Examen de statistiques descriptives

(Jérôme Lacaille et Florent Forest)

Vous soumettrez votre réponse sous la forme d'un fichier notebook (MACS3-SD20-Prenom_Nom.ipynb) que vous enverez par courrier électronique avant le 1er décembre aux deux adresses suivantes :

- jerome.lacaille@gmail.com (mailto:jerome.lacaille@gmail.com)
- forest@lipn-univ-paris13.fr (mailto:forest@lipn-univ-paris13.fr).

Ce notebook utilise la toolbox 'tabata' que vous trouverez sur GitHub comme précisé dans le cours : https://github.com/jee51/tabata (https://github.com/jee51/tabata).

L'examen se compose de plusieurs questions, chaque question compte pour un nombre de points défini. La note totale sera la somme des points obtenus par votre présence lors des cours et TD (1/2 point par séance, soit 6 points en tout en comptant les TD), plus les points accumulés par cet examen (20 points), la note finale sera majorée à 20.

Attention cependant, toute journée de retard compte pour un demi point de moins et une recopie évidente (plagiat) du travail d'un de vos camarades compte pour -4 points par contrevenant.

Deadline avant pénalité : le 1er décembre à 0h00 (30 novembre minuit).

Bon travail !

Corrections

Le tableau ci-dessous servira d'évaluation, vous voyez ainsi comment la notation sera découpée en éléments de base. Surtout n'hésitez pas à commenter vos codes et ne laissez pas d'affichages sans légende.

Question	Elément	Points	Note
A Théorie	A.2 Proba/Stats	+1	
	A.3 Modèles linéaires	+1	
	A.4 Modèles non linéaires	+1	
	A.5 Apprentissage	+1	

Question	Elément	Points	Note
	A.6 Machines de Boltzmann	+1	
---	---	---	---
B Pratique	B.1 Une interface data	+3	
	B.2 Comprendre un algo	+6	
	B.3 Créer un modèle	+6	
---	---	---	---

Les zones en jaune, rédigées en rouge, ci-dessous sont les questions qui vous sont posées.

A. Questions de cours

Cette partie contient quelques questions de cours permettant de m'assurer que les étudiants qui n'ont pas pu suivre un cours ont visionné la présentation en différé. Ces questions sont numérotées de 2 à 6, sachant que le premier cours a été présenté à l'université, les absents s'étant excusés (ceux qui ont suivi tous les cours et TD ont déjà 6 points d'avance).

Il s'agit de 5 couples de questions simples auxquelles vous répondrez dans la zone de texte juste après la question. Pour chaque cours une question porte sur la partie théorique et une autre sur les présentations industrielles.

A.2. "Probabilités et statistiques"

Jeudi 24 septembre.

A.2.1. Qu'est-ce qu'un boxplot ?

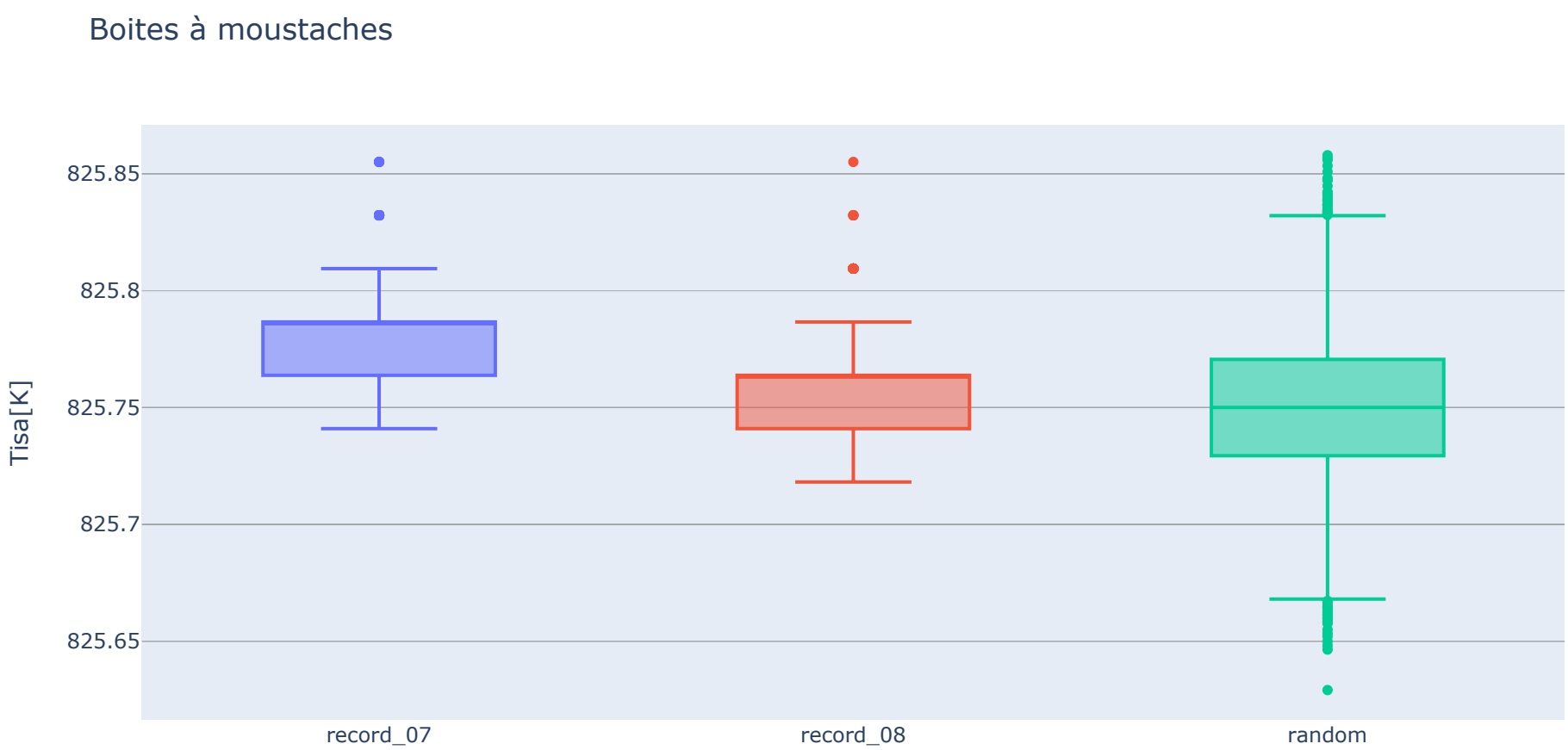
Un boxplot (boite à moustache en français) est un affichage graphique statistique qui permet facilement de représenter la moyenne quelques quantiles empirique de la distribution sous-jacente à un ensemble de mesures.

Entrée [1]:
```
import numpy as np
import pandas as pd
import plotly.graph_objects as go
import tabata as tbt
```

Je vais prendre ici comme exemple deux avions qui sont restés au sol et regarder la distribution des températures, puis comparer avec une distribution gaussienne.

Entrée [2]:
```
ds0 = tbt.Opset(tbt.opset.datafile('AFL1EB.h5'))
x0 = ds0[7]['Tisa[K]']
x1 = ds0[8]['Tisa[K]']
z = 825.75 + 0.03*np.random.randn(10000)
```

Entrée [3]:
```
fig = go.Figure(
    [go.Box(y=x0, name=x0.index.name),
     go.Box(y=x1, name=x1.index.name),
     go.Box(y=z, name='random')],
    go.Layout(title='Boites à moustaches', yaxis_title='Tisa[K]', showlegend=False))
fig.show()
```



La médiane se trouve au milieu de la boite (trait plein de la boite verte et en haut des boites bleues et rouges). Les limites hautes et basses de la

boite correspondent quax quantiles Q1=q(0.25) et Q3=q(0.75). Les traits (moustaches, *whiskers*) au dessus et en dessous ont une longueur égale à 1.5 fois l'IQR (Inter Quantile Range Q3-Q1) depuis la médiane et s'arrêtent au dernier point observé dans ce domaine. Les points au delà sont considérés comme des outliers.

A.2.2. Décrivez l'algorithme très simple qui a été utilisé pour surveiller des mesures sur des bancs d'essais de moteurs (on en a aussi un peu reparlé la semaine suivante).

Le problème était de s'assurer que les mesures faites pendant les essais étaient enregistrées convenablement et que les capteurs n'avaient pas de problème. On recherchait aussi des anomalies, mais il fallait que ce soit interprétable.

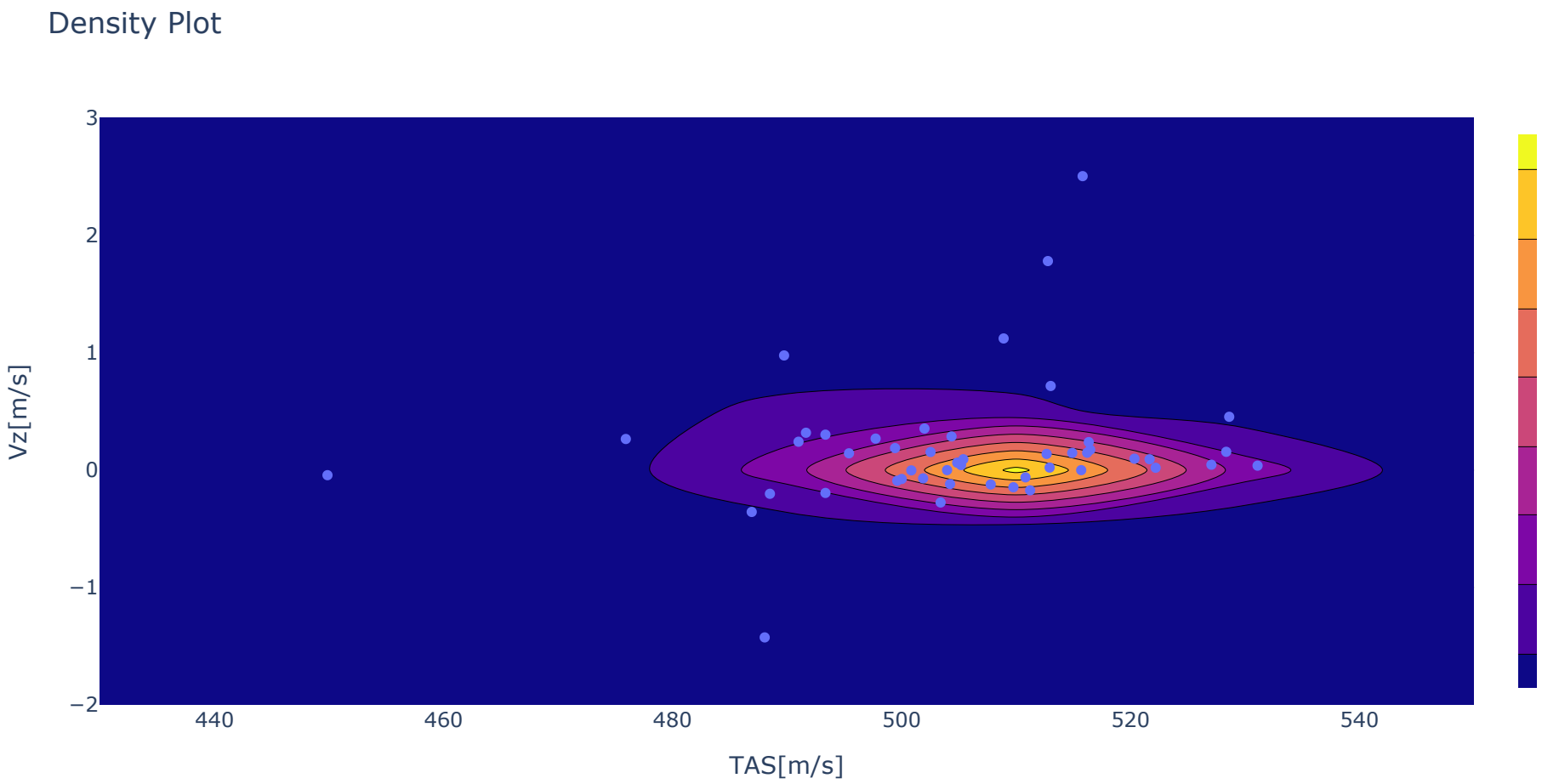
Pour cela on avait affiché les mesures par couples et défini les gabaris à 3-sigmas et 6-sigmas dans lesquels les points devaient se trouver.

Des outils graphiques classiques permettent d'avoir des statistiques équivalentes. Par exemple on peut afficher un 'density plot', c'est un histogramme en deux dimensions.

On crée une table correspondant aux observations mesurées à l'altitude maximale pour le jeu de données nettoyées puis on affiche un couple de variables avec des lignes de niveau correspondant à la densité locale d'observations.

