Multivariate_Analysis_Metabolomics_Workflow

August 15, 2022

1 Introduction

The purspose of this notebook was to ilustrate the data analysis perform along my MSc thesis, with a focus on unsupervised as well as supervised Multivariate Analysis (MVA) perform on a metabolomic data set. It was also shown how tools namely Probabilistics Quotient Normalization (PQN) or Viriable Importance in Projection (VIP) were implemented in Python. Furthermore, close to the end of the analysis it was described how to mass matching was carried out.

The data set used in this project came from a neurotoxicology experiment where astrocyte cells were exposed to different concentration of digoxin. Digoxin is one of the oldest cardiovascular medication used nowadays; digoxin is a common agent used to manage atrial fribrillation and the symptoms of heart failure as it is a positive ionotropic and negative chronotropic drug which means that digoxin increases the force of the heartbeat and decreases the heart rate.

The digoxin data set corresponded to 6 experimental groups by 4 replicates, leading to a total of twenty-four samples. The experimental groups could be summarized as *Negative Control, Exposed Groups, and Positive Control*. On the Exposed groups the concentration of digoxin ranged from 0 to 10 $\mu molar$. The Positive Control group was exposed to TNFα.

The different steps of the data analysis carried out in this project can be summarized as follow:

- Exploratory Data Analysis
  - Principal Component Analysis (PCA)
  - Hierarchical Clustering Analysis (HCA)
- Supervised modelling
  - Partial Least Squares (PLS)
  - Random Forest (RF)
- Variable Selection

Author: Christian Peralta

All rights reserved

1.1 Importing libraries

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

2 Data Preparation

The original data set contained several column which were not of interest for a first exploratory data analysis (EDA) where regardless of the model to use, the best practice is to have a data matrix containing only [observations x predictors]. In addition to perform data cleaning, observarions label was modified to a shorter but meaningful name, enabling a straighforward interpretation of the results.

[2]:
```
df = pd.read_csv("20201209_MGZ_720_RP.csv", sep= "\t", header= 4)
df.shape
```

[2]:
```
(3255, 71)
```

[3]:
```
df.columns
```

[3]:
```
Index(['Compound', 'Neutral mass (Da)', 'm/z', 'Charge',
       'Retention time (min)', 'Chromatographic peak width (min)',
       'Identifications', 'Anova (p)', 'q Value', 'Max Fold Change',
       'Highest Mean', 'Lowest Mean', 'Isotope Distribution',
       'Maximum Abundance', 'Minimum CV%', 'MSMS info available', 'Rt',
       'KEGG in silico', 'Waters CCS library', 'LipidBlast', 'A revoir',
       'Antibiotic', 'pH indicator', 'Accepted ID', 'Accepted Compound ID',
       'Accepted Description', 'Adducts', 'Formula', 'Score',
       'Fragmentation Score', 'Mass Error (ppm)', 'Isotope Similarity',
       'Retention Time Error (mins)', 'Compound Link', 'Privileged',
       'Control A_1-A,1_01_201', 'Control B_1-A,2_01_210',
       'Control C_1-A,3_01_216', 'Control D_1-A,4_01_191',
       'Digoxine conc 0.0 A _1-B,1_01_214',
       'Digoxine conc 0.0 B _1-B,2_01_218', 'Digoxine conc 0.0 C_1-B,3_01_190',
       'Digoxine conc 0.0 D _1-B,4_01_202', 'Digoxine conc 0.1 A_1-C,1_01_205',
       'Digoxine conc 0.1 B_1-C,2_01_209', 'Digoxine conc 0.1 C_1-C,3_01_198',
       'Digoxine conc 0.1 D_1-C,4_01_192', 'Digoxine conc 1.0 A_1-D,1_01_208',
       'Digoxine conc 1.0 B_1-D,2_01_200', 'Digoxine conc 1.0 C_1-D,3_01_193',
       'Digoxine conc 1.0 D_1-D,4_01_215', 'Digoxine conc 10.0 A_1-E,1_01_213',
       'Digoxine conc 10.0 B_1-E,2_01_206',
       'Digoxine conc 10.0 C_1-E,3_01_199',
       'Digoxine conc 10.0 D _1-E,4_01_194', 'TNF-alfa A_1-F,1_01_207',
       'TNF-alfa B_1-F,2_01_217', 'TNF-alfa C_1-F,3_01_197',
       'TNF-alfa D_1-F,4_01_189', 'dQC_1-A,8_01_186', 'dQC_1-A,8_01_188',
       'dQC_1-A,8_01_196', 'dQC_1-A,8_01_204', 'dQC_1-A,8_01_212',
       'QC_1-A,7_01_185', 'QC_1-A,7_01_187', 'QC_1-A,7_01_195',
       'QC_1-A,7_01_203', 'QC_1-A,7_01_211', 'QC_1-A,7_01_219',
       'QC_1-A,7_01_221'],
      dtype='object')
```

The data frame was cleaned to select those column corresponding to the experimental samples.

[4]:
```
df2 = df.iloc[:, 35:]
df2.columns
```

[4]:
```
Index(['Control A_1-A,1_01_201', 'Control B_1-A,2_01_210',
       'Control C_1-A,3_01_216', 'Control D_1-A,4_01_191',
       'Digoxine conc 0.0 A _1-B,1_01_214',
       'Digoxine conc 0.0 B _1-B,2_01_218', 'Digoxine conc 0.0 C_1-B,3_01_190',
       'Digoxine conc 0.0 D _1-B,4_01_202', 'Digoxine conc 0.1 A_1-C,1_01_205',
       'Digoxine conc 0.1 B_1-C,2_01_209', 'Digoxine conc 0.1 C_1-C,3_01_198',
       'Digoxine conc 0.1 D_1-C,4_01_192', 'Digoxine conc 1.0 A_1-D,1_01_208',
       'Digoxine conc 1.0 B_1-D,2_01_200', 'Digoxine conc 1.0 C_1-D,3_01_193',
       'Digoxine conc 1.0 D_1-D,4_01_215', 'Digoxine conc 10.0 A_1-E,1_01_213',
       'Digoxine conc 10.0 B_1-E,2_01_206',
       'Digoxine conc 10.0 C_1-E,3_01_199',
       'Digoxine conc 10.0 D _1-E,4_01_194', 'TNF-alfa A_1-F,1_01_207',
       'TNF-alfa B_1-F,2_01_217', 'TNF-alfa C_1-F,3_01_197',
       'TNF-alfa D_1-F,4_01_189', 'dQC_1-A,8_01_186', 'dQC_1-A,8_01_188',
       'dQC_1-A,8_01_196', 'dQC_1-A,8_01_204', 'dQC_1-A,8_01_212',
       'QC_1-A,7_01_185', 'QC_1-A,7_01_187', 'QC_1-A,7_01_195',
       'QC_1-A,7_01_203', 'QC_1-A,7_01_211', 'QC_1-A,7_01_219',
       'QC_1-A,7_01_221'],
      dtype='object')
```

As one can see, above, the names were long. The name was given following a certain experimental regulation which is not of interest for data analysis.

[5]:
```
new_names = ['Control_A', 'Control_B', 'Control_C', 'Control_D', 'Conc_0.0_A',␣
 ↪'Conc_0.0_B', 'Conc_0.0_C', 'Conc_0.0_D',
              'Conc_0.1_A', 'Conc_0.1_B', 'Conc_0.1_C', 'Conc_0.1_D', 'Conc_1.
 ↪0_A', 'Conc_1.0_B', 'Conc_1.0_C', 'Conc_1.0_D',
              'Conc_10.0_A', 'Conc_10.0_B','Conc_10.0_C', 'Conc_10.0_D',␣
 ↪'TNF-a_A', 'TNF-a_B', 'TNF-a_C', 'TNF-a_D', 'dQC_1',
              'dQC_2', 'dQC_3', 'dQC_4', 'dQC_5','QC_1', 'QC_2', 'QC_3', 'QC_4',␣
 ↪'QC_5', 'QC_6', 'QC_7']
```

[6]:
```
# using set_axis for a clean axis manipulation
df2 =df2.set_axis(new_names, axis=1)
```

Once the data set was filtered the last step to be ready for EDA was to transpose the data set, hence it was switched from [predictors x observations] to [observarions x predictors]. Where the observations were stored as the index of the pandas.DataFrame instance.

[7]:
```
# transposing the data frame
df3 = df2.T

# defining column name for the index
df3.index.name = "samples"
```

```
# checking the dimensions
df3.shape
```

[7]: (36, 3255)

3 Exploratory Data Analysis

Typical multivariate analysis workflow in metabolomics starts with exploratory data analysis (EDA) which consists on unsupervised modeling to provide a first overview of the data. Unsupervised statistical tools aim at building models summarizing the data set in an intelligible manner and hopefully finding natural partitions of the dataset to facilitate the understanding of the relationship between the samples and detect potential outliers. The models also provide information about the variables that are responsible for these relationships.

The EDA can be seen as an iterative process where sample groups were excluded from the model at each iteration until keeping only exposure-related samples whilst understanding the data. The first PCA model was built using the entire data set, which included QCs and dQCs, aming at having a first overview of the data focused on studing the quality of the data acquisition.

3.1 How to perform Principal Component Analysis in Python

[8]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

The data must be standardize before PCA

[9]:
```
#normalizing the data
df_pca = StandardScaler().fit_transform(df3)
```

[10]:
```
pca = PCA(n_components=.95)
Pcomponents = pca.fit_transform(df_pca)
```

[11]:
```
# computing explaining variable
var = pca.explained_variance_ratio_*100
```

The scores are returned directly from sklearn function, however, loadings have to be manually computed from sklearn output.

To obtain loadings, the loading matrix (also known as cross-correlation matrix) which is given by the equation $loadings = eigenvectors * \sqrt{eigenvalues}$ and can be done with sklearn as follows: `loadings = pca.components_.T * np.sqrt(pca.explained_variance_)`

[12]:
```
loadings = pd.DataFrame(pca.components_.T * np.sqrt(pca.explained_variance_))
```

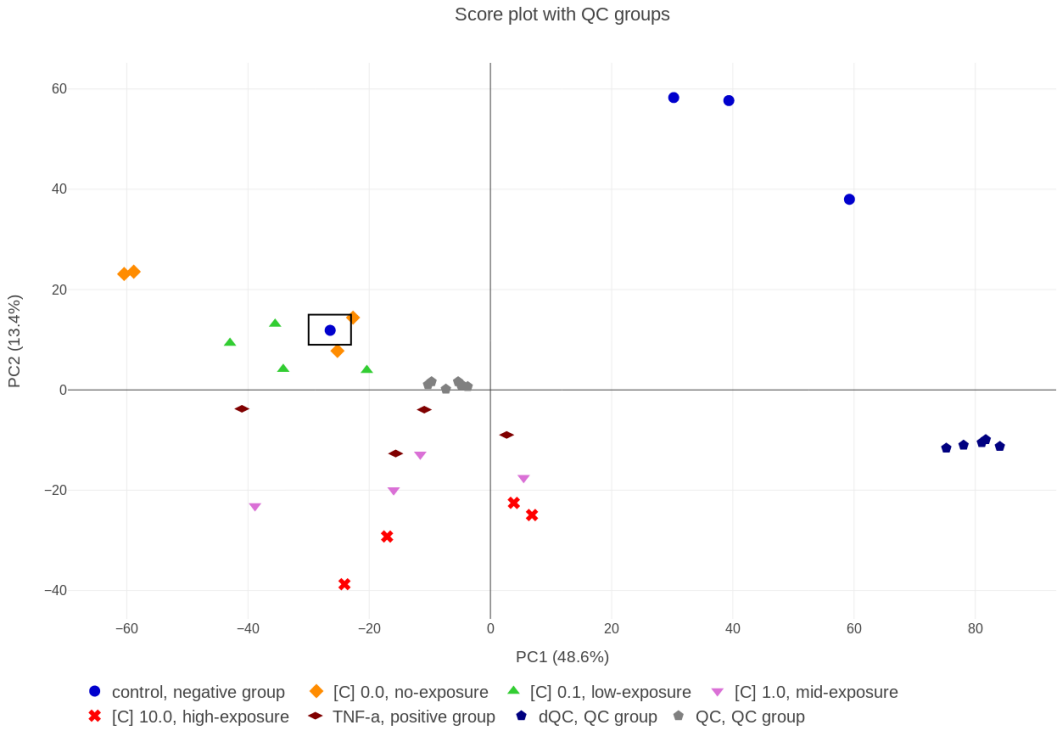
3.2 Complete data set

It was defined four list of names to help with visualization and interpretation of the results by using them to set color and shape of markers as well as color and symbol sequence

[13]:
```
color = ["control", "control", "control", "control", "[C] 0.0", "[C] 0.0","[C]␣
↪0.0","[C] 0.0", "[C] 0.1", "[C] 0.1",
                        "[C] 0.1","[C] 0.1", "[C] 1.0", "[C] 1.0","[C] 1.
↪0","[C] 1.0","[C] 10.0", "[C] 10.0",
                        "[C] 10.0","[C] 10.0", "TNF-a",␣
↪"TNF-a","TNF-a","TNF-a", "dQC", "dQC", "dQC", "dQC", "dQC","QC", "QC", "QC",␣
↪"QC",
                        "QC", "QC", "QC"]

color_seq=["mediumblue","darkorange", "limegreen", "orchid","red", "maroon",␣
↪"navy", "grey"]

symbol = ["negative group", "negative group", "negative group", "negative␣
↪group", "no-exposure","no-exposure","no-exposure",
       "no-exposure",␣
↪"low-exposure","low-exposure","low-exposure","low-exposure",
       "mid-exposure","mid-exposure","mid-exposure","mid-exposure",␣
↪"high-exposure","high-exposure","high-exposure",
       "high-exposure", "positive group", "positive group","positive␣
↪group","positive group", "QC group", "QC group", "QC group", "QC group", "QC␣
↪group",
       "QC group", "QC group", "QC group", "QC group", "QC group",
       "QC group", "QC group"]

symbol_seq=["circle","diamond", "triangle-up","triangle-down",␣
↪"x","diamond-wide", "pentagon"]
```

