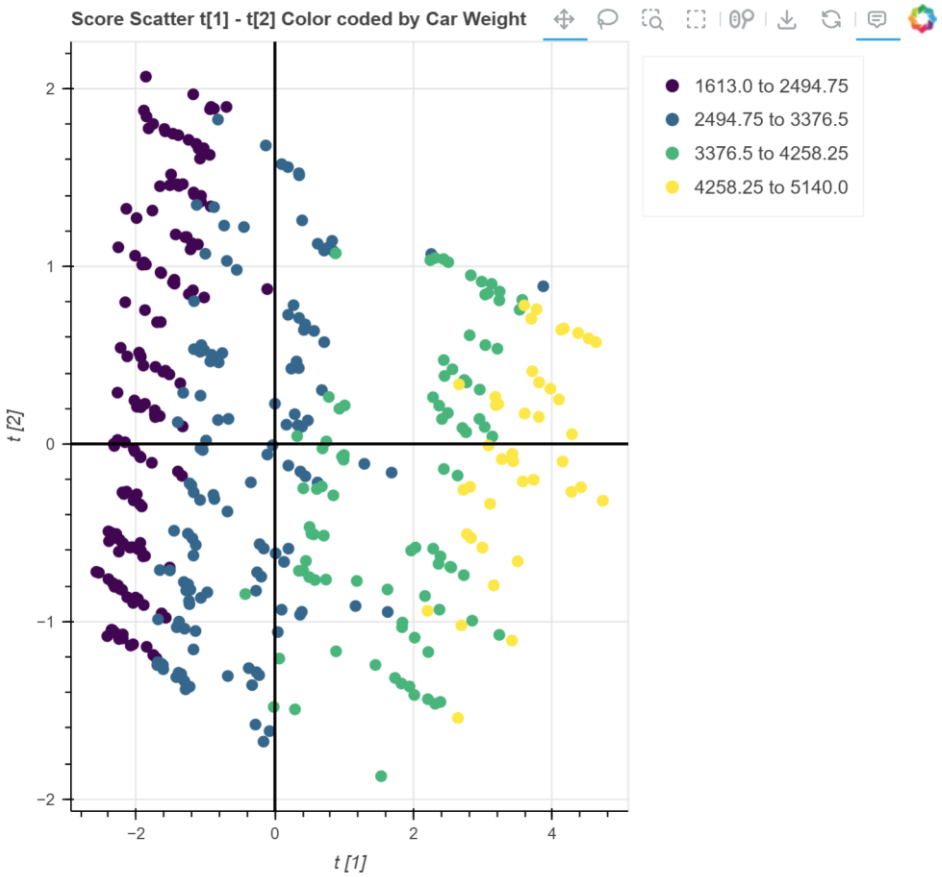
```
In [38]: pp.score_scatter(plsobj,[1,2],CLASSID=Cars_CLASSID,colorby='Origin')
```



Pyphi_plot – Plotting diagnostics – score scatters

```
In [6]: #t1 vs t2 color coded by Car Wegith using 4 bins to group
   ...: pp.score_scatter(pcaobj,[1,2],CLASSID=Cars_Features,colorby='Weight',nbins=4,
   ...:                  addtitle='Color coded by Car Weight')
```

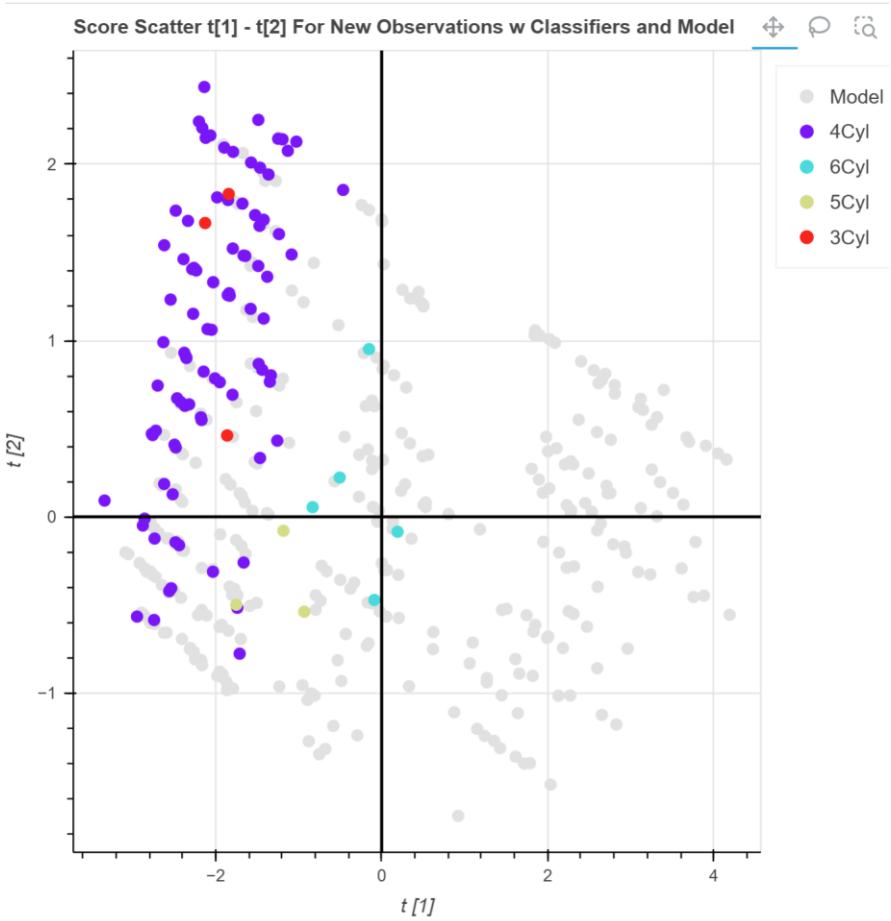
- Coloring can also be done based on a numeric value; the number of bins needs to be established to determine grouping of observations.



Pyphi_plot – Plotting diagnostics – score scatters

```
pp.score_scatter(pcaobj,[1,2],Xnew=Cars_Features_new, CLASSID=Cars_CLASSID_new,
                 colorby='Cylinders',include_model=True,
                 addtitle='For New Observations w Classifiers and Model')
```

- When plotting scores for observations that were not part of the original model, one can also include the scores for the model with the flag : include_model=True



Pyphi_plot – Plotting diagnostics - Loadings

- Loadings

Three options:

loadings(mvmobj, *, plotwidth=600, xgrid=False)

Column plots of all loadings for the model

weighted_loadings(mvmobj, *, plotwidth=600, xgrid=False)

Column plots of the loadings weighted by the R2 per varible per compoment

loadings_map(mvmobj, dims, *, plotwidth=600)

scatter plot of the loadings for a model

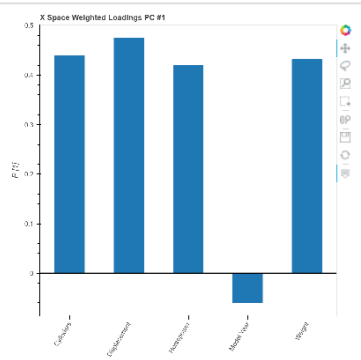
dims: What dimensions to plot in xaxis and yaxis, e.g. [1,2] will plot loadings for 1st LV vs loadings for 2nd LV

Pyphi_plot – Plotting diagnostics - Loadings

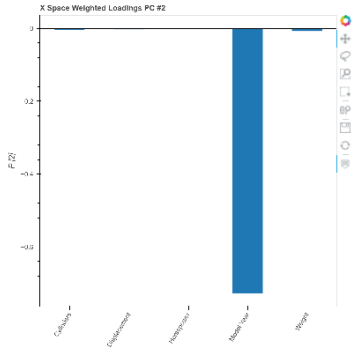
- Loadings

```
In [42]: pp.weighted_loadings(pcaobj)
```

Weighted loadings for 1st PC



Weighted loadings for 2nd PC



Weighted loadings for 3rd PC

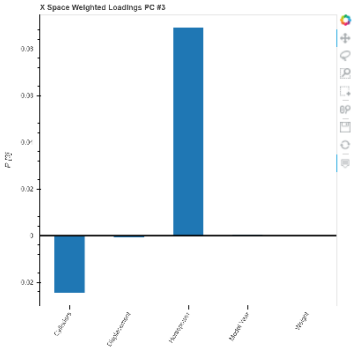
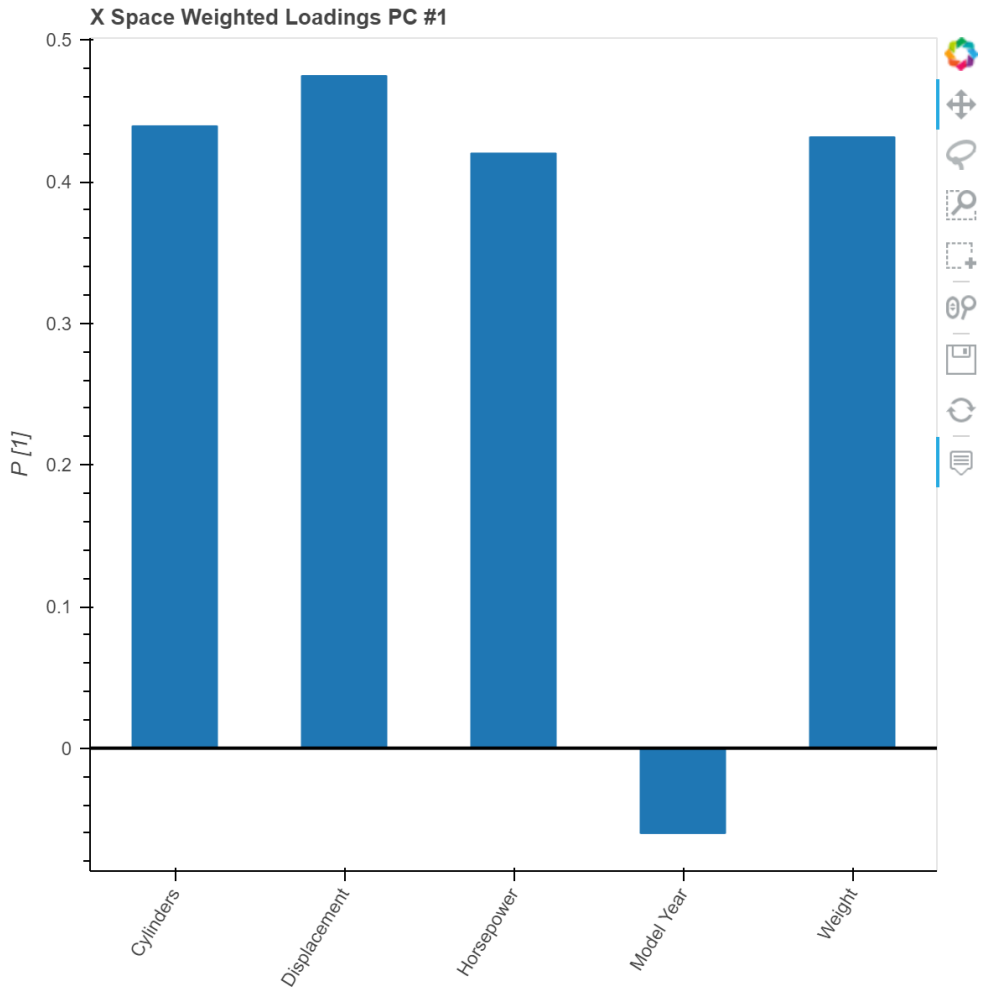


Figure is zoomed out for illustration

Pyphi_plot – Plotting diagnostics - Loadings

- Loadings

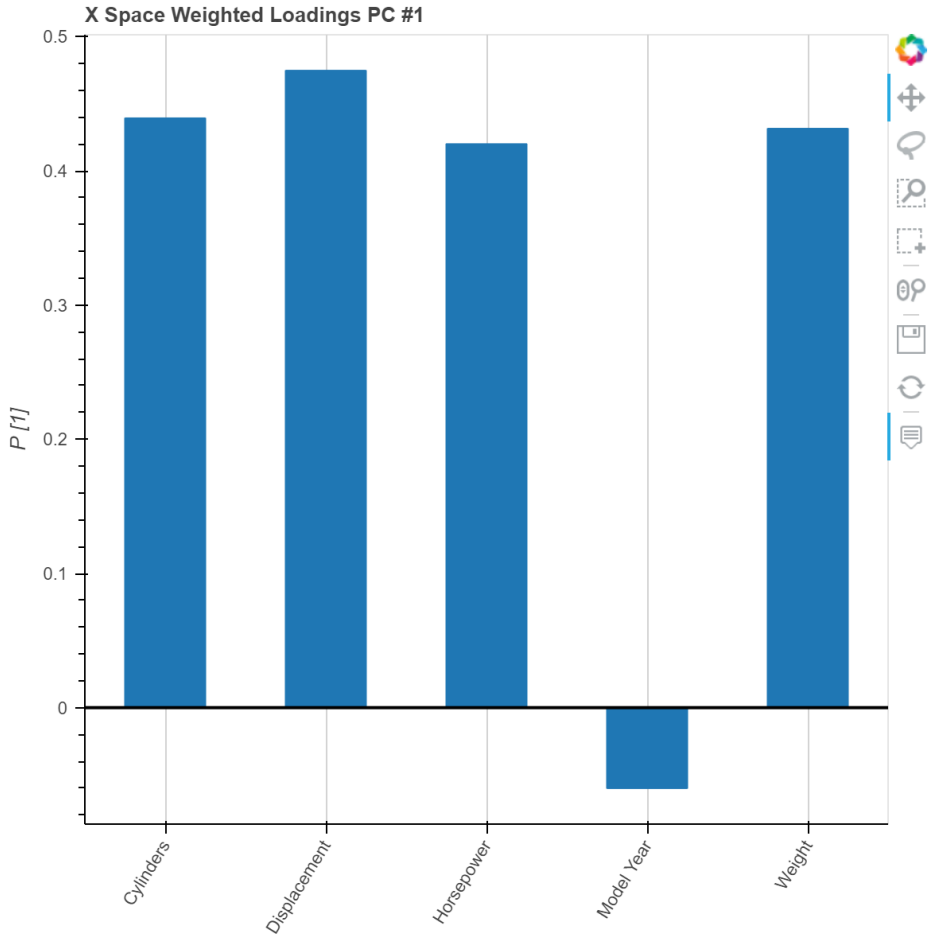
```
In [42]: pp.weighted_loadings(pcaobj)
```



