Assignment 3

Group Members

Abdul Wahab Madni, Hamza Badar, Aleksandr Semenikhin

In [1]:
```
import pandas as pd
import plotly.express as px
import seaborn as sns
import matplotlib.pyplot as plt
```

Ex.1: Parallel Coordinates Plot in Plotly

a)

In [2]:
```
df=pd.read_excel("Data_Cortex_Nuclear.xls")
df
```

Out[2]:

	MouseID	DYRK1A_N	ITSN1_N	BDNF_N	NR1_N	NR2A_N	pAKT_N	pBRAF_N	pCAMKII_N	pCREB_N	...	pCFOS_N	SYP_N	H3AcK18_N	EGR1_
0	309_1	0.503644	0.747193	0.430175	2.816329	5.990152	0.218830	0.177565	2.373744	0.232224	...	0.108336	0.427099	0.114783	0.1317
1	309_2	0.514617	0.689064	0.411770	2.789514	5.685038	0.211636	0.172817	2.292150	0.226972	...	0.104315	0.441581	0.111974	0.1351
2	309_3	0.509183	0.730247	0.418309	2.687201	5.622059	0.209011	0.175722	2.283337	0.230247	...	0.106219	0.435777	0.111883	0.1333
3	309_4	0.442107	0.617076	0.358626	2.466947	4.979503	0.222886	0.176463	2.152301	0.207004	...	0.111262	0.391691	0.130405	0.1474
4	309_5	0.434940	0.617430	0.358802	2.365785	4.718679	0.213106	0.173627	2.134014	0.192158	...	0.110694	0.434154	0.118481	0.1403
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1075	J3295_11	0.254860	0.463591	0.254860	2.092082	2.600035	0.211736	0.171262	2.483740	0.207317	...	0.183324	0.374088	0.318782	0.2046
1076	J3295_12	0.272198	0.474163	0.251638	2.161390	2.801492	0.251274	0.182496	2.512737	0.216339	...	0.175674	0.375259	0.325639	0.2004
1077	J3295_13	0.228700	0.395179	0.234118	1.733184	2.220852	0.220665	0.161435	1.989723	0.185164	...	0.158296	0.422121	0.321306	0.2291
1078	J3295_14	0.221242	0.412894	0.243974	1.876347	2.384088	0.208897	0.173623	2.086028	0.192044	...	0.196296	0.397676	0.335936	0.2513
1079	J3295_15	0.302626	0.461059	0.256564	2.092790	2.594348	0.251001	0.191811	2.361816	0.223632	...	0.187556	0.420347	0.335062	0.2529

1080 rows × 82 columns

In [3]:
```
tcss_group = df[df['class'] == 't-CS-s']
ccss_group = df[df['class'] == 'c-CS-s']
print("Number of mice for class t-CS-s is:",len(tcss_group.index))
print("Number of mice for class c-CS-s is:",len(ccss_group.index))
```

```
Number of mice for class t-CS-s is: 105
Number of mice for class c-CS-s is: 135
```

b)

To give different color to each mice class, we'll have to combine both mouse classes into one dataframe so that we can give the data easily to the plotly function.

The original code from plotly website contained a 'color' paramete which passes the color to different classes depend on the values. Problem we encounter was that this 'color' parameter don't accept string values. That's why we changed the classes of mice from string values to integer values assigning the 't-CS-s' class value of '1' and 'c-CS-s' class value of '2' precisely.