Score plot visualization

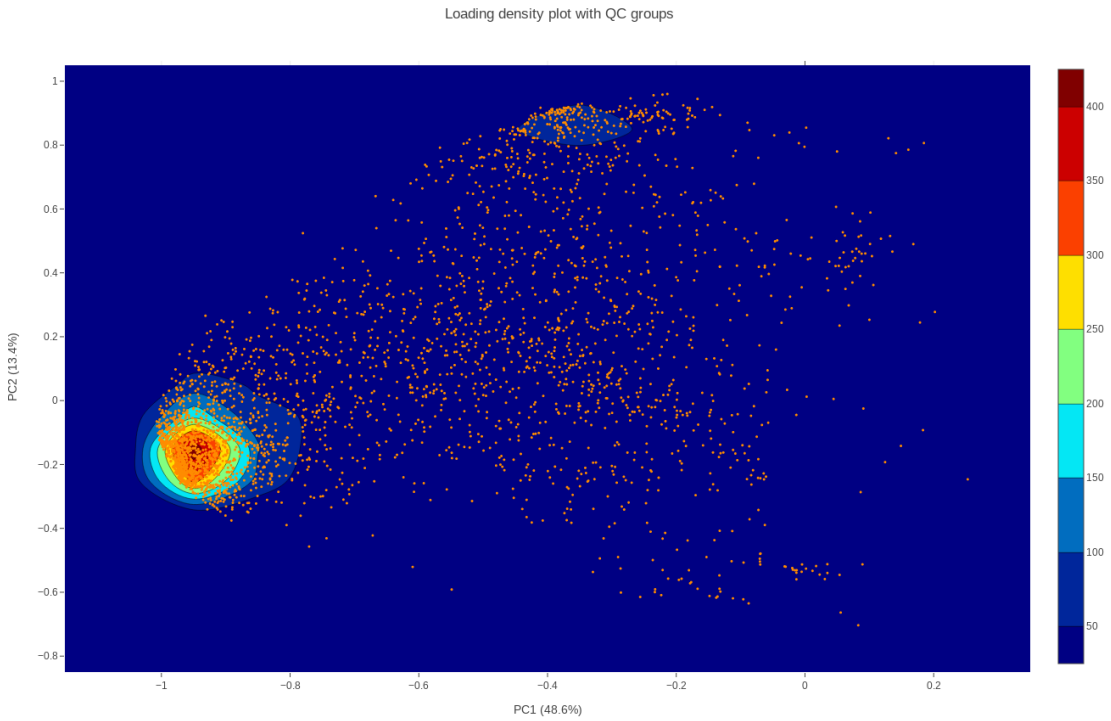
[14]:
```
score_plot = px.scatter(Pcomponents, x= Pcomponents[:,0], y=Pcomponents[:,1]␣
↪,labels = {"x" : f"PC1 ({var[0]:.1f}%)", "y" : f"PC2 ({var[1]:.1f}%)"},
                color=color, template= "presentation", width= 900, height= 900,
                title= "Score plot with QC groups", hover_name= df3.index,␣
↪symbol=symbol,
                symbol_sequence=symbol_seq, color_discrete_sequence=color_seq)

score_plot.update_layout(legend_title=None, font=dict(size=16),␣
↪legend=dict(yanchor="top", y=-.1, xanchor="left", x=0.01, orientation="h",␣
↪font_size=20))
score_plot.update_traces(marker=dict(size=13))
score_plot.add_shape(type="rect", x0=-30, x1=-23, y0=9, y1=15)
score_plot.show()
```



PCA model was built on the complete data set where the above score plot, showed how PC1 was driven by the dQCs samples, whereas PC2 was mainly driven by the control group, which seemed to be orthogonal to digoxin effect. In addition, on the PC2 the trending of digoxin concentration was reflected; on top there were the samples with none or lower digoxin concentration, followed by samples with the higher concentration. One sample belonging to the control group - highlighted within a rectangle - was found in between [C] 0.0 and [C] 0.1 samples, hence it was considered as an outlier. In the middle of the trending, there were located the positive control group, e.g. TNF- , and close to the origin one could find QCs samples which were a pooled mixture of all the samples and acted as a biological mean.

Loading plot visualization

[15]:
```
loading_plot = go.Figure()
loading_plot.add_trace(go.Scatter(x=loadings.iloc[:,0], y=loadings.iloc[:,1],␣
 ↪mode="markers", marker=dict(size=3, color="darkorange"), xaxis="x",␣
 ↪yaxis="y"))
loading_plot.add_histogram2dcontour(x=loadings.iloc[:,0], y=loadings.iloc[:,1],␣
 ↪ colorscale="Jet", xaxis="x", yaxis="y")
loading_plot.update_layout(template="presentation", width= 900, height= 900,␣
 ↪hovermode=False, font=dict(size=12), title="Loading density plot with QC␣
 ↪groups")
loading_plot.update_xaxes(title_text=f"PC1 ({var[0]:.1f}%)")
loading_plot.update_yaxes(title_text=f"PC2 ({var[1]:.1f}%)")
```



Through the loadings plot, it was confirmed that the PC1 was driven by dQC, as aforementioned, since the concentration of the loading at the positive side was notably lower than at the negative side. This was related to the fact that dQC were a diluted mixture of all samples, hence containing lower concentration of compounds.

After a first exploration of the data, it was highlighted that the data was acquired with high quality as the QCs and dQCs samples were well grouped. Moreover, one negative control sample was considered as an outlier and that the negative control group seemed to behave orthogonally to digoxin effect. Consequently, next exploratory steps included removal of QCs/dQCs to observe how the first 2 PCs behaved.

3.3 Data set excluding: *QCs/dQCs*

[16]:
```
# using pd.filter method with negative regex matching to filter out QC samples
df_filtered = df3.filter(axis=0, regex="(?m)^(?!(^dQC|QC).)")
```

[17]:
```
color = ["control", "control", "control", "control", "[C] 0.0", "[C] 0.0","[C]␣
↪0.0","[C] 0.0", "[C] 0.1", "[C] 0.1",
                        "[C] 0.1","[C] 0.1", "[C] 1.0", "[C] 1.0","[C] 1.
↪0","[C] 1.0","[C] 10.0", "[C] 10.0",
                        "[C] 10.0","[C] 10.0", "TNF-a", "TNF-a","TNF-a","TNF-a"]

color_seq=["mediumblue","darkorange", "limegreen", "orchid","red", "maroon"]
```

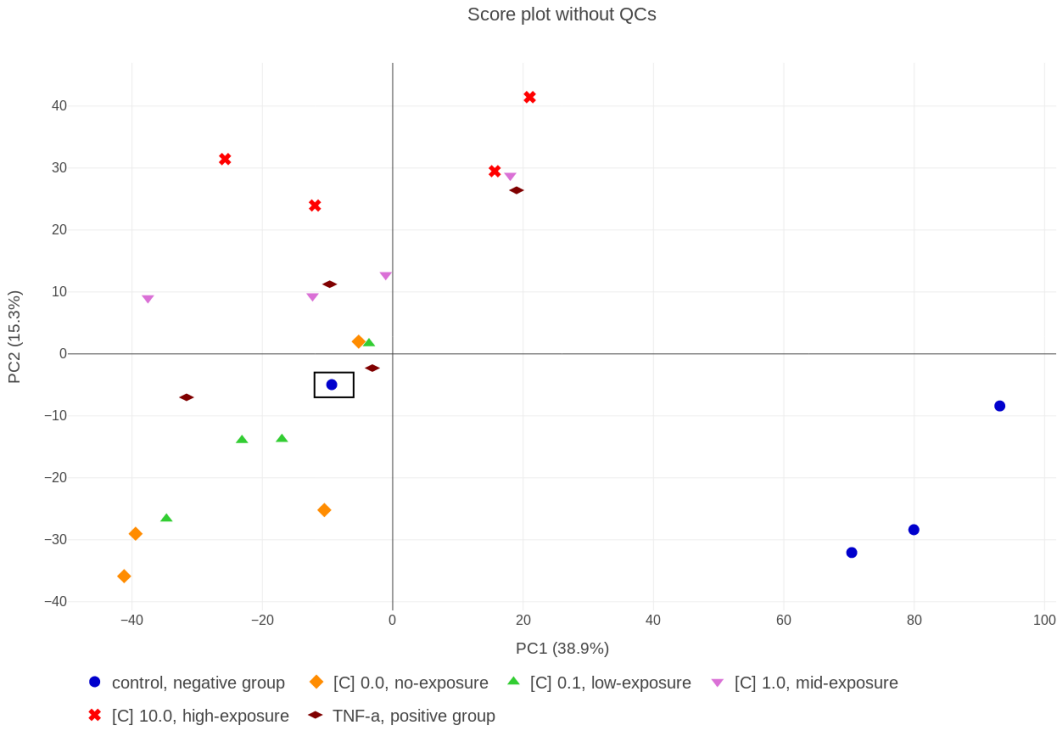
```
symbol = ["negative group", "negative group", "negative group", "negative␣
 ↪group", "no-exposure","no-exposure","no-exposure",
       "no-exposure",␣
 ↪"low-exposure","low-exposure","low-exposure","low-exposure",
       "mid-exposure","mid-exposure","mid-exposure","mid-exposure",␣
 ↪"high-exposure","high-exposure","high-exposure",
       "high-exposure", "positive group", "positive group","positive␣
 ↪group","positive group"]

symbol_seq=["circle","diamond", "triangle-up","triangle-down",␣
 ↪"x","diamond-wide", "pentagon"]
```

[18]:
```
#Let's create a pipeline to concatenate scaler and pca
from sklearn.pipeline import Pipeline

pca_out= PCA(n_components=.95)
pca_pipe = Pipeline([("scale", StandardScaler()), ("pca", pca_out)])
```

[19]:
```
pcomponents = pca_pipe.fit_transform(df_filtered)
var = pca_out.explained_variance_ratio_*100
```

Score plot visualization

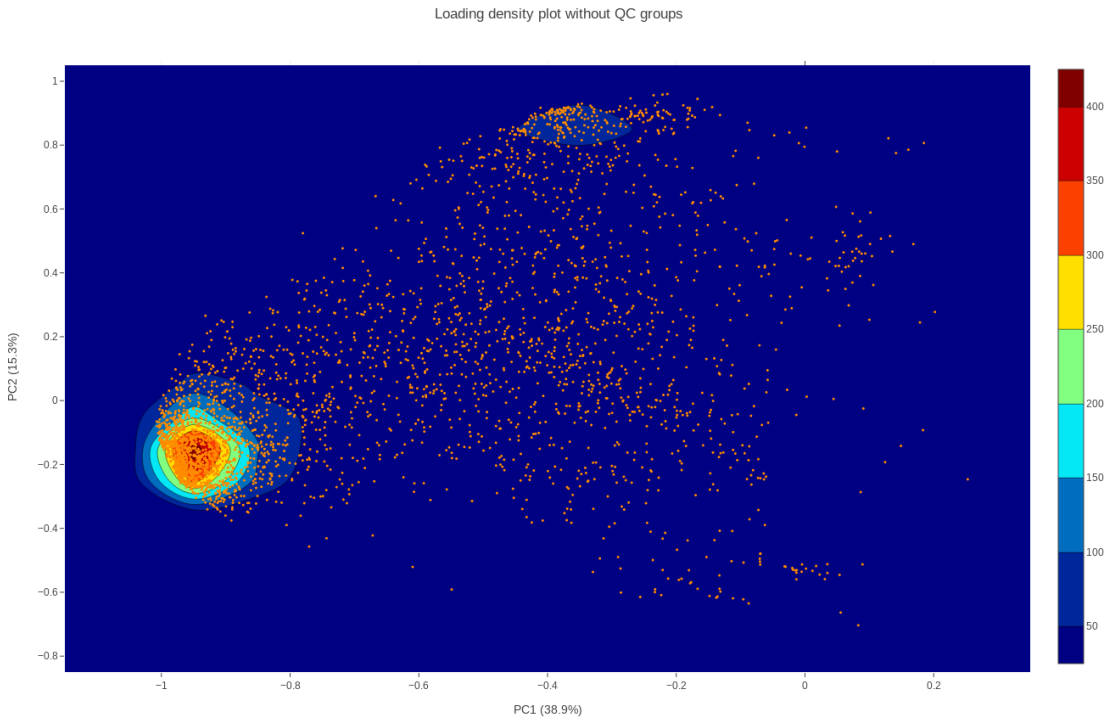
[20]:
```
score_plot = px.scatter(data_frame= pcomponents, x=pcomponents[:,0],␣
 ↪y=pcomponents[:,1], color=color,
                        labels= {"x": f"PC1 ({var[0]:.1f}%)", "y":f"PC2␣
 ↪({var[1]:.1f}%)"}, template= "presentation",  width= 900, height= 900,
 title= "Score plot without QCs", hover_name= df_filtered.index, symbol=symbol,␣
 ↪symbol_sequence=symbol_seq, color_discrete_sequence=color_seq)
score_plot.update_layout(legend_title= None, font=dict(size=16),
                         legend=dict(yanchor="top", y=-.1, xanchor="left", x=0.
 ↪01, orientation="h", font_size=20), overwrite=True)
score_plot.update_traces(marker=dict(size=13))
score_plot.add_shape(type="rect", x0=-12, x1=-6, y0=-7, y1=-3)
score_plot.show()
```



Exclusion of QC groups from the data set, allowed a better visualization of the digoxin trending by the score plot. At the same time, the orthogonality of the negative group to the toxicity of digoxin was plain to see. However, it was hard to interpret what was driving the PCs as the trending in concentration was crossing both PCs rather than been represented by only one.

Loading plot visualization

[21]:
```
loading_plot = go.Figure()
loading_plot.add_trace(go.Scatter(x=loadings.iloc[:,0], y=loadings.iloc[:,1],␣
 ↪mode="markers", marker=dict(size=3, color="darkorange"), xaxis="x",␣
 ↪yaxis="y"))
loading_plot.add_histogram2dcontour(x=loadings.iloc[:,0], y=loadings.iloc[:,1],␣
 ↪ colorscale="Jet", xaxis="x", yaxis="y")
loading_plot.update_layout(template="presentation", width= 900, height= 900,␣
 ↪hovermode=False, font=dict(size=12), title="Loading density plot without QC␣
 ↪groups")
loading_plot.update_xaxes(title_text=f"PC1 ({var[0]:.1f}%)")
loading_plot.update_yaxes(title_text=f"PC2 ({var[1]:.1f}%)")
loading_plot.show()
```

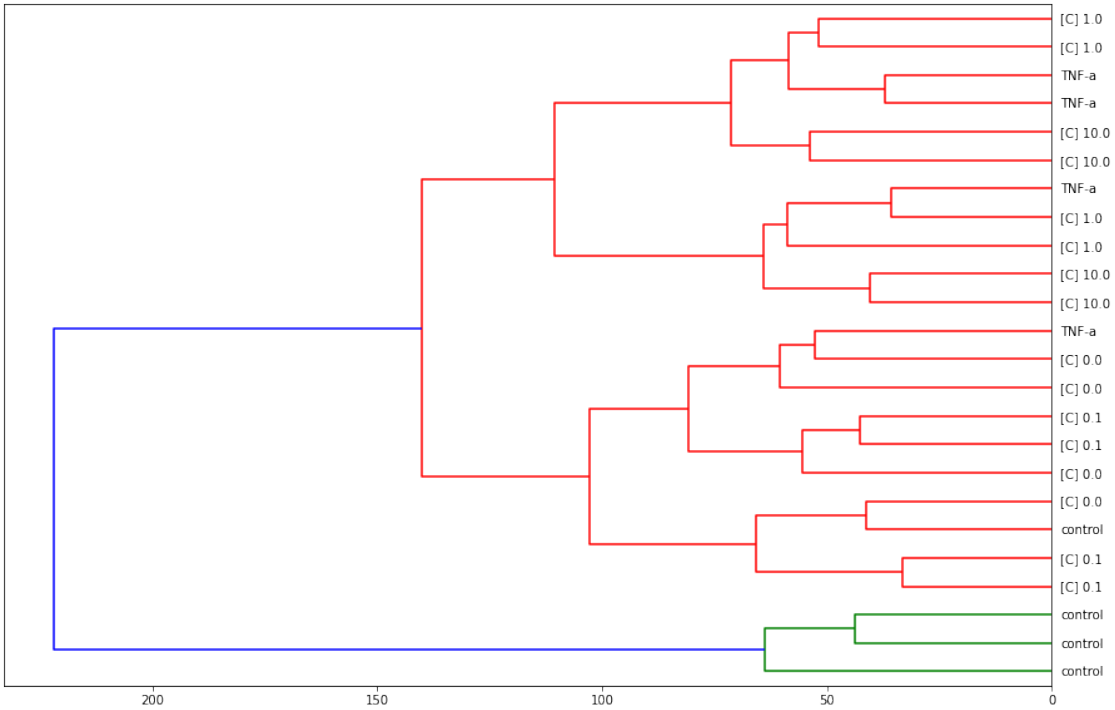


On the other hand, the loading plot, Figure 3.3b, showed an unusual behavior of the loadings. It was observed a hotspot located on the left side of the PC1 which was not related to any sample or group of samples in particular as it was the case in the first PCA model.