Pyphi_plot – Plotting diagnostics - Loadings

- Loadings

```
In [43]: pp.weighted_loadings(pcaobj,xgrid=True)
```

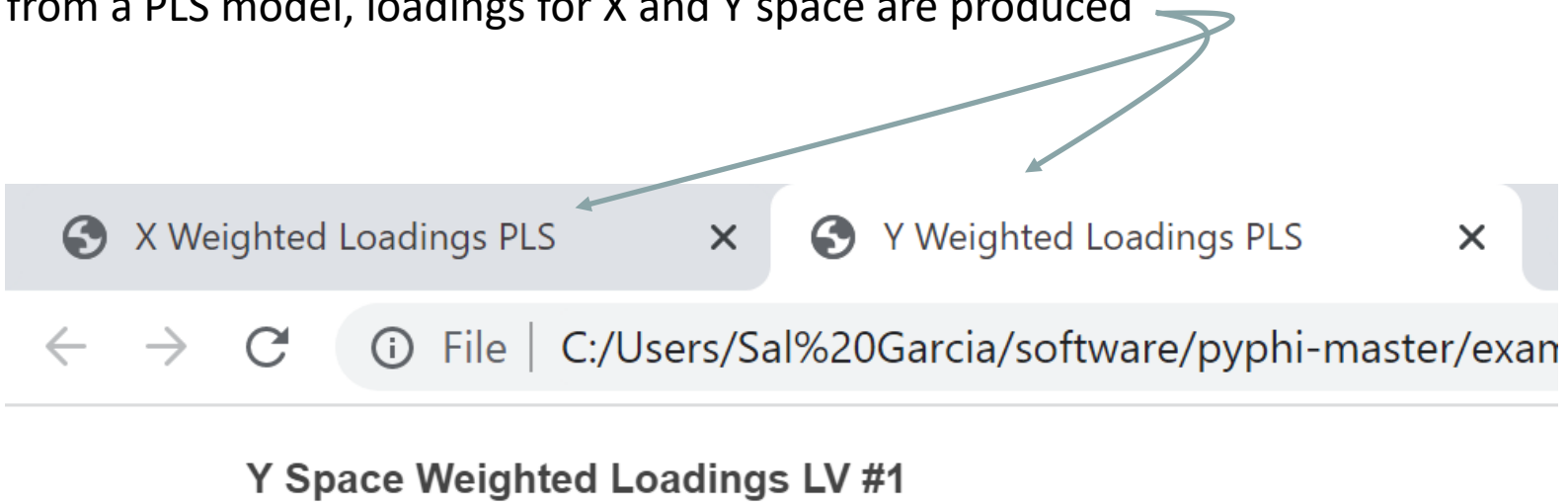


Pyphi_plot – Plotting diagnostics - Loadings

- Loadings

```
In [44]: pp.weighted_loadings(plsobj,xgrid=True)
```

When plotting loadings from a PLS model, loadings for X and Y space are produced

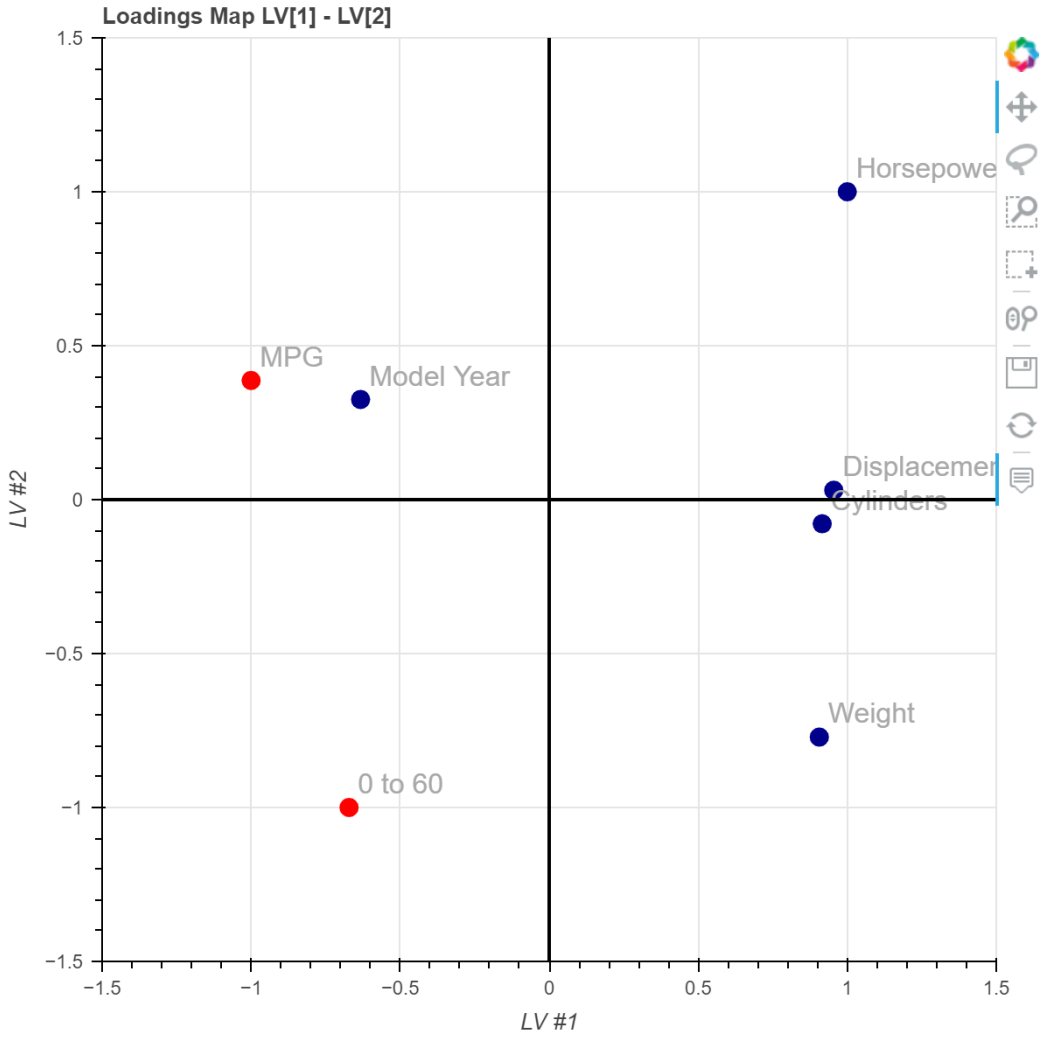


Pyphi_plot – Plotting diagnostics - Loadings

- Loadings

```
In [45]: pp.loadings_map(plsobj,[1,2])
```

When plotting loadings from a PLS model, loadings for X and Y space are overlayed



Pyphi_plot – Plotting diagnostics – Hotelling's T^2 and SPE

- Global diagnostics: Hotelling's T^2 and SPE

```
diagnostics(mvmobj, *, Xnew=False, Ynew=False, score_plot_xydim=False,
plotwidth=600, ht2_logscale=False, spe_logscale=False)
```

Single routine produces HT2 and SPE

Inputs:
mvmobj: Model object produced by either phi.pca or phi.pls

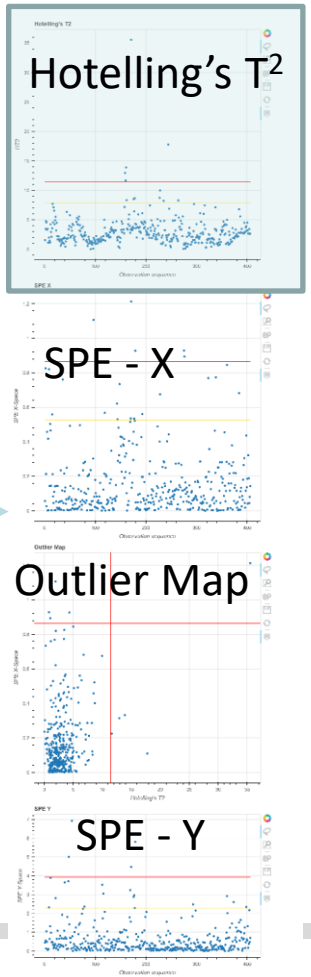
Optional inputs:
Xnew: X Data set different from training set [if diagnostics are needed for data different from training data set] – used in Hotelling's T2 and SPE X
Ynew: Y Data set different from training set [if diagnostics are needed for data different from training data set] - used for SPE Y
score_plot_xydim: list of dimensions (e.g.[1,2]), if sent it adds a score scatter plot at the bottom of the page
ht2_logscale/ spe_logscale: if 'True' will apply log10 to plots

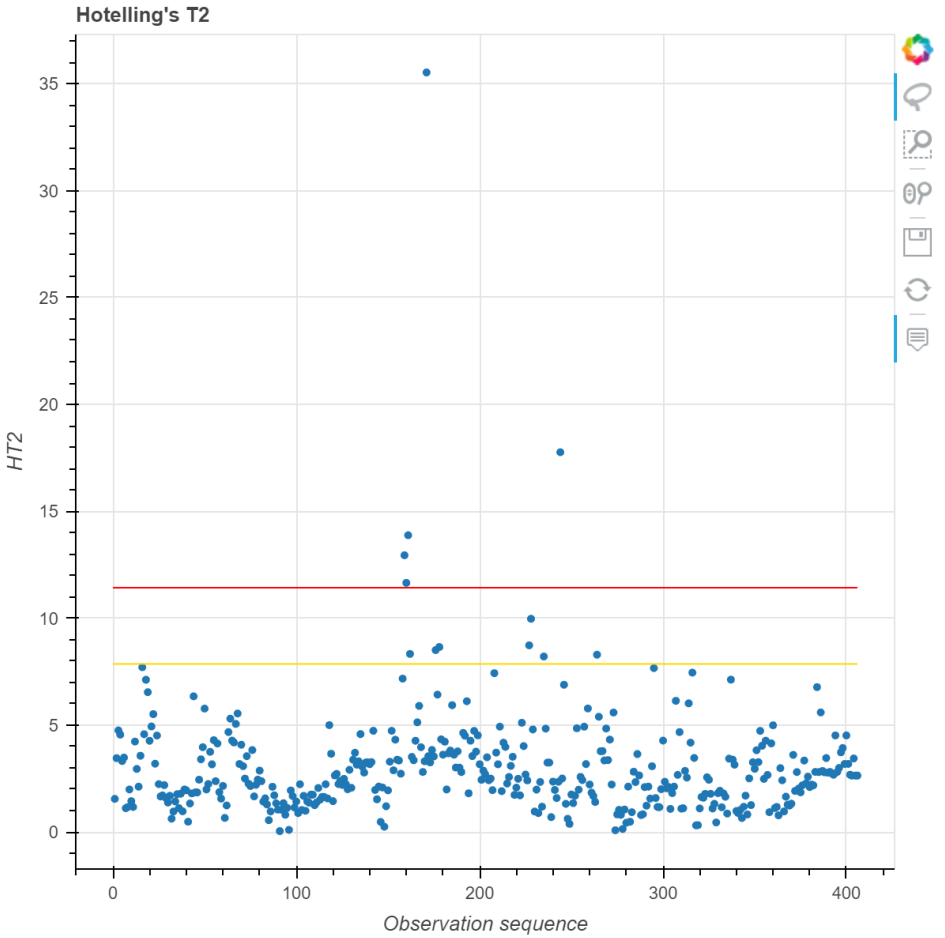
Pyphi_plot – Plotting diagnostics – Hotelling's T^2 and SPE

- Global diagnostics: Hotelling's T^2 and SPE

```
In [47]: pp.diagnostics(plsobj)
```