In [4]:
```
df2=pd.concat([tcss_group, ccss_group])
df2
```

Out[4]:
```
     MouseID  DYRK1A_N  ITSN1_N   BDNF_N    NR1_N   NR2A_N   pAKT_N  pBRAF_N  pCAMKII_N  pCREB_N ...   pCFOS_N    SYP_N  H3AcK18_N  EGR1_

840  18899_1  0.506200  0.696046 0.316019 2.204591 4.154323 0.254859 0.180295  2.557473  0.192694 ...       NaN 0.397663   0.155484    Na

841  18899_2  0.523760  0.746212 0.324897 2.285640 4.322314 0.268767 0.194387  2.648244  0.198864 ...       NaN 0.388920   0.164507    Na

842  18899_3  0.518612  0.733233 0.356137 2.330148 4.631455 0.272468 0.186284  2.624078  0.192656 ...       NaN 0.387212   0.156970    Na

843  18899_4  0.436986  0.626614 0.295108 2.008023 3.605088 0.258317 0.183562  2.648141  0.194521 ...       NaN 0.428589   0.187594    Na

844  18899_5  0.505599  0.719826 0.314600 2.194110 3.908544 0.281833 0.200539  2.804231  0.210701 ...       NaN 0.416420   0.182469    Na

...      ...       ...      ...      ...      ...      ...      ...      ...       ...       ... ...       ...      ...        ...

430 50810F_11 0.393097  0.506367 0.303954 2.169236 3.417560 0.292225 0.208110  3.700737  0.269102 ...  0.119256 0.385777        NaN    Na

431 50810F_12 0.390873  0.512566 0.296296 2.112434 3.481481 0.284722 0.198743  3.588624  0.266865 ...  0.096600 0.363439        NaN    Na

432 50810F_13 0.350444  0.456195 0.356233 1.959475 2.934774 0.290621 0.192590  3.191432  0.255886 ...  0.146874 0.489381        NaN    Na

433 50810F_14 0.399414  0.496445 0.368883 2.116688 2.801757 0.314095 0.230029  3.525721  0.291092 ...  0.146512 0.505635        NaN    Na

434 50810F_15 0.347810  0.469789 0.344411 1.916918 2.724698 0.311178 0.197130  3.182779  0.274924 ...  0.161435 0.492377        NaN    Na

240 rows × 82 columns
```

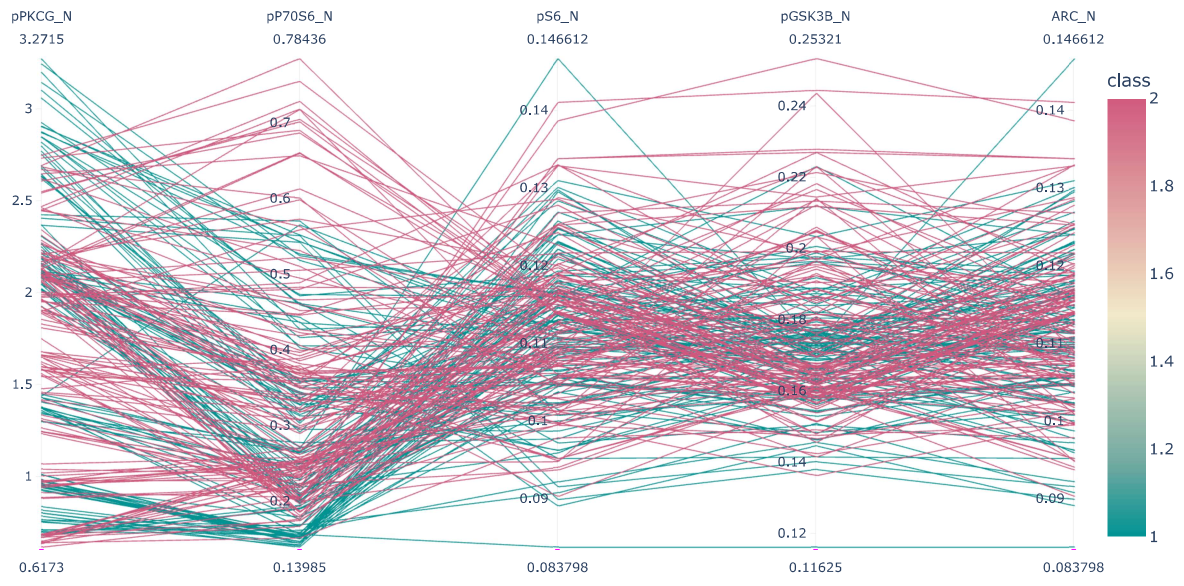
In [5]:
```
df2['class'] = df2['class'].replace('t-CS-s',1)
df2['class'] = df2['class'].replace('c-CS-s',2)
df2
```