Entrée [4]:
```
cols = ds0[0].columns # On ne garde que les colonnes initiales (suppression des phases CR et CL)
dsc = tbt.Opset(tbt.opset.datafile('AFL1EB_C.h5')) # jeu de données nettoyées.
df = pd.DataFrame([df.loc[df['ALT[m]'].idxmax(),cols].values for df in dsc],
                  columns=cols,index=dsc.records)
```

Entrée [5]:
```
vx = cols[2] # La vitesse air.
vy = cols[3] # La vitesse de montée.
fig = go.Figure([
    go.Scatter(x=df[vx],y=df[vy],mode='markers',hoverinfo='text',hovertext=df.index),
    go.Histogram2dContour(x=df[vx],y=df[vy],
                          histnorm='probability',hoverinfo='skip')],
    go.Layout(title='Density Plot',xaxis_title=vx, yaxis_title=vy,showlegend=False)
)
fig.show()
```



Chaque ligne de niveau correspond à une probabilité de se trouver dans la zone délimitée. On voit tout de suite que certaines données sont un peu décalées. Il sera intéressant de regarder ces vols.

A.3. "Modèles linéaires"

Jeudi 1er octobre

A.3.1. A quoi sert le Lasso ?

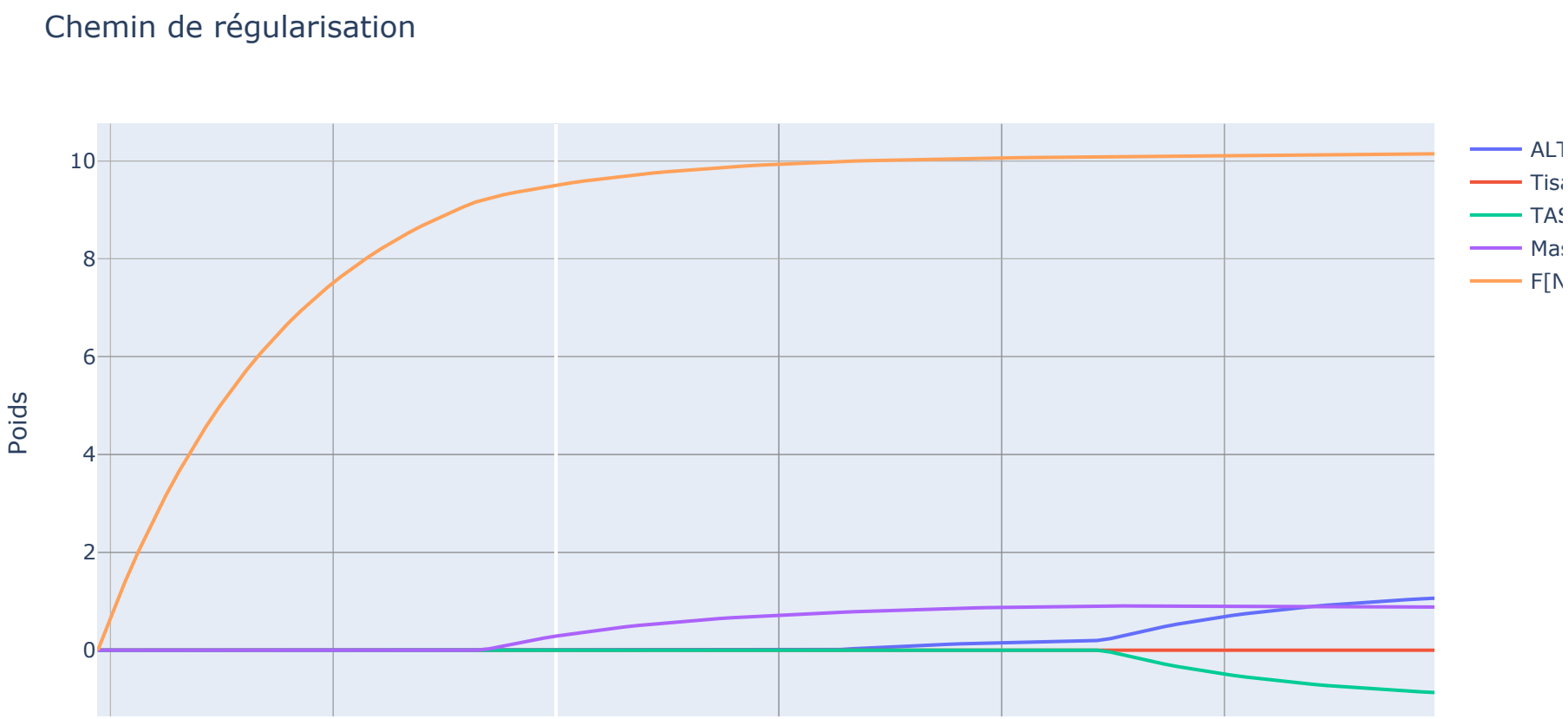
Le LASSO est un algorithme de sélection de variables pour un modèle linéaire. On ajoute au carré des erreurs une pénalité correspondant à la norme L1 des poids (la somme des valeurs absolue des poids). Du coup minimiser l'erreur et une pondération de cette norme va conduire à choisir un vecteur de poids sur l'hypercube défini par une iso-norme L1, donc probablement, un coin, un coté, et donc un nombre important de paramètres nuls. Cela permet d'identifier les facteurs importants pour un modèle linéaire en éliminant ceux qui ne le sont pas.

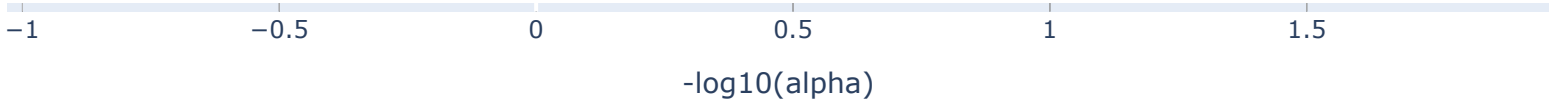
Pour illustrer cela on utilise les données du premier vol et on essaye de comprendre la vitesse de montée.

Entrée [6]:
```
from sklearn.linear_model import lasso_path
from sklearn.preprocessing import StandardScaler
```

Entrée [7]:
```
df = dsc[0][cols] # On ne garde que les colonnes numériques.
vz = 'Vz[m/s]'
y = df[vz]
X = df.drop(columns=[vz])
SC = StandardScaler() # On normalise les données pour avoir des poids d'ordre équivalents.
XS = SC.fit_transform(X)
alphas_lasso, coefs_lasso, _ = lasso_path(XS, y)

# Affichage
data = [go.Scatter(x=-np.log10(alphas_lasso), y=coefs_lasso[i], mode="lines", name=X.columns[i])
        for i in range(len(X.columns))]
fig = go.Figure(data)
fig.update_layout(title="Chemin de régularisation",
                  yaxis_title="Poids", xaxis_title="-log10(alpha)")
fig.show()
```





L'axe des abscisse correspond au paramètre de compensation α dont le coût optimisé est défini par la formule suivante dans le package scikit-learn :

$$\frac{1}{2n} * ||y - Xw||_2^2 + \alpha||w||_1$$

Ainsi, plus l'abscisse augment, plus α est petit et donc moins on a de contrainte sur les poids w, c'est pourquoi le nombre de paramètres non nuls augmente.

Comme prévu, la vitesse de montée dépend essentiellement de la poussée, puis un peu de la masse de l'avion. Les autres facteurs n'arrivent que tardivement.

A.3.2. Pourquoi doit-on rendre des mesures indépendantes du contexte d'acquisition pour surveiller un moteur ? Quelle solution est proposée dans le cours ?

Si les mesures dépendent du contexte, alors elles ne sont plus comparables. Une solution consiste à supprimer la dépendance au contexte. Par exemple, dans le cours on a utilisé une régression sur le contexte, puis on travaille sur les résidus.

Bien sûr, pour que les données soient a nouveau bien dimensionnées on les ramène dans leur domaine et dimensions respectives.

Une application très simple consiste à étudier la consommation moyenne par passager et par heure. On voudrait savoir si elle est toujours à peu près identique. Hors, on sait que cette consommation est égale à la quantité de kérosène utilisée divisée par la durée du vol. Mais elle dépend aussi de la façon de piloter l'avion, regardons par exemple l'altitude de croisière et l'énergie dépensée. Il faudrait supprimer ces informations pour avoir une consommation moyenne comparable de vol à vol.

Entrée [8]:
```
from sklearn.linear_model import LinearRegression
from plotly.subplots import make_subplots
```

Entrée [9]:
```
densite_kerosene = 800/1000 # kg/l
nb_pax = 150 # Nombre de passagers d'un A320.
m = 'Masse[kg]'
df = pd.DataFrame([[(dfi.index[-1]-dfi.index[0]).total_seconds(),
                    dfi[m].iloc[0]-dfi[m].iloc[-1],
                    dfi['ALT[m]'].max(),
                    dfi['F[N]'].sum()]
                   for dfi in dsc],
                   columns=['Durée','Kero','ALT','E'],
                   index=dsc.records)
df.head()
```

Out[9]:

	Durée	Kero	ALT	E
/record_00	7428.0	1425.053343	10785.335808	550879.956303
/record_01	7458.0	1455.083250	10790.407680	433904.604054
/record_02	7041.0	1320.714059	11164.458240	350041.886511
/record_03	6456.0	1168.311592	11489.058048	-873092.861898
/record_04	4859.0	897.886362	9196.571904	271790.367146

Entrée [10]:
```
y = df['Kero']
X = df.drop(columns=['Kero'])
cols = X.columns
SC = StandardScaler()
X = pd.DataFrame(SC.fit_transform(X),columns=cols, index=df.index)
reg = LinearRegression().fit(X,y)
yh = reg.predict(X)
conso_normalisee = y.mean() + (y-yh)
reg.coef_
```

Out[10]:
```
array([383.4367107 ,  -3.40496388,  34.29306553])
```

Les coefficients ne semblent pas négligeables. Pour le vérifier le plus simple en Python est d'utiliser statmodels.

