```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import cross_val_score

# Supervised learning
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.model_selection import KFold, train_test_split
import statsmodels.formula.api as smf
import statsmodels.api as sm

# Unsupervised learning
from sklearn.decomposition import PCA, FastICA
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler, Normalizer
from scipy.optimize import linear_sum_assignment
```

Project

Madeleine Ekblom, Matias Jääskeläinen, Jakub Kubečka

Time used: many hours

Read in the data

```
dummy = pd.read_csv("/content/sample_data/dummy.csv")
npf_train = pd.read_csv("/content/sample_data/npf_train.csv")
npf_test_hidden = pd.read_csv("/content/sample_data/npf_test_hidden.csv")
```

See the columns

```
npf_train.head(2)
```

	id	date	event	partlybad	HYY_META.CO2168.mean	HYY_META.CO2168.std	HYY
0	1	2000-01-23	nonevent	False	373.496585	0.189497	
1	2	2000-01-25	nonevent	False	381.752738	1.701439	

```
npf_test_hidden.head(2)
```

	id	date	event	partlybad	HYY_META.CO2168.mean	HYY_META.CO2168.std	HYY_
0	725	NaN	NaN	False	372.575187	10.051405	
1	726	NaN	NaN	False	382.408306	0.752684	

2 rows × 104 columns

Split the data into features and labels

```
#del X
X = npf_train.iloc[:,4::2].copy()
#X = npf_train.iloc[:,4:]

#del test
test = npf_test_hidden.iloc[:,4::2].copy()
#test = npf_test_hidden.iloc[:,4:]

#del Y
Y = npf_train.iloc[:,2]
```

```
X.head(1)
```

	HYY_META.CO2168.mean	HYY_META.CO2336.mean	HYY_META.CO242.mean	HYY_META.C
0	373.496585	373.382593	373.961481	

```
Y[:2]
```

```
0    nonevent
1    nonevent
Name: event, dtype: object
```

```
test.head(1)
```

	HYY_META.CO2168.mean	HYY_META.CO2336.mean	HYY_META.CO242.mean	HYY_META.C
0	372.575187	372.324439	376.135989	

Make Y (labels) categorical

```
labeler = LabelEncoder()
Y = labeler.fit_transform(Y.astype('str'))
list(labeler.classes_), set(Y)
```

```
(['II', 'Ia', 'Ib', 'nonevent'], {0, 1, 2, 3})
```

```
Y[:2]
```

```
array([3, 3])
```

Scale X (features)

```
scaler = StandardScaler()
X = scaler.fit(X).transform(X)

test = scaler.transform(test)
```

```
X.shape, test.shape
```

```
((724, 50), (724, 50))
```

Reduce the number of dimensions

First test how many PCA-components should be used

```
pca_test = PCA(n_components=30)
pca_test.fit(X)
```

```
PCA(copy=True, iterated_power='auto', n_components=30, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

```
np.sum(pca_test.explained_variance_ratio_)
```

```
0.9999269315979487
```

```
plt.plot(np.arange(1,31),pca_test.explained_variance_ratio_,'o-');
plt.plot(np.arange(1,31),np.cumsum(pca_test.explained_variance_ratio_),'o-');
```

Transform the labels and the test set into 4 PC:s

```
pca = PCA(n_components=4)
npf_pca = pca.fit_transform(X)