Output shown in default web browser





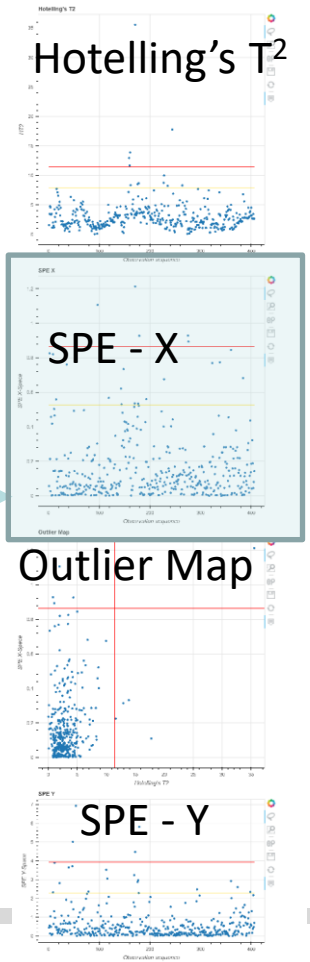
Zoomed out for illustration

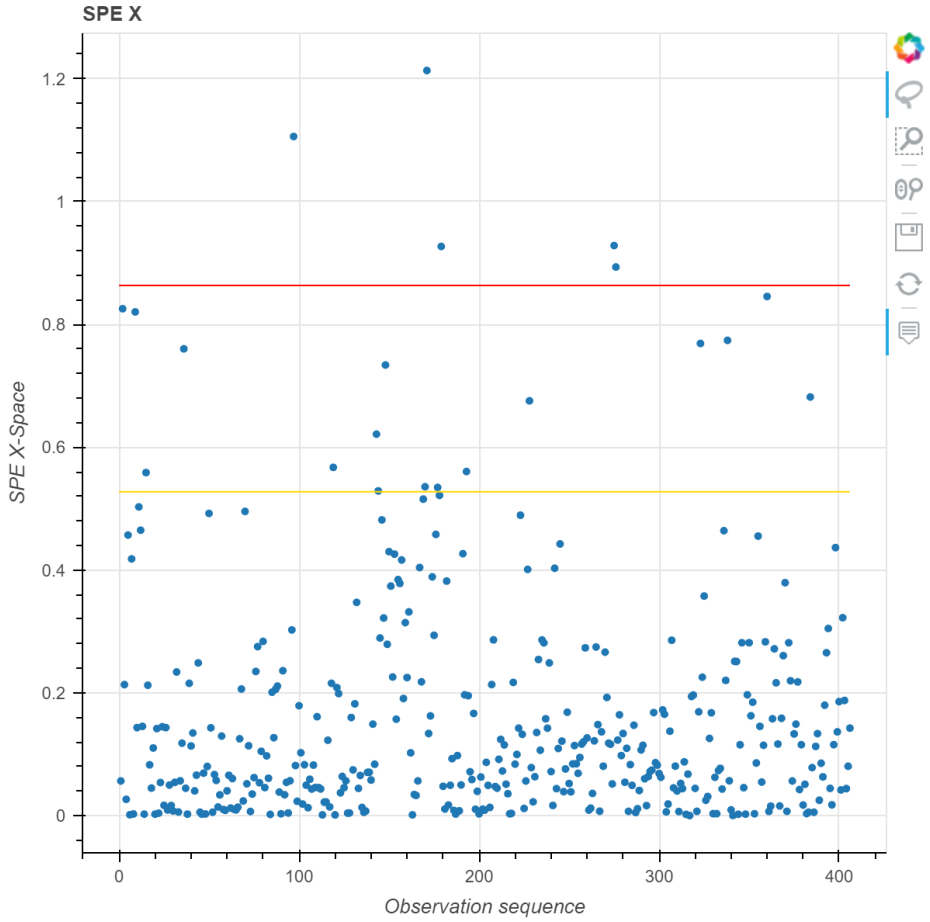
Pyphi_plot – Plotting diagnostics – Hotelling's T^2 and SPE

- Global diagnostics: Hotelling's T^2 and SPE

```
In [47]: pp.diagnostics(plsobj)
```

Output shown in default web browser





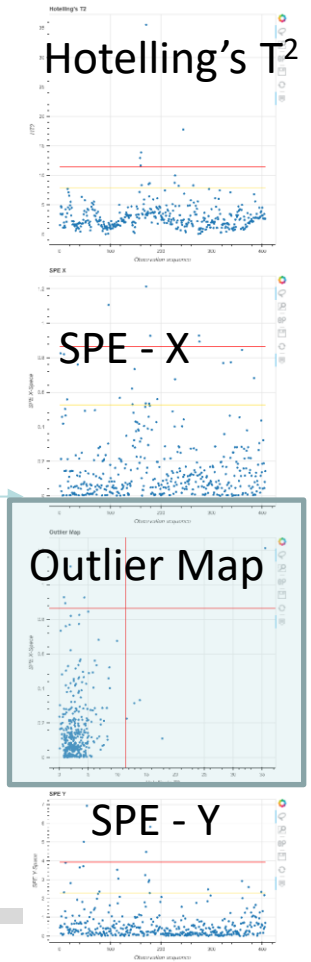
Zoomed out for illustration

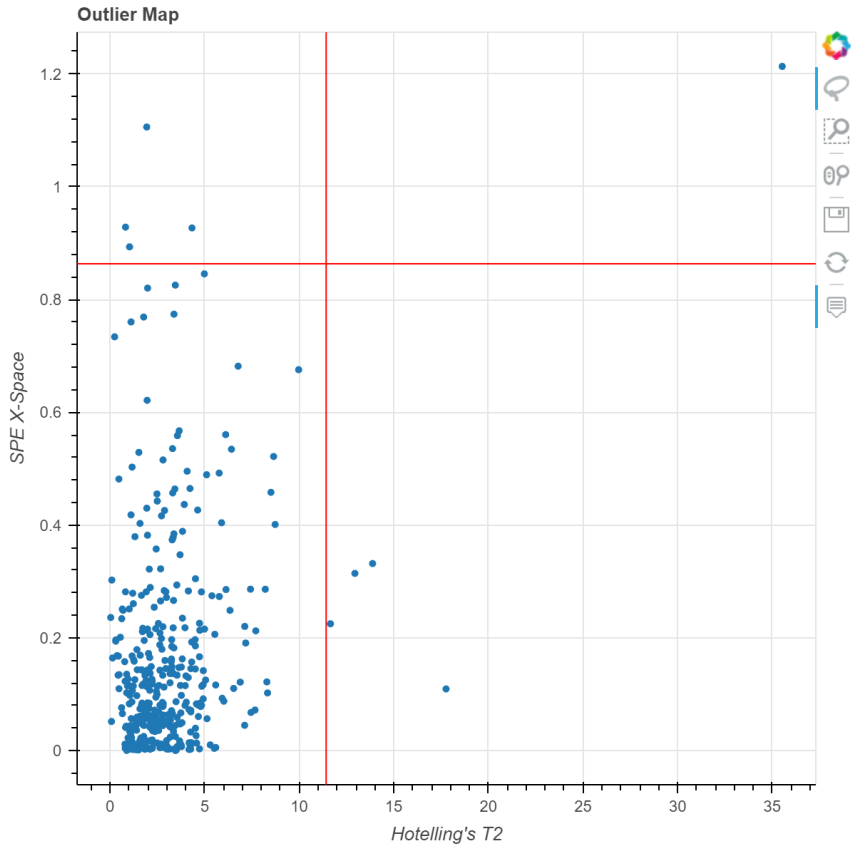
Pyphi_plot – Plotting diagnostics – Hotelling's T^2 and SPE

- Global diagnostics: Hotelling's T^2 and SPE

```
In [47]: pp.diagnostics(plsobj)
```

Output shown in default web browser





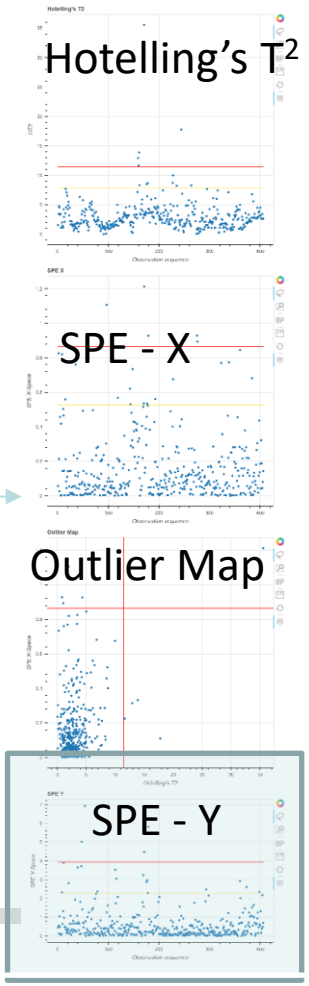
Zoomed out for illustration

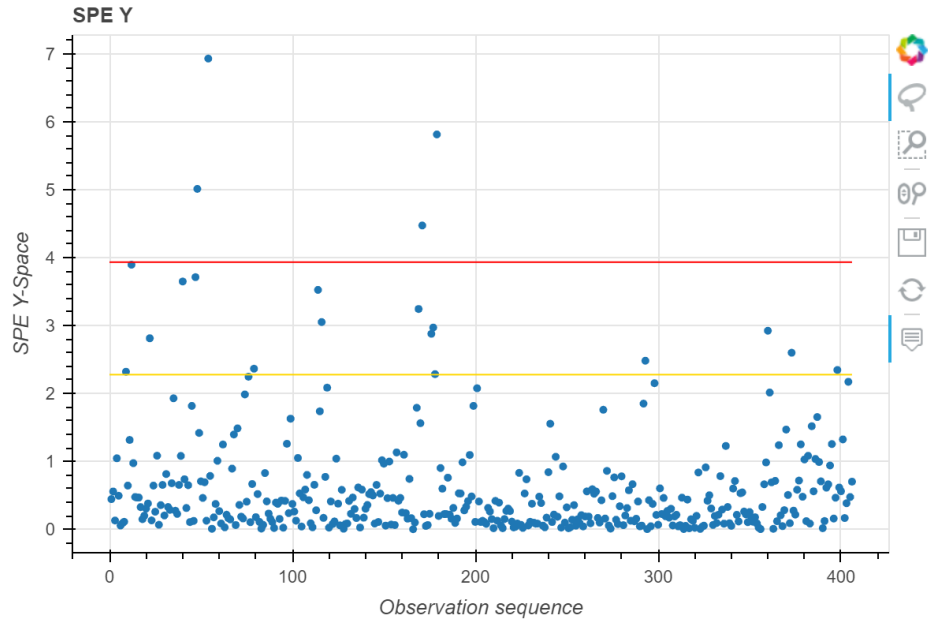
Pyphi_plot – Plotting diagnostics – Hotelling's T^2 and SPE

- Global diagnostics: Hotelling's T^2 and SPE

```
In [47]: pp.diagnostics(plsobj)
```

Output shown in default web browser





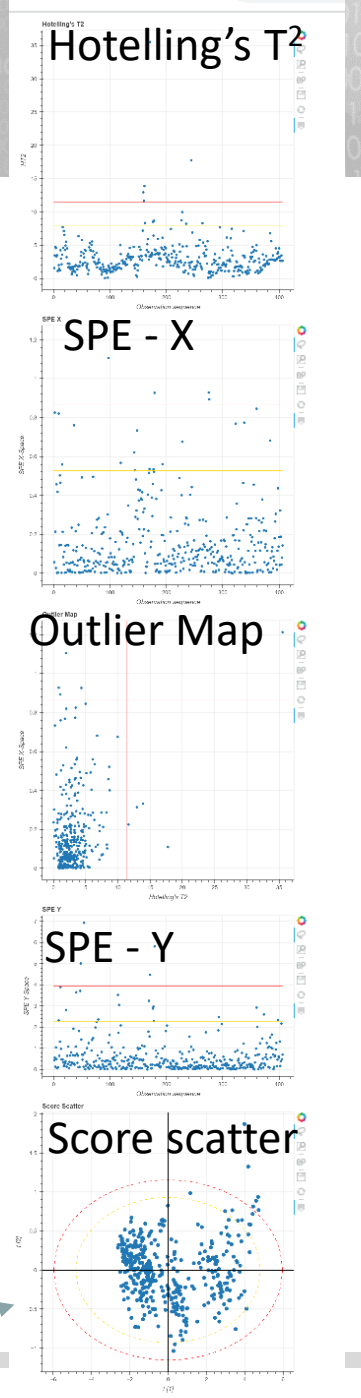
Zoomed out for illustration

Pyphi_plot – Plotting diagnostics – Hotelling's T^2 and SPE

- Global diagnostics: Hotelling's T^2 and SPE

```
In [48]: pp.diagnostics(plsobj, score_plot_xydim=[1,2])
```

Adds a score scatter at the bottom
In this case plotting 1st vs 2nd PC's



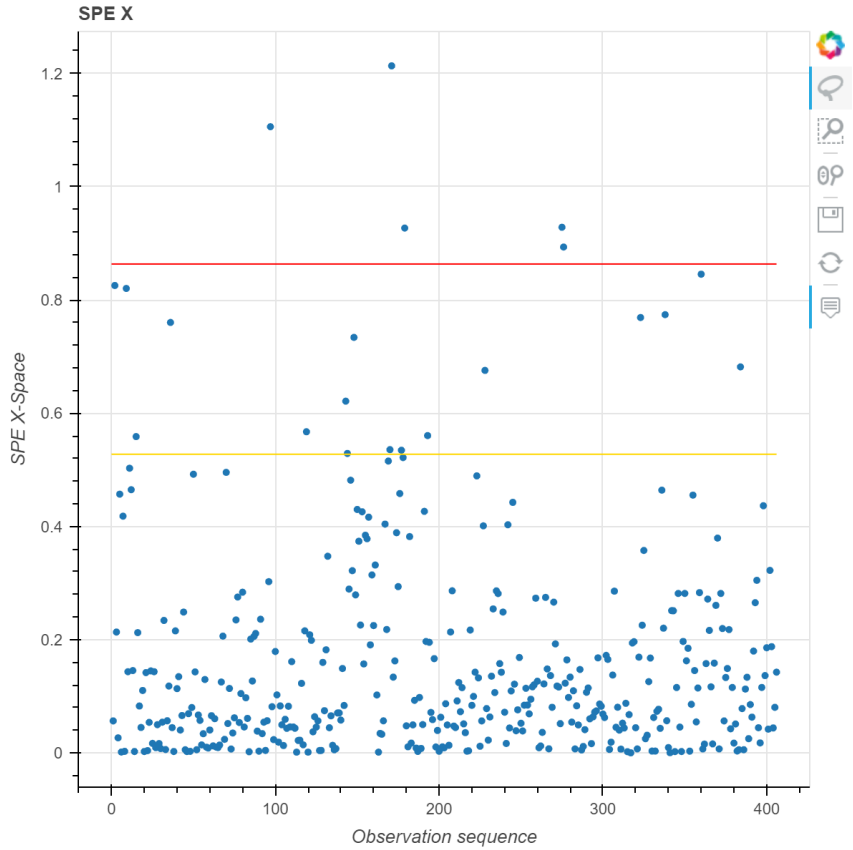
Pyphi_plot – Plotting diagnostics – Hotelling's T^2 and SPE