HCA was perform of the PCA components to prove what was observed, that control group was orthogonal to the rest of the groups.

[22]:
```
import scipy.cluster.hierarchy as sch
```

[23]:
```
plt.figure(figsize=(15,10))

dendrogram_pca = sch.dendrogram(sch.linkage(pcomponents, method='ward'),␣
↪labels=color, orientation="left")
plt.show()
```



Hence, to continue with the EDA, the negative control group was removed from subsequent PCA models.

3.4 Data set excluding: *QCs/dQCs, negative control groups*

[24]:
```
df_filtered = df_filtered.filter(axis=0,regex="(?m)^(?!^Control*)")
```

[25]:
```
color = ["[C] 0.0", "[C] 0.0","[C] 0.0","[C] 0.0", "[C] 0.1", "[C] 0.1",
                        "[C] 0.1","[C] 0.1", "[C] 1.0", "[C] 1.0","[C] 1.
 ↪0","[C] 1.0","[C] 10.0", "[C] 10.0",
                        "[C] 10.0","[C] 10.0", "TNF-a", "TNF-a","TNF-a","TNF-a"]

color_seq=["darkorange", "limegreen", "orchid","red", "maroon"]

symbol = ["no-exposure","no-exposure","no-exposure",
        "no-exposure",␣
 ↪"low-exposure","low-exposure","low-exposure","low-exposure",
        "mid-exposure","mid-exposure","mid-exposure","mid-exposure",␣
 ↪"high-exposure","high-exposure","high-exposure",
        "high-exposure", "positive group", "positive group","positive␣
 ↪group","positive group"]

symbol_seq=["diamond", "triangle-up","triangle-down", "x","diamond-wide",␣
 ↪"pentagon"]
```

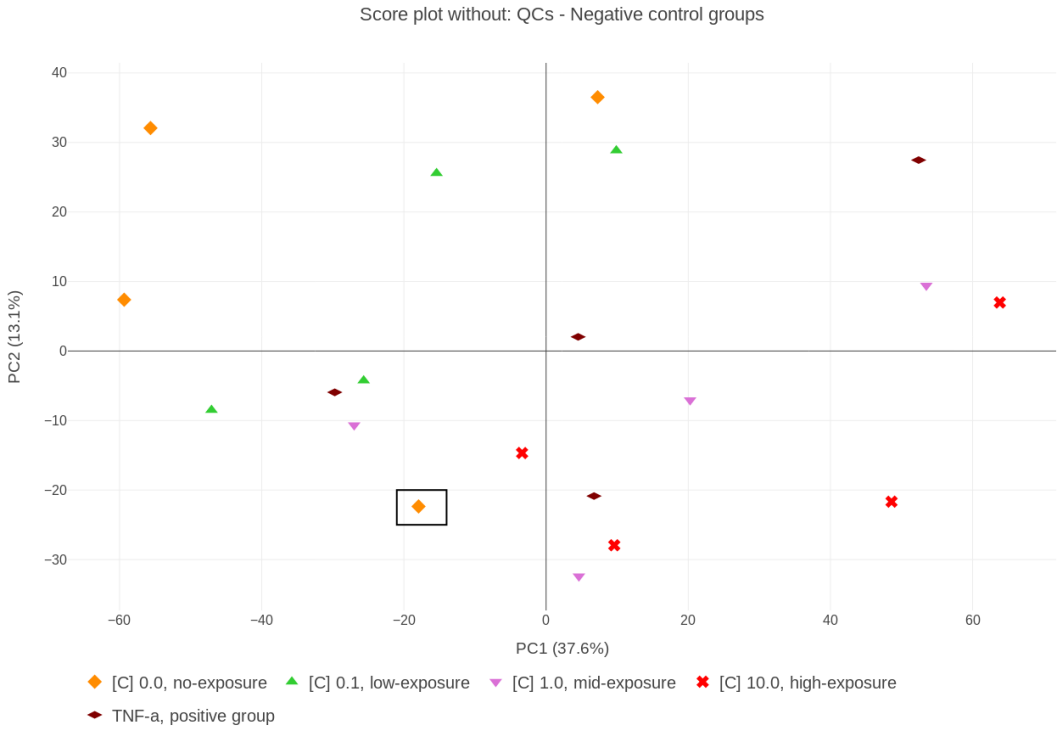
From now on, the analysis was done using a Python library developed during this project to speed up data analysis workflow. Hence, following steps also show how to use `chemometrics library`.

[26]:
```
from chemometrics.unsupervised import unsupervised
```

[27]:
```
# creating the object
output = unsupervised()

# computing pca, pc variable contains the scores
pca = output.PCA_ready(df = df_filtered)
```

Score plot visualization

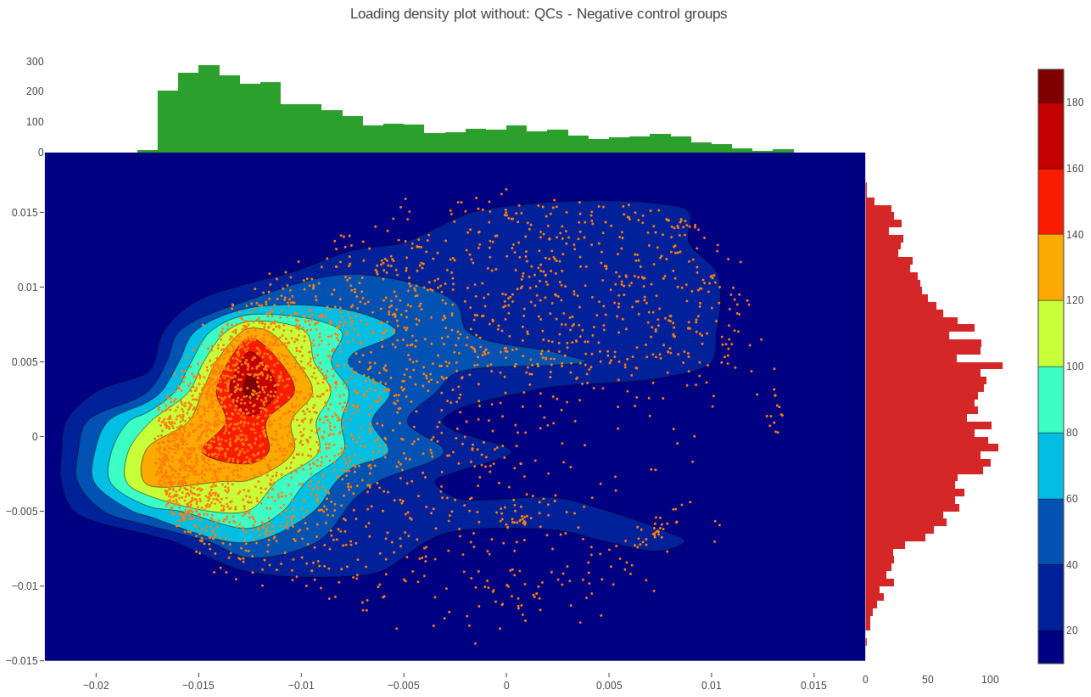
[28]:
```
# now the object containin the results of pca has to be used, here named␣
 ↪'output'
score_plot = output.score_viz(pcx= 0, pcy= 1, color=color, symbol=symbol,␣
 ↪label= None, label_position = None, symbol_sequence=symbol_seq,
                               color_discrete_sequence=color_seq,␣
 ↪hover_name=df_filtered.index)
score_plot.update_layout(width= 900, height= 900,
                         title="Score plot without: QCs - Negative control␣
 ↪groups", font=(dict(size=16)),
                         overwrite=True, legend_title=None,␣
 ↪legend=dict(yanchor="top", y=-.1, xanchor="left", x=0.01, orientation="h",␣
 ↪font_size=20))
score_plot.update_traces(marker=dict(size=13))
score_plot.add_shape(type="rect", x0=-21, x1=-14, y0=-25, y1=-20)
score_plot.show()
```



When the negative control group, which was driving the PC1, was removed from the model the complexity of the data set was reflected on the score plot. It was observed how the effect of digoxin on the metabolism of astrocyte cells was scattered over the two first PCs, without a clear correlation with any PC. However, the trending of the concentration was still clear, on the top left was located the no-exposure group; whereas the group exposed to the highest dose was located on the bottom right. TNF was encountered in the middle of the trending, showing that it had an impact on the metabolism similar to low-mid concentrations of digoxin; making it a suitable positive control for the experiment. Furthermore, it was observed that a sample from the no-exposure group - highlighted within a rectangle - behave as an outlier, since it was located far from the rest of the samples of its own group.

Loading plot visualization

[29]:
```
loading_plot = output.loading_contourplot(pcx=0, pcy=1, histogram=True)
loading_plot.update_layout(width= 900, height= 900,overwrite=True,
                           title="Loading density plot without: QCs - Negative␣
 ↪control groups", font=(dict(size=12)))
loading_plot
```



The loading plot kept showing an unusual behavior of the variables, and the decision to remove the positive control group as well as the outlier was taken. Hence, the PCA model would only include samples of good quality which belonged strictly to the digoxin effect.

3.5 Data set excluding: *QCs/dQCs, negative/positive control groups & outlier*

[30]:
```
df_filtered = df_filtered.filter(axis=0, regex="(?m)^(?!^T)")
```

[31]:
```
df_filtered=df_filtered.iloc[1:, :]
```

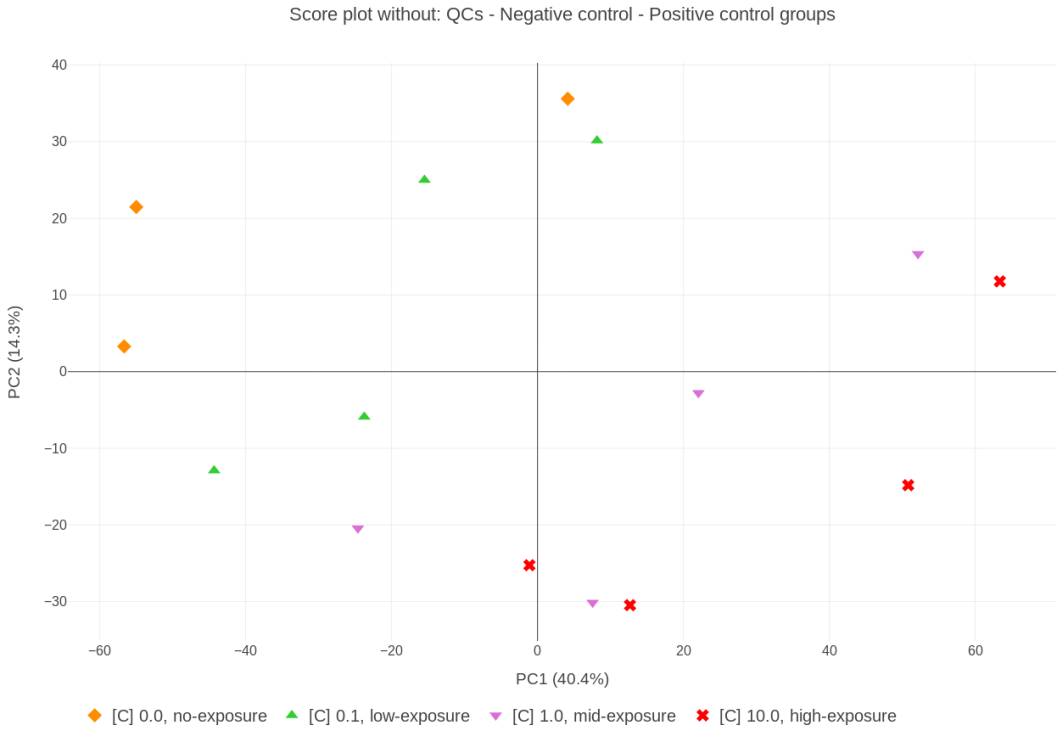
[32]:
```
color = ["[C] 0.0","[C] 0.0","[C] 0.0", "[C] 0.1", "[C] 0.1",
                        "[C] 0.1","[C] 0.1", "[C] 1.0", "[C] 1.0","[C] 1.
 ↪0","[C] 1.0","[C] 10.0", "[C] 10.0",
                        "[C] 10.0","[C] 10.0"]

color_seq=["darkorange", "limegreen", "orchid","red"]

symbol = ["no-exposure","no-exposure",
        "no-exposure",␣
 ↪"low-exposure","low-exposure","low-exposure","low-exposure",
        "mid-exposure","mid-exposure","mid-exposure","mid-exposure",␣
 ↪"high-exposure","high-exposure","high-exposure",
        "high-exposure"]
```

```
symbol_seq=["diamond", "triangle-up","triangle-down", "x","diamond-wide"]
```

[33]:
```
output_2 = unsupervised()
pca_2 = output_2.PCA_ready(df_filtered)
```

Score plot visualization

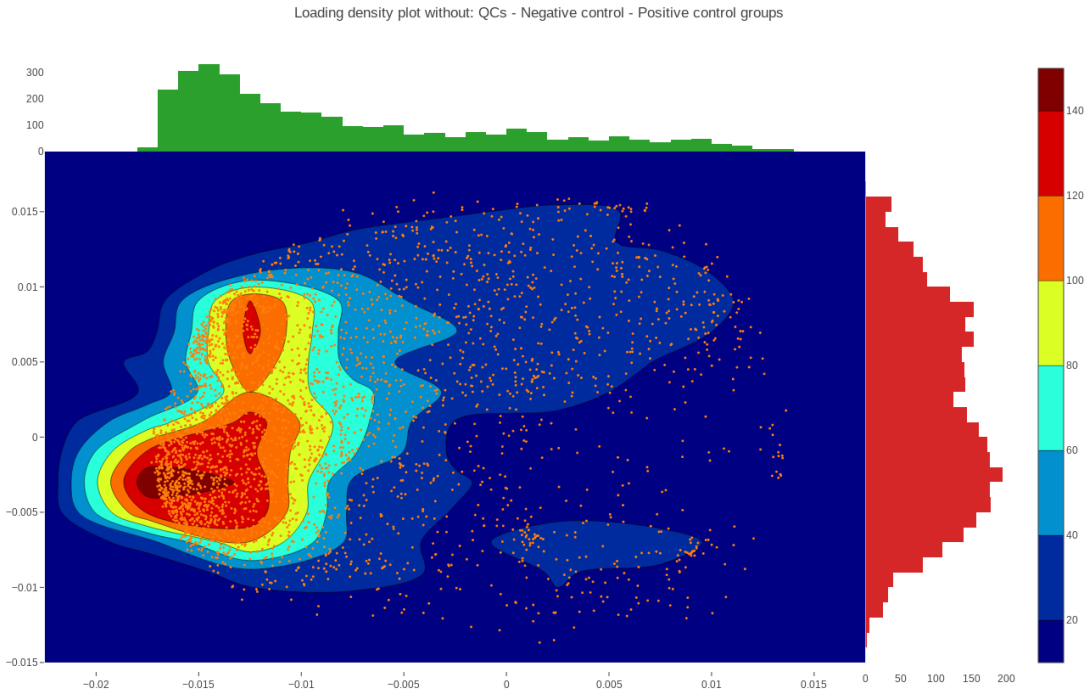
[34]:
```
scores_plot = output_2.score_viz(pcx= 0, pcy= 1, color=color, symbol=symbol,␣
 ↪label= None, label_position = None, symbol_sequence=symbol_seq,
                                  color_discrete_sequence=color_seq,␣
 ↪hover_name=df_filtered.index)
scores_plot.update_layout(width= 900, height= 900, legend=dict(yanchor="top",␣
 ↪y=-.1, xanchor="left", x=0.01, orientation="h", font_size=20),
                          title="Score plot without: QCs - Negative control -␣
 ↪Positive control groups", font=(dict(size=16)),
                          overwrite=True, legend_title=None)
scores_plot.update_traces(marker=dict(size=13))
scores_plot.show()
```



The last PCA model of the EDA revealed that positive control group as well as the outlier sample were masking an unusual behavior of the loadings.