test_pca = pca.transform(test)
```

```
npf_pca.shape, type(npf_pca), test_pca.shape, type(test_pca)
```

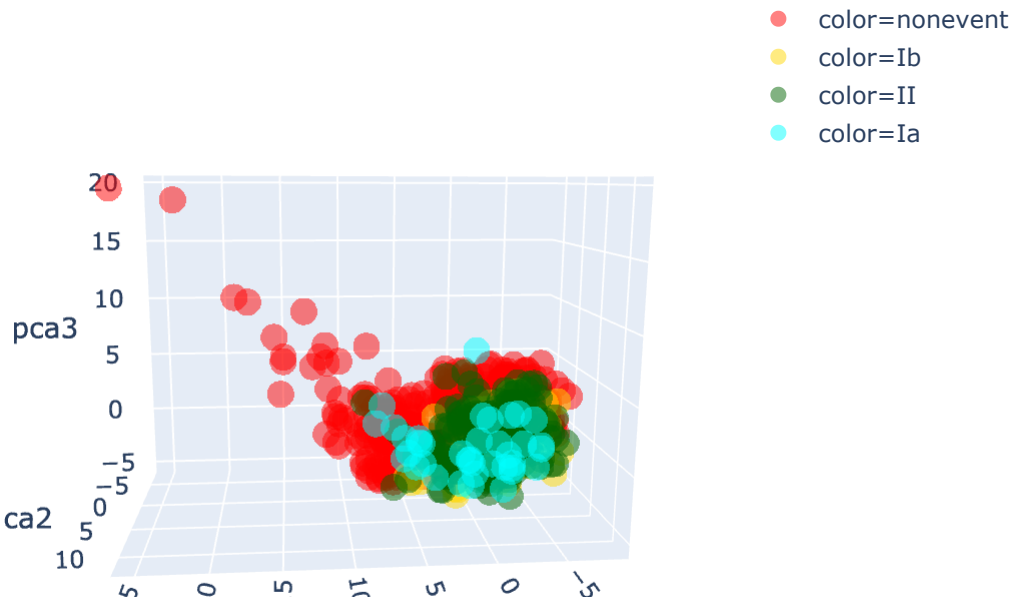
```
((724, 4), numpy.ndarray, (724, 4), numpy.ndarray)
```

```
own_cmap = [
    "red",
    "gold",
    "darkgreen",
    "cyan",
    "dodgerblue",
    "royalblue",
    "blue",
    "blueviolet",
    "plum",
    "deeppink",
    "magenta",
    "purple",
    "saddlebrown",
    "lightsalmon",
    "k"]
```

```
npf_pca = pd.DataFrame(npf_pca, columns=["pca1","pca2","pca3","pca4"])

test_pca = pd.DataFrame(test_pca, columns=["pca1","pca2","pca3","pca4"])

klabels = labeler.inverse_transform(Y)
px.scatter_3d(data_frame=npf_pca,
              x="pca1",
              y="pca2",
              z="pca3",
              color=klabels,
              color_discrete_sequence=own_cmap,
              hover_name=npf_pca.index,
              opacity=0.5,
              width= 600,
              height= 500)
```



Test ICA also

```
ica = FastICA(n_components=4)
npf_ica = ica.fit_transform(X)

test_ica = ica.transform(test)
```

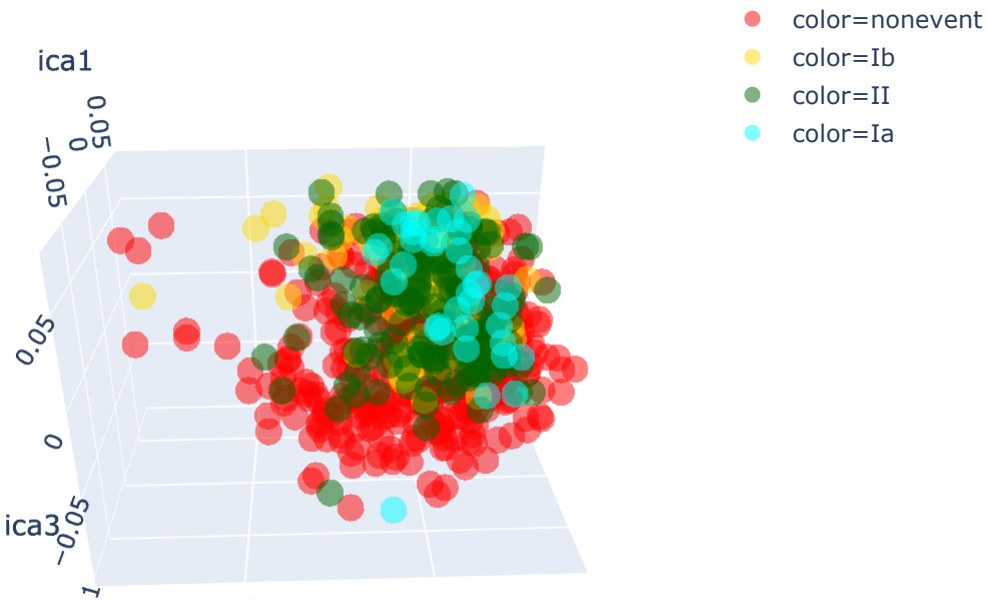
```
npf_ica.shape, type(npf_ica), test_ica.shape, type(test_ica)
```