- Global diagnostics: Hotelling's T^2 and SPE

```
In [48]: pp.diagnostics(plsobj, score_plot_xydim=[1,2])
```

Adds a score scatter at the bottom
In this case plotting 1st vs 2nd PC's

Why? To visualize where observations fall in different diagnostics **plots are linked**

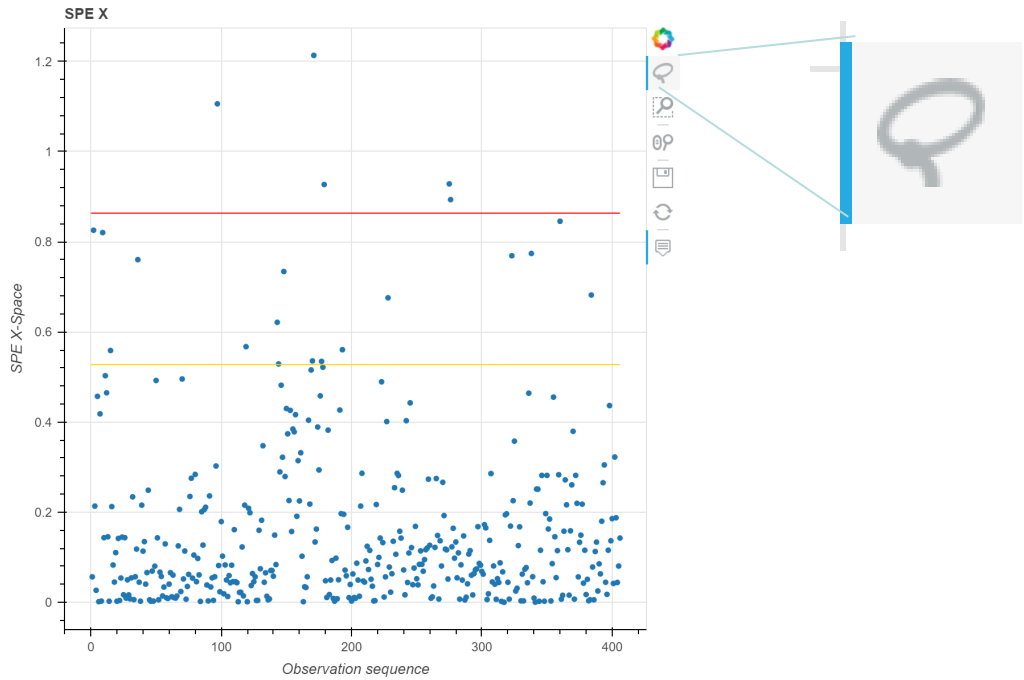


Pyphi_plot – Plotting diagnostics – Hotelling's T^2 and SPE

- Global diagnostics: Hotelling's T^2 and SPE

```
In [48]: pp.diagnostics(plsobj, score_plot_xydim=[1,2])
```

Why? To visualize where observations fall in different diagnostics
plots are linked



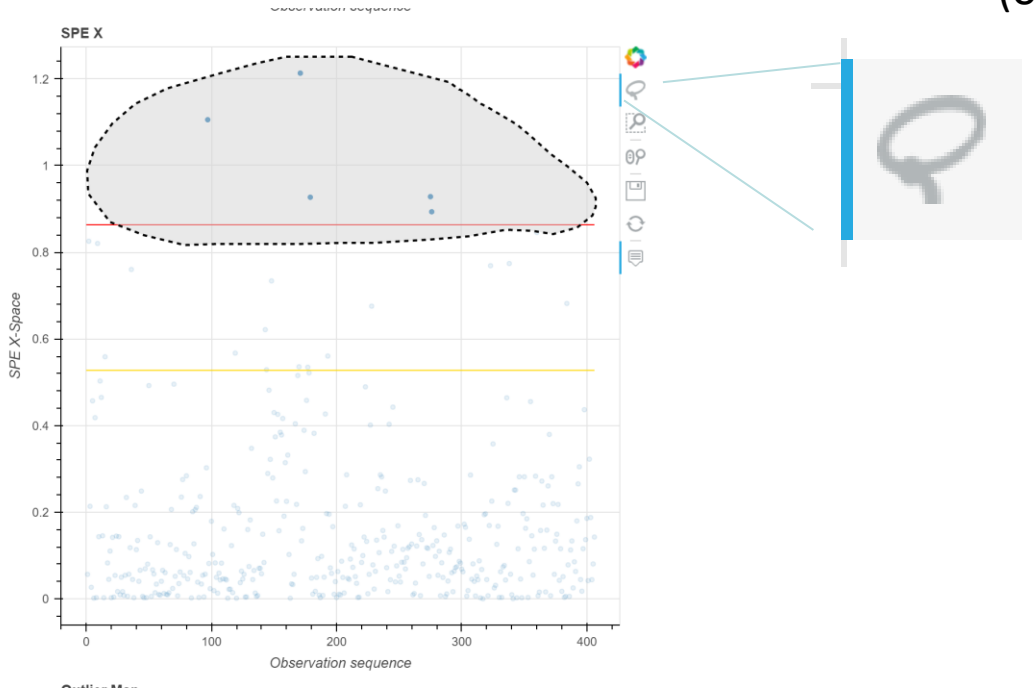
Using the lasso, select observations in the SPE X plot (or any other)

Pyphi_plot – Plotting diagnostics – Hotelling's T^2 and SPE

- Global diagnostics: Hotelling's T^2 and SPE

```
In [48]: pp.diagnostics(plsobj, score_plot_xydim=[1,2])
```

Why? To visualize where observations fall in different diagnostics **plots are linked**



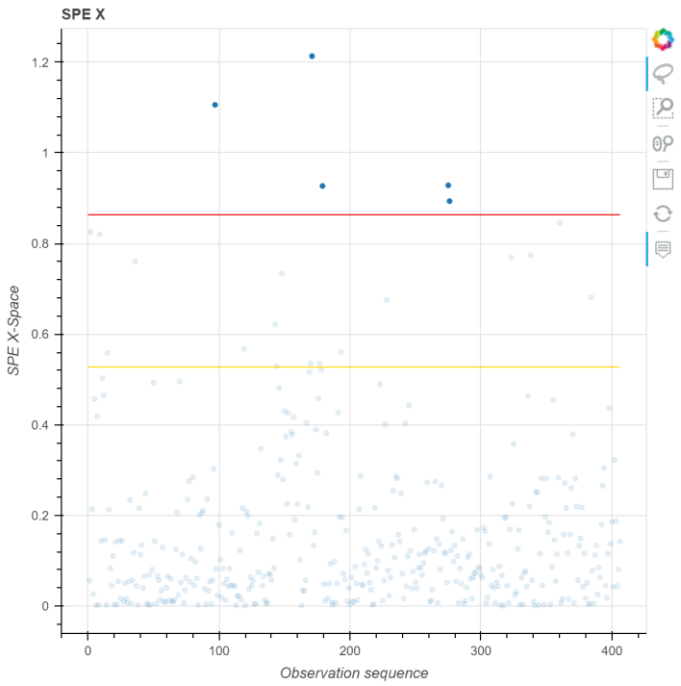
Using the lasso, select observations in the SPE X plot (or any other)

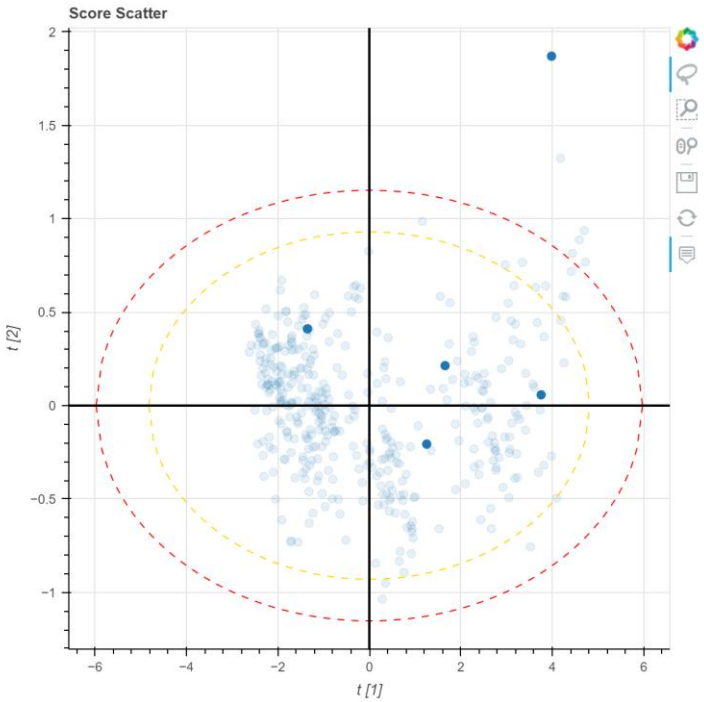
Pyphi_plot – Plotting diagnostics – Hotelling's T^2 and SPE

- Global diagnostics: Hotelling's T^2 and SPE

```
In [48]: pp.diagnostics(plsobj, score_plot_xydim=[1,2])
```

Why? To visualize where observations fall in different diagnostics
plots are linked





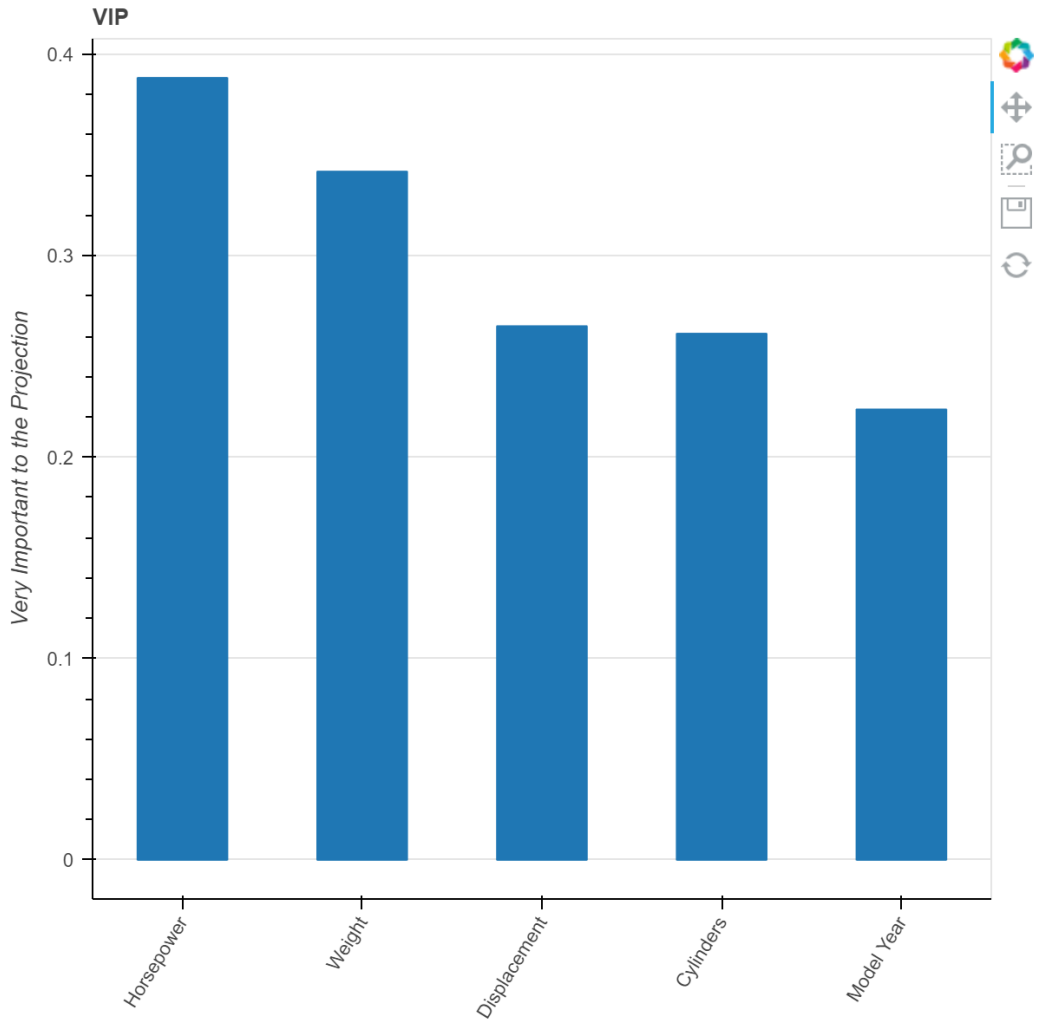
Selected observations are highlighted in all plots

Pyphi_plot – Plotting diagnostics – VIP Plot for PLS models

- Very Important to the Projection (VIP)

```
vip(mvmobj, *, plotwidth=600)
```

```
In [50]: pp.vip(plsobj)
```



Pyphi_plot – Plotting diagnostics – Contribution Plots

- Contributions to Scores, Hotelling's T2 and SPE

```
contributions_plot(mvmobj, X, cont_type, *, Y=False, from_obs=False,
to_obs=False, lv_space=False, plotwidth=800, plotheight=600, xgrid=False)
```

mvmobj: Model object created with phi.pca or phi.pca

X: Data over which contributions are calculated

cont_type: Type of contribution to be calculated : 'ht2' | 'spe' | 'scores'

to_obs: Name of observation(s) to calculate the contribution [string or list of strings]

from_obs: Name of observation(s) relative to which the contributions are calculated [string or list of strings]. If omitted, default is model origin. Only applicable to 'ht2' and 'scores'

lv_space: Latent variables over which the calculation is done. Default is all. Only applicable to 'ht2' and 'scores'

