TP3 Credit Card #Finance (partie 2)

Détection de fraude de carte bancaire

Mise en œuvre des algorithmes d'apprentissage automatique supervisé et non supervisé en respectant toutes les étapes de construction d'un modèle

Dans Anaconda Prompt :

- pip install eli5
- pip install sklearn
- pip install scipy
- pip install collections
- pip install seaborn

In [83]:

```
# Import des librairies
import numpy as np # librairie de calcul numérique
import pandas as pd # librairie de statistiques
from pandas.plotting import scatter_matrix
import seaborn as sns
import matplotlib.pyplot as plt # librairie de tracé de figures
import matplotlib.patches as mpatches
import time

# Librairies Machine learning
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.decomposition import PCA, TruncatedSVD # librairie d'analyse factorielle
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from sklearn.svm import SVC
from sklearn.manifold import TSNE
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import eli5 #for purmutation importance
from eli5.sklearn import PermutationImportance
from sklearn.pipeline import make_pipeline
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score, accurac
from sklearn.model_selection import KFold, StratifiedKFold, train_test_split, cross_val_sco
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import load_iris
from scipy.cluster.hierarchy import dendrogram
from sklearn.metrics import confusion_matrix #for model evaluation
import collections
from collections import Counter
```

In [84]:

```
# chargement de la base de données
df = pd.read_csv('creditcard.csv')
```

Etape 1 : Traitement des données

Centrage-réduction des données

In [85]:

```
#Fonction de centrage-réduction
def centrereduit(x):
    return (x - np.mean(x)) / np.std(x)

#Centrer réduire les variables sauf la classe
for i in range(0,len(df.columns)-1):
    df[df.columns[i]] = centrereduit(df[df.columns[i]])
```

In [86]:

```
np.mean(df['V1'])
```

Out[86]:

```
-2.263246277900502e-15
```

In [87]:

```
np.std(df['V1'])
```

Out[87]:

```
0.999999999999998
```

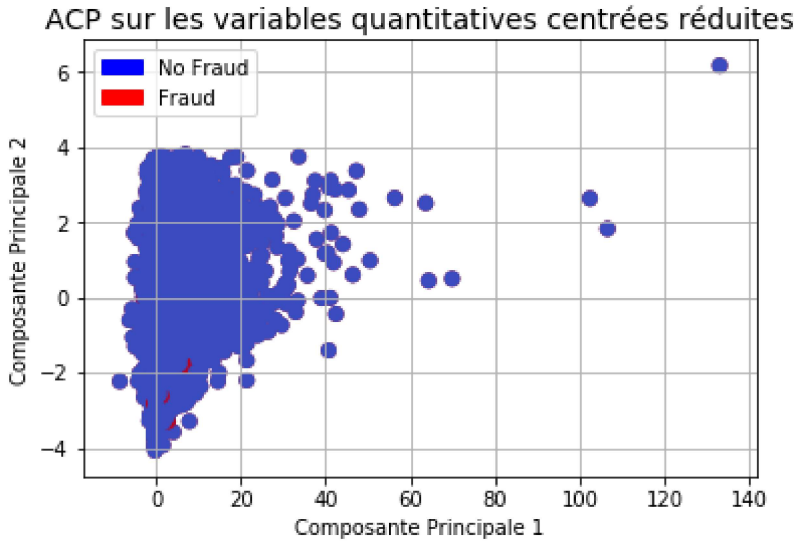
Centrer réduire de sorte à ce que la moyenne = 0 et l'écart-type = 1

In [88]:

```
X_reduced_pca = PCA(n_components=2, random_state=42).fit_transform(df.drop("Class", axis=1)
```

In [89]:

```
plt.figure()

plt.scatter(X_reduced_pca[:,0], X_reduced_pca[:,1], c=(df["Class"] == 0), cmap='coolwarm',
plt.scatter(X_reduced_pca[:,0], X_reduced_pca[:,1], c=(df["Class"] == 1), cmap='coolwarm',
plt.title('ACP sur les variables quantitatives centrées réduites', fontsize=14)
plt.grid(True)
plt.xlabel("Composante Principale 1")
plt.ylabel("Composante Principale 2")

blue_patch = mpatches.Patch(color='blue', label='No Fraud')
red_patch = mpatches.Patch(color='red', label='Fraud')
plt.legend(handles=[blue_patch, red_patch])
plt.show()
```



Etape 2 : Création d'une base d'apprentissage et d'une base de test qui soient représentatif du nombre de transactions bancaires frauduleuses

En effet, il y a 492 transactions frauduleuses sur 284.315 transactions bancaires normales. Pour cette application spécifique à notre cas d'étude, peut-être est-ce pertinent de découper une base d'apprentissage où seront représentées 80% des transactions bancaires frauduleuses.