Out[5]:
```
     MouseID  DYRK1A_N  ITSN1_N   BDNF_N    NR1_N   NR2A_N   pAKT_N  pBRAF_N  pCAMKII_N  pCREB_N ...   pCFOS_N    SYP_N  H3AcK18_N  EGR1_
840  18899_1  0.506200  0.696046 0.316019 2.204591 4.154323 0.254859 0.180295  2.557473  0.192694 ...       NaN 0.397663   0.155484    Na
841  18899_2  0.523760  0.746212 0.324897 2.285640 4.322314 0.268767 0.194387  2.648244  0.198864 ...       NaN 0.388920   0.164507    Na
842  18899_3  0.518612  0.733233 0.356137 2.330148 4.631455 0.272468 0.186284  2.624078  0.192656 ...       NaN 0.387212   0.156970    Na
843  18899_4  0.436986  0.626614 0.295108 2.008023 3.605088 0.258317 0.183562  2.648141  0.194521 ...       NaN 0.428589   0.187594    Na
844  18899_5  0.505599  0.719826 0.314600 2.194110 3.908544 0.281833 0.200539  2.804231  0.210701 ...       NaN 0.416420   0.182469    Na
...      ...       ...      ...      ...      ...      ...      ...      ...       ...       ... ...       ...      ...        ...
430 50810F_11 0.393097  0.506367 0.303954 2.169236 3.417560 0.292225 0.208110  3.700737  0.269102 ...  0.119256 0.385777        NaN    Na
431 50810F_12 0.390873  0.512566 0.296296 2.112434 3.481481 0.284722 0.198743  3.588624  0.266865 ...  0.096600 0.363439        NaN    Na
432 50810F_13 0.350444  0.456195 0.356233 1.959475 2.934774 0.290621 0.192590  3.191432  0.255886 ...  0.146874 0.489381        NaN    Na
433 50810F_14 0.399414  0.496445 0.368883 2.116688 2.801757 0.314095 0.230029  3.525721  0.291092 ...  0.146512 0.505635        NaN    Na
434 50810F_15 0.347810  0.469789 0.344411 1.916918 2.724698 0.311178 0.197130  3.182779  0.274924 ...  0.161435 0.492377        NaN    Na

240 rows × 82 columns
```

```
In [6]: fig = px.parallel_coordinates(df2, color="class",
                                   dimensions=['pPKCG_N','pP70S6_N', 'pS6_N',
                                               'pGSK3B_N', 'ARC_N'],
                                    color_continuous_scale=px.colors.diverging.Tealrose,
                                    color_continuous_midpoint=1.5
                                     )
        fig.show()
```

```
C:\Users\madni\anaconda3\lib\site-packages\plotly\express\_core.py:279: FutureWarning: iteritems is deprecated and will be rem
oved in a future version. Use .items instead.
  dims = [
```



c)

The suspecious thing we found was that the protien "pS6_N" and "ARC_N" have identical range of protien values and have a positive corelation between them.

Ex.2: Comparing RadViz and Star Coordinates

a)

They can be interpreted as dimensionality reduction techniques because both Star Coordinates and RadViz represents high dimensional data into lower dimension while trying to preserve the orignality of data in higher dimension. or we could say they visualize the P dimensional data on Plane.

b)

The authors prefer star coordinates over RadViz because:

1. Star coordinates preserves the shape of the distribution of points along a line segment while RadViz may not preserve the uniform distribution of points along a line segment in the data space when mapping them. This can result in points
   concentrating towards one of the endpoints, resulting in nonlinear distortions. This can alter the relative distances between the data points and lead to misleading plots.