Pyphi_plot – Plotting diagnostics – Contribution Plots

- Contributions to Scores, Hotelling's T^2 and SPE

- Contributions to movement in the score space across all components from average (origin) to observation 'Car159'

```
In [55]: pp.contributions_plot(plsobj,Cars_Features,'scores',to_obs=['Car159'])
```

- Contributions to movement in the score space only across 2nd LV, from average (origin) to observation 'Car159'

```
In [56]: pp.contributions_plot(plsobj,Cars_Features,'scores',to_obs=['Car159'],lv_space=2)
```

- Contribution to difference (movement) in the score space from 'Car121' to 'Car151' only across 2nd LV

```
pp.contributions_plot(plsobj,Cars_Features,'scores',from_obs=['Car121'],to_obs=['Car159'],lv_space=[2])
```

- Contribution to differences in the score space, across all components from observations 'Car120' and 'Car121' , to observations 'Car159' , 'Car160' , and 'Car161'

```
In [60]: pp.contributions_plot(plsobj,Cars_Features,'scores',from_obs=['Car120','Car121'],to_obs=['Car159','Car160','Car161'])
```

Pyphi_plot – Plotting diagnostics – Predicted vs Observed

- Predicted vs Observed

```
predvsobs(mvmobj, X, Y, *, CLASSID=False, colorby=False, x_space=False)
```

mvmobj: A PLS model object created with phi.pls
X/Y: Data to use in the calculations

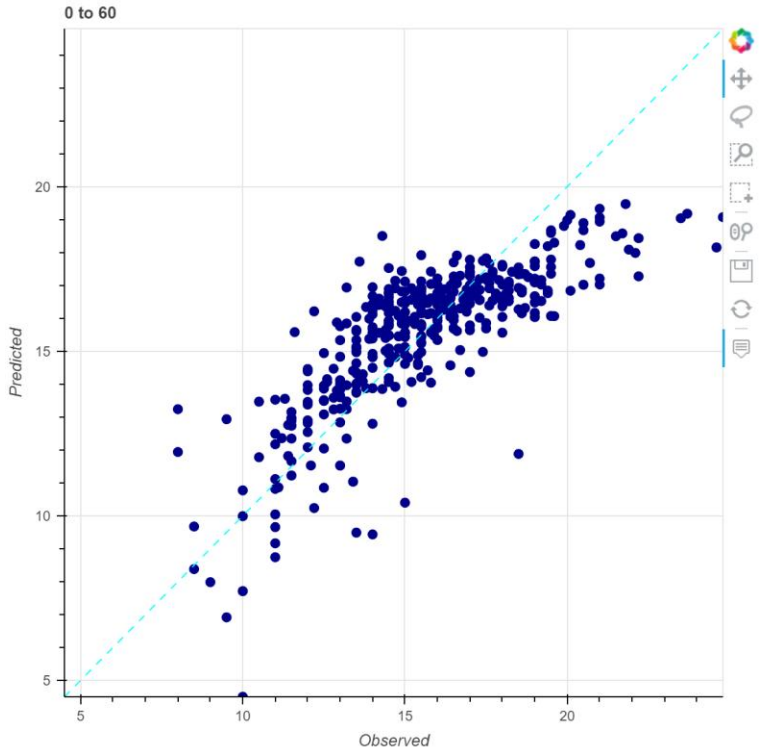
Optional
CLASSID*: A matrix with categorical data, e.g.
colorby*: Criteria to plot by (e.g. 'Origin')
x_space: 'True' will also include X Space variables in the plot. Default is False

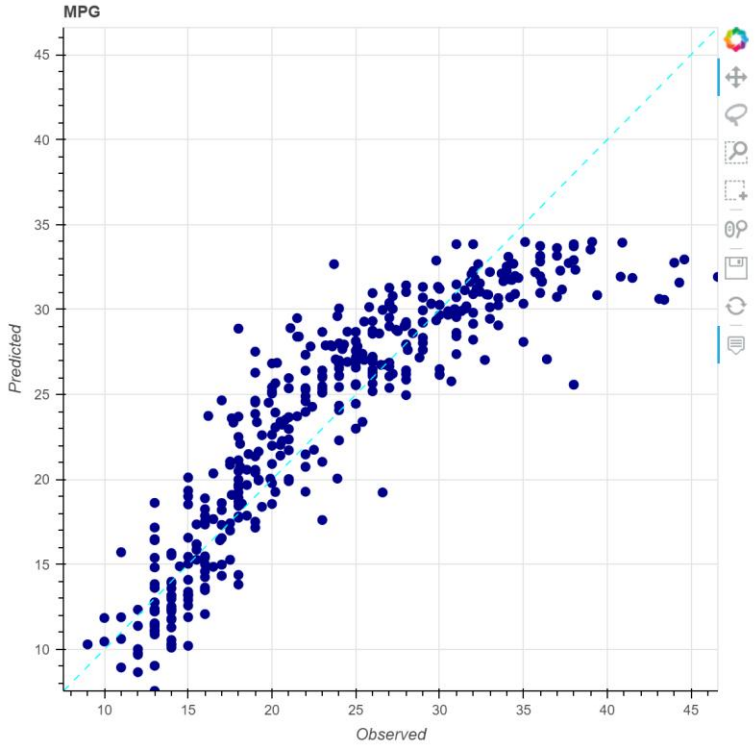
* Refer to slide 31 "Color coding with categories"

Pyphi_plot – Plotting diagnostics – Predicted vs Observed

- Predicted vs Observed

```
[63]: pp.predvsobs(plsobj,Cars_Features,Cars_Performance)
```

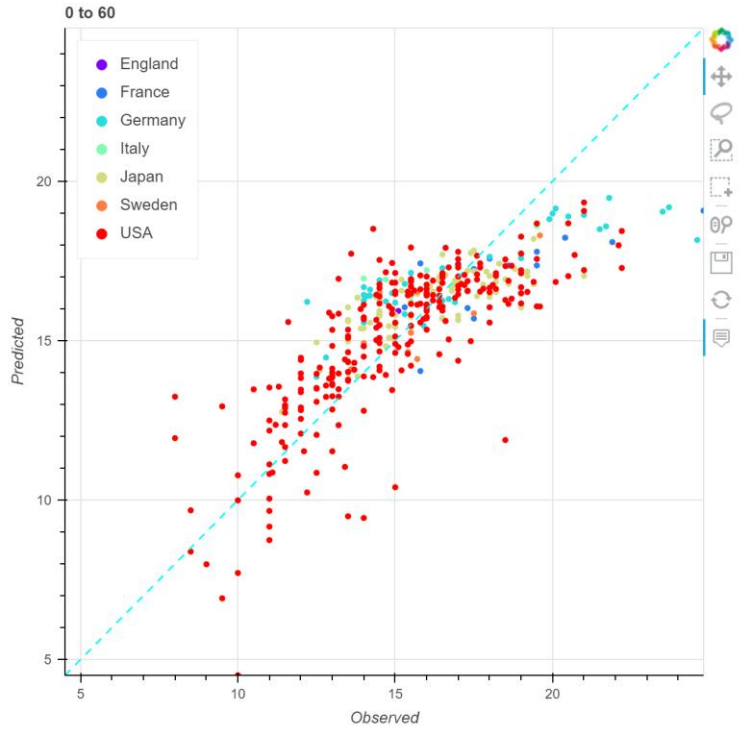


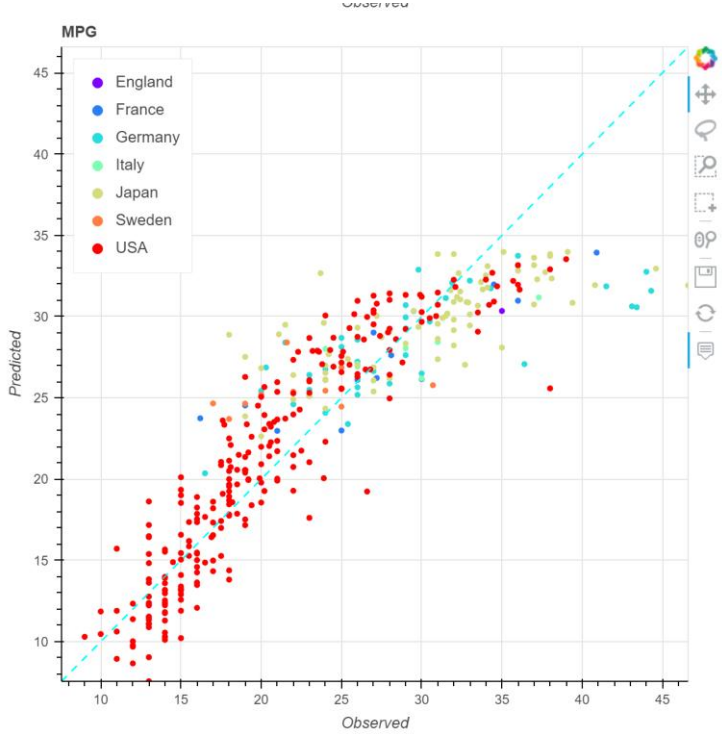


Pyphi_plot – Plotting diagnostics – Predicted vs Observed

- Predicted vs Observed

```
[62]: pp.predvsobs(plsobj,Cars_Features,Cars_Performance,CLASSID=Cars_CLASSID,colorby='Origin')
```





Pyphi_plots – Plotting diagnistics – Multi-Block PLS

- All plotting functions work seamlessly with an mbpls object:

```
mbpls_obj=phi.mbpls(mbdata,y_data,2)
preds=phi.pls_pred(mbdata, mbpls_obj)
pp.score_scatter(mbpls_obj, [1,2])
pp.weighted_loadings(mbpls_obj)
pp.loadings(mbpls_obj)
pp.vip(mbpls_obj)
```

- Multi-block specific plots:

mb_r2pb(mvmobj, *, plotwidth=600, plotheight=400) R2 per block

mb_weights(mvmobj, *, plotwidth=600, plotheight=400) Super-weights on blocks

mb_vip(mvmobj, *, plotwidth=600, plotheight=400) VIP per block

```
pp.r2pv(mbpls_obj)
pp.mb_r2pb(mbpls_obj)
pp.mb_weights(mbpls_obj)
pp.mb_vip(mbpls_obj)
```

Pyphi_plots – Plotting diagnistics – LPLS

- All plotting diagnostics work with an LPLS model

```
pp.loadings_map(lpls_obj, [1,2],addtitle='LPLS Model')
pp.loadings(lpls_obj,addtitle='LPLS Model')
pp.weighted_loadings(lpls_obj)
pp.vip(lpls_obj)
pp.r2pv(lpls_obj)
pp.score_scatter(lpls_obj, [1,2],add_labels=True,addtitle='Scores for blends')
pp.score_scatter(lpls_obj, [1,2],add_labels=True, rscores=True, addtitle='Scores for Materials')
```

- Loadings plotted are the material loadings.
- To make a score-scatter for the material space (X) use the flag **rscores = True**
  - Otherwise, the **t** scores are plotted

Pyphi_plots – Plotting diagnostics – JRPLS and TPLS

Flag "material" will plot the diagnostic for a specific material, available in:

```
pp.r2pv(jrplsobj,material='MAT4')
pp.loadings( jrplsobj,material='MAT4')
pp.loadings_map( jrplsobj, [1,2],material='MAT2')
pp.weighted_loadings( jrplsobj,material='MAT2')
pp.score_scatter(jrplsobj,[1,2],rscores=True,material='MAT2')
pp.vip(jrplsobj,plotwidth=1000 ,material='MAT4',addtitle='Mat4')
```

Pyphi_plots – Plotting diagnostics –TPLS

Flag **zspace=True** will plot the diagnostic for the process block, available in:

pp.r2pv(tplsobj,zspace=True)

pp.loadings(tplsobj,zspace=True)

pp.weighted_loadings(tplsobj,zspace=True)

pp.loadings_map(tplsobj,[1,2],plotwidth=800,zspace=True)

pp.vip(tplsobj,plotwidth=1000,zspace=True)

Imperial College London

PyPhiBatch – Multivariate Analysis of Batch Processes

Unfolding things the right way.