Entrée [11]:
```
import statsmodels.api as sm
```

Entrée [12]:
```
X1 = sm.add_constant(X)
rg = sm.OLS(y,X1).fit()
print(rg.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   Kero   R-squared:                       0.978
Model:                            OLS   Adj. R-squared:                  0.977
Method:                 Least Squares   F-statistic:                     659.3
Date:                Sun, 06 Dec 2020   Prob (F-statistic):           1.43e-36
Time:                        16:43:28   Log-Likelihood:                -261.11
No. Observations:                  48   AIC:                             530.2
Df Residuals:                      44   BIC:                             537.7
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       1428.0775      8.404    169.935      0.000    1411.141    1445.014
Durée        383.4367     10.700     35.836      0.000     361.873     405.001
ALT           -3.4050     10.747     -0.317      0.753     -25.063      18.253
E             34.2931      8.824      3.886      0.000      16.509      52.077
==============================================================================
Omnibus:                        0.488   Durbin-Watson:                   1.769
Prob(Omnibus):                  0.784   Jarque-Bera (JB):                0.502
Skew:                          -0.222   Prob(JB):                        0.778
Kurtosis:                       2.769   Cond. No.                         2.15
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

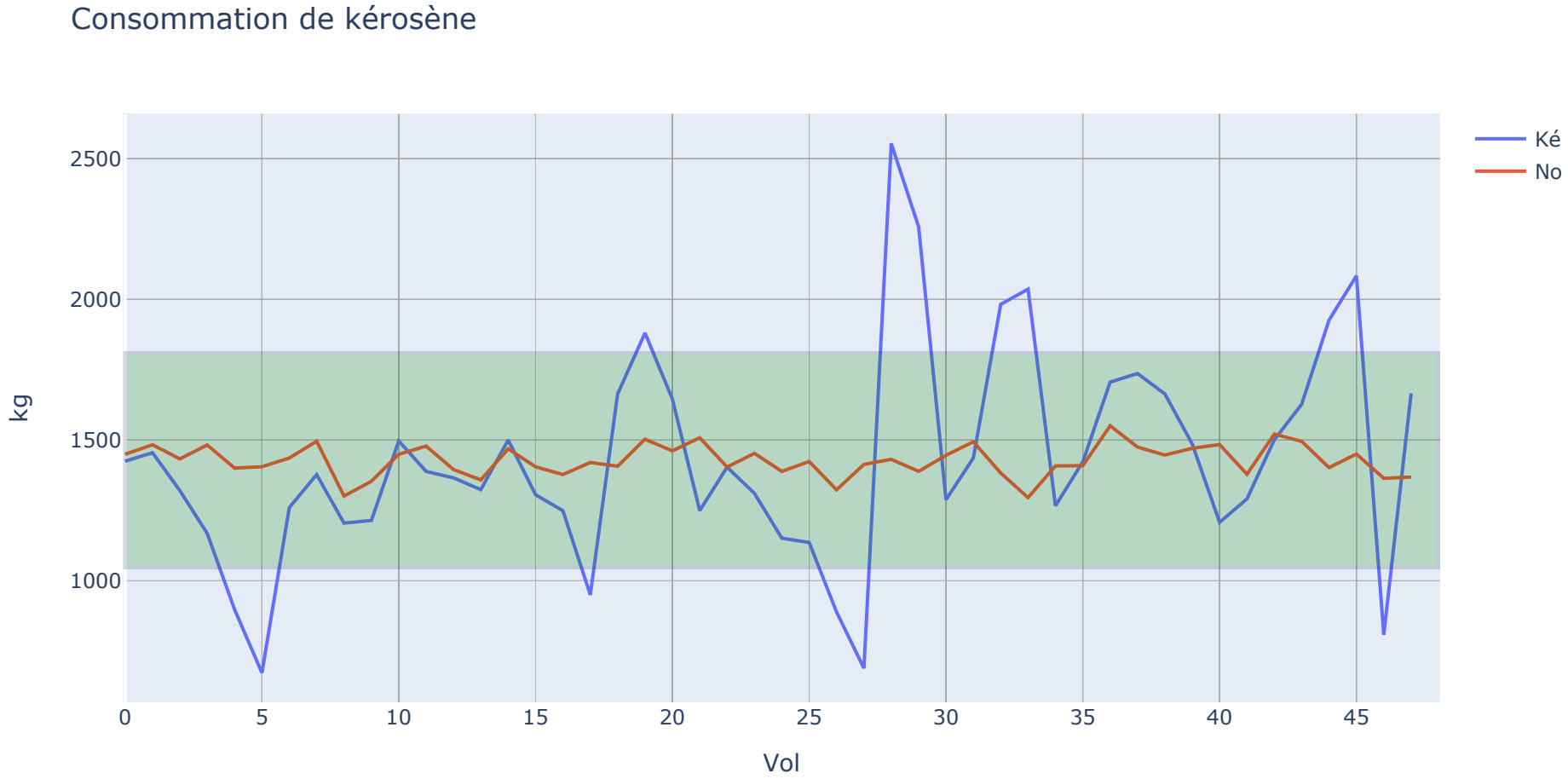
La régression est très informative car elle explique presque 98% de la variance. On retrouve bien les coefficients obtenus avec scikit-learn, mais on a des informations supplémentaires, par exemple, la dépendance en fonction de l'altitude est finalement négligeable : la p-value est trop grande et l'intervalle de confiance contient 0.

Le package `scikit-learn` est très vaste et permet de développer de nombreux algorithmes. Par contre le package `statsmodels`, beaucoup plus restreint, est pratique pour obtenir des résultats statistiques comme sous R.

Entrée [13]:
```
y.var(), conso_normalisee.var()
```

Out[13]:
```
(145831.1055747204, 3173.4500651779795)
```

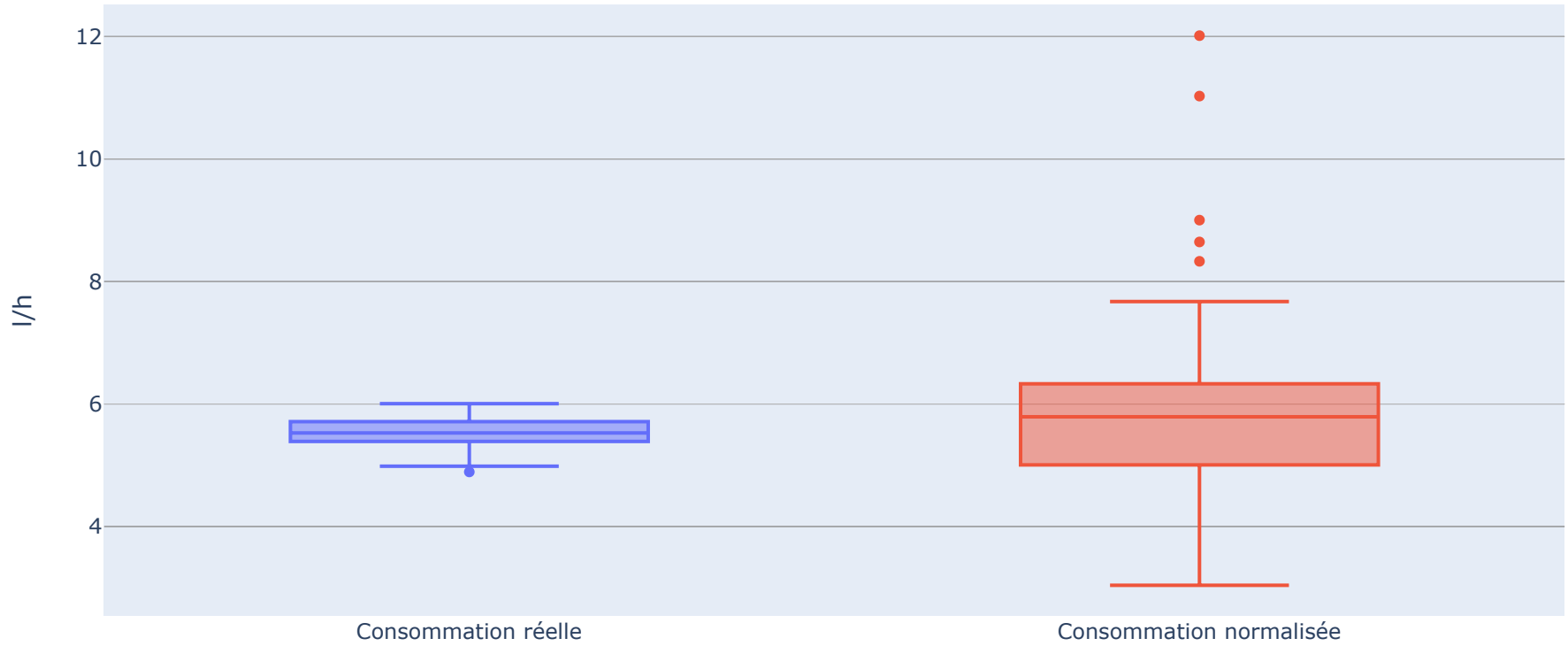
Entrée [14]:
```
fig = go.Figure()
fig.add_trace(go.Scatter(y=y,name='Kérosène'))
fig.add_trace(go.Scatter(y=conso_normalisee,name='Normalisé'))
m = y.mean()
s = y.std()
fig.add_shape(type='rect',x0=0,x1=len(y),y0=m-s,y1=m+s,fillcolor='green',opacity=0.2)
fig.update_layout(title="Consommation de kérosène",
                  yaxis_title="kg", xaxis_title="Vol")
fig.show()
```



On remarque que la variance du calcul de la quantité de kérosène estimée diminue quand on normalise par la poussée et l'altitude de croisière.

Entrée [15]:
```
c_normal = 3600/densite_kerosene/nb_pax*conso_normalisee/df['Durée']
c_reelle = 3600/densite_kerosene/nb_pax*y/df['Durée']
```

Entrée [16]:
```
fig = go.Figure(
    [go.Box(y=c_reelle, name='Consommation réelle'),
     go.Box(y=c_normal, name='Consommation normalisée')],
    go.Layout(title='Calcul de la consommation moyenne par passager',
              yaxis_title='l/h', showlegend=False))
fig.update_traces(hoverinfo='text',hovertext=df.index)
fig.show()
```



En divisant par la durée pour avoir la consommation instantanée, les données deviennent plus intéressantes, on peut d'ailleur remarquer certains

vols ayant surconsommé et d'autres qui semblent être plus efficaces. Il serait intéressant de comprendre pourquoi et de proposer de bonnes pratiques de pilotage. (Passez la souris sur les outliers pour repérer les enregistrements correspondants.)

A.4. "Modèles non linéaires"

Jeudi 8 octobre

A.4.1. Expliquez comment on construit une forêt aléatoire pour faire de la classification d'observations ?

La forêt aléatoire est fomée d'une série d'arbres de décision. Pour que tous les arbres soient différents, à chaque étape on tire aléatoirement un sous-ensemble de facteurs. Les chemins de décisions vont donc être assez différents parmis les arbres.

On crée un tableau de données correspondant à des indicateurs descriptifs de la croisière et on va essayer de trouver un moyen de différentier les vols qui consomment le plus.

Entrée [17]:
```
data = []
for df in dsc:
    df = df[df['CR']].copy()
    data.append([(df.index[-1]-df.index[0]).total_seconds(), # Durée de croisière
                 df['Masse[kg]'].iloc[0]-df['Masse[kg]'].iloc[-1], # Kérosène
                 df['ALT[m]'].max(), # Altitude maximale
                 df['ALT[m]'].mean(), # Altitude moyenne
                 df['Vz[m/s]'].max(),
                 df['Vz[m/s]'].mean(),
                 df['Vz[m/s]'].std(),
                 df['Tisa[K]'].mean(),
                 df['TAS[m/s]'].max(),
                 df['TAS[m/s]'].mean(),
                 df['F[N]'].sum(), # énergie
                ])
df = pd.DataFrame(data,columns=['Durée','Kero',
                                'ALT_max','Alt_mean',
                                'Vz_max','Vz_mean','Vz_std',
                                'Tisa_mean',
                                'TAS_max', 'TAS_mean',
                                'E'
                               ],index=dsc.records)
df['Conso'] = 3600/densite_kerosene/nb_pax*df['Kero']/df['Durée']
df.drop(columns=['Kero','Durée'],inplace=True)
df.head()
```

Out[17]:

	ALT_max	Alt_mean	Vz_max	Vz_mean	Vz_std	Tisa_mean	TAS_max	TAS_mean	E	Conso
/record_00	10785.335808	10671.133888	0.887659	0.000313	0.155514	637.791590	534.716032	527.918642	-277397.344400	6.376886
/record_01	10790.407680	10593.096765	0.934226	-0.004575	0.183917	639.196258	541.596938	531.647961	-419385.567227	6.303238
/record_02	11164.458240	11093.924898	0.939448	-0.000386	0.099563	630.181352	505.954178	504.782817	-420324.494034	5.934126
/record_03	11489.058048	11410.850362	0.967301	-0.003160	0.143552	624.477115	513.802544	506.824279	-235914.137465	5.745844
/record_04	9196.571904	9192.230074	0.978181	0.001537	0.127055	664.411859	498.221244	497.990777	-70947.402673	6.377413

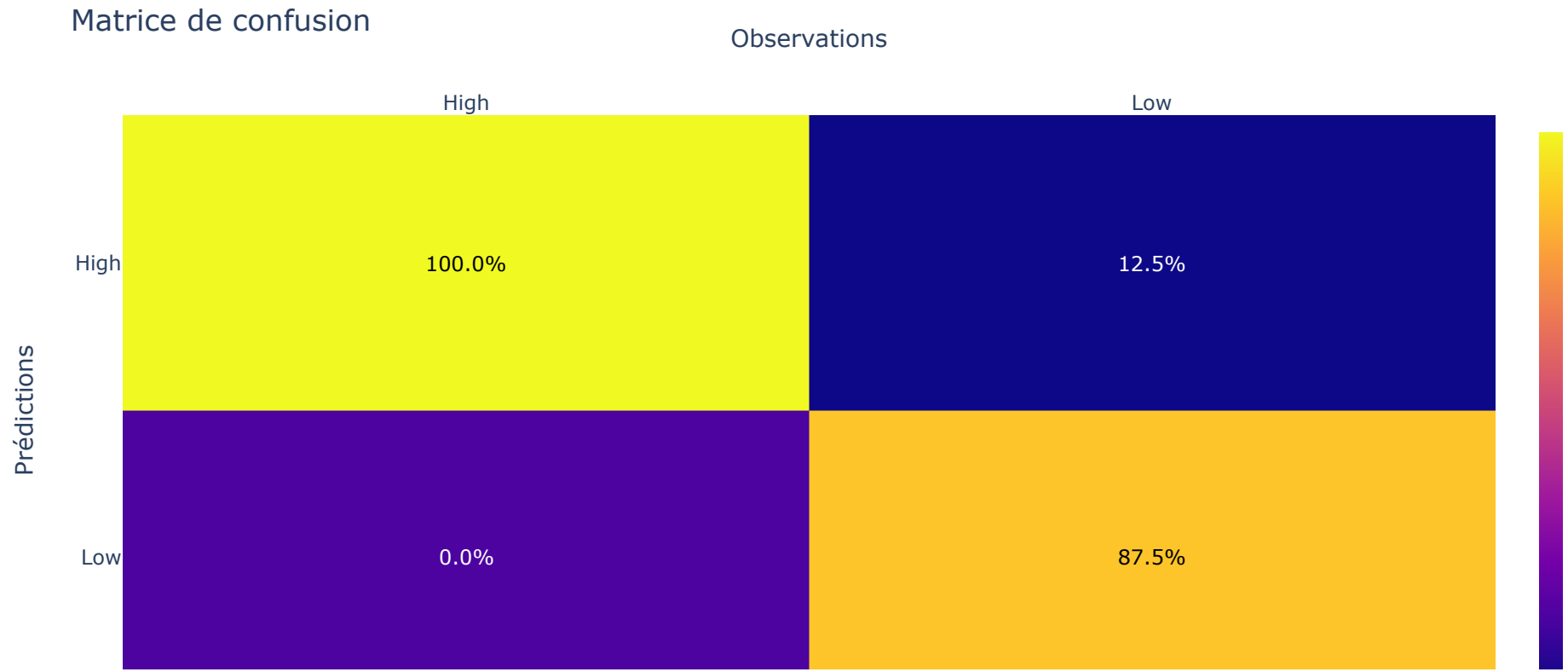
Entrée [18]:
```
thr = df['Conso'].median()
print("La consommation médiane pendant la croisière est de " +
      "{:.1f} litres par passager et par heure de vol.".format(thr))
```