2. The RadViz has poor performance in showing accurate outliers than the star coordinate. To prove this, authors ran two model strategies for searching outliers. The first was with respect to the center of mass and the second was with respect to any other mapped point. the result showed that the star coordinate showed more frequent outliers even with extreme values whereas RazVid misinterpreted some points as outliers and showed less performance.

c)

In the case of showing sparse data clearly, RadViz is better than star coordinate. The case goes as the following: If very few of the attributes values(depends on the anchor points distance) of data are really large than the others, then the RadViz would map these points close to their corresponding anchors and other smaller values(non-sparse) close to the origin. As the result, the difference in the values can be observed very easily by the viewers. For star coordinate, it's not the same case as it plots the larger data close to the tip of the corresponding axis vector but non-sparse data can also be present there.That's why authors prefer RadViz over star coordinate.

d)

1. Another ambiguity of RadViz is that it cannot obtain the correct high-dimensional attribute values visually. The normalization step in RadViz transforms the data so that the sum of its elements is 1, which eliminates information about the original attribute values. That's why, in RadViz, users can try to compare attribute values with each other in plot regarding their size, but cannot estimate their original values. Whereas, the Star coordinate axis can be labeled, allowing users to estimate original values visually.

2. Because of non-linear nature of RadViz, the data can become highly overlapped and it cannot be separates visually especially lighter colors and no matter whichever arrangement of anchor points we choose, there will still be this ambiguity.

Extended RadViz has the potential to resolve them by spreading out the anchor points along the line segments and adding partial spring lines.

e)

The two algorithm reported out like the following:

1. First algorithm was building regular configurations and permuting attributes. The computational effort was huge in this as ordering variables in visualizations is complex and time-consuming. It also did'nt achieved class separation for RadViz.

2. Second algorithm was “class discriminationlayout” (CDL) algorithm, which groups similar variables into sectors according to the t-statistic. The computational effort for this algorithm was significantly efficient. Class separation was achieved.

f)

This approach combines star coordinates and Linear Discriminant Analysis (LDA) to perform feature selection manually. Short axis vectors in the plot are removed as they could be least discriminative and don't affect the plot very much. This simplify the visualization and class overlap may reduce.

g)

To make sure that the projection of data points onto RadViz axes covers the whole range of each axis, anchor points in RadViz are selected at the corners of the overall convex hull. This ensures that the data distribution has no empty spaces that could impact the projection. this makes it simpler to map the data points to the RadViz axis.

This property does not meet for Star Coordinate axis in Fig 12(a), and in Fig 12(b) it does seem to form a convex hull with little bit of angle tolerance.

Ex.3: Principal Component Analysis

a)

In [7]:
```
df = pd.read_excel("breast-cancer-wisconsin.xlsx")
df['bareNuc'].fillna(int(df['bareNuc'].mean()), inplace = True)
df.head()
```

Out[7]:

	code	thickness	uniCelS	uniCelShape	marAdh	epiCelSize	bareNuc	blaChroma	normNuc	mitoses	class
0	1000025	5	1	1	1	2	1.0	3	1	1	2
1	1002945	5	4	4	5	7	10.0	3	2	1	2
2	1015425	3	1	1	1	2	2.0	3	1	1	2
3	1016277	6	8	8	1	3	4.0	3	7	1	2
4	1017023	4	1	1	3	2	1.0	3	1	1	2

In this case, I decided to use mean function because in bareNuc we have data only in range of 1 - 10. So we wouldn't have outliers that could affect our mean value.

b)

In [8]:
```
from sklearn.decomposition import PCA
pca = PCA()
```

In [9]:
```
df_new = df.drop(columns=['code', 'class'])
pca.fit(df_new)
```

Out[9]:
```
▾ PCA
PCA()
```

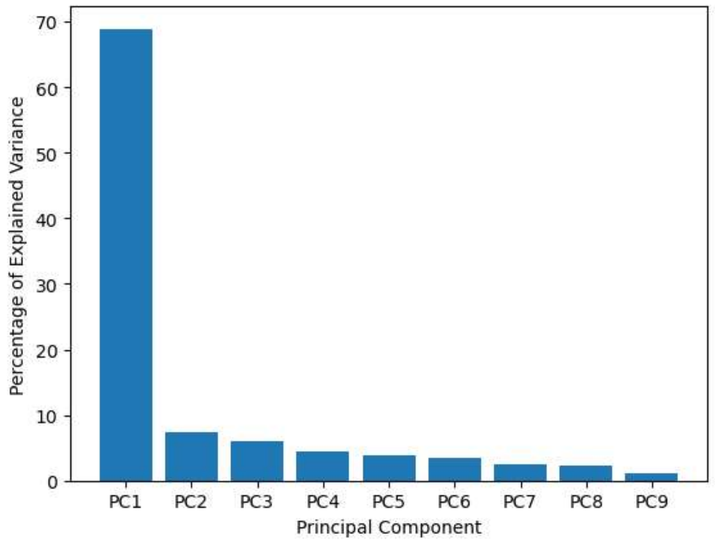
In [10]:
```
x_pca = pca.transform(df_new)
x_pca.shape
```