Loading plot visualization

[35]:
```
loading_plot= output_2.loading_contourplot(pcx=0, pcy=1, histogram=True)
loading_plot.update_layout(title="Loading density plot without: QCs - Negative␣
 ↪control - Positive control groups", font=(dict(size=12)),
                           overwrite=True, width= 900, height= 900)
loading_plot
```



On the loading plot, it was observed two hotspots of loadings. The top hotspot was considered as related to digoxin exposure, whereas the bottom hotspot seemed to have an orthogonal effect to digoxin exposure. It was hypothesized that such effect could be induced by a wide variety of possibilities, among them, it could be an uneven cell growth or mistakes on the sample preparation, etc.

Thus, it was concluded that the data was not normalized accordingly, to correct for such undesired variability. The method applied to normalize the data set was probabilistic quotient normalization

4 Probabilistic Quotient Normalization (PQN)

Normalization is a preprocessing method which accounts for different dilutions of samples by scaling the spectra to the same virtual overall concentration. Probabilistic Quotient Normalization (PQN) was introduced as a robust normalization method more suitable for metabolomics studies compared to previous methods. PQN was based on the fact that the vast majority of analytes would not vary among samples. Then PQN computes the most probable dilution factor by looking at the distribution of the quotients of a test sample by those of a reference profile.

4.1 Implementation

[36]:
```
def PQN(path: str, sep=","):
    """
    PQN is a wrapper function for median-based Probabilistic Quotient␣
↪Normalization.

    As input, file named as "xxx1_xxx2.csv" which contains tranposed data,␣
↪where the first column is the sample names.
    Quality controls and diluted quality controls have to be named as "*QC*"␣
↪and "dQC*" respectively.

    The output is the normalised data named as "PQN_xxx1".

    Parameters
    ----------
    path: path of the file to be normalised.
    sep: character separator

    References
    ----------
    Frank Dieterle, Alfred Ross, Götz Schlotterbeck, Hans Senn. Probabilistic␣
↪Quotient Normalization as Robust Method to Account
    for Dilution of Complex Biological Mixtures. Application in 1H NMR␣
↪Metabonomics. Anal. Chem. 2006, 78, 4281-4290.

        """
    df = pd.read_csv(path, sep=sep, index_col=0)
    #importing the data frame from path, passing the first column as index to␣
↪use filter over it
    #qcs = df.filter(axis=0, regex="^dQC*|^QC")
    qcs = df.filter(axis=0, regex="^QC")
    #filtering QCs and dQCs to concatenate with the normalized df later
    ref_vector = np.asarray(df.filter(axis=0, regex="^QC*").median(axis=0))
    #filtering df to obtain only QCs and computing the reference vector which␣
↪is the mean across all QCs samples
    quotients = df.filter(axis=0, regex="(?m)^(?!QC).*$").apply(lambda x: x/
↪ref_vector, axis=1, result_type="expand")
    #filtering the data to exclude dQCs and QCs and using apply to compute the␣
↪quotients, variables divided by ref_vector
    coef_vector= np.asarray(quotients.median(axis=1))
    #compting the coeficient vector for each sample, the scalar.
    pqn_df = df.filter(axis=0, regex="(?m)^(?!QC).*$").apply(lambda x: x/
↪coef_vector, axis=0, result_type="expand")
    #filtering to exclude QC samples and computing normalization of each sample␣
↪dividing by coef_vector
    export_df = pd.concat([pqn_df, qcs])
```

```
    #concatenating pqn_df with QC samples to obtain a final df to be exported
    file_name=f'{path.split("_")[0]}_pqn.csv'
    #setting file name
    #export_df.to_csv(file_name,header=True, index=True)
    return export_df
```

[37]:
```
df_pqn=PQN("digoxin_to_pca.csv")
```

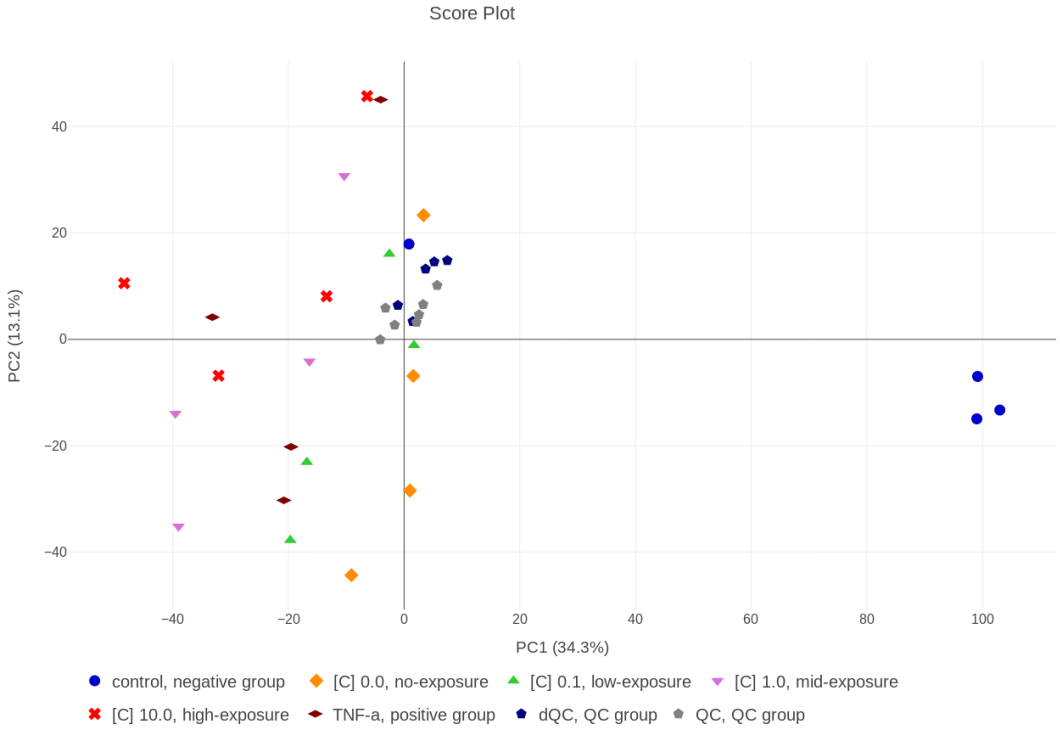
[38]:
```
color = ["control", "control", "control", "control", "[C] 0.0", "[C] 0.0","[C]␣
 ↪0.0","[C] 0.0", "[C] 0.1", "[C] 0.1",
                        "[C] 0.1","[C] 0.1", "[C] 1.0", "[C] 1.0","[C] 1.
 ↪0","[C] 1.0","[C] 10.0", "[C] 10.0",
                        "[C] 10.0","[C] 10.0", "TNF-a",␣
 ↪"TNF-a","TNF-a","TNF-a", "dQC", "dQC", "dQC", "dQC", "dQC","QC", "QC", "QC",␣
 ↪"QC",
                        "QC", "QC", "QC"]

color_seq=["mediumblue","darkorange", "limegreen", "orchid","red", "maroon",␣
 ↪"navy", "grey"]

symbol = ["negative group", "negative group", "negative group", "negative␣
 ↪group", "no-exposure","no-exposure","no-exposure",
       "no-exposure",␣
 ↪"low-exposure","low-exposure","low-exposure","low-exposure",
       "mid-exposure","mid-exposure","mid-exposure","mid-exposure",␣
 ↪"high-exposure","high-exposure","high-exposure",
       "high-exposure", "positive group", "positive group","positive␣
 ↪group","positive group", "QC group", "QC group", "QC group", "QC group", "QC␣
 ↪group",
       "QC group", "QC group", "QC group", "QC group", "QC group",
       "QC group", "QC group"]

symbol_seq=["circle","diamond", "triangle-up","triangle-down",␣
 ↪"x","diamond-wide", "pentagon"]
```

[39]:
```
output = unsupervised()
pca = output.PCA_ready(df=df_pqn)
```

Score plot visualization

[40]:
```
score_plot= output.score_viz(pcx=0, pcy=1, color=color, label=None,␣
 ↪color_discrete_sequence=color_seq,
                              label_position=None, symbol=symbol,␣
 ↪symbol_sequence=symbol_seq)
```

```
score_plot.update_layout(width= 900, height= 900, font=(dict(size=16)),␣
 ↪legend=dict(yanchor="top", y=-.1, xanchor="left", x=0.01, orientation="h",␣
 ↪font_size=20),
                          overwrite=True)
score_plot.update_traces(marker=dict(size=13))
score_plot.show()
```



The suitable manner to check whether PQN was performed correctly is by having a look at the dQCs. Since dQCs are diluted samples of QCs, when applying PQN dQCs and QCs should cluster together.

This effect was observed when normalizing the data and a secon EDA was carried out. However, for simplycity and readability, in this notebook it was only included the last model of EDA, containing solely digoxin exposure-related samples. By this means an easier discussion was achieved, focusing on how PQN affected the data and impaired data interpretation.

4.2 Before vs After PQN

Score plot before vs after PQN

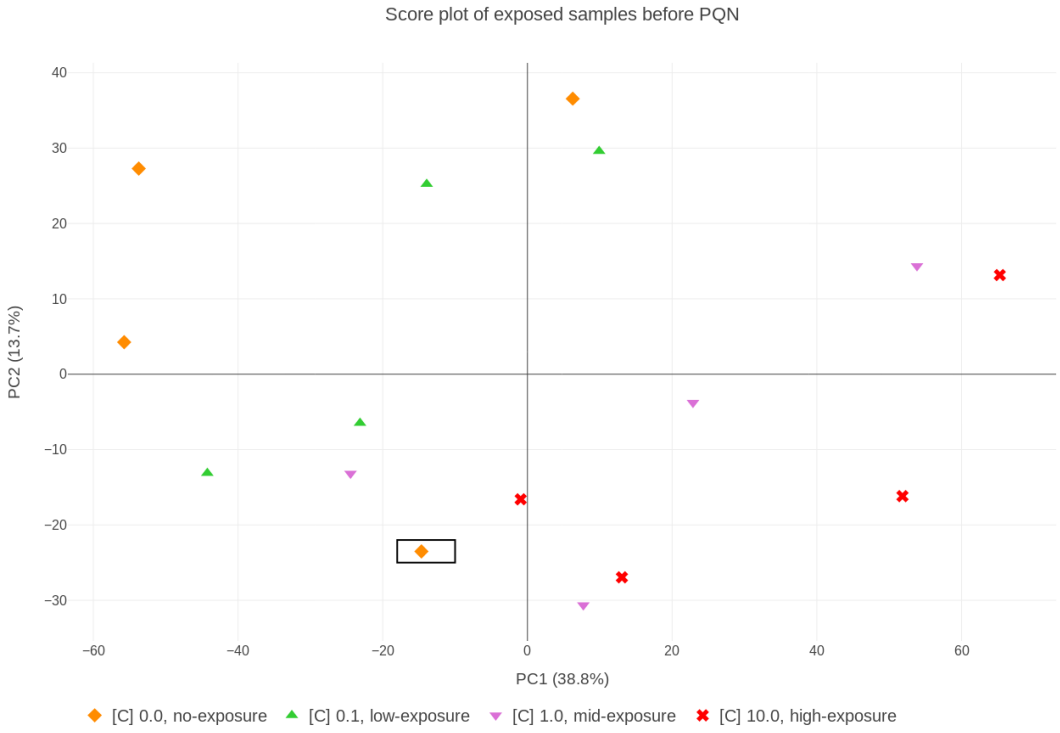
[41]:
```
df_before_pqn=df3.filter(axis=0, regex="(?m)^(?!(^dQC|QC|^Control*|^T))")
```

[42]:
```
```

```
color = ["[C] 0.0","[C] 0.0","[C] 0.0","[C] 0.0", "[C] 0.1", "[C] 0.1",
                        "[C] 0.1","[C] 0.1", "[C] 1.0", "[C] 1.0","[C] 1.
 ↪0","[C] 1.0","[C] 10.0", "[C] 10.0",
                        "[C] 10.0","[C] 10.0"]

color_seq=["darkorange", "limegreen", "orchid","red"]

symbol = ["no-exposure","no-exposure","no-exposure",
        "no-exposure",␣
 ↪"low-exposure","low-exposure","low-exposure","low-exposure",
        "mid-exposure","mid-exposure","mid-exposure","mid-exposure",␣
 ↪"high-exposure","high-exposure","high-exposure",
        "high-exposure"]

symbol_seq=["diamond", "triangle-up","triangle-down", "x","diamond-wide"]
```

[43]:
```
output_before = unsupervised()
pca= output_before.PCA_ready(df=df_before_pqn)
```