Centre for
Process Systems Engineering

Batch Data

- Batch Process:
  - Process that ends in a different state than the one it started
  - Non-stationary processes with no constant steady state
  - Operation is recipe based, recipe is repeated batch after batch
- Typical Dataset consists of:
  - **Trends or trajectories:** Multiple variables are measured throughout the batch and for multiple batches. (i.e. Variables change with time).
  - **Phase information:** Most automated processes operate by "steps" or "phases"
  - **Initial Conditions**: Characteristics that are not necessarily measured throughout the batch, but are only available at the begining.
  - **Product characteristics (or product quality measurements):** Variables not necessarily measured throughout the batch but at the end of the batch and are representative of the quality of the product

PyPhi Batch

- Basic implementation of Multi-way PCA and Multi-way PLS as published by Nomikos and MacGregor in a series of papers at the end of decade of the 90's.
- Supports unfolding in both directions (although variable wise unfolding is no different than "plain" PCA)
- Provides simple routines for batch alignment
- Can simulate the monitoring performance unfolding batch-wise:
  - Time-varying monitoring of process variables up to time k
    - Scores
    - Hoteliings T2
    - SPE (Global and instantaneous)
    - Contributions
  - Forecast for process variables from K+1 to end of batch
  - Forecasting of product quality from time k+1 to end of batch

PyPhi Batch

- Data format: Trajectories, one spreadsheet batch data runs downwards batch after batch

First column is always batch identifier, column name can be anything

If phase information is given, it is always the second column and must be named "Phase"

	A	B	C	D	E	F	G	H	I	J	K	L
1	Batch number	Phase	Tank level	Differential pressu	Dryer pressure	Power	Torque	Agitator Speed	Jacket Temp SP	Jacket Temp PV	Dryer Temp SP	Dryer Temp
44	Batch 1	Deagglomerate	46.783	2.896	56.508	102.325	8.016	31.043	61.297	84.995	87.893	24.691
45	Batch 1	Deagglomerate	47.63	2.988	56.061	102.207	8.016	30.636	61.134	84.995	87.957	24.573
46	Batch 1	Deagglomerate	48.444	3.03	55.615	101.533	8.016	29.097	60.971	84.994	88.021	24.485
47	Batch 1	Deagglomerate	49.258	3.086	55.163	99.783	8.016	28.587	60.808	84.993	88.085	24.398
48	Batch 1	Deagglomerate	50.009	3.068	54.709	98.851	8.016	27.473	60.645	84.993	88.126	24.325
49	Batch 1	Deagglomerate	50.766	3.173	54.256	96.821	8.017	26.023	60.482	84.992	88.167	24.252
50	Batch 1	Deagglomerate	51.419	3.135	53.802	96.668	8.017	25.364	60.32	84.992	88.207	24.098
51	Batch 1	Deagglomerate	52.037	3.248	53.246	95.876	8.017	24.433	60.11	84.991	88.247	23.948
52	Batch 1	Deagglomerate	52.572	3.301	52.684	94.283	8.017	20.613	60.077	84.99	88.313	23.831
53	Batch 1	Deagglomerate	53.107	3.384	52.123	94.074	8.017	25.767	60.09	84.99	88.399	23.647
54	Batch 1	Deagglomerate	53.642	3.437	51.413	92.105	8.017	20.504	59.67	84.989	88.486	23.48
55	Batch 1	Deagglomerate	54.177	3.612	50.665	91.222	8.017	24.797	59.347	84.988	88.577	23.369
56	Batch 1	Deagglomerate	54.638	3.698	49.708	89.475	8.018	24.313	59.299	84.988	88.668	23.299
57	Batch 1	Deagglomerate	54.872	3.904	48.579	88.597	8.018	24.202	59.324	84.987	88.703	23.229
58	Batch 1	Deagglomerate	55.106	4.089	46.511	86.025	8.018	20.521	59.215	84.987	88.63	23.158
59	Batch 1	Deagglomerate	55.34	4.465	44.33	83.985	8.018	15.75	59.142	84.986	88.557	23.325
60	Batch 1	Deagglomerate	55.574	4.872	42.15	82.149	8.018	19.237	59.075	84.985	88.359	23.763
61	Batch 1	Deagglomerate	55.619	5.213	39.969	80.32	8.018	15.617	59.009	84.985	88.03	24.502
62	Batch 1	Deagglomerate	55.6	5.856	37.788	78.207	8.018	18.256	58.988	84.984	87.698	25.241
63	Batch 1	Deagglomerate	55.581	6.376	34.74	76.87	27.067	19.625	58.967	84.983	87.356	25.98
64	Batch 1	Deagglomerate	55.562	6.702	32.958	74.757	31.98	13.763	58.947	84.983	86.847	26.856
65	Batch 1	Heat	55.543	6.845	30.864	74.464	32.029	17.448	58.927	84.982	85.844	28.785
66	Batch 1	Heat	55.524	7.06	29.625	74.219	32.029	16.926	58.906	84.982	86.952	30.875
67	Batch 1	Heat	55.505	7.427	28.605	74.029	32.028	16.736	58.886	84.981	85.864	32.97
68	Batch 1	Heat	55.486	7.789	28.316	73.876	32.028	16.365	58.865	84.98	86.739	35.066
69	Batch 1	Heat	55.481	8.191	28.351	73.622	32.028	11.966	58.845	84.98	85.707	37.161
70	Batch 1	Heat	55.485	8.652	28.386	73.33	32.027	15.419	58.825	84.979	86.517	39.257

Batch 1

Batch 2

Batch 3

Batch n

*If data is missing leave cell **empty**, the software will handle it*

Identifier for batch and phase must me non-numeric

PyPhi Batch

- Data Format: Initial Conditions , Product Quality and categories
  - Plain 2D matrix, batch identifier must match those in trajectories and **be in the same order of appearance as in the Trajectories spreadsheet**

	A	B	C	D	E	F	G	H
	Batch number	**Y1**	**Y2**	**Y3**	**Y4**	**Y5**	**Y6**	**Y7**
2	Batch 1	91.52	2.81	0.07	0.74	0.09	98.6	0.03
3	Batch 2	91.39	1.61	0.06	0.75	0.11	98.82	0
4	Batch 3	91.06	1.65	0.06	0.64	0.1	99.01	0
5	Batch 4	91.68	2.26	0	0.57	0	99.16	0
6	Batch 5	91.71	2.95	0.07	0.71	0.09	98.9	0
7	Batch 6	91.55	1.97	0.05	0.35	0.07	99.31	0
8	Batch 7	90.62	1.26	0.08	0.7	0.08	98.94	0
9	Batch 8	91.03	1.75	0.05	0.36	0.07	98.69	0
10	Batch 9	91.38	2.37	0	0.74	0.08	98.73	0
11	Batch 10	91.18		0	0.34	0	99.35	0
12	Batch 11	92.07		0	0.27	0	99.15	0
13	Batch 12	91.44		0	0.2	0	99.44	0
14	Batch 13	91.36	1.89	0.02	0.52	0.09	98.89	0
15	Batch 14	92.42	1.73	0	0.09	0	99.65	0
16	Batch 15	91.73	2.11	0	0.3	0.05	99.45	0
17	Batch 16	91.63		0	0.3	0	99.49	0
18	Batch 17	91.85	1.9	0	0.29	0	99.57	0
19	Batch 18	91.46	1.87	0	0.2	0	99.51	0
20	Batch 19	91.04	1.84	0	0.4	0.09	97.81	0.22
21	Batch 20	92.33	2.8	0	0.5	0	99.11	0
22	Batch 21	91.8		0	0.83	0.07	98.57	0
23	Batch 22	91.55		0	1.04	0.05	98.23	0
24	Batch 23	91.19		0	0.65	0.05	98.97	0
25	Batch 24	91.01		0	0.27	0	98.97	0
26	Batch 25	90.48	2.09	0	0.71	0	98.91	0

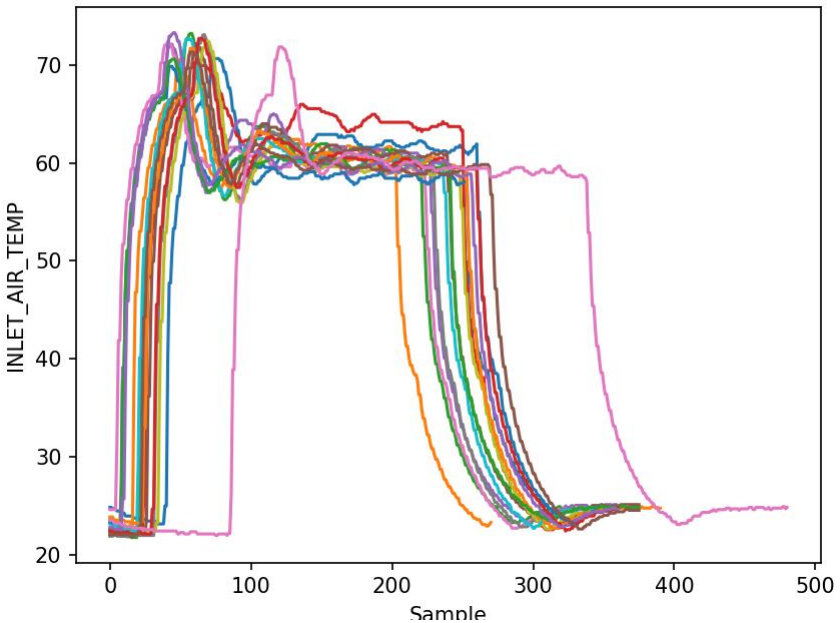
Example of data for the quality of multiple batches

	A	B	C
1	**Batch number**	**Quality**	
2	Batch 1	On-spec	
3	Batch 2	On-spec	
4	Batch 3	On-spec	
5	Batch 4	On-spec	
6	Batch 5	On-spec	
7	Batch 6	On-spec	
8	Batch 7	On-spec	
9	Batch 8	On-spec	
10	Batch 9	On-spec	
11	Batch 10	On-spec	
12	Batch 11	On-spec	
13	Batch 12	On-spec	
14	Batch 13	On-spec	
15	Batch 14	On-spec	
16	Batch 15	On-spec	
17	Batch 16	On-spec	
18	Batch 17	On-spec	
19	Batch 18	On-spec	
20	Batch 19	On-spec	
21	Batch 20	On-spec	
22	Batch 21	On-spec	
23	Batch 22	On-spec	
24	Batch 23	On-spec	
25	Batch 24	On-spec	

Example of categorical data multiple batches

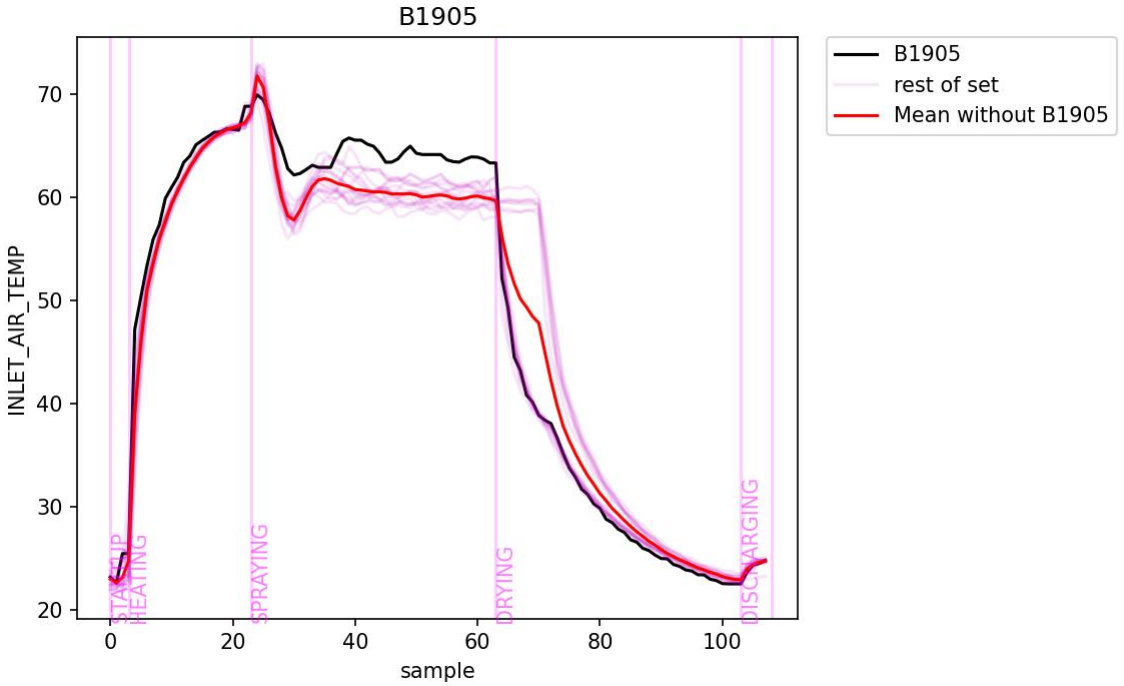
Simple plotting of batch trajectories

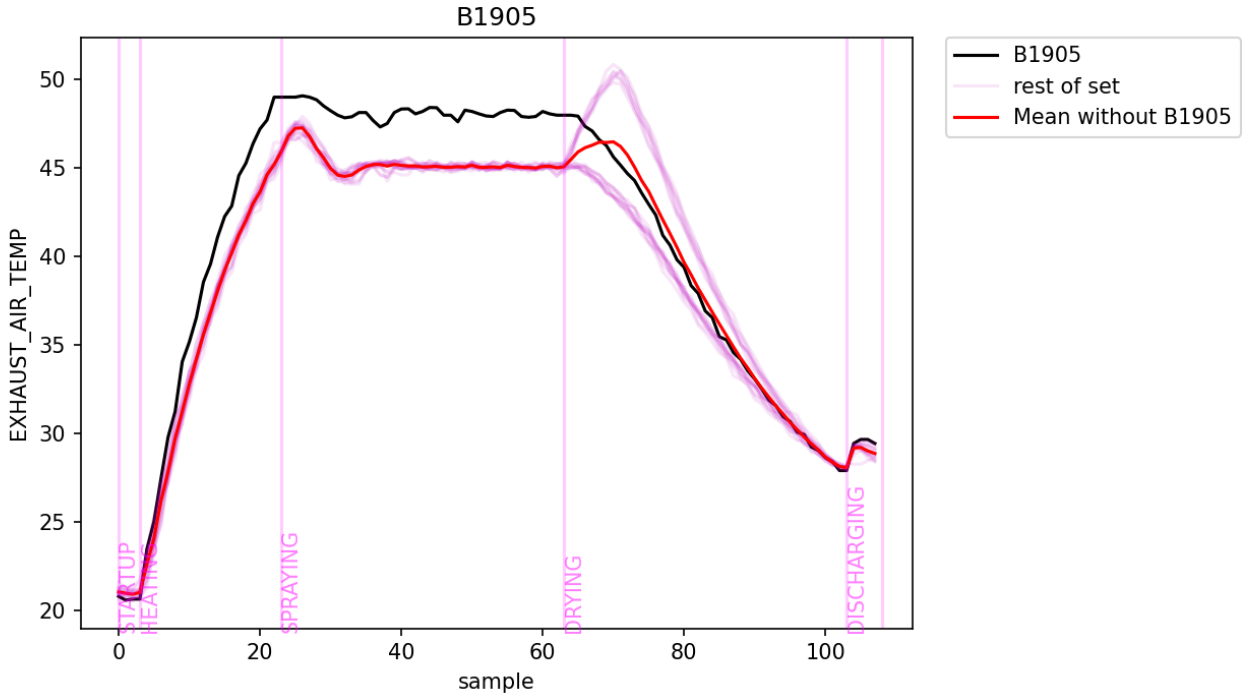
```python
import pandas as pd
import numpy as np
import pyphi_batch as phibatch
import pyphi_plots as pp
import matplotlib.pyplot as plt
import time

bdata=pd.read_excel('Batch Film Coating.xlsx')
#plot all variables
phibatch.plot_var_all_batches(bdata)
#plot a variable
phibatch.plot_var_all_batches(bdata,var_list='INLET_AIR_TEMP')
#plot some variables
phibatch.plot_var_all_batches(bdata,var_list=['INLET_AIR_TEMP','EXHAUST_AIR_TEMP'])
```