Out[10]:
```
(699, 9)
```

In [11]:
```
import numpy as np

per_var = np.round(pca.explained_variance_ratio_ * 100, decimals=1)
labels = ['PC' + str(x) for x in range(1,len(per_var)+1)]
```

In [12]:
```
plt.bar(x=range(1,len(per_var)+1),height=per_var, tick_label = labels)
plt.ylabel("Percentage of Explained Variance")
plt.xlabel("Principal Component")
plt.show()
```



To cover >=90% we need to take only four components PC1,PC2,PC3,PC4

c)

In [13]:
```
pca_df = pd.DataFrame(x_pca, columns = labels)
temp_map = {2:"benign", 4:"malignant"}
pca_df['class'] = df['class'].map(temp_map)
pca_df
```

Out[13]:

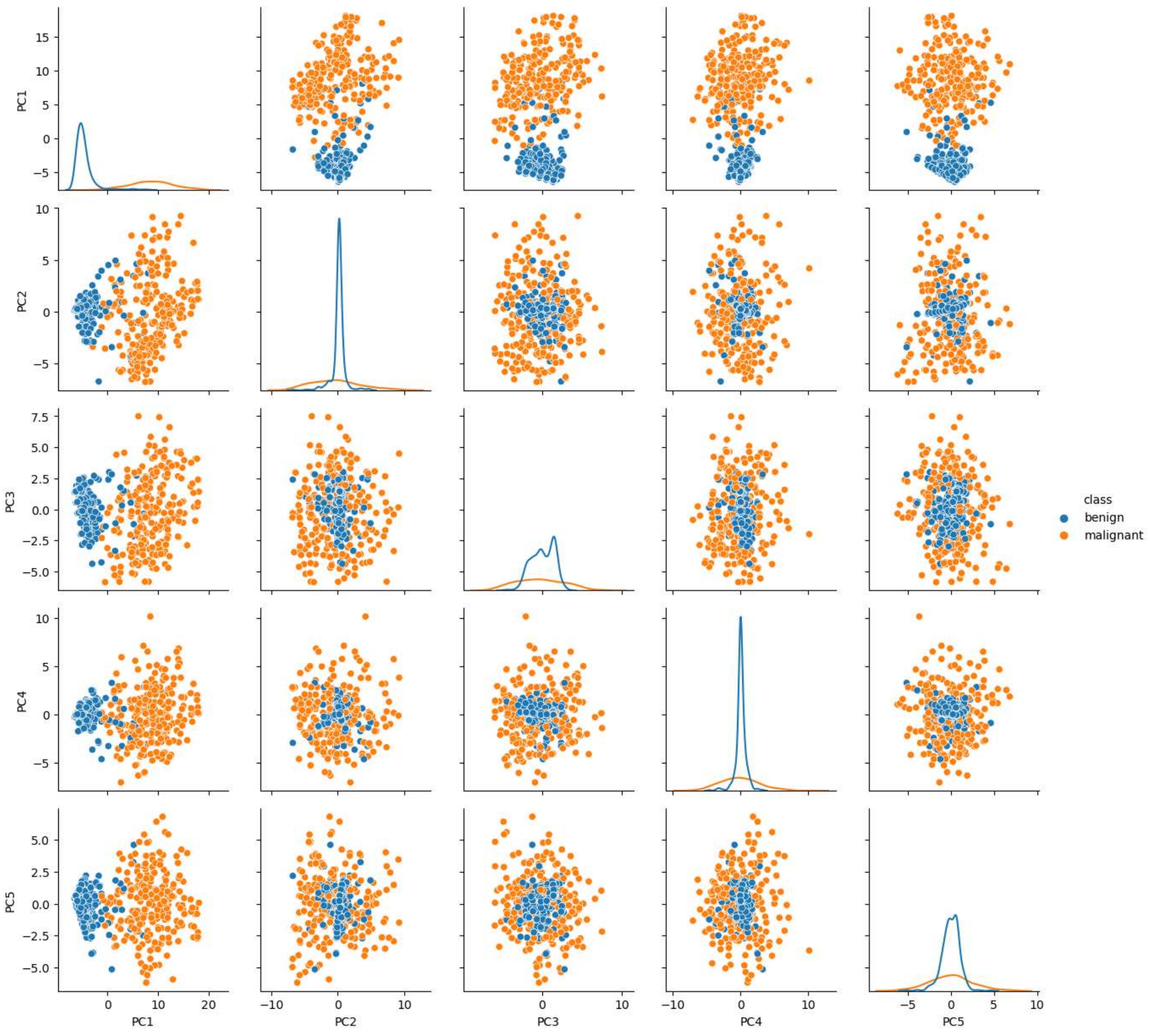
	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	class
0	-4.437445	0.084785	-1.739309	0.138475	-0.838819	-0.631344	-1.047331	0.077648	-0.014361	benign
1	4.829608	-4.822108	1.005076	0.658851	1.048190	2.223318	0.481139	3.110049	-0.004093	benign
2	-4.596621	-0.586992	0.042879	-0.313398	0.129666	-0.238086	-0.944825	-0.115098	0.037803	benign
3	5.151897	3.414987	-2.154582	-1.727735	3.261505	-0.800324	3.312871	-1.353263	0.617302	benign
4	-4.072421	-0.062545	0.069841	0.895564	-1.684328	-0.933017	-0.634688	0.197117	-0.106027	benign
...	...	...	...	...	...	...	...	...	...	...
694	-4.931526	-0.414221	-0.124669	-0.015401	0.171680	1.108924	0.365856	0.982274	0.090761	benign
695	-5.917762	0.239921	0.594248	0.022269	0.224915	0.543724	0.621374	0.150814	0.138471	benign