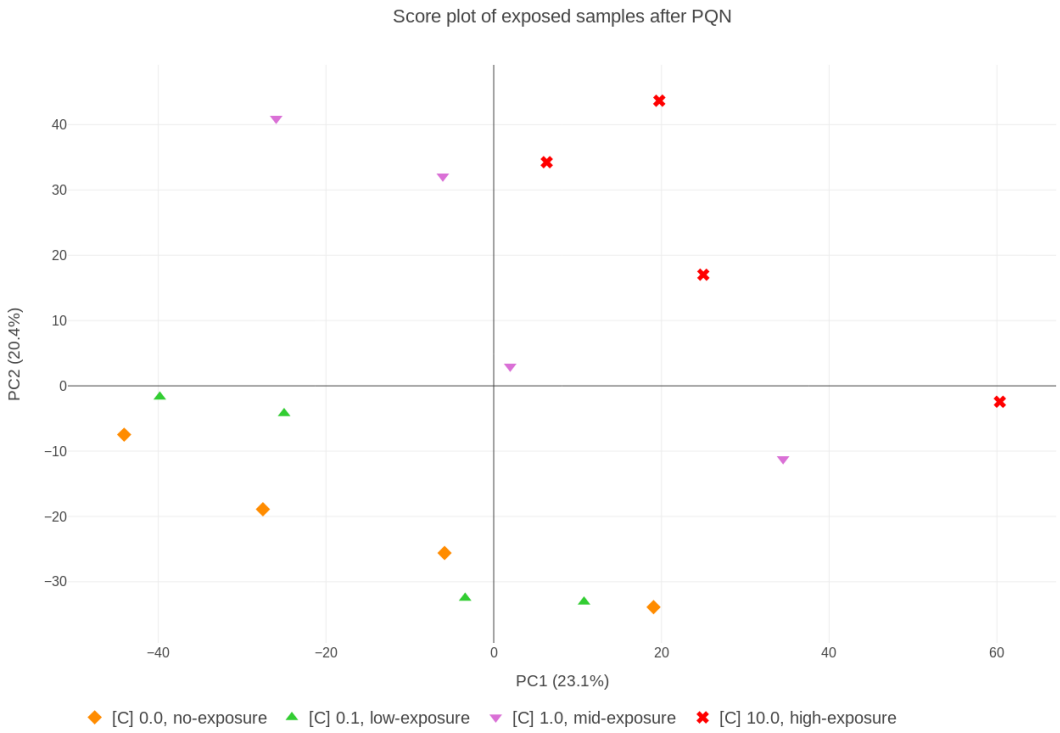
[44]:
```
score_plot = output_before.score_viz(pcx= 0, pcy= 1, color=color,␣
 ↪symbol=symbol, label= None, label_position = None,␣
 ↪symbol_sequence=symbol_seq,
                                color_discrete_sequence=color_seq,␣
 ↪hover_name=df_before_pqn.index)
score_plot.update_layout(width= 900, height= 900, legend=dict(yanchor="top",␣
 ↪y=-.1, xanchor="left", x=0.01, orientation="h", font_size=20),
                         title="Score plot of exposed samples before PQN",␣
 ↪font=(dict(size=16)),
                         overwrite=True, legend_title=None)
score_plot.update_traces(marker=dict(size=13))
score_plot.add_shape(type="rect", x0=-18, x1=-10, y0=-25, y1=-22)
score_plot.show()
```



[45]:
```
df_pqn=df_pqn.filter(axis=0, regex="(?m)^(?!(^dQC|QC|^Control*|^T))")
```

[46]:
```
output_pqn = unsupervised()
pca= output_pqn.PCA_ready(df=df_pqn)
```

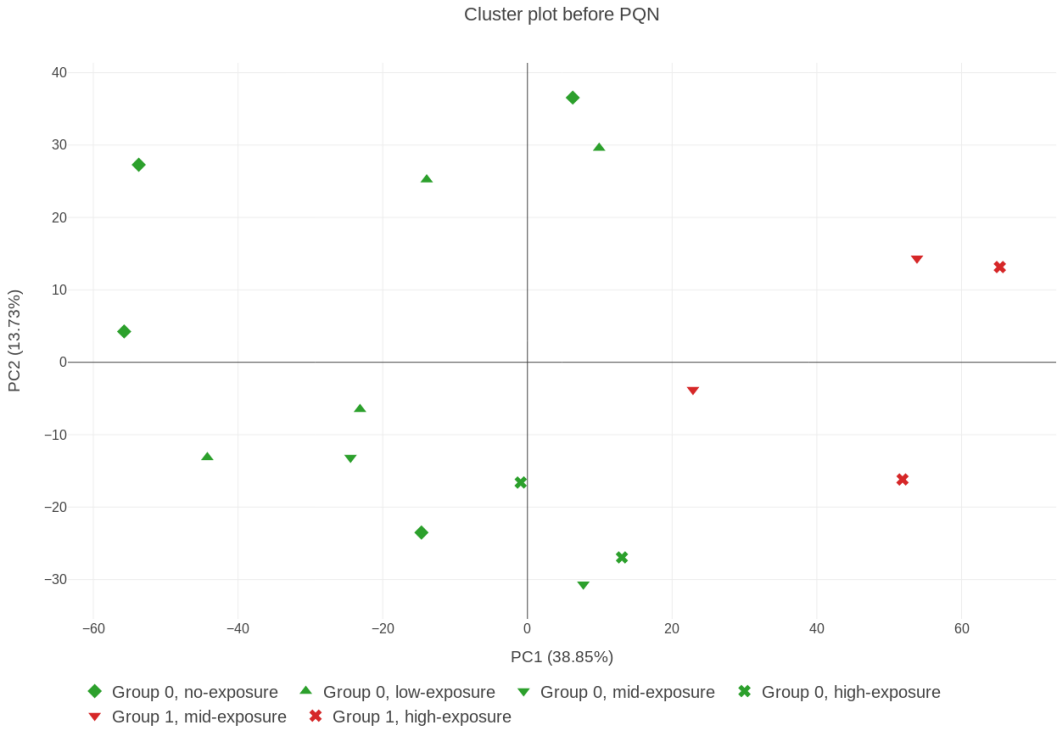
[47]:
```
score_plot= output_pqn.score_viz(pcx=0, pcy=1, color=color, label=None,␣
 ↪label_position="top center", symbol=symbol,
                                symbol_sequence=symbol_seq,␣
 ↪color_discrete_sequence=color_seq, hover_name=df_pqn.index)
score_plot.update_layout(legend_title=None, width= 900, height= 900,
                         title="Score plot of exposed samples after PQN",␣
 ↪font=(dict(size=16)),
                         overwrite=True, legend=dict(yanchor="top", y=-.1,␣
 ↪xanchor="left", x=0.01, orientation="h", font_size=20))
score_plot.update_traces(marker=dict(size=13))
score_plot.show()
```



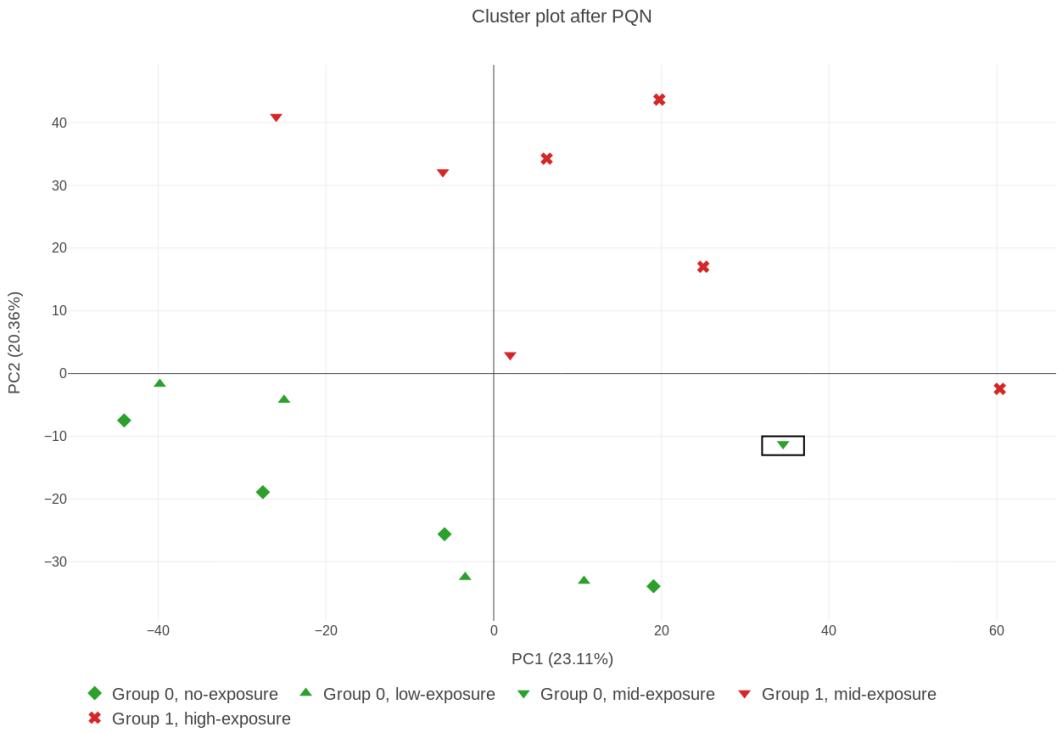
On one hand, on the score plot after PQN, it was detected no outliers; on the contrary, before applying PQN a sample belonging to no-exposure group was considered as an outlier, highlighted within a rectangle..

Cluster plot before vs after PQN

[48]:
```
cluster_plot = output_before.cluster_viz(pcx=0, pcy=1, label=None,␣
↪symbol=symbol, symbol_sequence=symbol_seq)
cluster_plot.update_layout(width= 900, height= 900, title="Cluster plot before␣
↪PQN",legend_title=None,
                           legend=dict(yanchor="top", y=-.1, xanchor="left",␣
↪x=0.01, orientation="h",font_size=20), font=dict(size=16))
cluster_plot.update_traces(marker=dict(size=13))
cluster_plot.show()
```



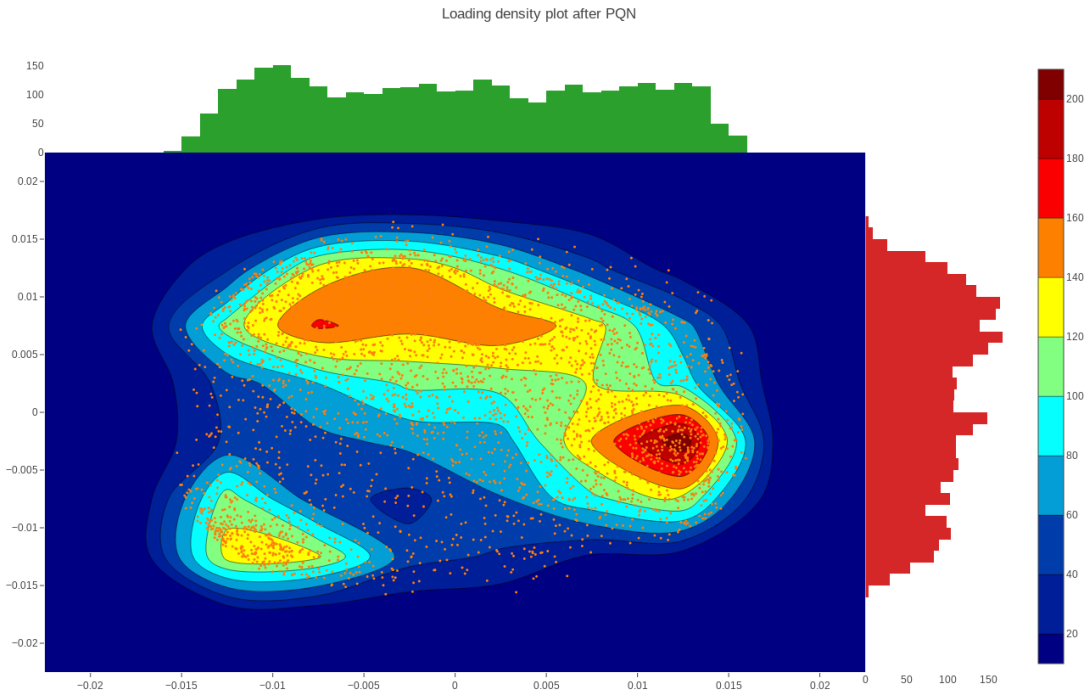
[49]:
```
cluster_plot = output_pqn.cluster_viz(pcx=0, pcy=1, label=None, symbol=symbol,␣
↪symbol_sequence=symbol_seq)
cluster_plot.update_layout(width= 900, height= 900,title="Cluster plot after␣
↪PQN", legend_title=None,
                           legend=dict(yanchor="top", y=-.1, xanchor="left",␣
↪x=0.01, orientation="h", font_size=20), font=dict(size=16))
cluster_plot.add_shape(type="rect", x0=32, x1=37, y0=-13, y1=-10)
cluster_plot.update_traces(marker=dict(size=13))
cluster_plot.show()
```



On the other hand, on the cluster plots, it was observed that before applying PQN, the HCA clustered together samples from mid/high-exposure groups with samples from the lower doses of digoxin. After applying PQN, however, HCA clustered together no/low-exposure groups separately from mid/high-exposure groups; except for a sample belonging to mid-exposure groups.

Loading plot after PQN

[50]:
```
loading_plot = output_pqn.loading_contourplot(pcx=0, pcy=1, histogram=True)
loading_plot.update_layout(width= 900, height= 900,title="Loading density plot␣
 ↪after PQN", overwrite=True, font=dict(size=12))
loading_plot
```



Through the loading plot, it was observed that there was a gradient, as now the highest concentration of loadings was related to groups with the highest concentration of digoxin. It was hypothesized that digoxin induced up-modulation of certain metabolites.

5 Supervised Analysis

The goal of supervised modelling was to obtain a robust model capable of retaining the highest amount of variability as well as presenting high prediction power. The importance of obtaining a robust model lies in the fact that it was the base to extract a subset of variables which would conform a metabolic fingerprint directly related to digoxin exposure. Since the main interest was to perform variable selection, PLS as well as RF models were chosen because one can use Variable Important in Projection (VIP) and Variable Importance, respectively, to carry out that task.

Within supervised modelling process, both frameworks can be applied, either classification or regression. A gold-standard model in metabolomics is PLS-DA, a classification model; however in this section it is discussed the limitations observed when used the classification framework on our metabolomic data, which presented highly unbalance structure with $p >> n$.

For a discussion comparing how supervised models performed before vs after PQN, please refer to the second supplementary notebook.

5.1 Classification Framework

Even though, neither PLS-DA nor RF-classifier performed good enough to be retained as final models, here it was addressed one of the hardest task when using a RF-classifier model, its interpretability.

For the classification framework, the response y variable was defined as discrete, referring the concentration groups, now classes, which were no-exposure, low-exposure, mid-exposure and high-exposure. The response variable was encoded to obtain a unitary dummy matrix.