```
La consommation médiane pendant la croisière est de 6.1 litres par passager et par heure de vol.
```

Entrée [19]:
```
from sklearn.ensemble import RandomForestClassifier
```

Entrée [20]:
```
CLS = RandomForestClassifier(n_estimators=10,max_depth=3)
y = df['Conso']>thr
X = df.drop(columns=['Conso'])
CLS.fit(X,y)
yh = pd.Series(CLS.predict(X),index=y.index,name=y.name+'_H')
```

Entrée [21]:
```
tpr = 100*np.sum(y & yh)/np.sum(y)    # True Positive Rate (Recall)
tnr = 100*np.sum(~y & ~yh)/np.sum(~y) # True Negative rate (Specificity)
z = [[tpr, 100-tpr],[100-tnr, tnr]]
levels = ['High', 'Low']
fig = go.Figure([go.Heatmap(z=z,x=levels,y=levels,hoverinfo='skip')])
annotations = []
for i, row in enumerate(z):
    for j, value in enumerate(row):
        fig.add_annotation(x=levels[i], y=levels[j],
                           font_color=('white' if value<50 else 'black'),
                           text="{:.1f}%".format(value),
                           showarrow=False)
fig.update_layout(title="Matrice de confusion")
fig.update_xaxes(title="Observations",side='top')
fig.update_yaxes(title="Prédictions",autorange="reversed")
fig.show()
```



On arrive assez bien à prédire la différence entre les vols qui consomment plus ou moins que la médiane, par contre on n'a pas d'information sur la causalité. Pour cela il vaut mieux utiliser un seul arbre de décision, mais là encore il peut y avoir d'autres solutions. L'outil d'analyse d'influence que je vous ai présenté en cours explore plein d'autres possibilités, la difficulté est de les présenter graphiquement.

(Je vais peut-être utiliser cette idée comme exercice pour le cours de l'an prochain).

A.4.2. Quel a été l'algorithme utilisé pour représenter l'état d'un moteur sur un écran ? A-t-il fallu opérer quelques prétraitements ?

Pour représenter l'état d'un moteur, j'ai utilisé une carte auto-organisatrice (ou SOM : Self Organizing Map). Cette méthode de catégorisation (*clustering* en anglais) utilise l'algorithme d'apprentissage non supervisé développé par Teuvo Kohonen.

L'état du moteur était issu de données récupérées pendant les vols (*snapshots* ACARS). Pour que la carte ait un sens il fallait que les données soient comparables d'un vol au suivant. J'ai donc normalisé ces données.

J'utilise le package `minisom` qui est assez simple et ne propose pas d'affichages spécifiques, ce qui me laisse gérer cela moi-même.

Entrée [22]:
```
from minisom import MiniSom
```

On standardise les données pour que la mesure de similarité n'ait pas à le gérer, par ailleurs, cela permettra par la suite d'avoir une homogénéité des affichages. On peut roujours reconstruire les données dans leur échelle d'origine vec la méthode `.inverse_predict()` .

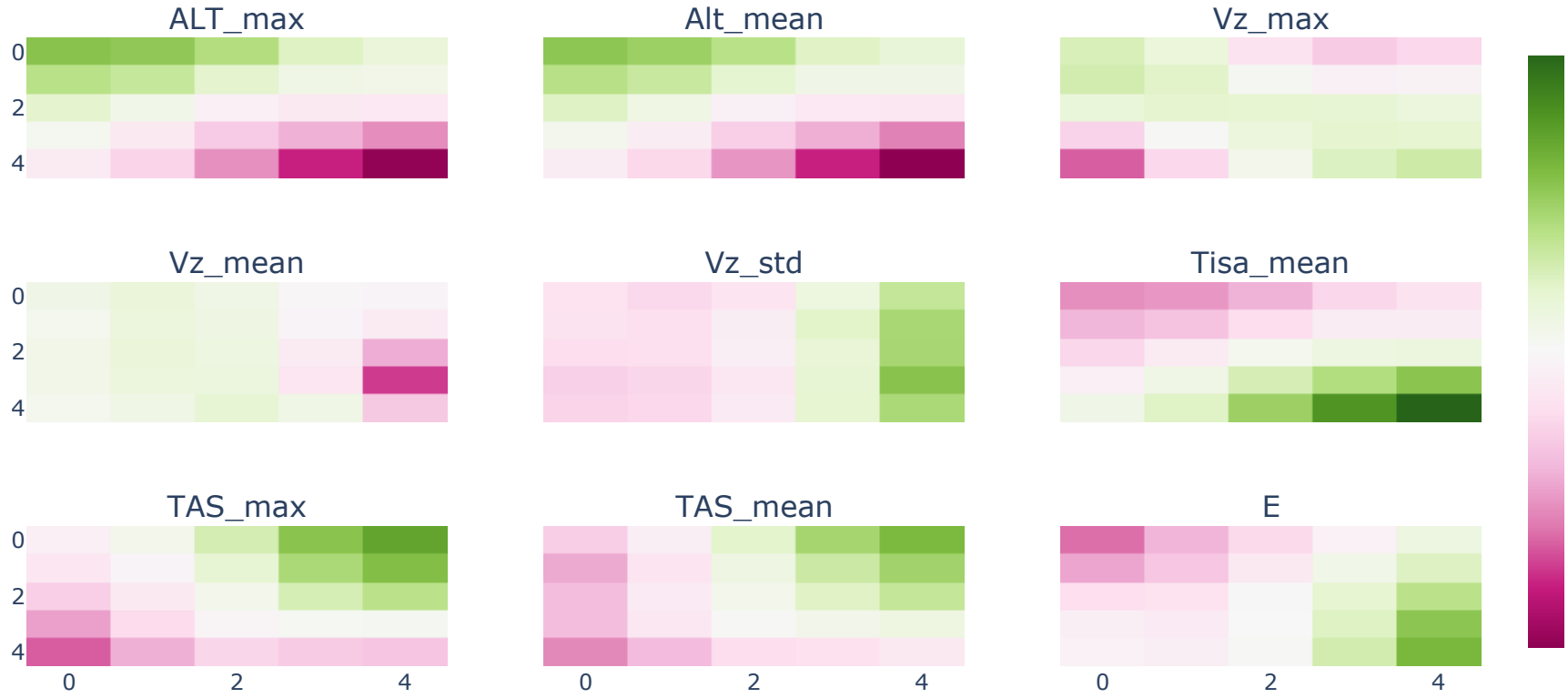
Entrée [23]:
```
SC = StandardScaler()
X = pd.DataFrame(SC.fit_transform(X),columns=X.columns, index=X.index)
```

Entrée [24]:
```
som = MiniSom(5, 5, X.shape[1], sigma=1.5, learning_rate=.5)
som.pca_weights_init(X.values) # Initialisation par une PCA, c'est toujours ça de gagné.
som.train(X.values, 1000, verbose=True)
```

