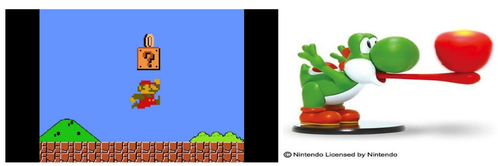
Interpretable machine learning

Vine

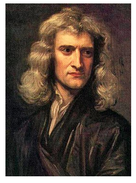
Thomas Bury
Pretending to be a DS

Coins, apples and bias



$\mathrm{P}[\mathrm{Head}] = \cdots$

0.5, obviously. But what about him?



Is the coin not a solid body? Is it not supposed to be ruled by Newton's laws and therefore the movement be deterministic? We are so used to it that we don't even consider the question "is it really probabilistic?" -> Example of misinterpretation and thought-bias

The three-dimensional dynamics of the die throw

Are human beings rational?

- Rationality definition
- Randomness human behaviour

Explainability vs Interpretability

- Interpretability: *predict* what would happen if you changed the input (i.e. predictions).
- Explainability: explain why this change happens, the internal mechanics (models that can summarize the reasons for specific behaviour)

If you can explain it, you can interpret it. The converse is not true.

However, interpretability is often not understood as such. Causation and correlation are often confused.

Explainability vs Interpretability cont'd

Example: throwing a ball with an initial speed $\vec{v}_0$, you observe it lands at $\vec{r}_0$. You change the speed from $\vec{v}_0$ to $\vec{v}_1$, same direction

- **Interpretation**: the ball will land farther, using historical data you predict the landing to occur at $\vec{r}_0$ (you collected data by making an experiment and made a regression and thus derive a phenomenological/empirical model)
- **Explanation**: the ball trajectory obeys Newton's laws. You know the exact effect of initial position and speed on the trajectory

Is interpretability a myth or useful?

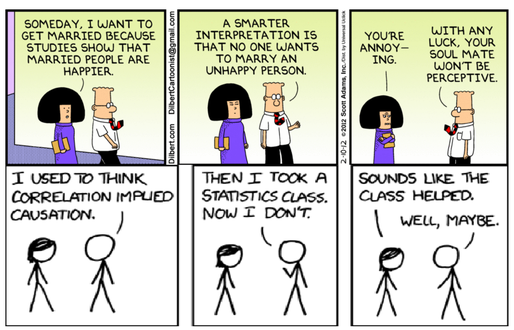
- What if many features ($10, 100, 10^6$), correlated and predictive power spread among them? Will you be able to summarize them with a *verbal template*?
- What if two very different models (GLM and GBM for instance) predict the same values but interpretations differ? Is the interpretation useful at all?
- What if you can interpret the model? Does it give you more insight on the causal relationship than you do not know? (and correlation doesn't mean causation)
- Machine learning and inference are iterative by nature, so is the interpretation
- Does the interpretation change if you include new information that actually truly matters? If yes can we rely on the interpretation?
- Do you consider your decision making as interpretable? Are you a rational being? (I'm far from being rational even if I'm more a robot than a human being according to my friends)
- Etc, see for instance this vlog, fast forward to 20' and this paper for instance

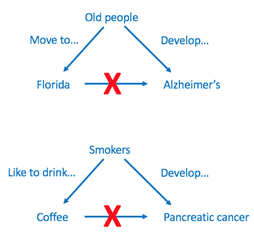
All is about being critical. Correlation is not causation. Interpretation is not as valuable as one might think at first glance if the purpose is not well defined beforehand (for marketing? for regulation? for common knowledge? for being reassured)

So what?

Even if we care about the predicted values and not the structure of the data generation process, it's however useful to know how the response is *correlated* to the predictors/information we know.

But keep in mind that...





However, if we have a *theory* (natural sciences) or enough observed *stylized facts* (finance):

Especially important when communicating with management and marketing

What is causal for a model $y = f(\quad)$ might just be a correlation in real-world because of an exogenous latent (not directly observed) variable

A simple artificial example

The Dummy model

$$y = \beta_0 + \beta_1 x_1 + \beta_{24} x_2 x_4 + \beta_3 x_3 + \epsilon$$

$$\begin{aligned}
\beta_0 &= 4 \\
\beta_1 &= 7 \\
\beta_{24} &= -6 \\
\beta_3 &= 5 \\
\epsilon &\sim \quad (0, 1) \\
\text{corr}(\quad) &= \begin{vmatrix} 1.0 & -0.4 & -0.9 \\ -0.4 & 1.0 & 0.0 \\ -0.9 & 0.0 & 1.0 \end{vmatrix}
\end{aligned}$$