[51]:
```
np.random.seed(42)
```

[52]:
```
df = pd.read_csv("digoxin_pqn.csv", index_col=0)
df = df.filter(axis=0, regex="(?m)^(?!(^dQC|QC|^Control*|^T).)")
```

[53]:
```
exposure = ["exposed_0.0", "exposed_0.0","exposed_0.0","exposed_0.0",
            "exposed_0.1", "exposed_0.1","exposed_0.1","exposed_0.1",␣
 ↪"exposed_1.0", "exposed_1.0","exposed_1.0","exposed_1.0",
            "exposed_10.0", "exposed_10.0","exposed_10.0","exposed_10.0"]
```

[54]:
```
color = ["[C] 0.0","[C] 0.0","[C] 0.0","[C] 0.0", "[C] 0.1", "[C] 0.1",
                        "[C] 0.1","[C] 0.1", "[C] 1.0", "[C] 1.0","[C] 1.
 ↪0","[C] 1.0","[C] 10.0", "[C] 10.0",
                        "[C] 10.0","[C] 10.0"]

color_seq=["darkorange", "limegreen", "orchid","red"]

symbol = ["no-exposure","no-exposure","no-exposure",
        "no-exposure",␣
 ↪"low-exposure","low-exposure","low-exposure","low-exposure",
        "mid-exposure","mid-exposure","mid-exposure","mid-exposure",␣
 ↪"high-exposure","high-exposure","high-exposure",
        "high-exposure"]

symbol_seq=["diamond", "triangle-up","triangle-down", "x","diamond-wide"]
```

[55]:
```
class_group=["no-exposure","low-exposure","mid-exposure","high-exposure"]
```

[56]:
```
from sklearn.preprocessing import LabelBinarizer
from chemometrics.supervised import Supervised
```

[57]:
```
encoder = LabelBinarizer()
y_classes = encoder.fit_transform(exposure)
y_classes
```

[57]:
```
array([[1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 0],
```

```
[1, 0, 0, 0],
[0, 1, 0, 0],
[0, 1, 0, 0],
[0, 1, 0, 0],
[0, 1, 0, 0],
[0, 0, 1, 0],
[0, 0, 1, 0],
[0, 0, 1, 0],
[0, 0, 1, 0],
[0, 0, 0, 1],
[0, 0, 0, 1],
[0, 0, 0, 1],
[0, 0, 0, 1]])
```

5.1.1 PLS-DA

[58]:
```
output = Supervised()
```

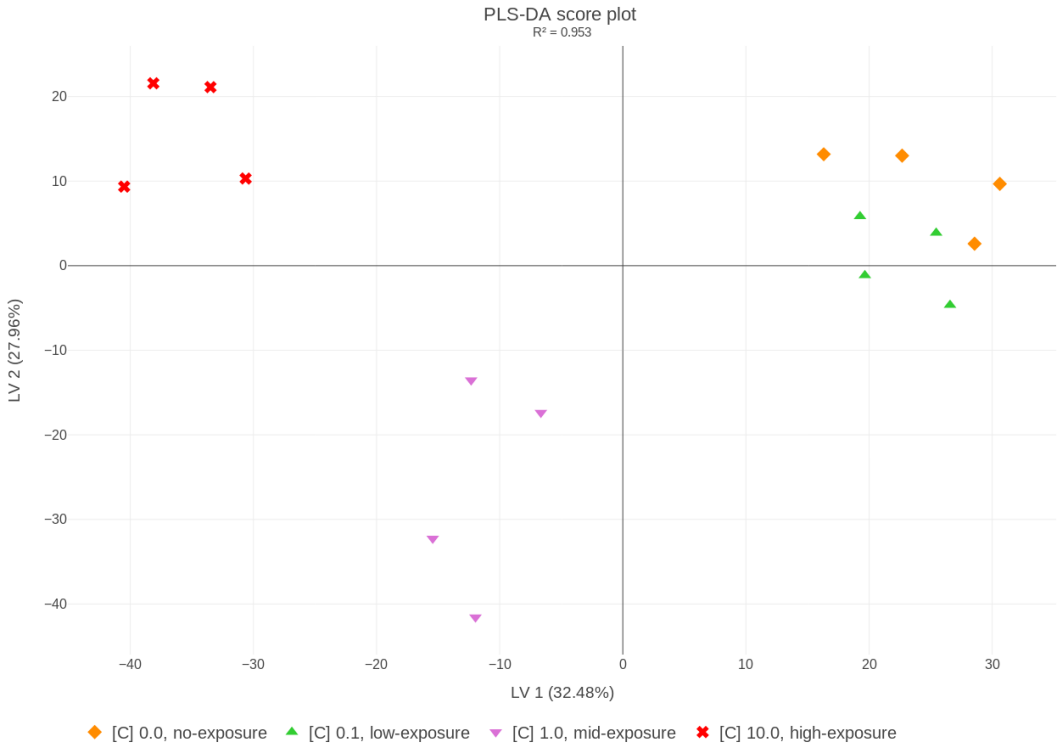
[59]:
```
pls_da = output.PLS_model(df, y_classes, n_components=4)
```

[60]:
```
print(f'Coefficient of determinarion R² for Y = {output.pls_coef_determ:.3f}')
```

```
Coefficient of determinarion R² for Y = 0.953
```

Score plot visualization

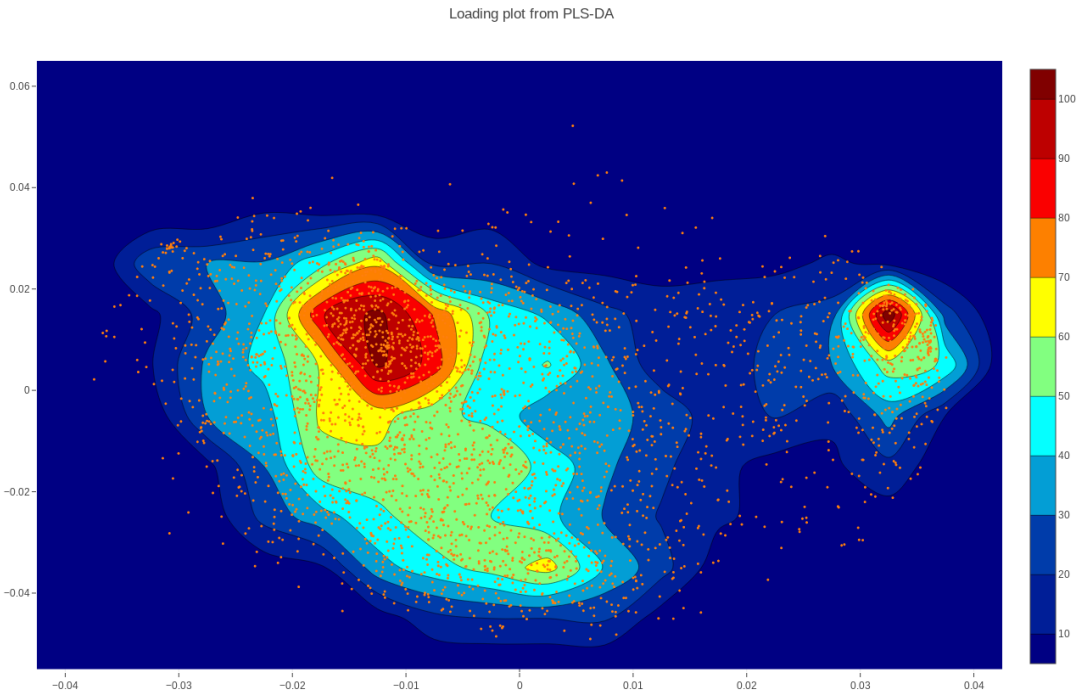
[61]:
```
pls_da_scoreplot = output.Score_viz(lv_1=0, lv_2=1,name_list=None,␣
↪symbol=symbol, symbol_sequence=symbol_seq, color=color,␣
↪color_discrete_sequence=color_seq)
pls_da_scoreplot.update_layout(title="PLS-DA score plot <br><sup>R² = 0.953</
↪sup>",
                               legend=dict(yanchor="top", y=-.1,␣
↪xanchor="left", x=0.01, orientation="h",font_size=20),
                          width= 900, height= 900, font=(dict(size=16)),␣
↪legend_title=None, overwrite=True)
pls_da_scoreplot.update_traces(marker=dict(size=13))
pls_da_scoreplot.show()
```



The PLS-DA model reported a $R^2 = 0.953$, and it managed to discriminate between the groups of high-exposure and no/low-exposure in the first Latent Variable (LV), localizing the mid-exposure group somewhere in the middle between the extreme groups

Loading plot visualization

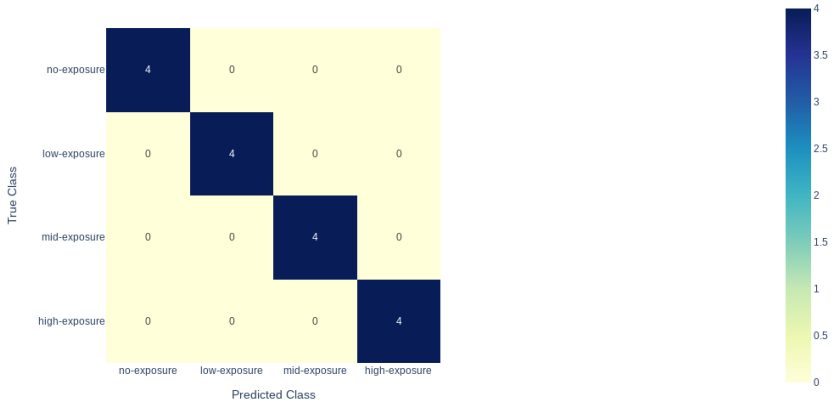
[62]:
```
pls_da_loadingplot = go.Figure()
pls_da_loadingplot.add_trace(go.Histogram2dContour(x=pls_da.x_loadings_[:,0],␣
 ↪y=pls_da.x_loadings_[:,1], colorscale="Jet", xaxis="x", yaxis="y"))
pls_da_loadingplot.add_trace(go.Scatter(x=pls_da.x_loadings_[:,0], y=pls_da.
 ↪x_loadings_[:,1], mode="markers", marker=dict(size=3), xaxis="x", yaxis="y"))
pls_da_loadingplot.update_layout(template="presentation", width= 900, height=␣
 ↪900, hovermode=False, font=dict(size=12), title= "Loading plot from PLS-DA")
pls_da_loadingplot.show()
```



Classification performance metrics

[63]:
```
from chemometrics.utils import ConfusionMat_viz, ROC_AUC_viz
```

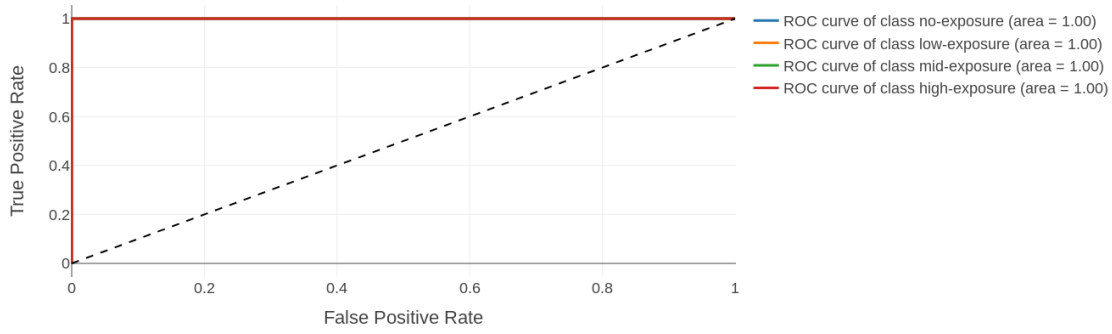
[64]:
```
ConfusionMat_viz(y=y_classes, y_hat=output.y_pred, classes_names=class_group)
```



[65]:
```
output.Classification_report(y_variable=y_classes, y_hat=output.y_pred,␣
↪target_names=class_group)
```

	precision	recall	f1-score	support
no-exposure	1.00	1.00	1.00	4
low-exposure	1.00	1.00	1.00	4
mid-exposure	1.00	1.00	1.00	4
high-exposure	1.00	1.00	1.00	4
accuracy			1.00	16
macro avg	1.00	1.00	1.00	16
weighted avg	1.00	1.00	1.00	16

[66]:
```
ROC_AUC_viz(n_classes=class_group, y_classes=y_classes, y_hat=output.y_pred)
```



It was observe a perfect classification from PLS-DA, where no single sample was misclassified.

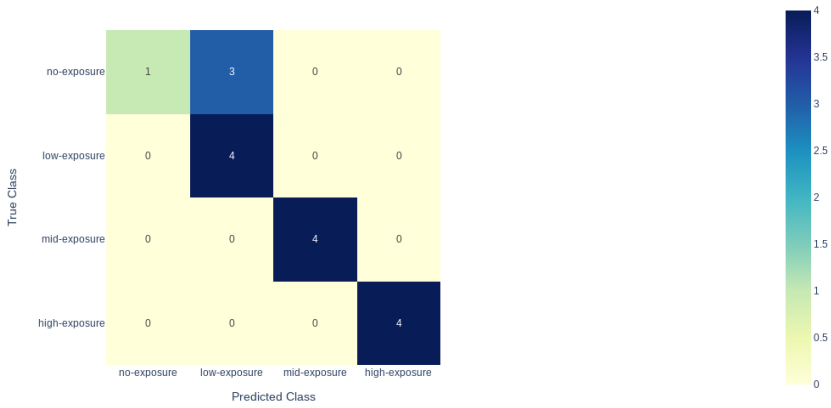
Cross-Validation

[67]:
```
from sklearn.model_selection import cross_val_predict, LeaveOneOut
```

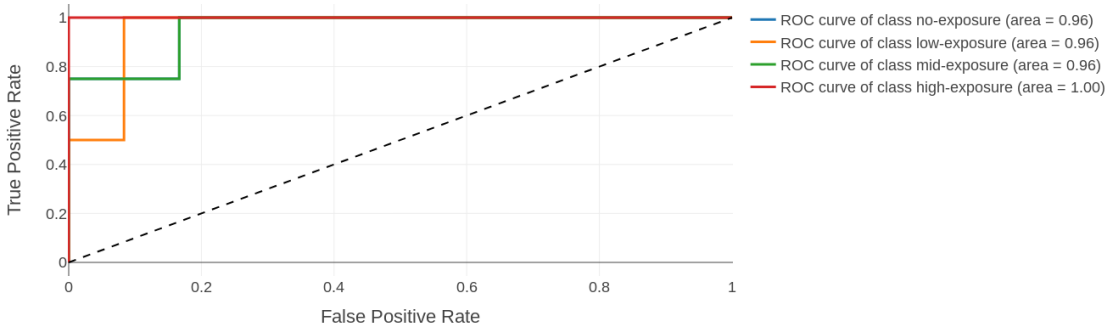
[68]:
```
cv = LeaveOneOut()
```

[69]:
```
pls_da_cv_pred = cross_val_predict(estimator=pls_da, X=df, y=y_classes, cv=cv)
```

[70]:
```
ConfusionMat_viz(y= y_classes, y_hat=pls_da_cv_pred, classes_names=class_group)
```



[71]:
```
ROC_AUC_viz(n_classes=class_group, y_classes=y_classes, y_hat=pls_da_cv_pred)
```



[72]:
```
output.Classification_report(y_variable=y_classes, y_hat=pls_da_cv_pred,␣
↪target_names=class_group)
```

```
                precision    recall  f1-score   support

   no-exposure       1.00      0.25      0.40         4
  low-exposure       0.57      1.00      0.73         4
  mid-exposure       1.00      1.00      1.00         4
 high-exposure       1.00      1.00      1.00         4

      accuracy                           0.81        16
     macro avg       0.89      0.81      0.78        16
  weighted avg       0.89      0.81      0.78        16
```

Quoting scikit-learn:

"Precision is the ability of the classifier no to label as positive a sample that is negative and recall is the ability of the classifier to find all positive samples.