```
[ 1000 / 1000 ] 100% - 0:00:00 left
quantization error: 1.5549592125110543
```

Le retour de la méthode `.get_weights()` est un tableau de tableaux, chacun représentant les poids des vecteurs templates (*codevectors*) de chaque cellule. On peut donc observer l'allure de la classification des vols.

Entrée [25]:
```
W = som.get_weights()
fig = make_subplots(rows=3,cols=3,
                    subplot_titles=X.columns,shared_xaxes=True,shared_yaxes=True)
for k in range(3):
    for l in range(3):
       fig.add_trace(go.Heatmap(z=W[:,:,3*k+l]),row=k+1,col=l+1)
fig.update_traces(coloraxis='coloraxis')
fig.update_yaxes(autorange="reversed")
fig.show()
```

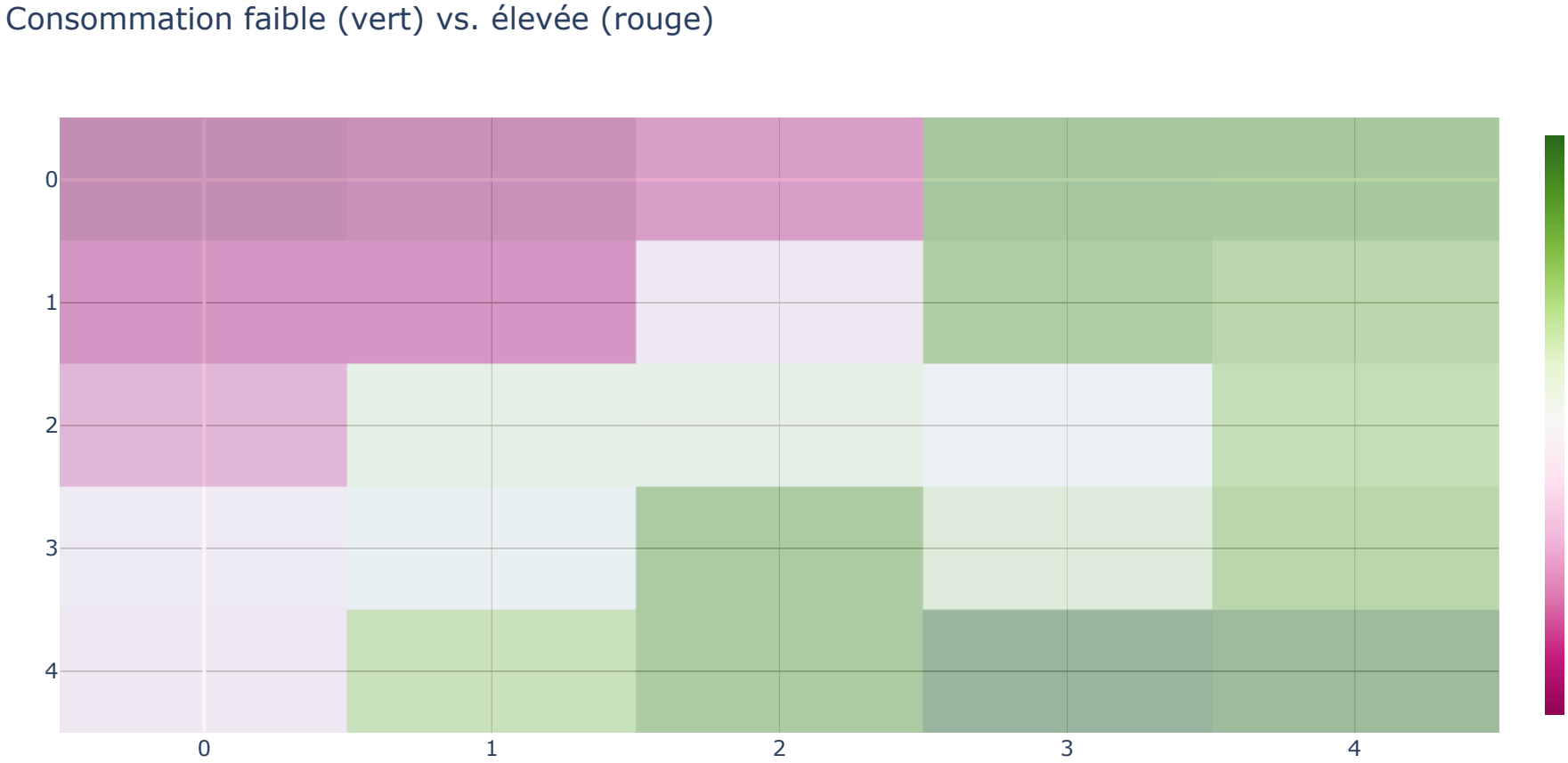


Ensuite on peut regarder la consommation comme un calque que l'on superpose aux images précédentes. Pour cela il faut par exemple calculer la consommation moyenne des observations au sein de chaque cellule, mais comme on a un bon prédicteur calculé à partir d'un RF autant l'utiliser.

Entrée [26]:
```
W0 = SC.inverse_transform(W)
W1 = np.reshape(W0,(-1,9))
C1 = CLS.predict_proba(W1)
```

J'ai pris el parti de renvoyer les probabilités de chacune des deux classes plutôt que la valeur booléenne. La seconde colonne `C[:,1]` correspond à la probabilité que la consommation soit faible et donc à la couleur verte pour l'échelle de couleurs choisie.

Entrée [27]:
```
fig = go.Figure(go.Heatmap(z=np.reshape(C1[:,1],(5,5))+0,
                           colorscale='PiYG', #['green','red'],
                           #showscale=False,
                           opacity=0.4))
fig.update_yaxes(autorange="reversed")
fig.update_layout(title='Consommation faible (vert) vs. élevée (rouge)')
fig.show()
```



Les cases vertes correspondent à de la faible consommation alors que les rouges sont des points de sur-consommation. Par exemple, on voit

immédiatement que quand l'altitude est élevée, la consommation est plus faible.

A.5. "Apprentissage"

Jeudi 15 octobre

A.5.1. Qu'entend-on par la notion de "confiance" en un apprentissage ? Donnez votre avis sur les relations existant entre confiance, précision et taille de l'échantillon d'apprentissage. Que signifie qu'un modèle est robuste ?

La confiance en le résultat d'un apprentissage est la probabilité $1-\delta$ avec laquelle on sait que l'erreur de généralisation est bornée (ici par ϵ). C'est un résultat obtenu par une inégalité de concentration. Dans le cours nous avons démontré celle-ci pour des ensembles de fonctions (ou modèles) paramétriques (w est le paramètre), typiquement des réseaux de neurones.

$$P(\sup_w |R_0(w) - R_n(w)| > \epsilon) \geq 1 - \delta$$

si

$$\epsilon \geq \sqrt{\frac{\log|F| + \log(2/\delta)}{2n}}$$

Dans cette équation $|F|$ est la complexité de l'ensemble des fonctions utilisées, c'est une fonction croissante de la dimension de Vapnik-Chervonenkis h qui dépend du type de fonction utilisées.

- $R_n(w)$ est l'erreur empirique calculable à partir d'un jeu de n observations ;
- $R_0(w)$ est le risque théorique que l'on souhaite connaitre.

Cette formulation un peu complexe a cependant l'avatage de ne pas dépendre d'une constante C inconnue (si l'on peut considérer que $|F|$ est connu). Mais nous en avons déduit des bornes dans le cours liées à la régularité des fonctions.

D'autres formulations sont possibles et font intervenir effectivement des bornes. Par exemple on retrouve souvent ce calcul adapté aux catégorisations binaires :

$$R_0(w_n) \leq R_n(w_n) + C\sqrt{\frac{h\log(n) + \log(2/\delta)}{n}}$$

où w_n est le paramètrage obtenu par minimisation du risque empirique et h la dimension de Vapnik-Chervonenkis de l'ensemble des classifieurs (étudiée en cours).

A.5.2. Quel technique (assez simple) a été utilisée pour construire des tubes de confiances autours d'enregistrements de mesures ?

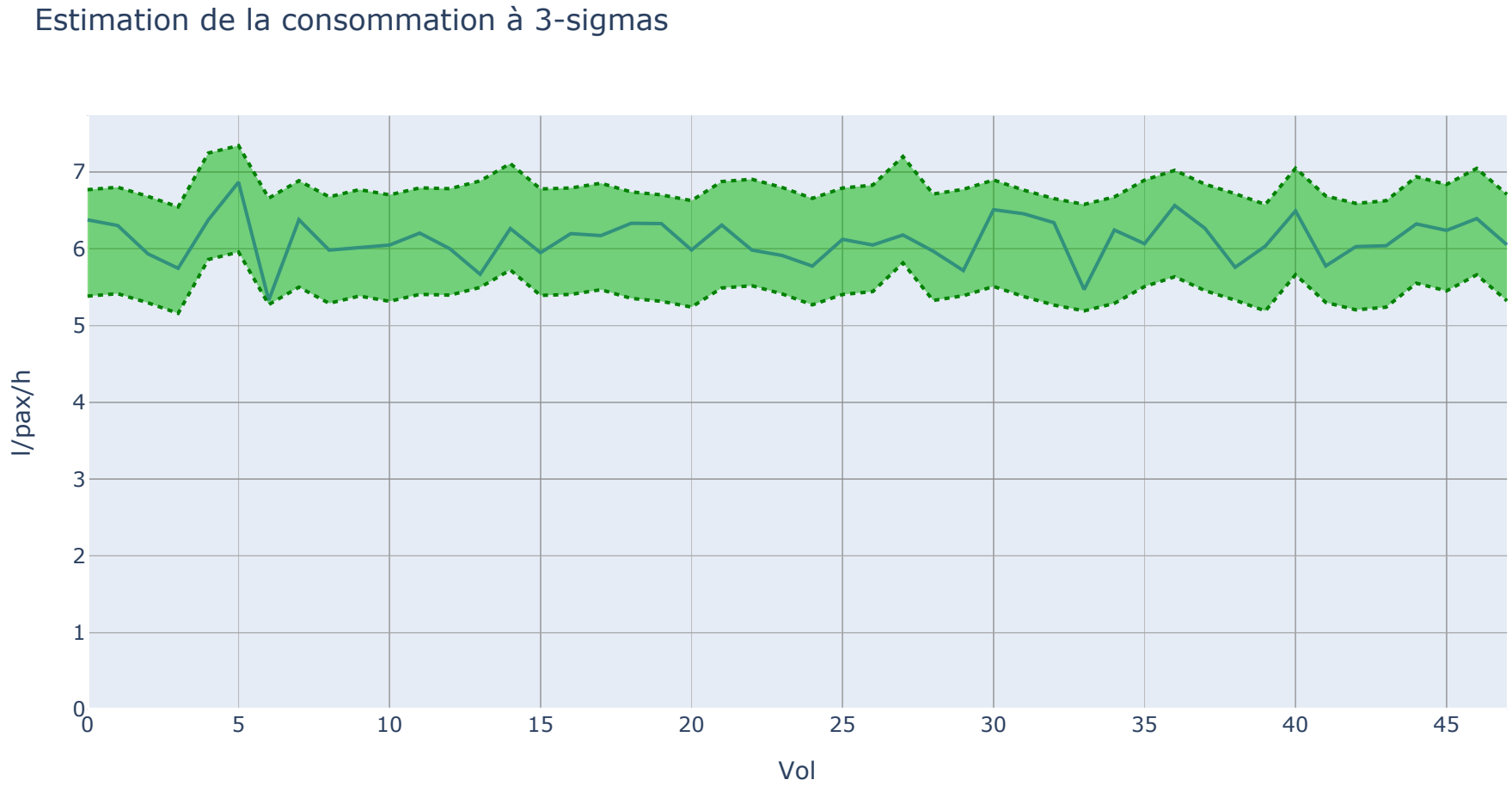
Une solution très simple consiste à construire un prédicteur de la variable autour de laquelle on souhaite mettre un tube de confiance. Ce prédicteur utilise des facteurs exogènes, comme des informations contextuelles, ou d'autres mesures prises au même instant (ou antérieures si on veut un tube anticipatif). D'ailleurs, un prédicteur anticipatif peut être un modèle autorégressif.

Ensuite on étudie l'erreur de ce prédicteur en analysant la distribution des résidus. Ceux-ci peuvent aussi être conditionnés par des facteurs exogènes. Cela permet de définir des quantiles inférieurs et supérieurs qui correspondent aux bornes du tube de confiance.

Entrée [28]:
```
y = df['Conso']
X = df[['ALT_max', 'Vz_max', 'E']]
SC = StandardScaler()
X = pd.DataFrame(SC.fit_transform(X),columns=X.columns)
reg = LinearRegression().fit(X,y)
yh = reg.predict(X)
```

Entrée [29]:
```
r = (y-yh).std()
```

Entrée [47]:
```
fig = go.Figure([go.Scatter(y=y),
                 go.Scatter(y=yh-3*r,opacity=0.7,stackgroup='tube',fill='none',
                            line_color='green', line_dash='dot'),
                 go.Scatter(y=np.ones(yh.shape)*r*6,opacity=0.7,stackgroup='tube',fillcolor='rgba(0,180,0,0.5
                            line_color='green', line_dash='dot')],
                go.Layout(showlegend=False,title="Estimation de la consommation à 3-sigmas",
                          yaxis_title="l/pax/h", xaxis_title="Vol"))
fig.show()
```



A.6. "Machines de Boltzmann"

Jeudi 22 octobre

A.6.1. Quel est le rôle de la température dans un recuit simulé ?

Dans un recuit simulé, la température sert à applanir la distributions du champ aléatoire modélisé quand elle augmente. Quand la température diminue, la distribution du champ se concentre sur les configurations fondamentales : celles d'énergie minimale. Du coup en faisant varier la température du modèle lentement d'une valeur élevée vers 0, tout en suivant l'état d'une chaine de Markov modélisée par les spécifications locales (dynamiques de Glauber ou de Métropolis) on va converger vers le minimum d'énergie. On optimise donc la fonction énergie.