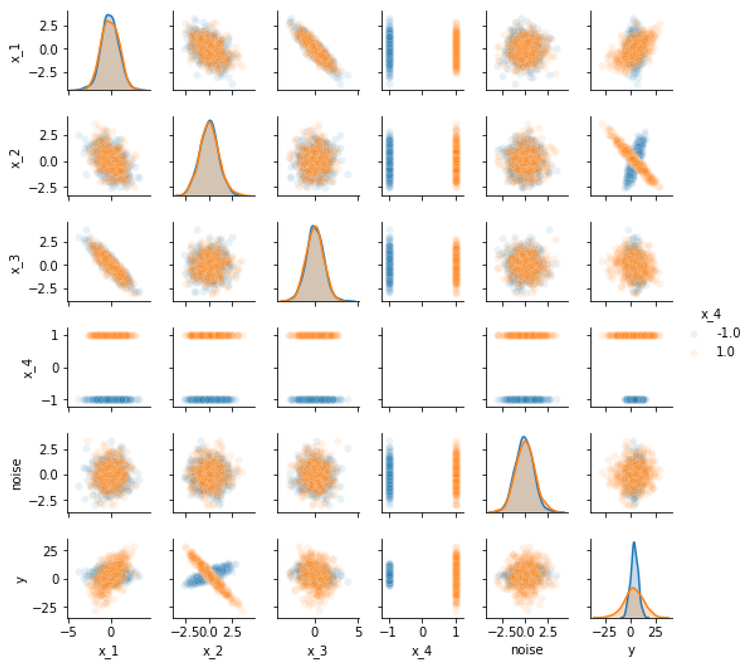
with correlated predictors (common case)

In [4]:
```
np.round(np.corrcoef(corr_data[ ['y', 'x_1', 'x_2', 'x_3', 'noise']].T), 2)
```

Out[4]:
```
array([[ 1.  ,  0.38, -0.45, -0.19,  0.12],
       [ 0.38,  1.  , -0.43, -0.9 ,  0.02],
       [-0.45, -0.43,  1.  ,  0.05,  0.  ],
       [-0.19, -0.9 ,  0.05,  1.  , -0.03],
       [ 0.12,  0.02,  0.  , -0.03,  1.  ]])
```

In [31]:
```
sns.pairplot(corr_data, hue="x_4", plot_kws={'alpha': 0.1}, height=1.25)
plt.show()
```

```
:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py:48
: RuntimeWarning: invalid value encountered in true_divide
 binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kdetools.
y:34: RuntimeWarning: invalid value encountered in double_scalars
 FAC1 = 2*(np.pi*bw/RANGE)**2
```



From the pairplot, we deduce that there is an interaction between x_2 and x_4. Let's see what happens when we do not include all the relevant predictors and what are the consequences of collinearity on the coefficients and the model interpretation.

Collinearity and the variance inflation factor

Collinearity increases the variance of estimator and inflates the t-stat. Moreover, correlation can induce significantly different interpretation than the reality if we don't know exactly the data generation process (the majority of cases). Compare the regression coefficient values to the true values and have a look at their confidence interval.

Be careful

In [6]:

```
# Variance Inflation Factor
#
train = corr_data[corr_data.set=='train']
test = corr_data[corr_data.set=='test']
y = train['y']
X = train[['x_1', 'x_2', 'x_3', 'x_4']]

print(pd.Series([variance_inflation_factor(X.values, i) for i in range(X.shape[1
])], index=X.columns))
```

```
x_1    37.589518
x_2     7.241415
x_3    30.426052
x_4     1.001271
dtype: float64
```

VIF are large. We must pay attention to the results and not consider the coefficients as independent from each other.

In [7]:
```
y = train['y']
X = train[['x_3']]

X = sm.add_constant(X)
ols_dummy = sm.OLS(y, X).fit()
print(ols_dummy.summary())
```