In [90]:

```
# Option pour créer une base d'apprentissage : celle qui permet d'avoir une base de données
# et équilibrée par rapport à la variable de sortie "Class"

ratio_apprentissage = 0.8 # proportion de la base de données utilisée pour l'apprentissage

# sous-calculs
df_shuffle = df.sample(frac=1) # désordre aléatorie du jeu de données
num_fraude = df_shuffle.groupby('Class').size()[1] #combien de fraudes dans notre jeu de do
num_train = int(ratio_apprentissage * num_fraude + 1)

# base d'entrainement
df_fraude = df_shuffle.loc[df_shuffle['Class'] == 1][0:num_train]
df_non_fraude = df_shuffle.loc[df_shuffle['Class'] == 0][0:num_train]
df_train = pd.concat([df_fraude, df_non_fraude])
df_train = df_train.sample(frac=1)

# base de test
df_fraude = df_shuffle.loc[df_shuffle['Class'] == 1][num_train:num_fraude]
df_non_fraude = df_shuffle.loc[df_shuffle['Class'] == 0][num_train:num_fraude]
df_test = pd.concat([df_fraude, df_non_fraude])
df_test = df_test.sample(frac=1)

print( "Répartition dans le jeu d'apprentissage :", df_train.groupby('Class').size(),
      "\nRépartition dans le jeu de test :", df_test.groupby('Class').size() )
```

```
Répartition dans le jeu d'apprentissage : Class
0    394
1    394
dtype: int64
Répartition dans le jeu de test : Class
0    98
1    98
dtype: int64
```

In [91]:

```
#Fonction qui détecte les outliers
def detect_outliers(df, variables):
    '''détection les individus dont une des variables d'entrée est en dehors de l'intervall
    individus = [] # initialisation de la liste des individus outliers"
    # ...
    for var in variables:
        m = df[var].mean(axis=0)
        std = df[var].std(axis=0)
        mini = m - 3*std # seuil bas pour un outlier
        maxi = m + 3*std # seuil haut pour un outlier
        # ...
        individus_nouveaux = np.where( np.logical_or( df[var] < mini, df[var] > maxi ) )[0]
        individus = np.unique( individus + individus_nouveaux ).tolist()
    # ...
    return individus

# exécution de la fonction
variables = df_train.columns[1:29]
outliers = detect_outliers(df_train, variables)
#print( len(outliers), 'outliers sur', df_train.shape[0])

#df_train = df_train.drop(df_train.index[outliers]); # suppressions des valeurs
```

In [92]:

```
X = df.drop('Class', axis=1)
y = df['Class']

X_train = df_train.drop('Class', axis=1)
y_train = df_train['Class']

X_test = df_test.drop('Class', axis=1)
y_test = df_test['Class']
```

Etape 3 : Réduction de dimensions

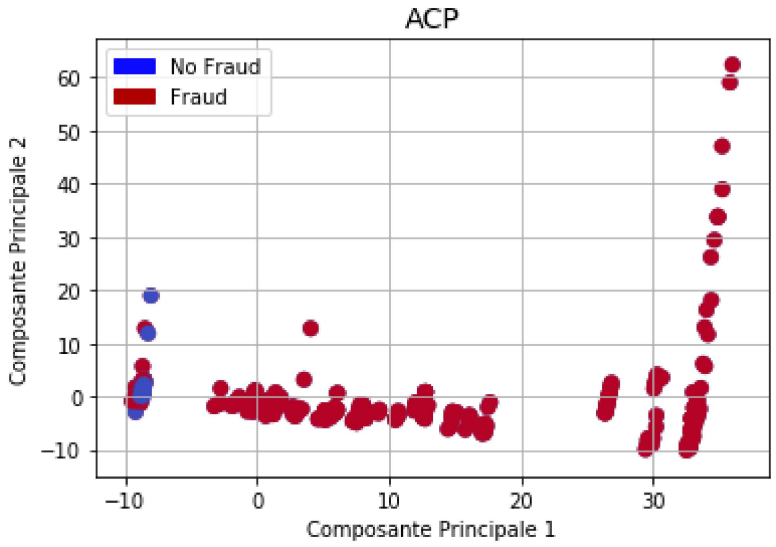
1) Analyse en composantes principales (ACP)