A.6.2. Comment peut-on tester si un capteur est en panne ? Que faire dans ce cas ?

Une solution présentée dans le cours est de construire un capteur virtuel utilisant d'autres mesures. Ensuite on compare la vraisemblance des données observées avec la vraisemblance des mêmes données, mais après avoir remplacé le capteur douteux par sa version virtuelle. Si le second calcul donne une meilleur vraisemblance c'est sans doute que le capteur à un problème.

Dans le cours j'ai utilisé un modèle Gaussien a priori sur le vecteur des résidus entre les mesures observées et les estimations. Du coup j'ai pu me contenter de faire le rapport des distances de Mahalanobis ce qui donne un test de Fisher. Si le rapport entre les deux distances (avec remplacement / sans remplacement) est suffisemment faible, on peut conclure à une anomalie.

B. Questions techniques

Cette question va vous obliger à regarder un code en construction, pas forcément écrit de manière parfaite mais vous allez devoir comprendre et expliquer son fonctionnement. Le code en question est dans la toolbox 'tabata' (https://github.com/jee51/tabata (https://github.com/jee51/tabata)) que vous pouvez installer sur votre machine ou cloner dans un environnement "Google Colab" par exemple. Vous avez du apprendre à faire cela avec Florent en TD.

Entrée [31]:
```
# Sous Colab, supprimez le # de la ligne suivante.
#!git clone https://github.com/jee51/tabata
```

Entrée [32]:
```
import os
import tabata as tbt
```

B.1 Une interface data

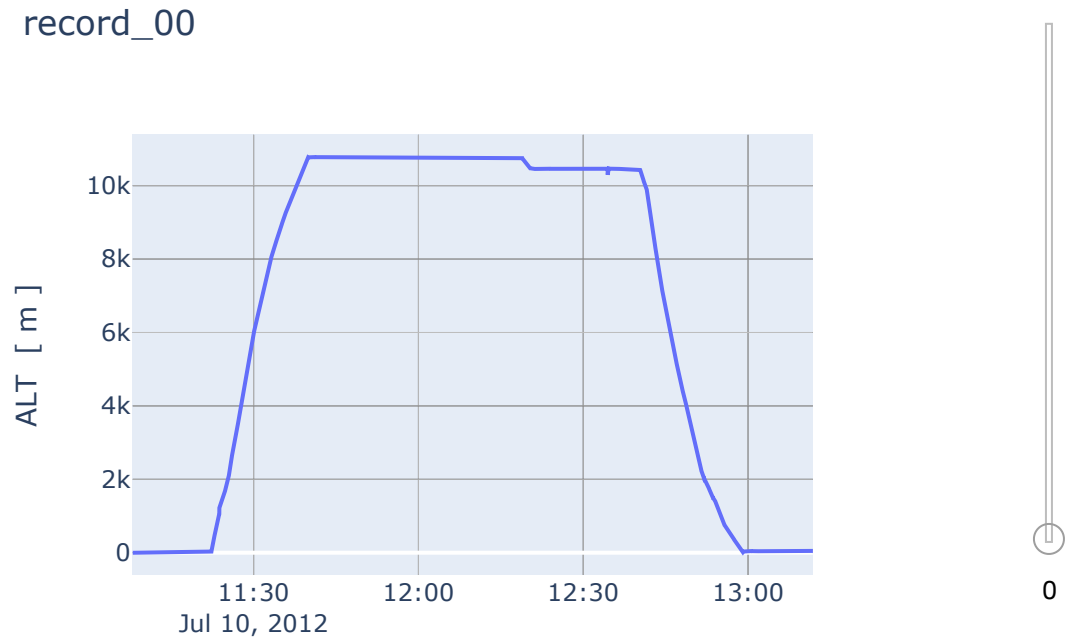
Nous avons vu en cours que pour collaborer facilement à plusieurs, il vaut mieux fixer à l'avance quelques contraintes sur les interfaces. Dans 'tabata' on va travailler sur des listes de signaux : chaque observation est un signal multivarié formé de plusieurs variables enregistrées au cours d'un intervalle de temps. Les signaux sont stockés naturellement dans des DataFrames 'pandas', pour les listes de signaux on a construit un objet `Opset` assez sympatique car il contient en particulier une fonction d'affichage interactive.

Un exemple de jeu de données est fourni avec 'tabata', ce sont des vols successifs d'un avion. Il y a 6 variables : l'altitude, une température mesurée dans le moteur qui reflète un peu la température extérieure, la vitesse air (Total Air Speed), la vitesse verticale, le poids de l'avion et une mesure de poussée du moteur.

Entrée [33]:
```
# Exécutez ce code pour charger les données et voir l'affichage interactif.
datafile = os.path.join(os.path.dirname(tbt.__file__),'notebooks/data/in/AFL1EB.h5')
S = tbt.Opset(datafile)
S.plot()
```

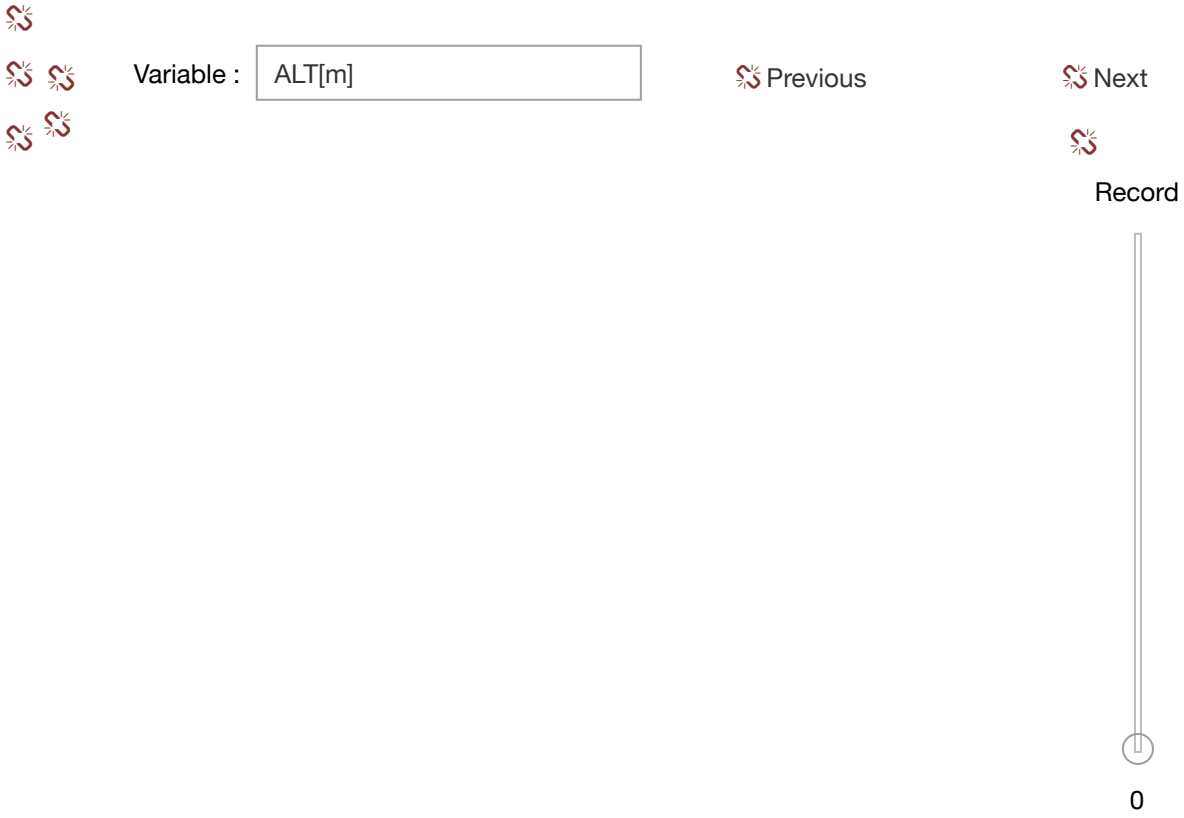
Variable : ALT[m] Previous Next

Record



(*Cette toolbox utilise Plotly pour les affichages ce qui peut poser certain problèmes si vous n'utilisez pas un environnement standard, JupyterLab par exemple. Il faudra si vous préférez cet environnement que vous installiez les extensions nécessaires. Des scripts d'installation sont proposés.*)

Entrée [34]:
```
# Sous "Google Colab" utilisez les méthodes '.ploc()'.
S.plotc()
```



L' `Opset` offre d'autres interfaces, vous pouvez les explorer en lançant le notebook que vous trouverez sous 'tabata/notebooks/opset_doc.ipynb'. L'exécution de ce code va créer une version épurée des données 'AFL1EB_C.h5' qui correspond au résultat d'une partie de l'examen de l'an passé.

(*Prenez le temps de regarder les autres notebooks exemples qui se trouvent dans ce répertoire.*)

B.1.1. Avez vous des idées pour améliorer cette API (Application Programming Interface) ? Pourquoi ai-je choisi d'utiliser la librairie Plotly plutôt que Matplotlib ? Que pensez-vous de l'interactivité réalisée sous le notebook ? Est-ce que le notebook n'est pas un moyen de collaboration pratique ?

Pourquoi ?

La solution "notebook" est pratique pour réaliser des expériences qui sont transférables entre utilisateurs. L'idée est de réaliser des exemples que l'on peut réexécuter par la suite pour comprendre comment on a résolu un problème ou comment fonctionne un code.

Evidemment il reste le problème des versions des différents packages dont certains d'entre vous ont fait l'expérience. C'est pourquoi il est indispensable de gérer des environnements stables que l'on évite de changer et de stocker un fichier "requirements.txt" qui rappelle le contenu de l'environnement.

- pip freeze > requirements.txt # Permet de créer une liste des package avec leur version installées.
- pip install -r requirement.txt # Permet de réinstaller l'environnement initial.

Concernant le package graphique `plotly` , c'est d'abord une question d'esthétique. Vous pouvez choisir les outils qui vous conviennent le mieux. Mais l'environnement "Dash" développé par Plotly pourra réutiliser directement vos codes si vous souhaitez créer une application web. Par ailleurs, je trouve l'interactivité plus facile à gérer entre `plotly` et `ipywidgets` et surtout, on peut zoomer sur les graphes comme en Matlab.

Un problème cependant, quand on stocke les notebook, queques graphes (avec interactivité notamment) ne sont pas sauvegardés (au contraire de `matplotlib`) et cela donne donc des exemples plus difficile à pérenniser sur GitHub.

Sinon, comme vous avez pu le remarquer, queques éléments ne sont pas encore pris en charge par Google Collab, c'est le cas du `FigureWidget` bien pratique. On peut esperer que ce soit le cas plus tard.

B.1.2. Il est facile avec des DataFrames pandas de faire des statistiques sur une variable, mais imaginons que l'on souhaite par exemple sortir la distribution des altitudes maximales à l'aide d'un graphe sympatique : une boîte à moustaches par exemple. Comment feriez vous ? Proposez un code.