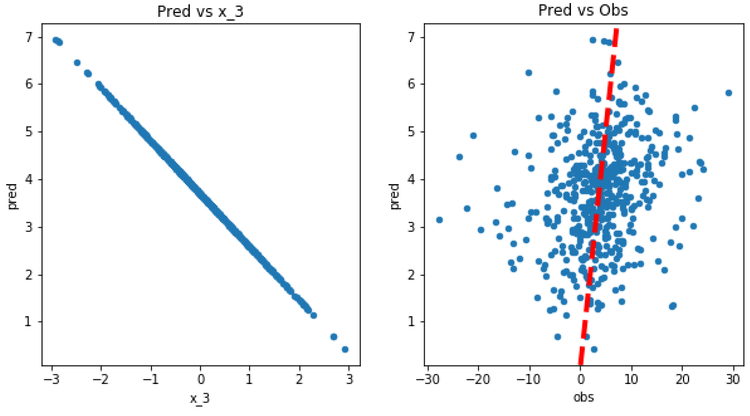
```
:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389: Fut
reWarning: Method .ptp is deprecated and will be removed in a future version.
se numpy.ptp instead.
 return ptp(axis=axis, out=out, **kwargs)
```

```
                            OLS Regression Results
==============================================================================
ep. Variable:                      y   R-squared:                       0.026
odel:                            OLS   Adj. R-squared:                  0.024
ethod:                 Least Squares   F-statistic:                     13.40
ate:                Fri, 27 Sep 2019   Prob (F-statistic):           0.000278
ime:                        11:45:32   Log-Likelihood:                -1667.2
o. Observations:                 500   AIC:                             3338.
f Residuals:                     498   BIC:                             3347.
f Model:                           1
ovariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
onst           3.6906      0.304     12.120      0.000       3.092       4.289
_3            -1.1153      0.305     -3.661      0.000      -1.714      -0.517
==============================================================================
mnibus:                       27.447   Durbin-Watson:                   1.986
rob(Omnibus):                  0.000   Jarque-Bera (JB):               59.266
kew:                          -0.294   Prob(JB):                     1.35e-13
urtosis:                       4.581   Cond. No.                         1.04
==============================================================================

arnings:
[1] Standard Errors assume that the covariance matrix of the errors is correct
y specified.
```

The estimation is $\hat{\beta}_3 = -1.12 \pm 0.30$. However the true value is $\beta_3 = 5$. Predicted values are grossly in line with observations and the coefficient has the wrong is negative instead of being positive. In this case, the interpretation will be wrong even if it predicted values are indeed decreasing with x_3.

In [9]:

```
pred = ols_dummy.predict(sm.add_constant(test[['x_3']]))
pred_df = test.loc[:, ['x_3']]
pred_df['pred'] = pred.values
pred_df['obs'] = test.y.values
# Plot
fig, (ax1, ax2) = plt.subplots(1, 2)
pred_df.plot(kind='scatter', x='x_3', y='pred', ax=ax1, title="Pred vs x_3", figsi
ze=(10,5))
pred_df.plot(kind='scatter', x='obs', y='pred', ax=ax2, title="Pred vs Obs")
add_identity(ax2, color='r', ls='--', linewidth=4)
plt.show(fig)
```



In [10]:

```
# Using more predictors
y = train['y']
X = train[['x_1', 'x_2', 'x_3']]

print(pd.Series([variance_inflation_factor(X.values, i) for i in range(X.shape[1
])], index=X.columns))

X = sm.add_constant(X)
ols_wo_int = sm.OLS(y, X).fit()
print(ols_wo_int.summary())
```

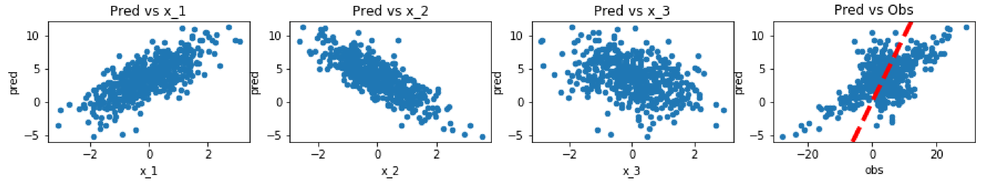
```
_1    37.573071
_2     7.240498
_3    30.408380
type: float64
                            OLS Regression Results
==============================================================================
ep. Variable:                      y   R-squared:                       0.180
odel:                            OLS   Adj. R-squared:                  0.175
ethod:                 Least Squares   F-statistic:                     36.36
ate:                Fri, 27 Sep 2019   Prob (F-statistic):           2.98e-21
ime:                        11:45:33   Log-Likelihood:                -1624.1
o. Observations:                 500   AIC:                             3256.
f Residuals:                     496   BIC:                             3273.
f Model:                           3
ovariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
onst           3.7816      0.281     13.479      0.000       3.230       4.333
_1             7.1738      1.669      4.297      0.000       3.894      10.454
_2             0.5649      0.730      0.774      0.439      -0.869       1.999
_3             5.4901      1.545      3.554      0.000       2.455       8.525
==============================================================================
mnibus:                        1.184   Durbin-Watson:                   1.922
rob(Omnibus):                  0.553   Jarque-Bera (JB):                1.112
kew:                          -0.115   Prob(JB):                        0.574
urtosis:                       3.015   Cond. No.                         12.2
==============================================================================

arnings:
[1] Standard Errors assume that the covariance matrix of the errors is correct
y specified.
```