In [93]:

```
# PCA Implementation
X_reduced_pca = PCA(n_components=2, random_state=42).fit_transform(X_train.values)
```

In [94]:

```
plt.figure()

plt.scatter(X_reduced_pca[:,0], X_reduced_pca[:,1], c=(y_train == 0), cmap='coolwarm', labe
plt.scatter(X_reduced_pca[:,0], X_reduced_pca[:,1], c=(y_train == 1), cmap='coolwarm', labe
plt.title('ACP', fontsize=14)
plt.xlabel("Composante Principale 1")
plt.ylabel("Composante Principale 2")
plt.grid(True)

blue_patch = mpatches.Patch(color='#0A0AFF', label='No Fraud')
red_patch = mpatches.Patch(color='#AF0000', label='Fraud')
plt.legend(handles=[blue_patch, red_patch])
plt.show()
```



In [95]:

```
mypca = PCA(n_components=2)
mypca.fit_transform(X_train.values)
mypca.explained_variance_ratio_
```

Out[95]:

```
array([0.61136221, 0.09758236])
```

In [96]:

```
sum(mypca.explained_variance_ratio_)
```

Out[96]:

```
0.7089445667069363
```

In [97]:

```
mypca.components_[1]
```

Out[97]:

```
array([ 0.02058156, -0.18536834,  0.12436665, -0.16752237,  0.00427454,
       -0.16186097,  0.09860942, -0.36960902, -0.45147059, -0.07872578,
       -0.14704832, -0.11982237,  0.12907787, -0.03126066,  0.33169356,
       -0.0588856 ,  0.10810196,  0.13158857,  0.01343278, -0.06585772,
        0.13176888, -0.43243725,  0.14007553,  0.10045657,  0.01638652,
       -0.04408526, -0.02844704, -0.30925105, -0.07579643, -0.00150026])
```

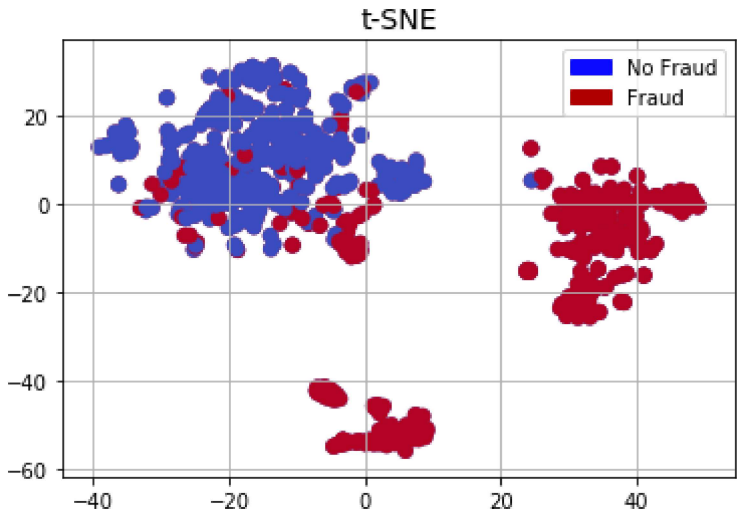
2) T-SNE

In [98]:

```
# T-SNE Implementation
X_reduced_tsne = TSNE(n_components=2, random_state=99).fit_transform(X_train.values)
#https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding
```