696	10.333555	7.220870	0.565126	-1.243139	4.030116	-1.715266	-0.227353	-0.191937	-0.233749	malignant
697	6.454953	2.483016	1.753074	-0.606009	2.188891	-4.360399	-2.459809	-1.997042	1.202542	malignant
698	7.545574	1.148008	1.939335	1.550752	3.146054	-4.146981	-2.114526	-1.739506	-0.278920	malignant

699 rows × 10 columns

In [14]:
```
df_show = pca_df[["PC1",'PC2','PC3','PC4','PC5','class']]

g= sns.PairGrid(df_show, hue = 'class')
g.map_diag(sns.kdeplot)
g.map_offdiag(sns.scatterplot)
g.add_legend()
```

Out[14]:
```
<seaborn.axisgrid.PairGrid at 0x1f15465ff70>
```



d)

The most difference between two types of samples we can see at PC1

In [15]:
```
df_show.loc[df_show['class'] == 'benign', 'PC1'].max()
```

Out[15]:
```
8.05047534796495
```

In [16]:
```
df_show.loc[df_show['class'] == 'benign', 'PC1'].min()
```

Out[16]:
```
-6.466337125162437
```

In [17]:
```
df_show.loc[df_show['class'] != 'benign', 'PC1'].min()
```

Out[17]:
```
-2.8644631403682967
```

In [18]:
```
df_show.loc[df_show['class'] != 'benign', 'PC1'].max()
```

Out[18]:
```
18.124744942925076
```

f)

If all variables will have different amount of digits, this will change Eigenvector at PCA. The correlation between variables could be proportional and it will be hard to understand dependencies between classes.

To avoid this it's better to use Scaler as pre-process. For example, there is such function at sklearn as "StandardScaler". It will bring all variables to a proper condition.