- $\hat{\beta}_1 = 7.2 \pm 1.7$, close to the true value 7 but the confidence interval is huge.
- $\hat{\beta}_2$ is not zero, might be misleading. However the t-stat is large and the truth is that x_2 has not direct effect on the response.
- $\hat{\beta}_3 = 5.5 \pm 1.5$ close to the true value 5 but the confidence interval is huge.

Including more information improves the quality of the interpretation, this conlcusion is made only because we know the true data generation process. However we cannot

In [11]:

```
pred = ols_wo_int.predict(sm.add_constant(test[['x_1', 'x_2', 'x_3']]))
pred_df = test.loc[:, ['x_1', 'x_2', 'x_3']]
pred_df['pred'] = pred.values
pred_df['obs'] = test.y.values
# Plot
fig, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4)
pred_df.plot(kind='scatter', x='x_1', y='pred', ax=ax1, title="Pred vs x_1", figsi
ze=(15,2))
pred_df.plot(kind='scatter', x='x_2', y='pred', ax=ax2, title="Pred vs x_2")
pred_df.plot(kind='scatter', x='x_3', y='pred', ax=ax3, title="Pred vs x_3")
pred_df.plot(kind='scatter', x='obs', y='pred', ax=ax4, title="Pred vs Obs")
add_identity(ax4, color='r', ls='--', linewidth=4)
plt.show(fig)
```



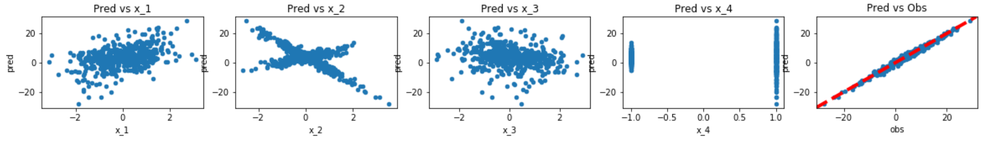
In [12]:

```
# We are lucky, we have all the relevant predictors and we know what is the exact
 DGP
# Using the pairplot, we can therefore build an even better model
y, X = patsy.dmatrices('y ~ x_1 + x_2 * x_4 + x_3', train)
full_ols = sm.OLS(y, X).fit()
print(full_ols.summary())
```

```
                            OLS Regression Results
==============================================================================
 ep. Variable:                      y   R-squared:                       0.979
 odel:                            OLS   Adj. R-squared:                  0.979
 ethod:                 Least Squares   F-statistic:                     4665.
 ate:                Fri, 27 Sep 2019   Prob (F-statistic):               0.00
 ime:                        11:45:33   Log-Likelihood:                -704.91
 o. Observations:                 500   AIC:                             1422.
 f Residuals:                     494   BIC:                             1447.
 f Model:                           5
 ovariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
 ntercept       3.9572      0.045     88.456      0.000       3.869       4.045
 _1             6.9524      0.266     26.123      0.000       6.430       7.475
 _2            -0.0310      0.116     -0.267      0.790      -0.260       0.198
 _4             0.0108      0.045      0.241      0.809      -0.077       0.099
 _2:x_4        -5.9881      0.043   -137.733      0.000      -6.074      -5.903
 _3             4.9619      0.246     20.146      0.000       4.478       5.446
==============================================================================
 mnibus:                        4.458   Durbin-Watson:                   1.996
 rob(Omnibus):                  0.108   Jarque-Bera (JB):                4.250
 kew:                           0.216   Prob(JB):                        0.119
 urtosis:                       3.133   Cond. No.                         12.3
==============================================================================

 arnings:
[1] Standard Errors assume that the covariance matrix of the errors is correct
 y specified.
```