In [99]:

```
plt.figure()

plt.scatter(X_reduced_tsne[:,0], X_reduced_tsne[:,1], c=(y_train == 0), cmap='coolwarm', la
plt.scatter(X_reduced_tsne[:,0], X_reduced_tsne[:,1], c=(y_train == 1), cmap='coolwarm', la
plt.title('t-SNE', fontsize=14)
plt.grid(True)

blue_patch = mpatches.Patch(color='#0A0AFF', label='No Fraud')
red_patch = mpatches.Patch(color='#AF0000', label='Fraud')
plt.legend(handles=[blue_patch, red_patch])

plt.show()
```



Etape 4 : Méthodes de clustering (apprentissage automatique non supervisé)

In [100]:

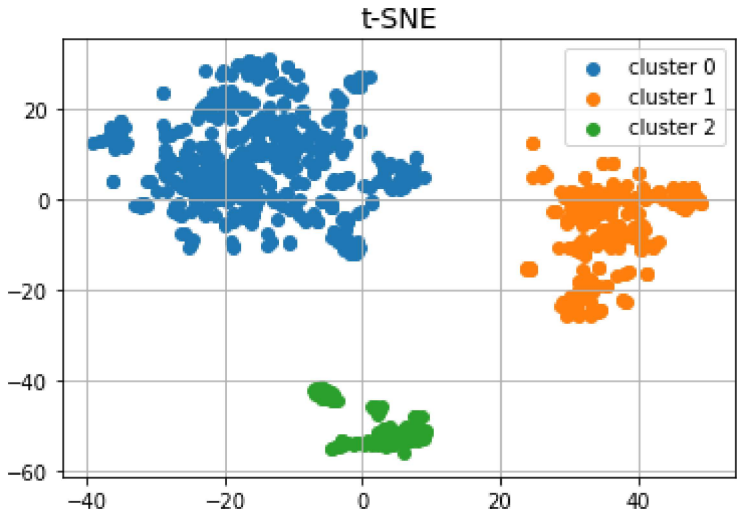
```
# exécution des k-means sur le nuage de point projeté par t-sne
kmeans = KMeans(n_clusters=3, random_state=0).fit( X_reduced_tsne[:,0:2] )

# résultats du clustering
kmeans.labels_ # numéro du cluster dans lequel sont chacun des individus (index : ligne) du
kmeans.cluster_centers_ # centres de chaque cluster
```

Out[100]:

```
array([[-14.630329 ,   8.859363 ],
       [ 35.74204  ,  -7.434634 ],
       [  1.6043257, -49.737083 ]], dtype=float32)
```

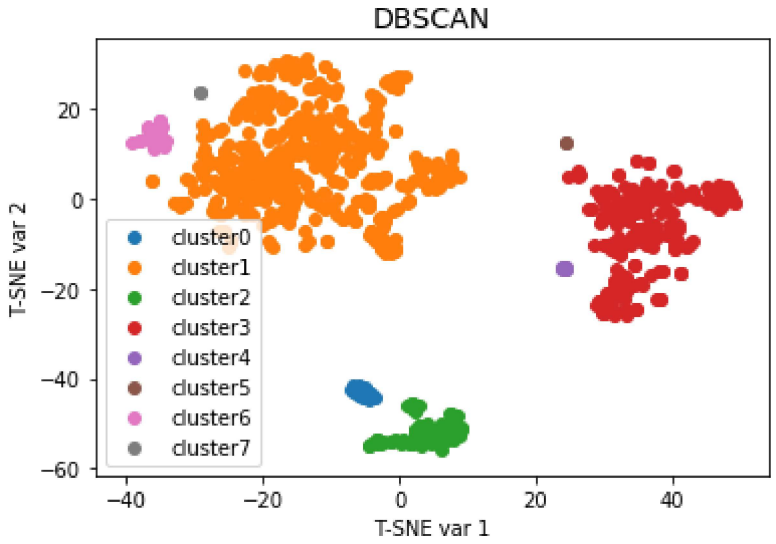
In [101]:

```
# représentation sur la projection de t-sne le résultat de k-means

cluster = kmeans.predict(X_reduced_tsne[:,0:2])

point0 = np.where(cluster == 0)[0]
point1 = np.where(cluster == 1)[0]
point2 = np.where(cluster == 2)[0]

plt.figure()

plt.scatter(X_reduced_tsne[point0,0], X_reduced_tsne[point0,1], cmap='coolwarm', label='clu
plt.scatter(X_reduced_tsne[point1,0], X_reduced_tsne[point1,1], cmap='coolwarm', label='clu
plt.scatter(X_reduced_tsne[point2,0], X_reduced_tsne[point2,1], cmap='coolwarm', label='clu

plt.title('t-SNE', fontsize=14)
plt.legend()
plt.grid(True)

plt.show()
```



In [102]:

```
from sklearn.cluster import DBSCAN
clustering = DBSCAN(eps=5, min_samples=2).fit(X_reduced_tsne)

#Plot
unique_labels = np.unique(clustering.labels_)
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_labels))]
for i in range(0,len(unique_labels)):
    plt.scatter(X_reduced_tsne[clustering.labels_==unique_labels[i]][:,0], X_reduced_tsne[c

plt.xlabel("T-SNE var 1")
plt.ylabel("T-SNE var 2")
plt.title('DBSCAN', fontsize=14)
plt.legend()

plt.show()
```



Etape 5 : Apprentissage automatique supervisé

1) Arbre de décision

In [103]:

```
# Classification par les arbres de décision
dtc = DecisionTreeClassifier()

dtc.fit(X_train, y_train)
```

Out[103]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

In [104]:

```
dtc.score(X_test, y_test)
```

Out[104]:

```
0.8826530612244898
```

In [105]:

```
#Prédiction
y_esti_tree = dtc.predict(X_test)
```

In [106]:

```
#Matrice de confusion
confusion_matrice_tree = confusion_matrix(y_test, y_esti_tree)
confusion_matrice_tree
```

Out[106]:

```
array([[88, 10],
       [13, 85]], dtype=int64)
```

In [107]:

```
sensitivity_tree = confusion_matrice_tree[1,1]/(confusion_matrice_tree[1,1]+confusion_matri
sensitivity_tree
```

Out[107]:

```
0.8673469387755102
```

In [108]:

```
specificity_tree = confusion_matrice_tree[0,0]/(confusion_matrice_tree[0,0]+confusion_matri
specificity_tree
```

Out[108]:

```
0.8979591836734694
```

2) Random Forest

In [109]:

```
#le modèle
rf = RandomForestClassifier(max_depth=10)

#Apprentissage
rf.fit(X_train, y_train)
```

```
d:\python\envs\myenvcnn\lib\site-packages\sklearn\ensemble\forest.py:245: Fu
tureWarning: The default value of n_estimators will change from 10 in versio
n 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

Out[109]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=10, max_features='auto', max_leaf_nodes=Non
e,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [110]:

```
#Prédiction sur les données de tests
y_pred_bin = rf.predict(X_test)

#class probability predictions
y_pred_quant = rf.predict_proba(X_test)[:,1] #"sorties souples"
#The predicted class probability is the fraction of samples of the same class in a leaf.
```

In [111]:

```
confusion_matrice_rf = confusion_matrix(y_test, y_pred_bin)
confusion_matrice_rf
```

Out[111]:

```
array([[98,  0],
       [14, 84]], dtype=int64)
```

In [112]:

```
sensitivity_rf = confusion_matrice_rf[1,1]/(confusion_matrice_rf[1,1]+confusion_matrice_rf[
sensitivity_rf
```

Out[112]:

```
0.8571428571428571
```

3) K plus proches voisins (KNN)

In [113]:

```
# Application de la méthode de prévision sur le jeu d'entrainement
neigh = KNeighborsClassifier(n_neighbors=3)

neigh.fit(X_train, y_train)
```

Out[113]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

In [114]:

```
# évaluation de la matrice de confusion sur le jeu de test
y_esti_knn = neigh.predict(X_test)
y_esti_knn
```

Out[114]:

```
array([0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1,
       0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1,
       0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0],
      dtype=int64)
```

In [115]:

```
confusion_matrice_knn = confusion_matrix(y_test, y_esti_knn)
confusion_matrice_knn
```

Out[115]:

```
array([[96,  2],
       [14, 84]], dtype=int64)
```

In [116]:

```
sensitivity_knn = confusion_matrice_knn[1,1]/(confusion_matrice_knn[1,1]+confusion_matrice_
sensitivity_knn
```

Out[116]:

```
0.8571428571428571
```

Réseau de neurones artificiel (ANN)

In [117]:

```
mlp = MLPClassifier(hidden_layer_sizes=(10), max_iter=20000,
                    solver='adam')
mlp.fit(X_train, y_train)
print("Training set score: %f" % mlp.score(X_train, y_train))
```

```
Training set score: 0.980964
```

In [118]:

```
#print("Test set score: %f" % mlp.score(X_test, y_test))
prediction = mlp.predict(X_test)
```

In [119]:

```
confusion_matrice = confusion_matrix(y_test, prediction)
print(confusion_matrice)
```

```
[[95  3]
 [11 87]]
```

In [120]:

```
specificity = confusion_matrice[0,0]/(confusion_matrice[0,0]+confusion_matrice[0,1])
print('Specificity : ', specificity)

sensitivity = confusion_matrice[1,1]/(confusion_matrice[1,1]+confusion_matrice[1,0])
print('Sensitivity : ', sensitivity )
```

```
Specificity :  0.9693877551020408
Sensitivity :  0.8877551020408163
```

In [121]:

```
from sklearn.model_selection import cross_validate
cross_validate(mlp, X, y, cv=5)
```

Out[121]:

```
{'fit_time': array([21.80803299, 21.86684465, 24.33326626, 25.44723725, 22.5
3883433]),
 'score_time': array([0.03989744, 0.03590322, 0.03590417, 0.0388968 , 0.0289
2399]),
 'test_score': array([0.99882378, 0.99954356, 0.99455768, 0.99945577, 0.9994
3821])}
```

Permutation d'importance

In [122]:

```
perm = PermutationImportance(mlp).fit(X_test, y_test)
eli5.show_weights(perm,feature_names = X_train.columns.tolist(),top=31)
```

Out[122]:

Weight	Feature
0.1622 ± 0.0362	V14
0.1184 ± 0.0510	V12
0.0959 ± 0.0489	V4
0.0337 ± 0.0247	V9
0.0337 ± 0.0153	V3
0.0255 ± 0.0171	V21
0.0235 ± 0.0426	V11
0.0214 ± 0.0119	V5
0.0214 ± 0.0135	V1
0.0194 ± 0.0176	V8
0.0133 ± 0.0082	V18
0.0102 ± 0.0065	Amount
0.0082 ± 0.0178	V17
0.0082 ± 0.0200	V27
0.0071 ± 0.0050	V13
0.0071 ± 0.0138	Time
0.0071 ± 0.0122	V10
0.0071 ± 0.0178	V7
0.0061 ± 0.0076	V28
0.0061 ± 0.0076	V24
0.0061 ± 0.0198	V2
0.0051 ± 0.0144	V16
0.0051 ± 0.0065	V6
0.0051 ± 0.0091	V22
0.0041 ± 0.0135	V20
0.0041 ± 0.0119	V19
0.0000 ± 0.0129	V23
-0.0010 ± 0.0176	V15
-0.0020 ± 0.0082	V26
-0.0020 ± 0.0138	V25

In [123]:

```
### Sans le drop
mlp = MLPClassifier(hidden_layer_sizes=(10), max_iter=100, solver='lbfgs', verbose=False)
mlp.fit(X_train, y_train)
print('Taux de précision sans drop', mlp.score(X_test, y_test)*100)

### Avec le drop
X_train_2=X_train.drop(['V18','V15','V25'],axis=1)
X_test_2=X_test.drop(['V18','V15','V25'],axis=1)

mlp2 = MLPClassifier(hidden_layer_sizes=(10), max_iter=20, solver='lbfgs', verbose=False)
mlp2.fit(X_train_2, y_train)
print('Taux de précision avec drop de V18, V15, V25', mlp2.score(X_test_2, y_test)*100)
```

```
Taux de précision sans drop 90.3061224489796
Taux de précision avec drop de V18, V15, V25 89.79591836734694
```

In [ ]:

```

```