Simple plotting of trajectories

```
phibatch.plot_batch(bdata_aligned_phase,
           which_batch=['B1905'],
           which_var=['INLET_AIR_TEMP','EXHAUST_AIR_TEMP','INLET_AIR'],
           include_mean_exc=True,
           include_set=True,
           phase_samples=samples_per_phase)
```





Simple plotting of batch trajectories

plot_var_all_batches(bdata,*,var_list=False,plot_title='',phase_samples=False ,alpha_=0.2)

```
import pandas as pd
import numpy as np
import pyphi_batch as phibatch
import pyphi_plots as pp
import matplotlib.pyplot as plt
import time

bdata=pd.read_excel('Batch Film Coating.xlsx')
#plot all variables
phibatch.plot_var_all_batches(bdata)
#plot a variable
phibatch.plot_var_all_batches(bdata,var_list='INLET_AIR_TEMP')
#plot some variables
phibatch.plot_var_all_batches(bdata,var_list=['INLET_AIR_TEMP','EXHAUST_AIR_TEMP'])
```

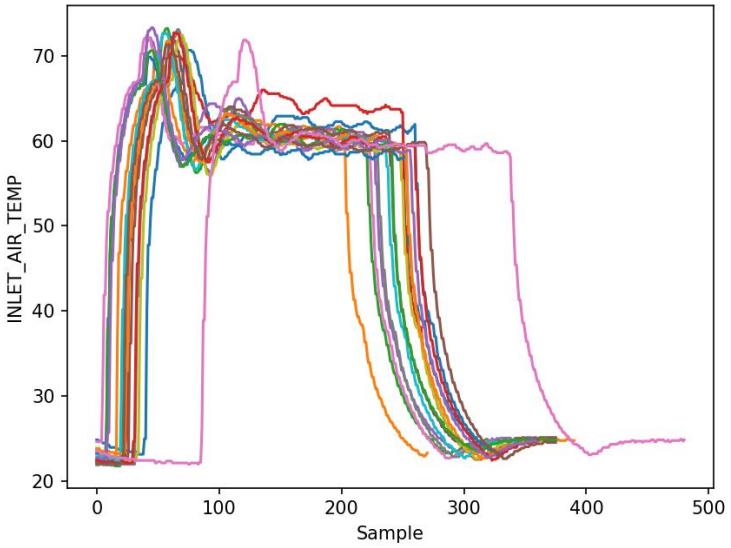
Simple plotting of batch trajectories

plot_batch(bdata,which_batch,which_var,*,include_mean_exc=False,include_set=False, phase_samples=False,single_plot=False):

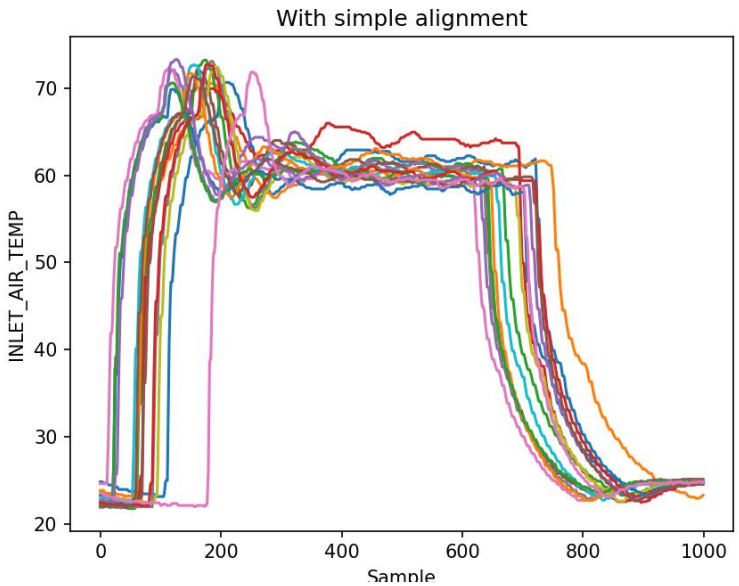
```
phibatch.plot_batch(bdata_aligned_phase,
          which_batch=['B1905'],
          which_var=['INLET_AIR_TEMP','EXHAUST_AIR_TEMP','INLET_AIR'],
          include_mean_exc=True,
          include_set=True,
          phase_samples=samples_per_phase)
```

Batch alignment

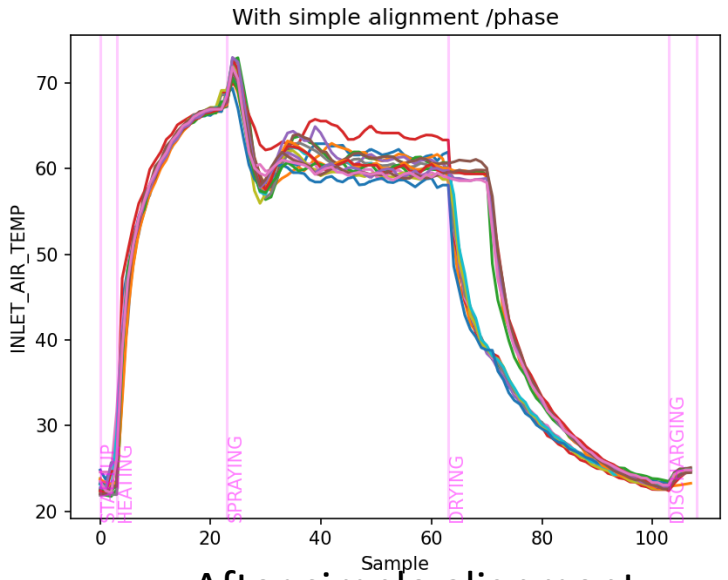
- Batch data must be aligned to prevent comparing variables at different stages of operation
- Data is normally not aligned due to normal variation in the execution
  - A simple time delay can cause a shift in the data



Raw Data



After simple alignment
resample all batches
n times throughout



After simple alignment
By phase
resample all batches:
n_a times during Phase *a*
n_b *times during Phase b*

Batch alignment

phibatch.simple_align(raw_batch_data,n)

raw batch data: As read from spreadsheet

n: Number of samples

```
bdata_aligned=phibatch.simple_align(bdata,1000)
```

phibatch.phase_simple_align(raw_batch_data,samples_per_phase)

raw batch data: As read from spreadsheet

samples_per_phase : dictionary with samples to take for each phase. Phase names must match spreadsheet

```
samples_per_phase={'STARTUP':3, 'HEATING':20,'SPRAYING':40, 'DRYING':40,'DISCHARGING':5}
bdata_aligned_phase=phibatch.phase_simple_align(bdata,samples_per_phase)
```

Both routines return a pandas dataframe with aligned data

Batch alignment **using indicator variable**

`phibatch.phase_iv_align(bdata, nsamples)`

bdata is a Pandas DataFrame where 1st column is Batch Identifier
the second column is a phase indicator and following columns are variables, each row is a new time sample. Batches are concatenated vertically
if nsamples is a dictionary: samples to generate per phase e.g.
nsamples = {'Heating':100,'Reaction':200,'Cooling':10}

If an indicator variable is used, with known start and end values
indicate it with a list like this:
[IVarID,num_samples,start_value,end_value]
example:
nsamples = {'Heating':['TIC101',100,30,50],'Reaction':200,'Cooling':10}
During the 'Heating' phase use TIC101 as an indicator variable
take 100 samples equidistant from TIC101=30 to TIC101=50
and align against that variable as a measure of batch evolution (instead of time)
* If an indicator variable is used, with unknown start but known end values
indicate it with a list like this:
[IVarID,num_samples,end_value]
example
nsamples = {'Heating':['TIC101',100,50],'Reaction':200,'Cooling':10}
During the 'Heating' phase use TIC101 as an indicator variable
take 100 samples equidistant from the value of TIC101 at the start of the phase
to the point when TIC101=50 and align against that variable as a measure
of batch evolution (instead of time)
Returns a pandas dataframe with batch data resampled (aligned)

If no IV is sent, the resampling is linear with respect to row number per phase

Multi-way PCA

`phibatch.mpca(xbatch,a,*,unfolding='batch wise',phase_samples=False,cross_val=0)`

Multi-way PCA for batch analysis

xbatch: Pandas dataframe with aligned batch data it is assumed that all batches have the same number of samples

a: Number of PC's to fit

unfolding: 'batch wise' or 'variable wise'

phase_samples: information about samples per phase [optional]

cross_val: percent of elements for cross validation (defult is 0 = no cross val)

Multi-way PLS

```
phibatch.mpls(xbatch,y,a,*,zinit=False,phase_samples=False,mb_each_var=False,
              cross_val=0,cross_val_X=False)
```

Multi-way PLS for batch analysis

xbatch: Pandas dataframe with aligned batch data it is assumed that all batches have the same number of samples

y : Response to predict, one row per batch

a: Number of PC's to fit

zinit: Initial conditions <optional>

phase_samples: alignment information

mb_each_var: if "True" will make each variable measured a block

otherwise zinit is one block and xbatch another

cross_val and cross_val_X are same as in phi.pls

Predictions

`predict(xbatch, mmvm_obj, *, zinit=False)`

Generate predictions for a Multi-way PCA/PLS model

Input:

xbatch: Batch data with same variables and alignment as model will generate predictions for all batches

mmvm_obj: Multi-way PLS or PCA

zinit: Initial conditions [if any]

Output:

preds: A dictionary with keys ['Yhat', 'Xhat', 'Tnew', 'speX', 'T2']

Plots

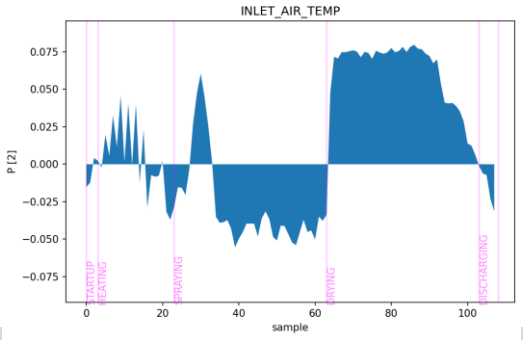
- Every plot in PyPhi plot works seemingly with PyPhi Batch, with the exception of the contribution plots (data must be unfolded first)
- Additional plots (since the PyPhi plot might prove to be difficult to read since loadings are time varying)

loadings(mmvm_obj, dim, *, r2_weighted=False, which_var=False)

```
    mmvm_obj:     Multiway PCA or PLS object
    dim:          What component or latent variable to plot
    r2_weighted:  If True => weight the loading by the R2pv
    which_var:    Variable for which the plot is done, if not
                  sent all are plotted
```

The routine will plot the loadings for all trajectories

and for initial conditions (if given)



Other Plots

r2pv(mmvm_obj, *, which_var=False)

Plot batch r2 for variables as a function of time/sample

mmvm_obj: Multiway PCA or PLS object

which_var: Variable for which the plot is done, if not sent all are plotted

loadings_abs_integral(mmvm_obj, *, r2_weighted=False, addtitle=False)

Plot the integral of the absolute value of loadings for a batch

Inputs:

mmvm_obj: A multiway PCA or PLS model

r2_weighted: Boolean flag, if True then in weights the loading by the R2pv

addtitle: Text to place in the title of the figure

Other plots

contributions(mmvmobj, X, cont_type, *, to_obs=False, from_obs=False, lv_space=False, phase_samples=False, dyn_conts=False, which_var=False)

Batch contribution plots to Scores, HT^2 or SPE

mmvmobj= Multiway Model

X = batch data

cont_type = 'scores' | 'ht2' | 'spe'

to_obs = Observation to calculate contributions to

from_obs = Relative basis to calculate contributions to [for 'scores' and 'ht2' only]

if not sent the origin of the model us used as the base.

Returns:

contribution_vector

Utilities for batch analysis

phase_sampling_dist(bdata, time_column=False, addtitle=False, use_phases=False)

Count and plot a histogram of the distribution of samples (or time if time_column is indicated) consumed per phase on a batch dataset

bdata: Batch data organized as: column[0] = Batch Identifier column name is unrestricted

column[1] = Phase information per sample must be called 'Phase','phase', or 'PHASE'

this information is optional

column[2:]= Variables measured throughout the batch

time_column: Indicates the name of the column with time, if not sent, counting is done in terms samples

add_title: Optional text to be placed as the figure title

use_phases: In case the user wants to only do counting for a subset of phases

Utilities for batch analysis

`build_rel_time(bdata, *, time_unit='min')`

Converts the column 'Timestamp' into 'Time' in time_units relative to the start of each batch

`descriptors(bdata, which_var, desc, *, phase=False)`

Get descriptor values for a batch trajectory

Inputs:

bdata: Dataframe of batch data, first column is batch ID, second column can be phase id

which_var: List of variables to get descriptors for

desc: List of descriptors to calculate, options are:

'min','max','mean','median', 'std','var','range', 'ave_slope'

phase: to specify what phases to do this for

Outputs: descriptors: A dataframe with the descriptors per batch

Batch monitoring

- Mimics the exercise of monitoring a batch in real-time.
- Done in two steps:
  a) Calculate confidence intervals using a set of "golden batches"
  b) Monitor a specific batch

- Both done with the same routine **phibatch.monitor**

a) Calculate confidence intervals using all data, and return a new model object augmented with the necessary information to mimic monitoring, examples:

```
phibatch.monitor(mpca_obj, clean_batch_data)
```
Simple MPCA model

```
phibatch.monitor(mpls_obj, bdata_aligned_phase)
```
Simple MPLS model

```
phibatch.monitor(mpls_obj, bdata_aligned_phase,zinit=initial_chem)
```
MPLS model with initial conditions

Batch monitoring

- Mimics the exercise of monitoring a batch in real-time.
- Done in two steps:
  a) Calculate confidence intervals using a set of "golden batches"
  b) Monitor a specific batch
- Both done with the same routine **phibatch.monitor**

b) Monitor a specific batch

Mimic monitoring for one batch

```
mon_1905 = phibatch.monitor(mpca_obj,dev_batch_data,which_batch='B1905')
```

Mimic monitoring for various batches

```
mon_1905_1805 = phibatch.monitor(mpca_obj,dev_batch_data,which_batch=['B1905','B1805'])
```

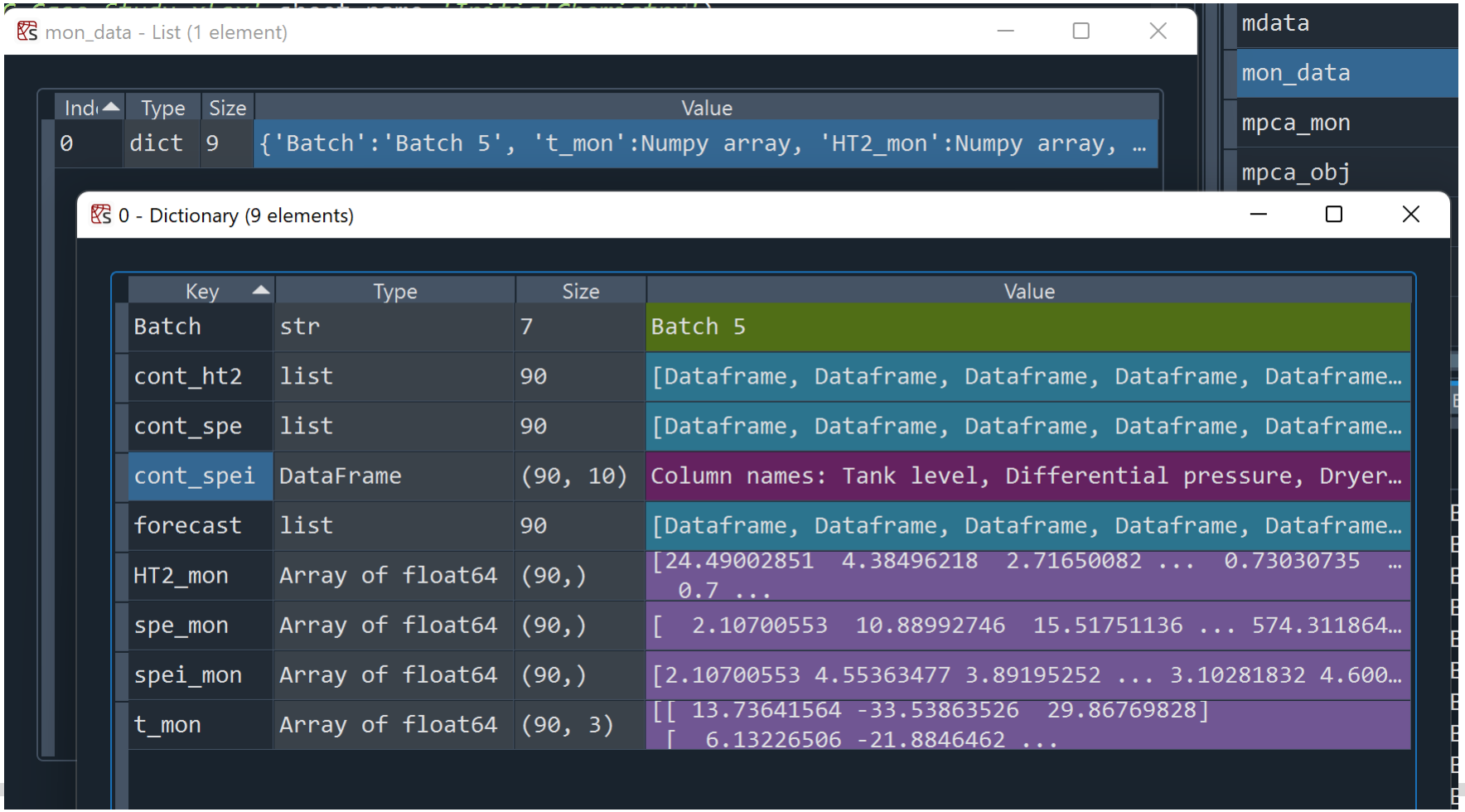
Mimic monitoring a batch if using a MPLS with initial conditions

```
mon_data =phibatch.monitor(mpls_obj, bdata_aligned_phase,which_batch='Batch 5',zinit=initial_chem)
```

Add the flag **soft_sensor=*variable_name*** to mimic the soft-sensor predictions of the variable

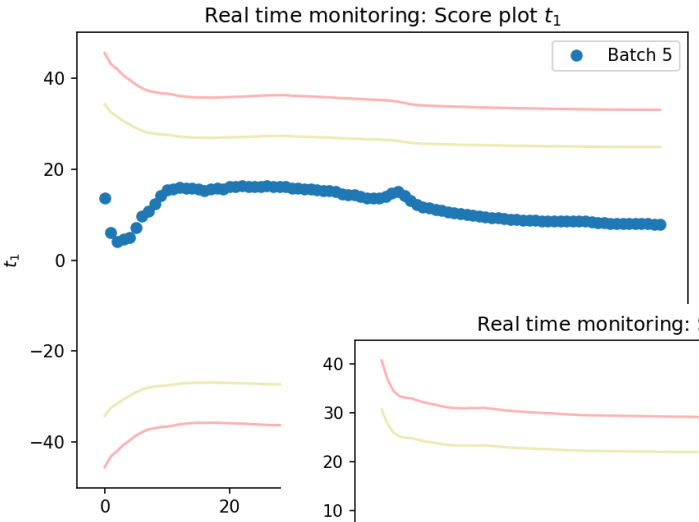
Batch monitoring

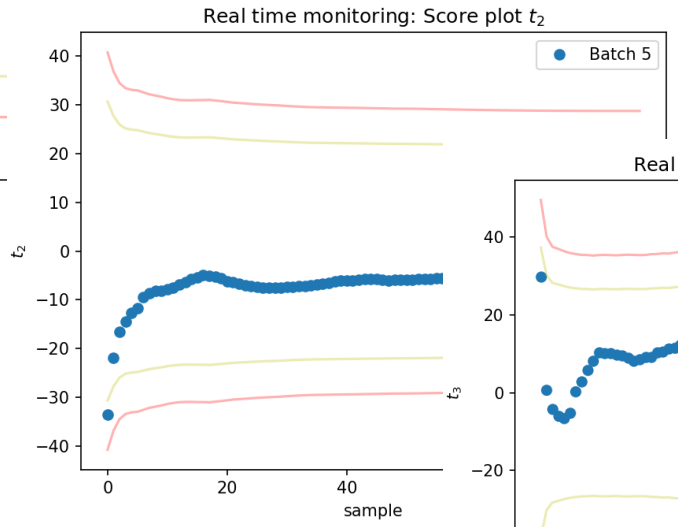
- The monitoring calculation returns a list of dictionaries (one per batch that was monitored) with all the time-varying diagnostics

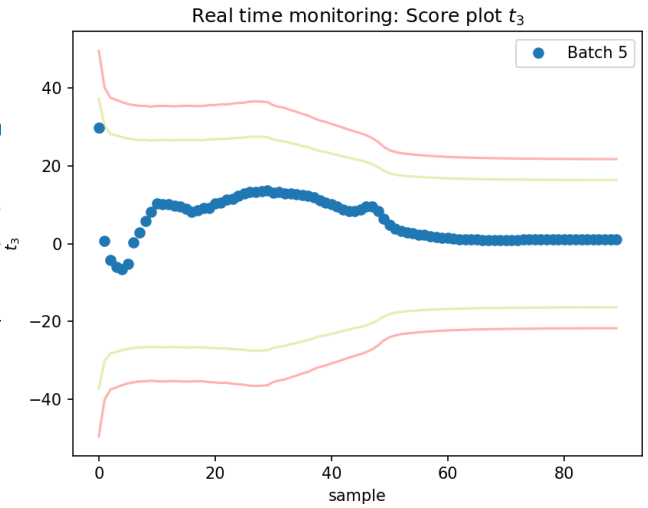


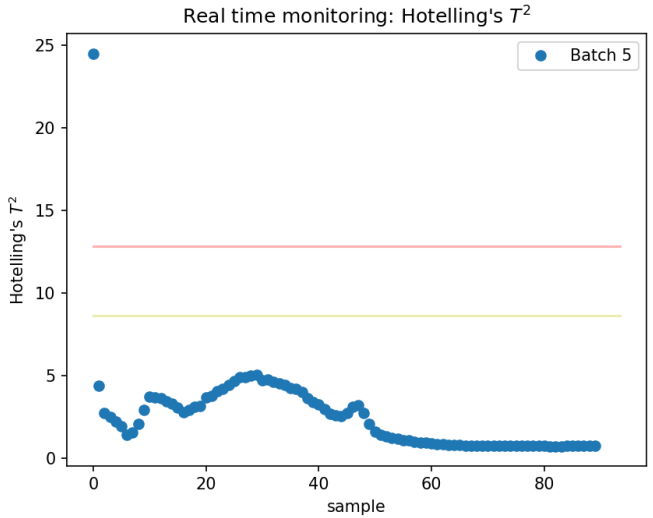
Batch monitoring

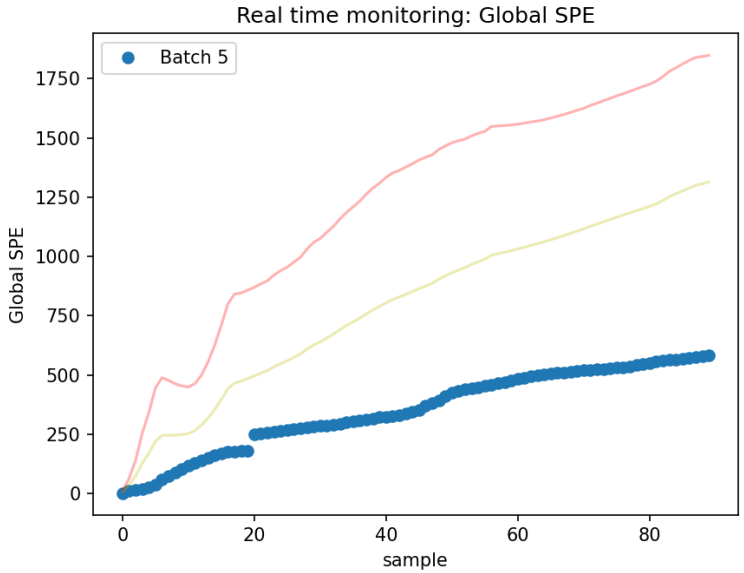
- And will also produce all the plots

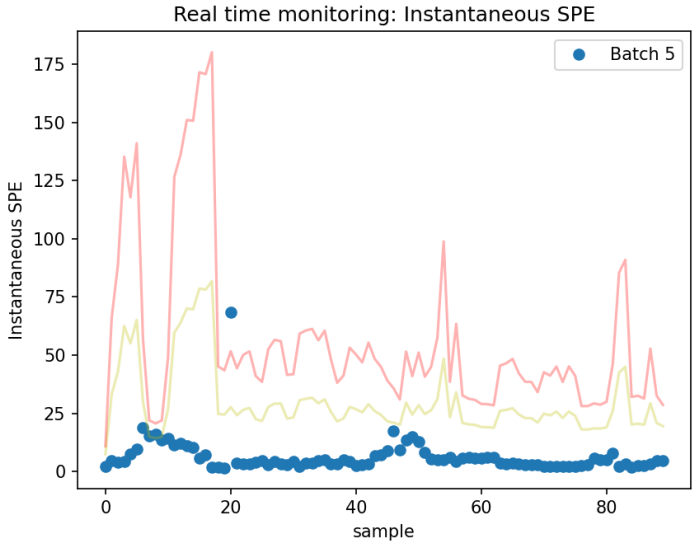












Batch Monitoring

- The information returned includes :
  - The contributions to all diagnostics for all times
  - The forecasted trajectory for all variables as the batch progresses
  - The forecasted quality of the end product as the batch progresses