In [13]:

```
# Full example
y, X = patsy.dmatrices('y ~ x_1 + x_2 * x_4 + x_3', test)
pred = full_ols.predict(X)
pred_df = pd.DataFrame(X)
pred_df.columns = ["Intercept", "x_1", "x_2", "x_4", "x_2:x_4", "x_3"]
pred_df['pred'] = pred
pred_df['obs'] = test.y.values
# Plot
fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(1, 5)
pred_df.plot(kind='scatter', x='x_1', y='pred', ax=ax1, title="Pred vs x_1", figsi
ze=(20,2))
pred_df.plot(kind='scatter', x='x_2', y='pred', ax=ax2, title="Pred vs x_2")
pred_df.plot(kind='scatter', x='x_3', y='pred', ax=ax3, title="Pred vs x_3")
pred_df.plot(kind='scatter', x='x_4', y='pred', ax=ax4, title="Pred vs x_4")
pred_df.plot(kind='scatter', x='obs', y='pred', ax=ax5, title="Pred vs Obs")
add_identity(ax5, color='r', ls='--', linewidth=4)
plt.show(fig)
```



In the last model, everything is right. We used the exact functional form for the regression which means that we knew the DGP beforehand (as we know Newton's law before performing a ballistic experiment). We can in this particular case *explain* the regression model. Note it's far to be frequent.

Unknown data generating process (DGP), non-linear models and models interpretation

Most of the time, we don't have a clue of what is the DGP and we often use a non-linear model. We want at least to know how the response is *correlated* to the set of predictors we have. For that, we use interpretable models (meta-models on top of the actual one or importance attribution).

NB: we don't have *full information* (here meaning, we might lack significant predictors). This and the lack of knowledge about the DGP is why we cannot conclude anything from the correlation between the response and the predictors.

LightGBM as an example

Pretend that we don't know anything about the DGP and fit a *naive* GBM (I'll use lightGBM flavour, the tastier for now with Catboost for those who like kittens)

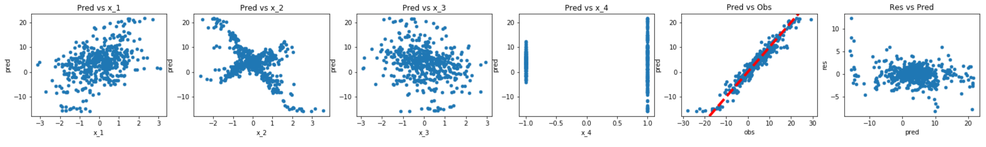
Side note: roughly an interaction is when the slope is modulated by other predictor(s)

In [14]:
```
# Naive lightGBM without optimization
y = train['y']
X = train[['x_1', 'x_2', 'x_3', 'x_4']]
regr = lgb.LGBMRegressor()
regr.fit(X, y)
```

Out[14]:
```
GBMRegressor(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
             importance_type='split', learning_rate=0.1, max_depth=-1,
             min_child_samples=20, min_child_weight=0.001, min_split_gain=0.
,
             n_estimators=100, n_jobs=-1, num_leaves=31, objective=None,
             random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
             subsample=1.0, subsample_for_bin=200000, subsample_freq=0)
```

In [15]:

```
y = test['y']
X = test.loc[:, ['x_1', 'x_2', 'x_3', 'x_4']]
y_lgb = regr.predict(X)
#y_xbart = xbart.predict(X)
pred_df = X.copy()
pred_df['pred'] = y_lgb #y_xbart #
pred_df['obs'] = test.y.values
pred_df['res'] = pred_df['pred'] - pred_df['obs']
# Plot
fig, (ax1, ax2, ax3, ax4, ax5, ax6) = plt.subplots(1, 6)
pred_df.plot(kind='scatter', x='x_1', y='pred', ax=ax1, title="Pred vs x_1", figsi
ze=(27,3))
pred_df.plot(kind='scatter', x='x_2', y='pred', ax=ax2, title="Pred vs x_2")
pred_df.plot(kind='scatter', x='x_3', y='pred', ax=ax3, title="Pred vs x_3")
pred_df.plot(kind='scatter', x='x_4', y='pred', ax=ax4, title="Pred vs x_4")
pred_df.plot(kind='scatter', x='obs', y='pred', ax=ax5, title="Pred vs Obs")
add_identity(ax5, color='r', ls='--', linewidth=4)
pred_df.plot(kind='scatter', x='pred', y='res', ax=ax6, title="Res vs Pred")
plt.show(fig)
```



Let's explain this lightGBM model (note that exact results exist in the litterature for tree based ensemble models)

ICE and PDP plots

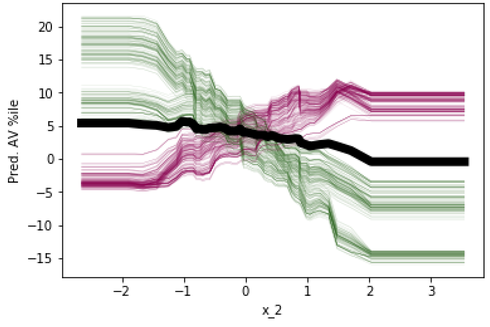
- ICE (Individual Conditional Expectation) plots display one line per instance that shows how the instance's prediction changes when a feature changes. So ICE plots are local interpretation, meaning at the row/policy level.
- The PDP (The partial dependence plot) is the average effect (average of ICE curves) of a feature and therefore is a global (at portfolio/dataset level) interpretation.