On a déjà vu comment faire une boxplot plus haut (A.2) et on a créé un tableau de synthèse statistique par vol (A.4). Voyon ce que cela donne pour l'latritude.

Entrée [35]:
```
df.describe()
```

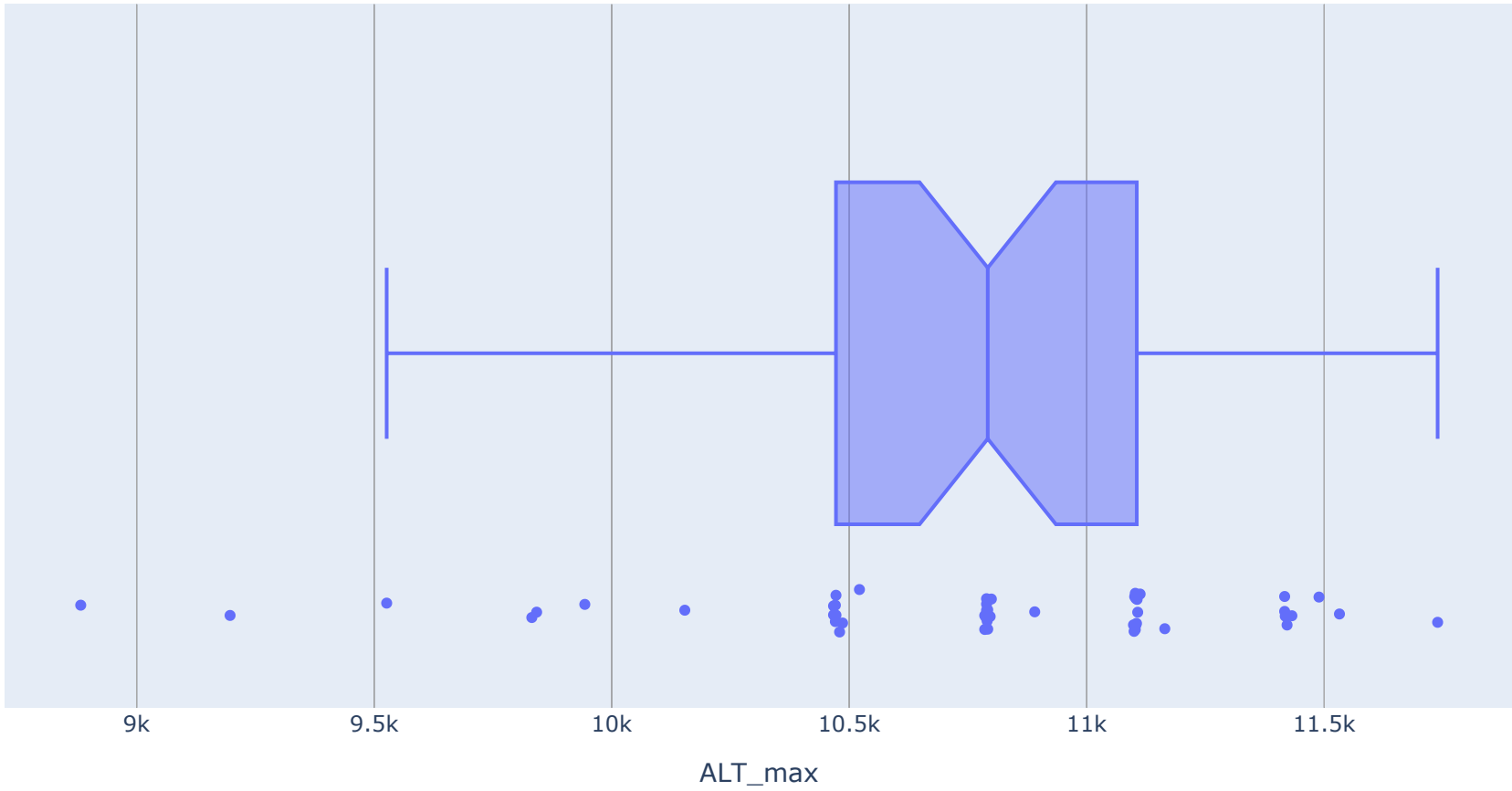
Out[35]:

	ALT_max	Alt_mean	Vz_max	Vz_mean	Vz_std	Tisa_mean	TAS_max	TAS_mean	E	Conso
count	48.000000	48.000000	48.000000	48.000000	48.000000	48.000000	48.000000	48.000000	48.000000	48.000000
mean	10759.210384	10688.273721	0.962786	-0.001298	0.159207	637.576673	517.271894	509.107626	-387156.365394	6.115293
std	596.250580	577.107123	0.029864	0.008572	0.050769	10.252667	10.383366	10.435102	196172.512665	0.289768
min	8882.115840	8874.689170	0.878230	-0.053945	0.094008	624.475280	489.543790	481.640683	-958690.300105	5.329779
25%	10472.147712	10459.988119	0.943655	-0.002450	0.133842	630.181308	509.815382	502.982379	-462625.565678	5.979155
50%	10791.675648	10776.766049	0.970855	-0.000570	0.148538	635.890211	516.540650	510.224063	-352857.887022	6.094629
75%	11105.180736	11093.927312	0.987392	0.001189	0.176368	641.592214	524.190431	515.622810	-277976.849407	6.324894
max	11738.847744	11621.771361	0.999941	0.015857	0.357700	670.127595	541.596938	531.647961	-44122.543371	6.868085

Pour changer un peu on va utiliser des racourçis à l'aide de l'API simplifiée `plotly.express` .

Entrée [36]:
```
import plotly.express as px
```

Entrée [37]:
```
px.box(df,x='ALT_max',notched=True,points='all',hover_name=df.index)
```



Petites astuces, on a fait un boxplot horizontal, on a rajouté les points avec un peu de bruit vertical (*jitter*) et on s'est assuré de pouvoir récupérer le nom de chaque vol en passant avec la souris sur les points.

Les encoches (*notches*) correspondent à $\frac{\pm 1.58 IQR}{\sqrt{n}}$ où IQR est l'*interquantile range*, donc Q3-Q1 (Q1=q(0.25), Q2=q(0.5) est la médiane et Q3=q(0.75)). La largeur entre les encoches représente un intervalle de confiance à 95% autour de la médiane (dans le cas gaussien) aussi, si deux boxplots on des encoches qui ne se superposent pas, on peut assurer que les deux médianes sont différentes à 95%.

B.2 Comprendre un algorithme

La toolbox 'tabata' contient un outil assez amusant qui permet de sélectionner des instants. Voyons comment cela fonctionne.

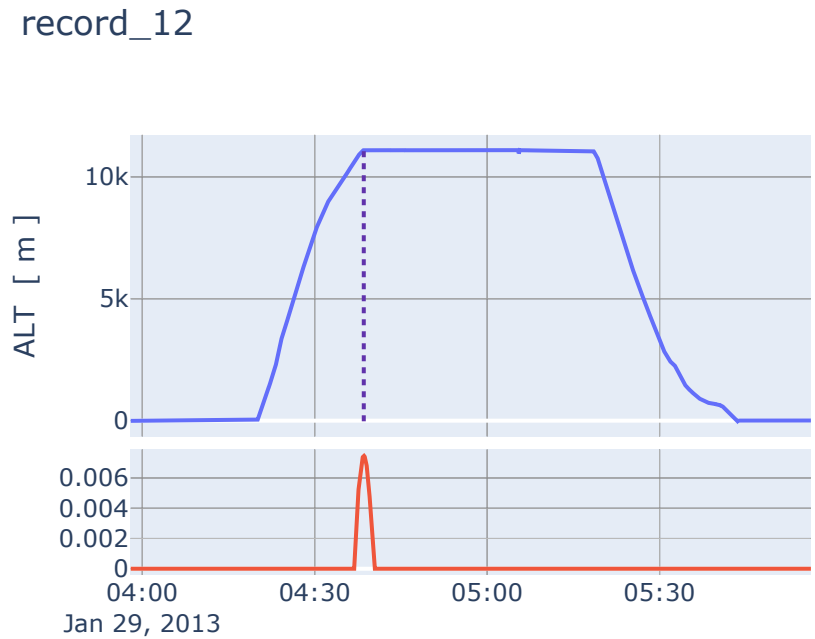
Entrée [38]:
```
cleanfile = tbt.opset.datafile('/AFL1EB_C.h5')
```

Entrée [39]:
```
# N'oubliez pas d'exécuter le code 'tabata/notebooks/opset_doc.ipynb' avant pour créer le fichier AFL1EB_C.h5
cleanfile = os.path.join(os.path.dirname(tbt.__file__),'notebooks/data/out/AFL1EB_C.h5')
S = tbt.Selector(cleanfile)
S.plot()
```

Plot | Param | Learn

Variable : ALT[m] Previous Next

Record

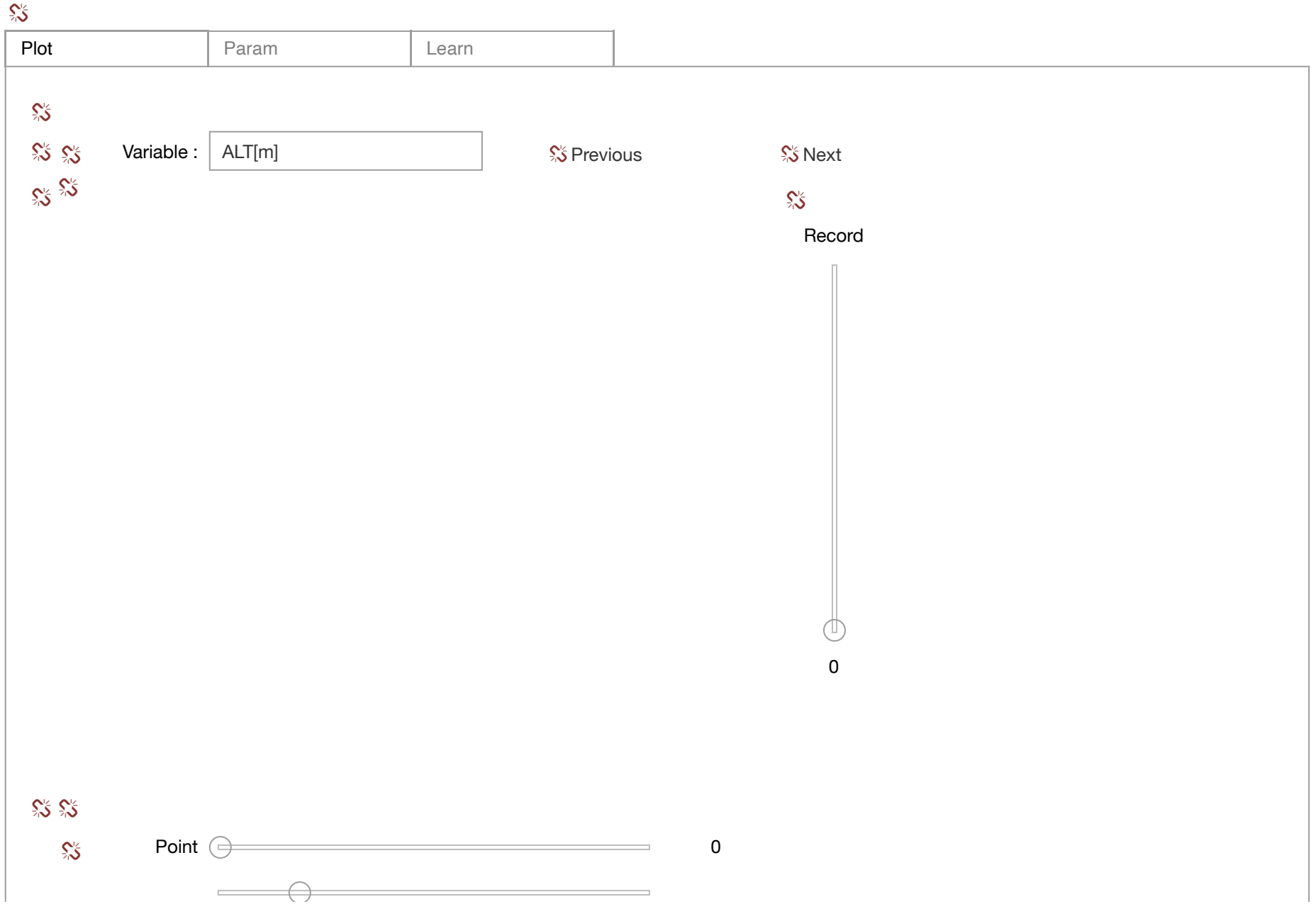


8

Filter 100

1. Depuis l'onglet 'Plot' cliquez sur un point qui vous intéresse de la courbe Altitude : par exemple la fin de la montée de l'avion. Puis passez à l'enregistrement suivant et recommencez, faites cela sur quelques enregistrements. Vous pouvez sauter des enregistrements qui ne vous plaisent pas, le 'record_10' par exemple (le n°6 dans la liste). pas la peine d'en faire trop, une dizaine de clics devraient suffire, mais sélectionnez bien toujours le même point d'intérêt sur chaque courbe.
2. Passez sur l'onglet 'Learn'. Tout en laissant ALT sélectionné, cliquez sur Learn. Le bouton doit être bleu, il va passer à l'orange quand l'apprentissage va s'exécuter, puis deviendra vert à la fin.
3. Retournez sur l'onglet 'Plot'. La courbe rouge en dessous de l'affichage de l'altitude montre une bosse. On dirait un probabilité de présence du point que l'on a cliqué.
4. Regarderz d'autres enregistrements. Voyez ! Sur des enregistrements que vous n'avez pas encore vu l'algorithme a deviné ce que vous souhaitiez lui monter.

Entrée [40]:
```
# Même chose que tout à l'heure.
# Sous "Colab" utilisez '.plotc()'
S.plotc()
```



Sous "Google Colab" la sélection d'un point sur une courbe ne fonctionne pas, vous utiliserez la scroll-barre 'Point'. Attention, d'une observation à l'autre le point de sélection est conservé, vous le verrez en faisant un aller-retour (Next-Previous). Pensez donc à en changer la position, et si vous voulez sauter une observation, ramenez le point en 0.

B.2.1. Comment ça marche ? Quelle a été l'astuce ?

(*Cette question est sans doute la plus difficile de l'épreuve. J'attends une réponse claire. Avez vous d'autres idées, pourquoi ce truc marche-t-il ?*)

Cette question attend une réponse simple (une astuce) :

- On va essayer de détecter si un point est à gauche ou à droite de la sélection attendue.

Voilà.

En fait il y a un avantages important à faire cela : on augmente fortement le nombre de données. On peut moduler ce volume en modifiant le paramètre "Sample" (onglet "Param"). Ce paramètre indique la proportion de points de chaque signal qui seront gardés pour l'apprentissage. Comme le choix est aléatoire, on peut refaire plusieurs essais et garder le meilleur, c'est le paramètre "Retry" qui gère ce nombre de tentatives.

En fait l'apprentissage se passe en deux étapes :

- On va d'abord chercher parmi les différents essais les indicateurs les plus importants : ceux qui sont le plus souvent utilisés et détectés par une proportion d'apparition minimale dans les différents essais égale au moins au paramètre "Percentile".
- Une fois une série d'indicateurs conservés, on va essayer des modèles de plus en plus simples en éliminant progressivement les caractéristiques inutiles.

Le paramètre "Split" est utilisé par les arbres de décision de `scikit-learn` pour élaguer les arbres à une hauteur pas trop grande en fixant une proportion d'examples minimale pour autoriser le découpage d'un noeud intérieur.

Entrée [41]:
```
# Je précharge des données toutes calculées pour faciliter la démonstration.
import pickle
idatafile = tbt.opset.datafile('interval.pkl')
with open(idatafile,'rb') as pkl:
    S,S1 = pickle.load(pkl)

# Attention le pickle a été sauvé dans la toolbox `tabata`,
# il faut adapter l'adresse en ajoutant le bon chemin.
# Désolé pour le problème.
S1.storename = S.storename = tbt.opset.datafile('AFL1EB_C.h5')
S._dsi.storename = tbt.opset.datafile('AFL1EB_C_I.h5')
```

Entrée [42]:
```
S.describe()
```

```
          Name  Filter  Order  Sigma       Std
Feature
0       ALT[m]      35      0     25  0.181580
1       ALT[m]     105      0      5  0.191481
2       ALT[m]     350      0      5  0.205967
|--- feature_1 <= 1.98
|   |--- feature_2 <= 1.90
|   |   |--- class: -1
|   |--- feature_2 >  1.90
|   |   |--- class: 1
|--- feature_1 >  1.98
|   |--- feature_0 <= 2.02
|   |   |--- class: -1
|   |--- feature_0 >  2.02
|   |   |--- class: 1
```

Ensuite, bien sûr, il y a les types d'indicateurs choisis :

- Les indicateurs comptent les bosses et les creux à différentes échelles.
- À différents ordres (ordre 2 = points d'inflexion, cela suffit visuellement).
- De la droite vers la gauche et de la gauche vers la droite.

Evidemment on va avoir un problème si le point d'intérêt se trouve au milieu d'un signal et que la gauche ou la droite n'ont pas d'importance, mais dans ce cas il suffit de faire des découpages progressifs.

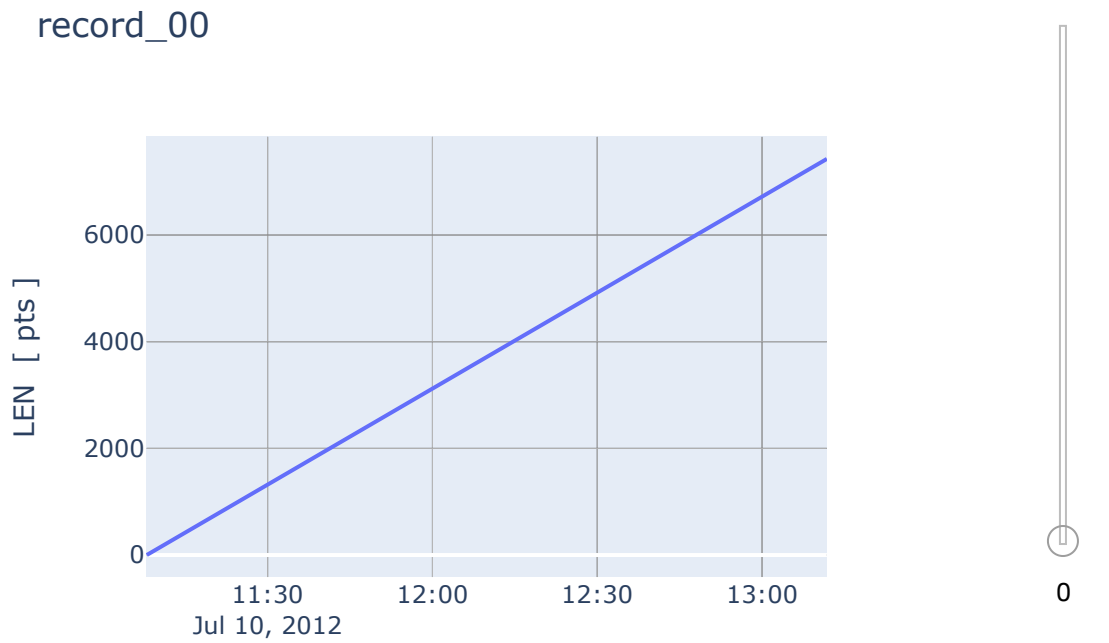
Le paramètre "Range" est calculé automatiquement à partir de la longueur minimale des signaux mais peut être fixé et défini les demi-tailles des moyennes mobiles utilisées par les filtres de Satvisky-Golay.

Le paramètre "Sigma" donne le niveau de passage par zéro (le haut d'une bosse ou le bas d'un creu correspond à un zéro de dérivée), du haut vers le bas ou du bas vers le haut. On crée ainsi plein d'indicateurs qui vont compter les numéros de bosses et de creux !

Entrée [43]:
```
tbt.Opset(S._dsi).plot()
```

Variable : LEN[pts] Previous Next

Record



Cet affichage donne l'allure des indicateurs calculés pour les signaux d'apprentissage. On voit qu'une position est interpolée linéairement entre creux ou bosse, ce qui permet d'avoir des références du type : au milieu de la bosse 2 et la bosse 3. Le code de l'indicateur donne son type, il est décrit dans le notebook 'instant_doc'.

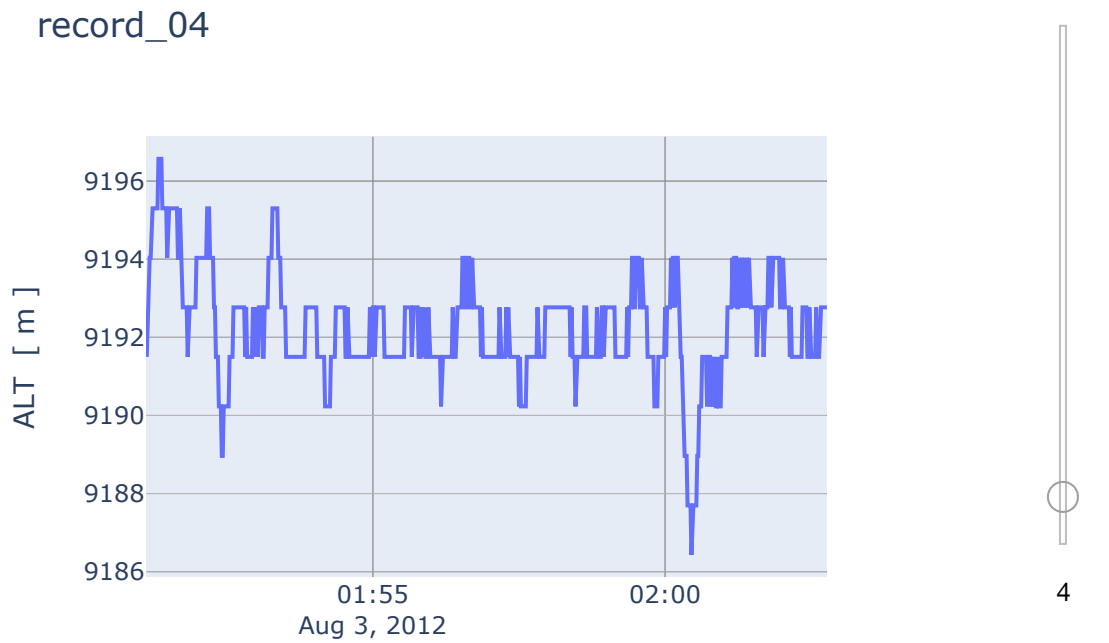
B.2.2. En étudiant l'API du `Selector`, vous allez voir qu'il est possible d'extraire un intervalle entre deux points d'intérêts. Faites cela, créez un dataset temporaire.

(*Facile ! C'est déjà fait dans les notebooks examples.*)

Entrée [44]:
```
E = S.between(S.predict(),S1.predict())
E.plot(pos=4)
```

Variable : ALT[m] Previous Next

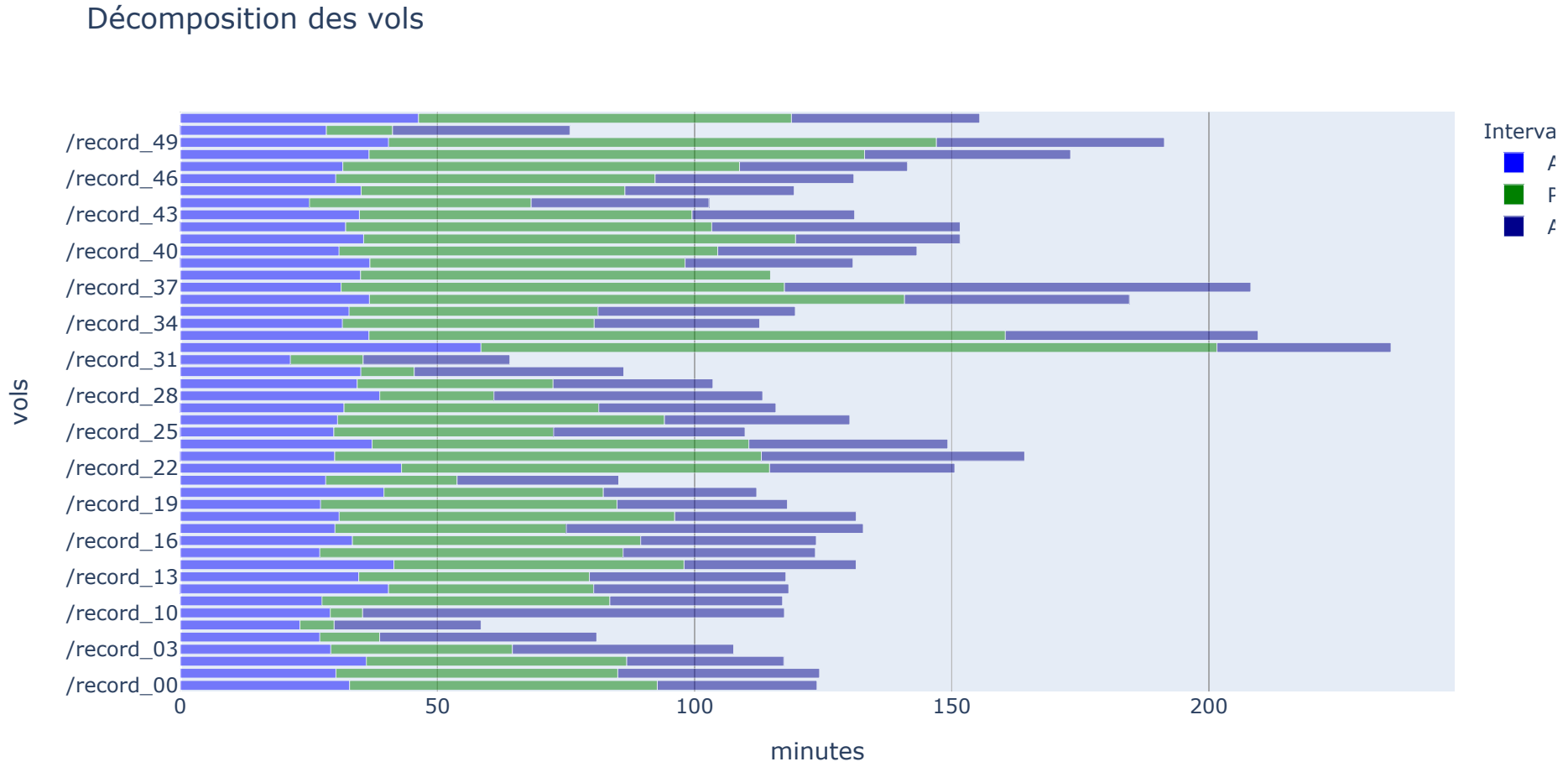
Record



On peut décrire visuellement les intervalles.

Entrée [45]:
```
df = pd.DataFrame([[(E[i].index[0]-S[i].index[0]).total_seconds()/60,
                    (E[i].index[-1]-E[i].index[0]).total_seconds()/60,
                    (S[i].index[-1]-E[i].index[-1]).total_seconds()/60]
                   for i in range(len(S))],
                  columns=['Avant', 'Pendant', 'Après'],
                  index = S.records)
```

Entrée [46]:
```
px.bar(df,x=df.columns,title='Décomposition des vols',
       labels={'variable':'Intervalle','value':'minutes','index':'vols'},
       opacity=0.5,
       color_discrete_sequence=['blue','green','darkblue'])
```



B.3 Créer un modèle.

Participer à un développement collaboratif.

B.3.1 Le détecteur d'instants que vous venez de manipuler est encore assez frustre, il faudrait peut-être créer des indicateurs plus malin. Avez-vous des idées ?

Le détecteur d'instants tel qu'il est construit utilise des indicateurs qui correspondent à un comptage du nombre de bosses ou de creux à partir de la gauche ou de la droite. En fait on recherche des patterns et comme on a décidé d'utiliser un modèle qui positionne un point comme étant à gauce ou à droite de la zone d'intérêt, il faut créer des indicateurs adaptés. Formellement, si on garde l'idée gauche/droite la technique consiste à construire des indicateurs croissants ou décroissants à partir des courbes.

- Supposons par exemple, qu'une variable soit positive, alors on peut la cumuler, on en déduit un nouvel indicateur.
- Pour la poussée, qui peut être négative, c'est un peu différent, on sait que la poussée intégrée donne une information sur l'énergie dépensée (il faut aussi prendre en compte le déplacement pour bien faire), l'avion monte puis descend, c'est un cycle que l'on peut modéliser par un vecteur angulaire toujours croissant.

On peut aussi construire des indicateurs "stochiométriques" à partir d'un index obtenu par le cumul précédent et qui remplacerait le temps.

Si maintenant on change de point de vue et que l'on essaye de comprendre le pattern à détecter, on peut envisager une décomposition des signaux en ondelettes.

- Le problème est qu'alors on ne peut plus utiliser l'astuce gauche/droite, il en faut une autre. Par exemple on peut considérer que les points qui sont "loin" des détections et qui sont beaucoup plus nombreux, sont "normaux", alors que nos détections sont des "anomalies" que l'on cherche à identifier. Cela devient un problème de modélisation stochastique sur les scalogrammes obtenus par ondelettes glissantes.
- L'utilisation d'un classifieur sera toujours délicat car on n'aura que très peu de points de la classe recherchée. Il faudra être très malin dans la sélection des variables, peut être avec un ensemble d'arbres de décision, un peu comme pour l'algorithme de `tabata`.
- Une dernière idée est de définir un coût qui dépend de la distance d'un point sur la courbe au point à détecter. Dans ce cas on peut essayer de faire une régression supervisée. Plusieurs fonctions de coût peuvent être testées.

Nous avons vu en cours qu'un tube de confiance est un outils assez sympatique pour détecter des anomalies sur des signaux temporels assez récurrents. La toolbox `tabata` a un module `Tubes` expérimental qui souhaite faire cela.

Dernière question bonus.

B.3.2. Proposez de vraies améliorations statistiques au tube de confiance comme on a vu dans le cours.

L'algorithme actuel établit pour chaque courbe surveillée, une série de prédictions linéaires. Les différences entre ces prédictions sont dans le choix des observations et celui des facteurs. Ces choix sont aléatoires et on garde les meilleurs résultats. Ensuite le tube produit est calculé à partir de l'enveloppe de ces prédictions que l'on a pris soin de lisser.

Cela n'est pas vraiment statistique, on obtient bien un gabari, mais on n'est pas certain de la qualité de celui-ci.

Voici plusieurs propositions :

1) Etudier la variance autour de la moyenne des prédictions et construire en chaque point un nouveau gabari correspondant à un multiple de l'écart-type. Si les écarts à la prédiction, sont localement gaussiens on peut en déduire un intervalle de confiance.

2) Extraire les points au delà du gabari, ces points doivent suivre une loi des extrèmes de type Gumble, on peut en déduite un intervalle de confiance au delà du gabari initial, il est aussi possible de modéliser ces écarts par une loi de Pareto.

D'autres approches consistent à travailler dans un espace de courbes mais il y a alors plusieurs étapes :

1) Ramener les courbes à une longeur identique, par exemple par un algorithme de transport comme DTW, ou en essayant de les synchroniser quitte à rajouter ou enlever des points à gauche et à droite.

2) Comprimer les courbes en effectuant par exemple une ACP pour remplacer chaque courbe par un vecteur de coefficients dans une base de courbes modèles "templates".

3) Construire un modèle de prédiction des paramètres de la courbe surveillée par les paramètres des courbes de supervision. Les résidus sont alors un bon indicateurs de la normalité d'une courbe.

4) Inverser le modèle de projection pour construire une estimation de la courbe. La largeur du tube peut s'obtenir comme au-dessus par une étude du résidus (local ou par loi des extrêmes).

5) Sinon, construire le tube en recréant un faisseau de courbes à partir du modèle de projection et un bruitage des paramètres pour construire une enveloppe autour de chaque observation. C'est moins juste mais cela fait l'affaire dans la plupart du temps.

Je suis sûr qu'il y a encore plein d'autres idées auxquelles je n'ai pas pensé...

N'oubliez pas d'enregistrer ce document et nous l'envoyer par mail. N'hésitez pas à nous contacter si vous avez besoin d'aide.

Je vous souhaite à tous de réussir dans votre domaine.

Bonnes fêtes.

Jérôme Lacaille