The F-1 measure can be interpreted as a weighted harmonic mean of the precision and recall. A F-1 measure reaches its best value at 1 and it worst score at 0.

In a binary classification problem, the terms "positive" and "negative" refer to the classifier's prediction, and terms *true* and *false* refer to whether that prediction corresponds to the observation. From a confusion matrix precision, recall and F-1 are derived as follows:

- $precision = \frac{TP}{TP+FP}$
- $recall = \frac{TP}{TP+FN}$
- F-1 $= 2 * \frac{precision * recall}{precision + recall}$

"

In this context, it was observed that PLS-DA failed to correctly predict samples belonging to `no-exposure` class, returning a high number of FN with a $recall = 0.25$.

The FN samples were misclassified as `low-exposure`; for this reason $precision = 0.57$ and the class `low-exposure` has a high number of FP.

Model evaluation and validation The aim of CV was to compare the goodness of fit R2 and goodness of prediction Q2 . R2 , also known as coefficient of determination, represents the fraction of variability which the model is capable of explaining, whereas the Q2 represents the prediction capability of the model. Goodness of fit and goodness of prediction metrics involves Total Sum of Squares (TSS), Residual Sums of Squares (RSS) as well as Predicted Error Sum of Squares (PRESS):

- $TSS = \sum(y_i - \overline{y})^2$
- $RSS = \sum(y_i - \hat{y})^2$
- $PRESS = \sum(y_i - \hat{y_i})^2$

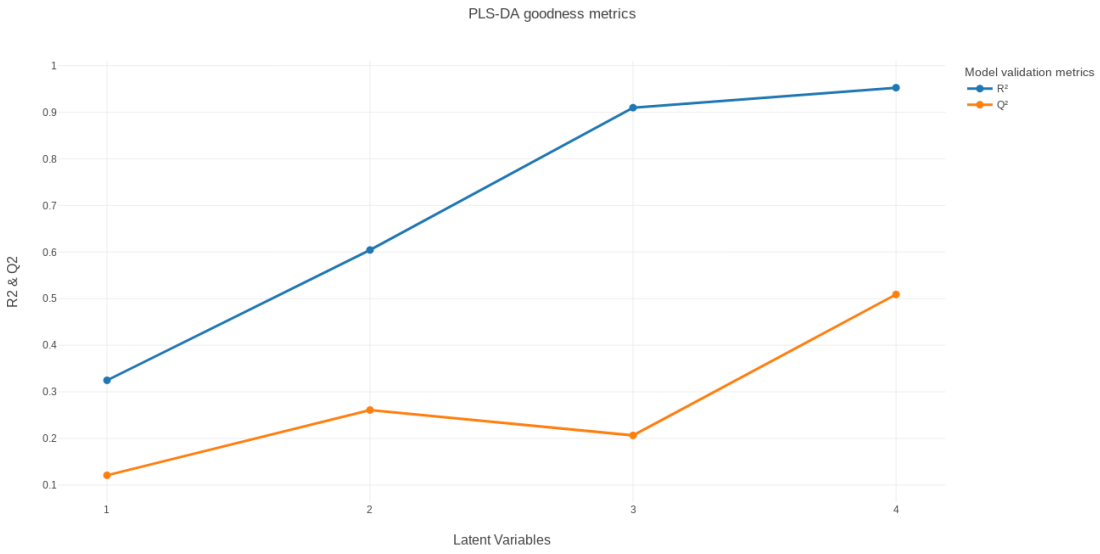
Where the difference between R^2 and Q^2 is that PRESS is computed during CV, when observations are kept out:

- $R^2 = 1 - \frac{RSS}{TSS}$
- $Q^2 = 1 - \frac{PRESS}{TSS}$

[73]:
```
output.PLS_goodness_metrics(X=df, y_variable=y_classes, n_models=4)
```

[74]:
```
goodness_met_viz = output.PLS_goodness_metrics_viz()
goodness_met_viz.update_layout(title="PLS-DA goodness metrics",
                               width= 400, height= 700, font=(dict(size=12)),␣
 ↪overwrite=True)
goodness_met_viz.update_xaxes(type="category")
```

```
goodness_met_viz.show()
```



When the goodness metrics were evaluated, the goodness of prediction behaved extremely unusual. A normal behavior of Q2 is that it starts to notably increase when the model includes 1 and 2 LV and afterwards it starts to reach a plateau or slightly decrease. For this reason, PLS-DA failed model validation and was rejected.

5.1.2 RF Classifier

[75]:
```
output_rf_c = Supervised()
```

[76]:
```
rf_c = output_rf_c.RandomForest_Classifier(X=df, y_variable=y_classes)
```

[77]:
```
print(f'OOB score = {rf_c.oob_score_}')
```

```
OOB score = 0.0625
```

Classification performance metrics

[78]:
```
y_pred_rf_c = rf_c.predict(df)
```

[79]:
```
output_rf_c.Classification_report(y_variable=y_classes, y_hat=y_pred_rf_c,␣
↪target_names=class_group)
```

```
              precision    recall  f1-score   support

 no-exposure       1.00      1.00      1.00         4
low-exposure       1.00      1.00      1.00         4
mid-exposure       1.00      1.00      1.00         4
```

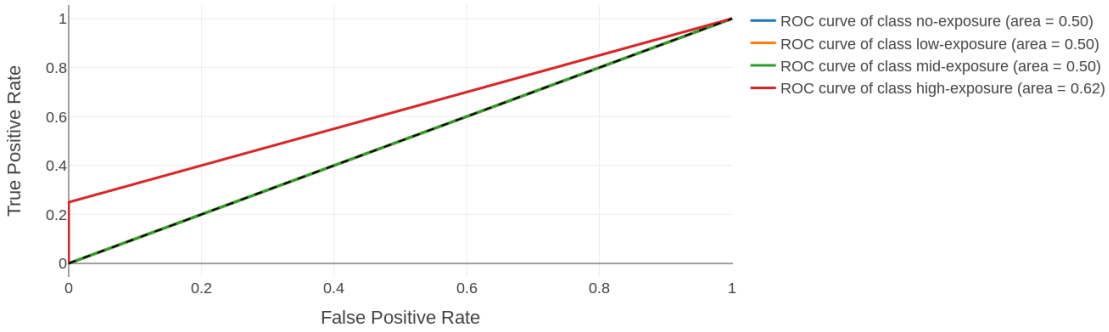
```
high-exposure       1.00      1.00      1.00         4

    accuracy                            1.00        16
   macro avg        1.00      1.00      1.00        16
weighted avg        1.00      1.00      1.00        16
```

[80]:
```
print(f'OOB score = {rf_c.oob_score_}')
```

```
OOB score = 0.0625
```

OOB score is really low, that means that the model had poor prediction power and that must be due to the low number of observations present in the data set. For that reason, even though RF managed to correctly classify each sample, it failed in prediction when CV was applied.

[81]:
```
ROC_AUC_viz(n_classes=class_group, y_classes=y_classes,␣
↪y_hat=cross_val_predict(rf_c, X=df, y=y_classes, cv=cv))
```



ROC curve showed that the RF-classifier would predict half of the time one class for each samples.

[82]:
```
import seaborn as sns
```

[83]:
```
def distanceMatrix(model, X):

    terminals = model.apply(X)
    #[n_samples, n_trees] containing the terminal node for each sample in each␣
↪tree
    nTrees = terminals.shape[1]

    #getting the nodes of the samples in the first tree
    a = terminals[:,0]
    #assest whether values are equal or not and by multiplying by 1 it obtaind␣
↪a binary matrix
```

```
    #initialitzing the first freq count
    proxMat = 1*np.equal.outer(a, a)

    for i in range(1, nTrees):
        a = terminals[:,i]
        proxMat += 1*np.equal.outer(a, a)

    distanceMat = np.absolute(np.round(1-(proxMat / nTrees), decimals=1))

    return distanceMat
```

[84]:
```
rf_c_distMat = distanceMatrix(rf_c, df)
rf_c_distMat=pd.DataFrame(rf_c_distMat)
```

[85]:
```
rf_c_distMat.set_axis(df.index, axis=1, inplace=True)
rf_c_distMat.set_axis(df.index, axis=0, inplace=True)
```

5.2 Regression Framework

For the regression framework, the response variable y was defined as the concentration to which the samples were exposed ranging from 0.0 to 10 µmolar.

The PLS-RA model reported an R2 = 0.999 which outperformed both PLS-DA as well as RF regressor which reported an $R^2 = 0.955$ with an OOB score still showing lack of prediction power, $OOB = 0.570$.

[86]:
```
y_cont = np.array([0.0, 0.0,0.0,0.0,0.1,0.1,0.1,0.1,1.0,1.0,1.0,1.0,10.0,10.
↪0,10.0,10.0])
```

5.2.1 PLS-RA

[87]:
```
output_2 = Supervised()
pls_ra = output_2.PLS_model(X=df, y_variable=y_cont, n_components=4)
```

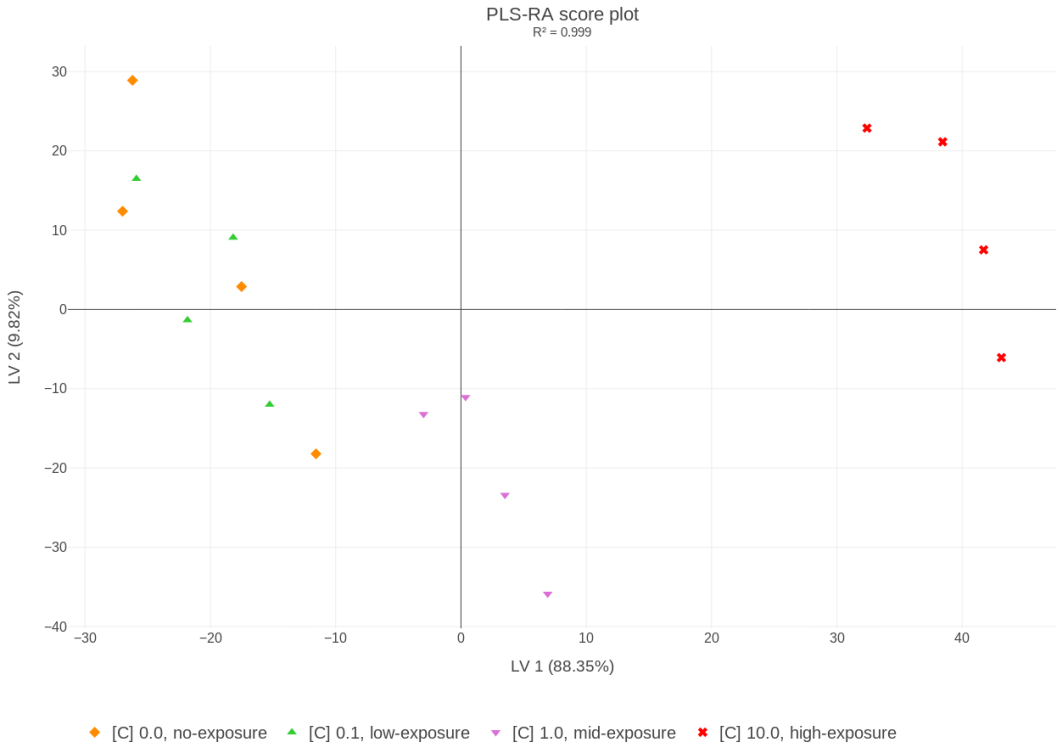
[88]:
```
print(f' Coefficient of determination R² for Y = {output_2.pls_coef_determ:.
↪3f}')
```

```
 Coefficient of determination R² for Y = 0.999
```

Score plot visualization

[89]:
```
score_plot_2 = output_2.Score_viz(lv_1=0, lv_2=1, name_list=None, color=color,␣
↪symbol=symbol, symbol_sequence=symbol_seq, color_discrete_sequence=color_seq)
score_plot_2.update_layout(title="PLS-RA score plot<br><sup>R² = 0.999",␣
↪legend=dict(yanchor="top", y=-.15, xanchor="left", x=0.01, orientation="h",␣
↪font_size=20),
                           width= 900, height= 900, font=(dict(size=16)),␣
↪overwrite=True)
```

```
score_plot_2.update_traces(marker=dict(size=10))
score_plot_2.show()
```



Strikingly, PLS-RA score plot, showed harmony with the classification result from RF-classifier; showing that the group of high exposure was the more dissimilar to the rest. Such situation was retained in the first LV, where on the left-hand side rested no/low-exposure groups and, extremely close to them, the mid-exposure group. On the right-hand side, however, it was located the high-exposure group.

Model evaluation and validation

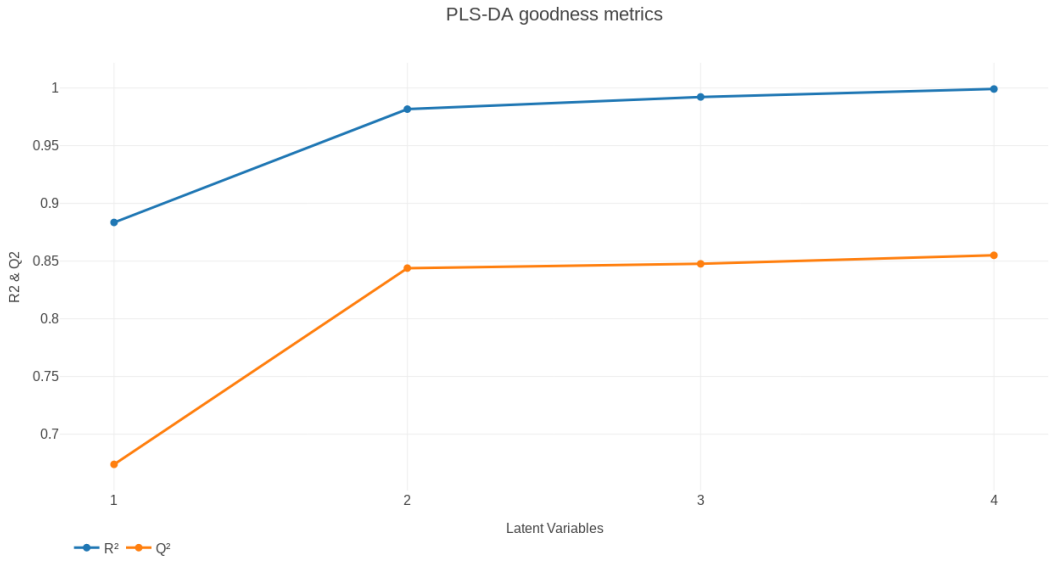
[90]:
```
output_2.PLS_goodness_metrics(X=df, y_variable=y_cont, n_models=4)
```

[91]:
```
output_2.r2_nested
```

[91]:
```
[0.8834663026277807, 0.9816321692364528, 0.9922095874067319, 0.999084079879463]
```

[92]:
```
goodness_met_viz=output_2.PLS_goodness_metrics_viz()
goodness_met_viz.update_layout(title="PLS-DA goodness metrics",
                               legend=dict(orientation="h", yanchor="top", y=-.
↪10, xanchor="left", x=0.01),
                           width= 400, height= 700, font=(dict(size=16)),␣
↪overwrite=True)
```

```
goodness_met_viz.update_xaxes(type="category")
goodness_met_viz.show()
```



PLS-RA goodness metrics were satisfactory. It was observed that a model including only two LV would perform well in terms of explained variability as well as prediction power; thus, accomplishing the parsimony criteria of the simplest model.