If you want more details, here is a great e-book

- The PDP limited to 2D visualisation, might not render interactions, assumes independence of the predictors, as it averages the individual effect it does not render heterogeneity
- The ICE cannot be surface plotted meaningfully, quickly overplotting when many rows in the data set (eventually, the plot will just be filled in with curves)

In [16]:

```
# purple and green light lines are the ICE curves, the black bold one is the PDP
# We can observe the lightGBM detects well the non-linearities in the test set
ice_df = ice(X, 'x_2', regr.predict, num_grid_points=50)
ice_plot(ice_df, alpha=.2, linewidth=0.3, plot_pdp=True, color_by='x_4', pdp_kwarg
s={'c': 'k', 'linewidth': 7}, cmap='PiYG')
plt.ylabel('Pred. AV %ile')
plt.xlabel('x_2');
```



- Bold black line: PDP, similar to the curve returned by Emblem. It cannont detect interactions.
- Light lines: ICE curves, render heterogeneity and interaction but quickly *overplot* if many observations

For a more complete discussion see Interpretable Machine Learning

VINE (Visual INteraction Effects)

A new method, brefly: it smartly consists in clustering ICE curves to avoid overplotting and provides a mesoscale interpretation (between local/row/policy and global/dataset/portfolio levels). Limited to tabular data (not suited to text or images).

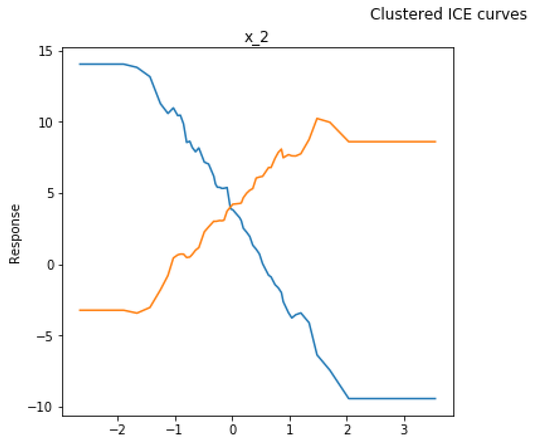
In [17]:
```
vine_dic=noglmvinelite.export(X, y, regr.predict, num_clusters=2, num_grid_points=
50)
```

```
------------------------------------------------
passed time vine for 500 rows:      1.25 s (   0.02 min, 0.00 h)
```

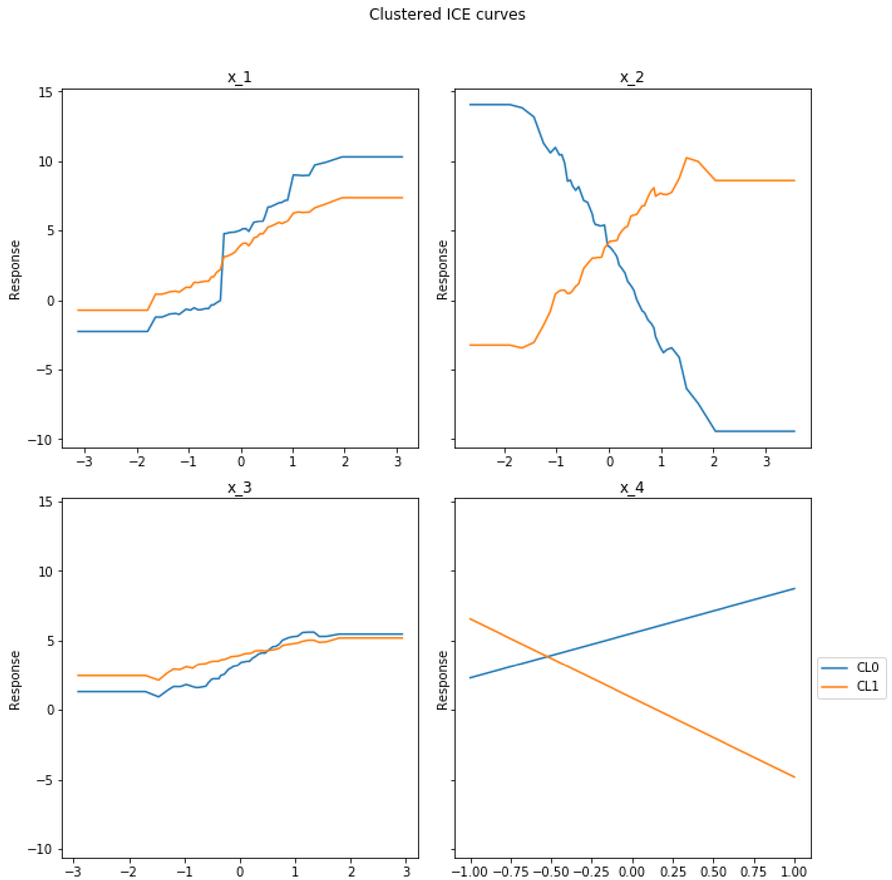
In [18]:

```
df = noglmvinelite.get_df_vine(vine_dic=vine_dic, features=['x_1', 'x_2', 'x_3',
'x_4'])
vine_df = noglmvinelite.get_df_vine(vine_dic, ['x_2'])
noglmvinelite.plot_vine(vine_dic, ['x_2'], ax_ylabel='Response', log_scale=False,
nrows=1, ncols=2, figsize=(10, 5))
```

```
No handles with labels found to put in legend.
```



In [19]:
```
noglmvinelite.plot_vine(vine_dic, ['x_1', 'x_2', 'x_3', 'x_4'], ax_ylabel='Respons
e', log_scale=False, nrows=2, ncols=2, figsize=(10, 10))
```



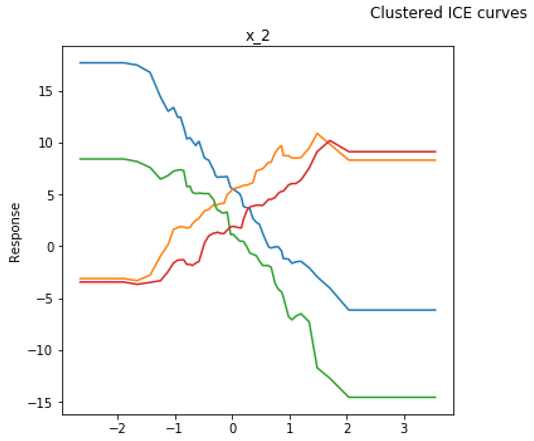
In the last figure, left panels, the curves are parellel. We can conclue that no interaction terms are involved. Moreover if we chose a different number of clusters (4, for instance) we would have as output:

In [20]:

```
vine_dic=noglmvinelite.export(X, y, regr.predict, num_clusters=4, num_grid_points=
50)
df = noglmvinelite.get_df_vine(vine_dic=vine_dic, features=['x_1', 'x_2', 'x_3',
'x_4'])
vine_df = noglmvinelite.get_df_vine(vine_dic, ['x_2'])
noglmvinelite.plot_vine(vine_dic, ['x_2'], ax_ylabel='Response', log_scale=False,
nrows=1, ncols=2, figsize=(10, 5))
```

```
--------------------------------------------------
passed time vine for 500 rows:      1.33 s (    0.02 min, 0.00 h)
```

```
No handles with labels found to put in legend.
```



The parallel curves can be grouped together and we end up with only two clusters. There is a nicer way to group clusters but I'll let that aside for the interested reader (see the VINE paper)

Shapley values

Old result of game theory (1953) which is the only possible consistent and locally accurate additive feature attribution method based on expectations.

Shapley interpreter, currently the golden interpreter. Provides both at the row (micro/local) and portfolio (macro/gloal) level explanations, explore interactions. It can be proven that the linear explainer built using Shapley values is the unique explainer, all the other linear explainers being a particular case of the SHAP one (see this reference and this reference).

Technical blabla

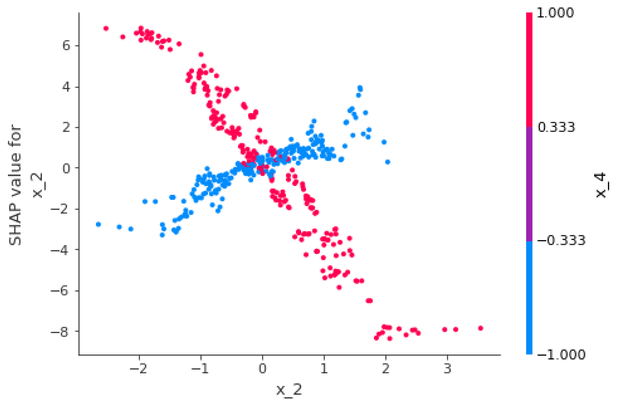
The SHAP interpreter is a linear additive one build on rational axioms and take the form:

$$g(z) = \phi_0 + \sum_{i=0} \phi_i z_i$$

where the z_i are simplified (scaled) predictors, ϕ_i are the so-called Shapley values and $g(z)$ the model's prediction (minus the average). In the case of a simple linear model, the Shapley values ϕ_i are nothing but the regression coefficients. Observing only a single feature at a time implies that dependencies between features are not taken into account, which could produce inaccurate and misleading explanations of the model's decision-making process. This is where cooperative game theory enters: feature values of an instance work together to cause a change in the model's prediction with respect to the model's expected output, and it divides this total change in prediction among the features in a way that is "fair" to their contributions across all possible subsets of features (see this blog). Note that as explained in the previous section, for tree-based models, there is an exact explainer which is much faster (and equivalent to) compute the Shapley values. See Tree SHAP paper

In [22]:

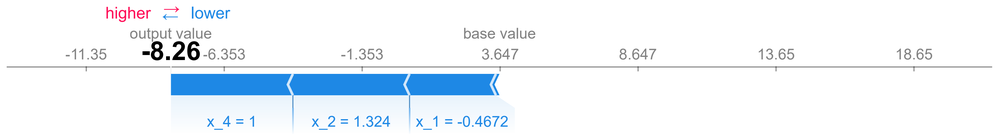
```
explainer, shap_values = treeshapvalues(regr, X)
shap.dependence_plot("x_2", shap_values, X) #, interaction_index="x_4")
```



The most relevant predictor for the interaction term is picked up automatically. This is a *global* interpretation (average effect over the whole data set). The Shapley interpreter is also able to interpret at the local scale (row/policy level) but it should not be generalized to the global/portfolio level. Indeed local and global interpretations might be in opposition. See next slide for an example.

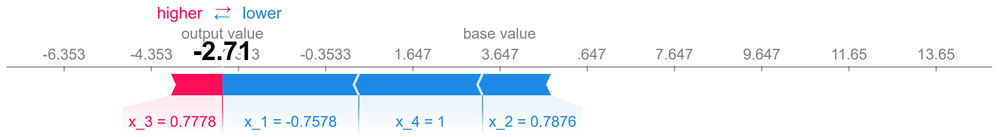
In [23]:
```
shap.force_plot(explainer.expected_value, shap_values[0,:], X.iloc[0,:])
```

Out[23]:



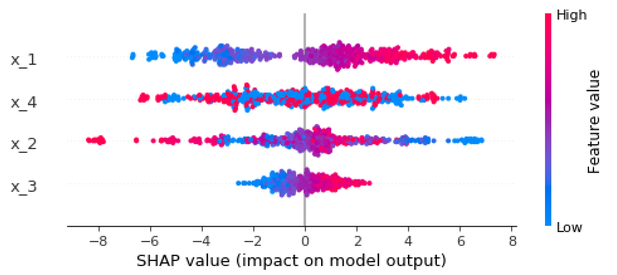
In [24]:
```
shap.force_plot(explainer.expected_value, shap_values[25,:], X.iloc[25,:])
```

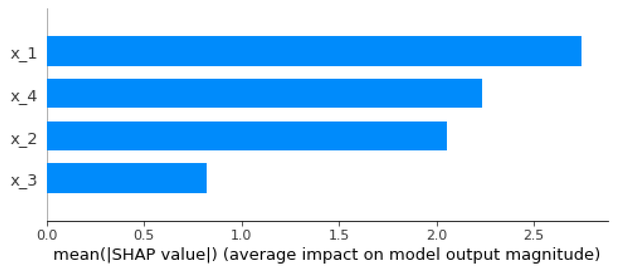
Out[24]:



The Shapley interpreter is versatile. It returns a summary plot and variable importance. For a complete review see the resources at the last slides.

In [26]:
```
summaryplot(shap_values, X)
```





Resources

- The Mythos of Model Interpretability
- Interpretable machine learning
- Peeking Inside the Black Box
- Shapley values
- VINE
- R DALEX
- Video tutorial