```
((724, 4), numpy.ndarray, (724, 4), numpy.ndarray)
```

```
npf_ica = pd.DataFrame(npf_ica, columns=["ica1","ica2","ica3","ica4"])

test_ica = pd.DataFrame(test_ica, columns=["ica1","ica2","ica3","ica4"])

px.scatter_3d(data_frame=npf_ica,
              x="ica1",
              y="ica2",
              z="ica3",
              color=klabels,
              color_discrete_sequence=own_cmap,
              hover_name=npf_ica.index,
              opacity=0.5,
              width= 600,
              height= 500)
```



Try t-SNE for visualisation

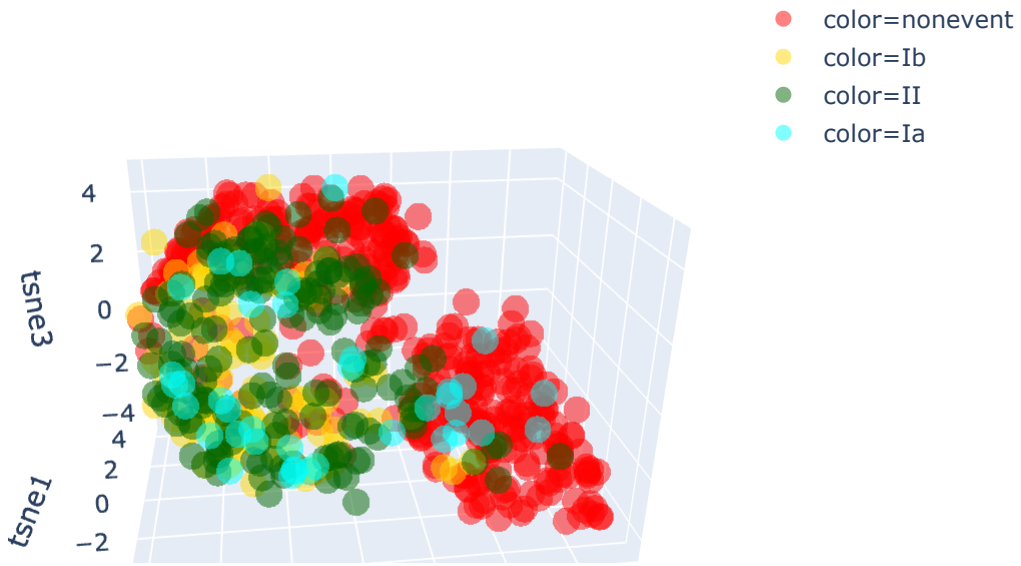
It does maybe make sense but it looks cool in the CV

```
 /\ /\
((ovo))
():::()
  VVV
```

```
npf_tsne = TSNE(n_components=3,perplexity=100,random_state=1).fit_transform(X)

npf_tsne = pd.DataFrame(npf_tsne,columns=['tsne1','tsne2','tsne3'])

px.scatter_3d(data_frame=npf_tsne,
              x="tsne1",
              y="tsne2",
              z="tsne3",
              color=klabels,
              color_discrete_sequence=own_cmap,
              hover_name=npf_tsne.index,
              opacity=0.5,
              width= 600,
              height= 500)
```



Split the train data into train and validation

`X_train` = training features

`X_test` = validation features

`Y_train` = training labels

`Y_test` = validation labels

```
X_train,X_test,Y_train,Y_test = train_test_split(npf_ica,Y,random_state=42)
```

```
X_train.shape,X_test.shape,Y_train.shape,Y_test.shape
```

```
((543, 4), (181, 4), (543,), (181,))
```

Try different classification models

```
names = ["Nearest Neighbors", "Linear SVM", "RBF SVM", "Gaussian Process",
         "Decision Tree", "Random Forest", "Neural Net", "AdaBoost",
         "Naive Bayes", "QDA"]

classifiers = [
    KNeighborsClassifier(3),
    SVC(kernel="linear", C=0.025),
    SVC(gamma=2, C=1),
    GaussianProcessClassifier(1.0 * RBF(1.0)),
    DecisionTreeClassifier(max_depth=5),
    RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1),
    MLPClassifier(alpha=1, max_iter=1000),
    AdaBoostClassifier(),
    GaussianNB(),
    QuadraticDiscriminantAnalysis()]
```

Use cross-validation to find the best model

```
# iterate over classifiers
for name, clf in zip(names, classifiers):
    clf.fit(X_train, Y_train)
    score = clf.score(X_test, Y_test)
    print("%s:  %0.2f %%" % (name,score*100))
```

```
Nearest Neighbors:  64.64 %
Linear SVM:  55.25 %
RBF SVM:  59.12 %
Gaussian Process:  65.75 %
Decision Tree:  61.33 %
Random Forest:  64.09 %
Neural Net:  61.33 %
AdaBoost:  58.56 %
Naive Bayes:  60.77 %
QDA:  63.54 %
```

Use 10-fold cross-validation to find the best model

```
#cross validation
# DECLARATION #
kf = KFold(n_splits=10)
# CODE #
scores=[]
# iterate over k-folds
for train, val in kf.split(npf_pca):
    scores_iter=[]
    # iterate over classifiers
    for name, clf in zip(names, classifiers):
        clf.fit(npf_pca.iloc[train], Y[train])
        score = clf.score(npf_pca.iloc[val], Y[val])
        scores_iter.append(score)
        #print("{}:  {}".format(name,score))
    scores.append(scores_iter)
#print(scores)
scores=np.mean(scores,axis=0)
#print(scores)
for name, score in zip(names, scores):
     print("%s:  %0.2f %%" % (name,score*100))
```

```
Nearest Neighbors:  62.87 %
Linear SVM:  61.08 %
RBF SVM:  57.52 %
Gaussian Process:  68.41 %
Decision Tree:  61.92 %
Random Forest:  64.82 %
Neural Net:  67.58 %
AdaBoost:  59.70 %
Naive Bayes:  64.81 %
QDA:  65.21 %
```

Train the model

We pick Gaussian processes

```
model = GaussianProcessClassifier(1.0 * RBF(1.0))
# Wanted to test also the neural net
#model = MLPClassifier(alpha=1, max_iter=1000)
```

Fit the model

```
model.fit(npf_ica,Y)
```

```
GaussianProcessClassifier(copy_X_train=True, kernel=1**2 * RBF(length_scale=1)
                          max_iter_predict=100, multi_class='one_vs_rest',
                          n_jobs=None, n_restarts_optimizer=0,
                          optimizer='fmin_l_bfgs_b', random_state=None,
                          warm_start=False)
```

Predict values for npf_test_hidden.csv

Using ICA transformed data

```
prediction = model.predict(test_ica)
#prediction = model.predict(npf_ica)    # this is a test
prediction = labeler.inverse_transform(prediction)

probs = model.predict_proba(test_ica)
#probs = model.predict_proba(npf_ica)    # this is a test
#probs = np.max(probs, axis=1)
```

```
# sum probabilities of event = p(1a) + p(1b) + p(2), p(nonevent) is 3rd columr
probs = np.sum(probs[:,:3], axis=1)
```

```
results_df = pd.DataFrame(zip(prediction,probs),columns=["event","p_event"])
```

```
results_df.head(10)
```

	event	p_event
0	nonevent	0.467028

Print out a csv

```
results_df.to_csv("/content/drive/My Drive/Colab Notebooks/answers.csv",index=
```

3	nonevent	0.334076
5	nonevent	0.153963
6	nonevent	0.166439
7	II	0.946102
8	nonevent	0.058983
9	II	0.710914