As shown, the PLS-RA model successfully passed model validation, for that reason it was retained and further explored to extract a metabolic fingerprint. The next step consisted on variable selection to obtain the metabolic fingerprint. Variable selection was done by computation of Variable Importance in Projection (VIP) of the PLS-RA model.

6 Metabolic Fingerprint Extraction

As aforementioned, PLS model was chosen since it allows variable selection, which means that it allows the extraction of a subset of variables of interest for further study. In the PLS framework, there is a wide variety of methods to perform variable selection, in this project it was use Variable Importance in Projection (VIP) which belongs to the category of filtering methods.

The idea behind VIP is to accumulate the importance of each variable being reflected by loading weights from each component. In the scikit-learn python library the computation of the VIP was not implemented yet, hence VIP were computed by the following the equation from where VIP measure, vj, is defined by the following equation:

$$v_j = \sqrt{p \sum_{a=1}^{A} [(q_a^2 t'_a t_a)(W_{aj}/\|W_a\|^2)] / \sum_{a=1}^{A} (q_a^2 t'_a t_a)}$$

Implemented as follows:

[93]:
```
def VIP(x, model):
    """
    Computes PLS Variable Importance in Projection (VIP)

    Parameter
    ---------
    x: data frames
    model: scikit-learn PLS model

    Return
    ------
    VIP

    References
    ---------
    Tahir Mehmood, Kristian Hovde Liland, Lars Snipen, and Solve Sæbø. A review␣
 ↪of variable
    selection methods in partial least squares regression. Chemometrics and␣
 ↪intelligent laboratory
    systems, 118:62-69, 2012

    Author
    ------
    https://github.com/scikit-learn/scikit-learn/issues/7050
    """
    t = model.x_scores_
    w = model.x_weights_
    q = model.y_loadings_

    m, p = x.shape
    _, h = t.shape

    vips = np.zeros((p,))

    s = np.diag(t.T @ t @ q.T @ q).reshape(h, -1)
    total_s = np.sum(s)

    for i in range(p):
        weight = np.array([ (w[i,j] / np.linalg.norm(w[:,j]))**2 for j in␣
 ↪range(h) ])
        vips[i] = np.sqrt(p*(s.T @ weight)/total_s)

    return vips
```

[94]:
```
vip = VIP(x=df, model=pls_ra)
```

Variable filtering was done based on three different thresholds which were tested to evaluate how

it affected the performance of the PLS-RA model and how many informative variables could be extracted. For a $VIP_{threshold} = X$, variables where retained if their $VIP_{value} \geq VIP_{threshold}$. New PLS-RA models were built and its goodness metrics where studied to assess how the reduction in the number of variables affected the models.

The following function was developed to automatize `VIP subsetting`:

[95]:
```
def VIP_subset(vips, threshold, X):
    """
    Performs df extraction from original df according to the treshold value␣
 ↪passed

    Parameter
    --------
    - vips: data frame or data matrix containing VIP values
    - threshold: threshod value, it can be integer or float
    - X: data frame containing annotated metabolites from where extraction is␣
 ↪carried out based on the VIP threshold and values

    Return
    ------
    Data frame containing the extracted metabolites of interest. The content is␣
 ↪the values of the original data frame

    Author:
    ------
    Christian Peralta

    """
    # vip marix is converted to a pd.DataFrame and transpose, so later the␣
 ↪variable name can be included in the output
    v = pd.DataFrame(vips).T
    #filtering according to a selected threshold
    v = v[v>=threshold].dropna(axis=1)
    # DF-v now contraind the variables of interest according to the VIP␣
 ↪threshold.
    # Variable name is set to object and stored in a variable for iteration
    features_subset = v.columns.astype("object")
    # setting an empty dictionary which will be converted to a DF
    d = {}
    # loop setting as key:value argument as variable_position:column of the DF-X
    for i in features_subset:
        d[i] = X[str(i)]
    d = pd.DataFrame(d)
    return d
```

[96]:
```
var_subset = VIP_subset(vips=vip,threshold=2, X=df)
```

The following functions was developed to automatically perform VIP feature selection which then in pass to PLS for model fitting. The main goal was to automatically compute `goodness metrics` for nested models at desired $VIP_{thresholds}$.

[97]:
```
def comparative_goodness_metrics(threshold=None):
    """
    Computes goodness metrics for specified VIP subsets, allowing the␣
 ↪comparison in terms of nested R² and Q².
    It takes as parameter treshold for VIP_subset function

    Parameter
    ---------
    - threshold: integer value set as VIP threshold

    Return
    ------
    List of dictionaries containing R², Q² values as well as beta-coefficients␣
 ↪for the corresponding VIP_threshold

    Author
    ------
    Christian Peralta
    """

    # first, variable subset extraction based on VIP threshold, using the␣
 ↪aforedeclared function 'VIP_subset'
    # subset DF will be use to fit PLS nested models
    subset = VIP_subset(vips = vip, threshold=threshold, X=df)
    # fitting and computation of goodness metrics
    results = output_2.PLS_goodness_metrics(X=subset, y_variable=y_cont,␣
 ↪n_models=4, vip_metrics=True, treshold=threshold)
    return results
```

[ ]:
```
# using map to iteratively compute metrics of the 3 desired thresholds
tr_1, tr_1_5, tr_2 = list(map(comparative_goodness_metrics, [1,1.5,2]))
```

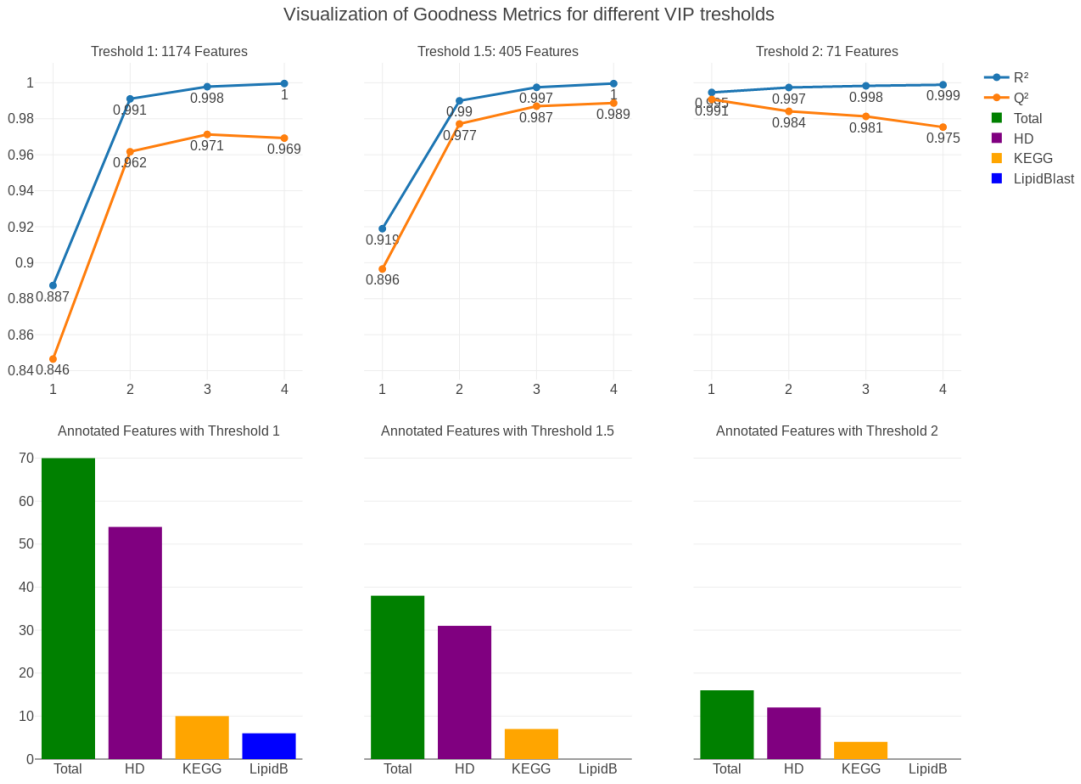
Visualization of VIP filtering

[99]:
```
vip_metrics = make_subplots(rows=2, cols=3, shared_yaxes=True,
                            subplot_titles=("Treshold 1: 1174 Features",␣
 ↪"Treshold 1.5: 405 Features", "Treshold 2: 71 Features",
                                            "Annotated Features with Threshold␣
 ↪1", "Annotated Features with Threshold 1.5","Annotated Features with␣
 ↪Threshold 2"),
                            vertical_spacing=0.09)
```

[100]:
```
n_model = [1,2,3,4]
```

[101]:
```
####################################
#treshold 1
####################################
vip_metrics.add_trace(go.Scatter(x=n_model, y=list(tr_1[0].values()),name="R²",
                                 mode="lines+markers+text",
                                 text=np.round(list(tr_1[0].values()),␣
 ↪decimals=3),
                                 textposition="bottom center",␣
 ↪marker_color="#1f77b4"), row=1, col=1)
vip_metrics.add_trace(go.Scatter(x=n_model, y=list(tr_1[1].values()), name="Q²",
                                 mode="lines+markers+text",
                                 text=np.round(list(tr_1[1].values()),␣
 ↪decimals=3),
                                 textposition="bottom center",␣
 ↪marker_color="#ff7f0e"), row=1, col=1)

vip_metrics.add_trace(go.Bar(x=["Total"], y=[70], name='Total',␣
 ↪marker_color="green"), row=2, col=1)
vip_metrics.add_trace(go.Bar(x=["HD"], y=[54], name='HD',␣
 ↪marker_color="purple"), row=2, col=1)
vip_metrics.add_trace(go.Bar(x=["KEGG"], y=[10], name='KEGG',␣
 ↪marker_color="orange"), row=2, col=1)
vip_metrics.add_trace(go.Bar(x=["LipidB"], y=[6], name='LipidBlast',␣
 ↪marker_color="blue"), row=2, col=1)

####################################
#treshold 1.5
####################################
vip_metrics.add_trace(go.Scatter(x=n_model, y=list(tr_1_5[0].values()),name="R²␣
 ↪for treshold 1.5",
                                 mode="lines+markers+text",
                                 text=np.round(list(tr_1_5[0].values()),␣
 ↪decimals=3),
                                 textposition="bottom center",␣
 ↪marker_color="#1f77b4", showlegend=False), row=1, col=2)
vip_metrics.add_trace(go.Scatter(x=n_model, y=list(tr_1_5[1].values()),name="Q²␣
 ↪for treshold 1.5",
                                 mode="lines+markers+text",
                                 text=np.round(list(tr_1_5[1].values()),␣
 ↪decimals=3),
                                 textposition="bottom center",␣
 ↪marker_color="#ff7f0e", showlegend=False), row=1, col=2)

vip_metrics.add_trace(go.Bar(x=["Total"], y=[38], name='Total',␣
 ↪marker_color="green", showlegend=False), row=2, col=2)
```

```
vip_metrics.add_trace(go.Bar(x=["HD"], y=[31], name='HD',␣
 ↪marker_color="purple", showlegend=False), row=2, col=2)
vip_metrics.add_trace(go.Bar(x=["KEGG"], y=[7], name='KEGG',␣
 ↪marker_color="orange", showlegend=False), row=2, col=2)
vip_metrics.add_trace(go.Bar(x=["LipidB"], y=[0], name='LipidBlast',␣
 ↪marker_color="blue", showlegend=False), row=2, col=2)

####################################
#treshold 2
####################################
vip_metrics.add_trace(go.Scatter(x=n_model, y=list(tr_2[0].values()),name="R²␣
 ↪for treshold 2",
                                 mode="lines+markers+text",
                                 text=np.round(list(tr_2[0].values()),␣
 ↪decimals=3),
                                 textposition="bottom center",␣
 ↪marker_color="#1f77b4", showlegend=False), row=1, col=3)
vip_metrics.add_trace(go.Scatter(x=n_model, y=list(tr_2[1].values()),name="Q²␣
 ↪for treshold 2",
                                 mode="lines+markers+text",
                                 text=np.round(list(tr_2[1].values()),␣
 ↪decimals=3),
                                 textposition="bottom center",␣
 ↪marker_color="#ff7f0e", showlegend=False), row=1, col=3)

vip_metrics.add_trace(go.Bar(x=["Total"], y=[16], name='Total',␣
 ↪marker_color="green", showlegend=False), row=2, col=3)
vip_metrics.add_trace(go.Bar(x=["HD"], y=[12], name='HD',␣
 ↪marker_color="purple", showlegend=False), row=2, col=3)
vip_metrics.add_trace(go.Bar(x=["KEGG"], y=[4], name='KEGG',␣
 ↪marker_color="orange", showlegend=False), row=2, col=3)
vip_metrics.add_trace(go.Bar(x=["LipidB"], y=[0], name='LipidBlast',␣
 ↪marker_color="blue", showlegend=False), row=2, col=3)


vip_metrics.update_layout(hovermode=False, font=dict(size=16), height=1000,␣
 ↪width=1320,
                          title_text="Visualization of Goodness Metrics for␣
 ↪different VIP tresholds",
                         template="presentation")
```



The graphs at the top represent the goodness metrics for different $VIP_{thresholds}$, including the number of variables which were present in the model. The validation metrics of first model, with $VIP_{thresholds} = 1$, seemed to outperform the base model, with roughly one third of original 3250 variables. Consecutive models, reported validation metrics equally good; however, for $VIP_{thresholds} = 2$ the Q2 metric started to decline when more than one LV was included in the model. Thus, the model obtained with a $VIP_{thresholds} = 2$ failed the model validation step.

At the bottom of the figure, it was represented the total number of annotated analytes which were present in the different models. Moreover, the graphs visualize the source of annotation, HD referring to in-house DB, KEGG referring to Kyoto Encyclopedia of Genes and Genomes DB and LipidB referring to LipidBlast DB.

The number of annotated analytes obtained with $VIP_{thresholds} = 1$ was notably higher, 70, when compared to the number of annotated variables obtained with $VIP_{thresholds} = 1.5$, ~ 40. Thus, the metabolic fingerprint extracted when setting a $VIP_{thresholds} = 1$ seemed to be more extensive, giving more chances of having highly informative digoxin-related biomarkers. For the final metabolic fingerprint only metabolites annotated at level 1, i.e HD annotation, where kept for graph-based analysis.

[ ]:

[ ]:

[ ]: