Data-driven assessment of rest-activity patterns

PSY 394S Machine Learning
Megan McMahon
Fall 2021

```
import pandas as pd
import numpy as np
import glob

import matplotlib.pyplot as plt
from matplotlib.dates import WeekdayLocator
import plotly.express as px
import seaborn as sns

from math import ceil
import random

from wearables import preproc

from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn import svm
```

```
actig_files = sorted(glob.glob('/Users/mcmahonmc/Box/CogNeuroLab/Aging Decision
Making R01/data/actigraphy/raw/*.csv'))

# actig_files_ = random.sample(actig_files, 10)

actig_files[:5]
```

```
['/Users/mcmahonmc/Box/CogNeuroLab/Aging Decision Making
R01/data/actigraphy/raw/30004_10_23_2017_10_20_00_AM_New_Analysis.csv',
 '/Users/mcmahonmc/Box/CogNeuroLab/Aging Decision Making
R01/data/actigraphy/raw/30008_11_9_2017_11_38_00_AM_New_Analysis.csv',
 '/Users/mcmahonmc/Box/CogNeuroLab/Aging Decision Making
R01/data/actigraphy/raw/30009_10_27_2017_6_35_00_PM_New_Analysis.csv',
 '/Users/mcmahonmc/Box/CogNeuroLab/Aging Decision Making
R01/data/actigraphy/raw/30012_10_30_2017_6_24_00_PM_New_Analysis.csv',
 '/Users/mcmahonmc/Box/CogNeuroLab/Aging Decision Making
R01/data/actigraphy/raw/30015_11_10_2017_4_15_00_PM_New_Analysis.csv']
```

```
# actdf = pd.DataFrame()

# weekday_map= {0:'Mon', 1:'Tue', 2:'Wed', 3:'Thu',
#                4:'Fri', 5:'Sat', 6:'Sun'}

# for file in actig_files:

#     subject = file.split('raw/')[1][:5]
# #     print(subject)

#     # read in actigraphy data
#     act = preproc.preproc(file, 'actiwatch', sr='.5T', truncate=False,
write=True, plot=True, recording_period_min=7)

#     # find first Monday midnight and start data from there so all subjects
starting on same day of the week
#     start = act[(act.index.dayofweek == 1) & (act.index.hour == 0)].index[0]

#     # cyclically wrap days that were cut off so not losing data
#     wrap = act[:start]
#     wrap.index = pd.date_range(start=act[start:].last_valid_index() +
pd.Timedelta(seconds=30),
#                                end=act[start:].last_valid_index() +
(wrap.last_valid_index() - wrap.first_valid_index()) + pd.Timedelta(seconds=30),
#                                freq='30S')

#     act = pd.concat((act[start:], wrap))

#     # keep only seven days of data
#     act = act[act.index <= (act.index[2] + pd.Timedelta(days=7))]

#     x_dates = [ weekday_map[day] for day in act.index.dayofweek.unique() ]

#     # plot
#     fig, ax = plt.subplots()
#     fig = sns.lineplot(x=act.index, y=act).set(title='sub-%s' % subject)
#     ax.set_xticklabels(labels=x_dates, rotation=45, ha='left')
#     ax.set_xlabel(''); ax.set_ylabel('Activity')
#     plt.show()

#     # if subject has < 7 days data, discard, else add to dataset
#     if ( (act.last_valid_index() - act.first_valid_index()) >=
pd.Timedelta(days=7) ):

#         actdf[subject] = act.values
#         print(actdf.shape)

#     else:

#         print('sub-%s discarded, recording period %s days' %
#               (subject, act.last_valid_index() - act.first_valid_index()))
```

```
#sns.heatmap(actdf.isnull(), cmap="YlGnBu")
```

```
# from sklearn.preprocessing import StandardScaler
# x = actdf[:-3].values
# x = StandardScaler().fit_transform(x) # normalizing the features
```

```
# x.shape
```

```
# x
```

```
7 * 24 * 60 * 2
```

```
20160
```

```
# np.save('/Users/mcmahonmc/Github/machine-learning-
2021/final/actigraphy_data.npy', x)
# actdf.to_csv('/Users/mcmahonmc/Github/machine-learning-
2021/final/actigraphy_data_df.csv', index=False)
```

Load data

```
x = np.load('/Users/mcmahonmc/Github/machine-learning-
2021/final/actigraphy_data.npy')
print('actigraphy data')
print(x.shape)

actdf = pd.read_csv('/Users/mcmahonmc/Github/machine-learning-
2021/final/actigraphy_data_df.csv')
print('actigraphy df')
print(actdf.shape)

targets = pd.read_csv('/Users/mcmahonmc/Github/machine-learning-
2021/final/target_data.csv')
print('targets')
print(targets.shape)

targets[:5]
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
/var/folders/ld/qzxjl7f92_5gzy756sfjkjfh0000gn/T/ipykernel_4163/3363168093.py in
<module>
----> 1 x = np.load('/Users/mcmahonmc/Github/machine-learning-
2021/final/actigraphy_data.npy')
      2 print('actigraphy data')
      3 print(x.shape)
      4 
      5 actdf = pd.read_csv('/Users/mcmahonmc/Github/machine-learning-
2021/final/actigraphy_data_df.csv')

/usr/local/lib/python3.9/site-packages/numpy/lib/npyio.py in load(file, mmap_mode,
allow_pickle, fix_imports, encoding)
    415             own_fid = False
    416         else:
--> 417             fid = stack.enter_context(open(os_fspath(file), "rb"))
    418             own_fid = True
    419 

FileNotFoundError: [Errno 2] No such file or directory:
'/Users/mcmahonmc/Github/machine-learning-2021/final/actigraphy_data.npy'
```

```
drop_subs = [ subject for subject in actdf.columns if int(subject) not in
targets.subject.values ]
drop_subs_idx = [ actdf.columns.get_loc(subject) for subject in actdf.columns if
int(subject) not in targets.subject.values ]

actdf = actdf.drop(drop_subs, axis=1)[:-3]
x = np.delete(x, drop_subs_idx, axis=1)

print(actdf.shape)
print(x.shape)
```

```
(20160, 116)
(20160, 116)
```

```
from sklearn.preprocessing import StandardScaler
x = actdf.values
x = StandardScaler().fit_transform(x) # normalizing the features
x.shape
```

```
(20160, 116)
```

```
x
```

```
array([[-0.61692969, -0.71235121, -0.5561038 , ..., -0.59062899,
         0.02914105,  1.82727309],
       [-0.63382197, -0.71235121, -0.5561038 , ..., -0.59062899,
        -0.24126602,  0.06096767],
       [-0.63382197, -0.71235121, -0.5561038 , ..., -0.59062899,
         0.31341515,  0.52851911],
       ...,
       [-0.63382197, -0.71235121, -0.5561038 , ..., -0.06502976,
        -0.67807743, -0.73776603],
       [-0.63382197, -0.69865941, -0.5561038 , ...,  0.1274432 ,
        -0.65727689,  0.28824962],
       [-0.63382197, -0.71235121, -0.5561038 , ..., -0.03541853,
        -0.68501095,  0.48955649]])
```

```
np.mean(x),np.std(x)
```

```
(4.350956133725299e-18, 1.0)
```

Compute traditional rest-activity measures

```
# from wearables import fitcosinor, npmetrics
# from datetime import datetime

# rar = pd.DataFrame()

# for subject in actdf.columns:

#     df = pd.DataFrame(actdf[subject][:-2]).set_index(pd.to_datetime(
#         pd.date_range(start = pd.to_datetime('2021-01-01 00:00:00'),
#                       end = pd.to_datetime('2021-01-01 00:00:00') +
pd.Timedelta(days=7),
#                       freq='30S'),
#         format = '%Y-%m-%d %H:%M:%S'))

#     df.columns = ['Activity']

#     cr = np.array(fitcosinor.fitcosinor(df)[0].T.values).T[0]
#     nonp = npmetrics.np_metrics_all(df['Activity'])

#     rar[subject] = np.concatenate((cr, nonp[:3]))

# rar = rar.T
# rar.columns = ['actmin', 'amp', 'alpha', 'beta', 'phi', 'IS', 'IV', 'RA']
# rar
```

```
rar = pd.read_csv('/Users/mcmahonmc/Github/machine-learning-2021/final/rar_df.csv',
index_col=0)

drop_subs = [ int(subject) for subject in rar.index if str(subject) not in
actdf.columns.values ]
drop_subs

rar2 = (rar[~rar.index.isin(drop_subs)])
print(rar2.shape)
```

```
(116, 8)
```

```
# rar.to_csv('/Users/mcmahonmc/Github/machine-learning-2021/final/rar_df.csv')
```

Define targets

target data uses output from rar dataframe merged with other variables of interest

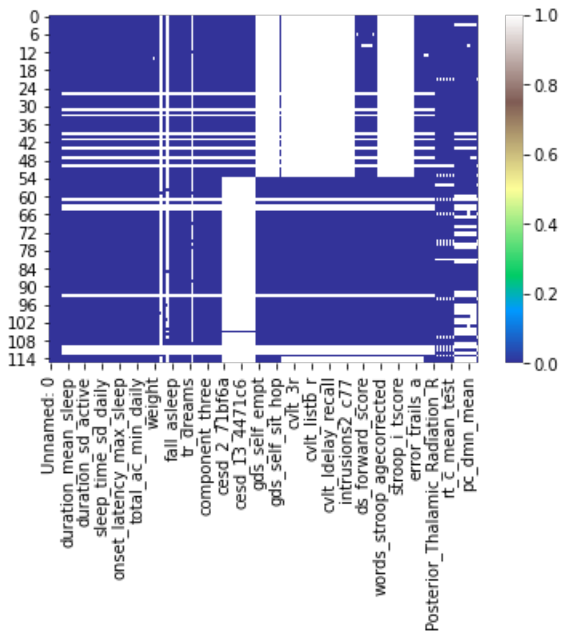
Missing data

Missing data here for CESD (this is intended for young adults only), GDS (this is intended for older adults only), and some of the MRI measures (due to poor image quality).

For targets of interest, will impute missing values with the mean.

```
sns.heatmap(targets.isnull(), cmap='terrain')
```

```
<AxesSubplot:>
```



```
# targets.to_csv('/Users/mcmahonmc/Github/machine-learning-
2021/final/target_data.csv', index=True)
```

Dimensionality Reduction

PCA

```
pca_act = PCA(n_components=2)
principalComponents_act= pca_act.fit_transform(x)
```

```
principal_act_Df = pd.DataFrame(data = principalComponents_act)
principal_act_Df.columns = ['component_' + str(colname + 1) for colname in
principal_act_Df.columns]
principal_act_Df
```

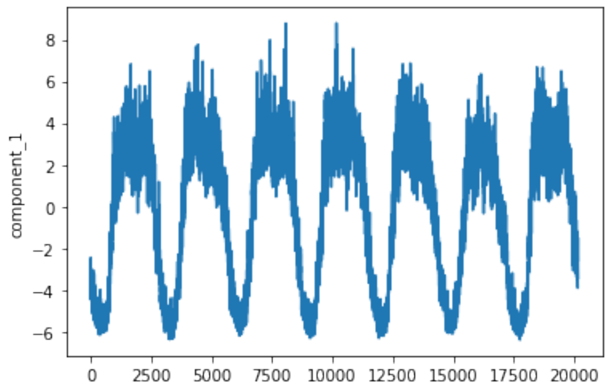
	component_1	component_2
0	-3.460750	2.269355
1	-3.305961	2.209452
2	-3.331143	2.420833
3	-3.435500	1.568814
4	-4.412418	1.668438
...	...	...
20155	-3.485638	1.707544
20156	-2.509299	2.652071
20157	-2.134925	3.531561
20158	-1.499098	1.755054
20159	-2.585157	3.124961

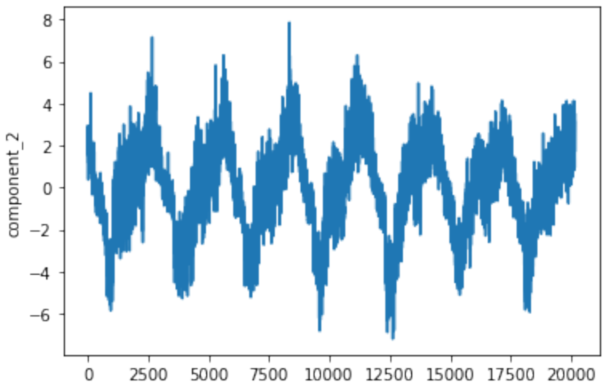
20160 rows × 2 columns

```
print('Explained variation per principal component:
{}'.format(pca_act.explained_variance_ratio_))
print('Silhouette score: %.3f' % metrics.silhouette_score(pca_act.components_.T,
targets['trails_b_group'], metric='euclidean'))
```

```
Explained variation per principal component: [0.11192743 0.03658112]
Silhouette score: -0.048
```

```
for col in principal_act_Df.columns:

    sns.lineplot(x=principal_act_Df.index, y=principal_act_Df[col])
    plt.show()
```





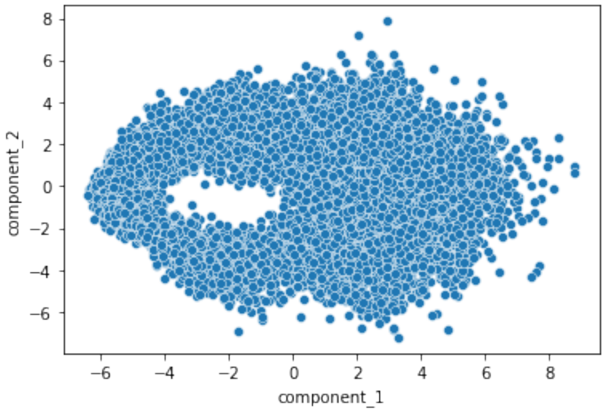
```
pca_act.components_.T.shape
```

Print to PDF

```
(116, 2)
```

```
sns.scatterplot(x=principal_act_Df['component_1'],
y=principal_act_Df['component_2'])
```

```
<AxesSubplot:xlabel='component_1', ylabel='component_2'>
```



```
print('Silhouette score: %.3f' %
metrics.silhouette_score(pca_act.components_[[0,1]].T, targets['trails_b_group'],
metric='euclidean'))
# print('Silhouette score: %.3f' %
metrics.silhouette_score(pca_act.components_[[1,2]].T, targets['trails_b_group'],
metric='euclidean'))
# print('Silhouette score: %.3f' %
metrics.silhouette_score(pca_act.components_[[0,2]].T, targets['trails_b_group'],
metric='euclidean'))
```

```
Silhouette score: -0.048
```

Comparison with traditional rest-activity measures

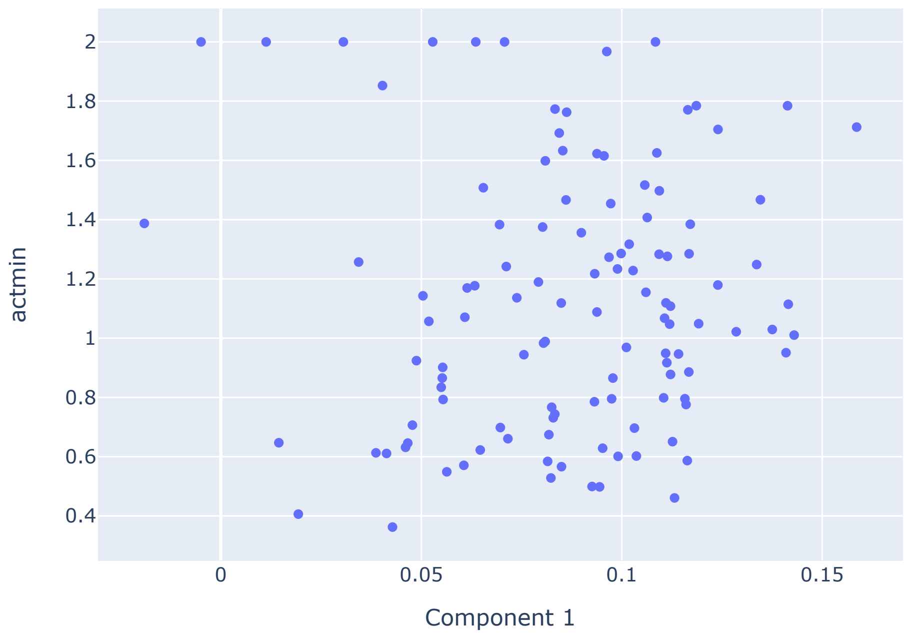
PCA

Component 1

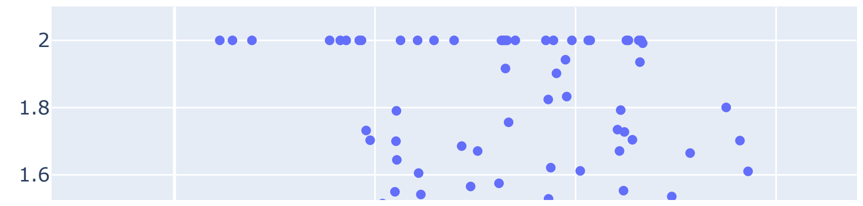
Shows some correspondance with IS, interdaily stability

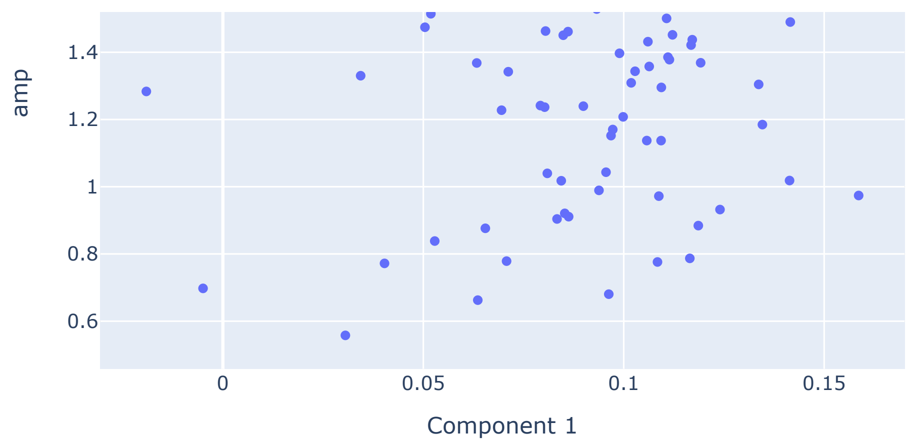
```
plt.figure(figsize=(10,10))
for col in rar2.columns:
    rarplt = px.scatter(x=pca_act.components_[0],
               y=rar2[col],
               labels = {
                     'x' : 'Component 1',
                     'y' : col
                  },
                  title='PCA Component 1 and %s' % col)

    display(rarplt)
```

PCA Component 1 and actmin

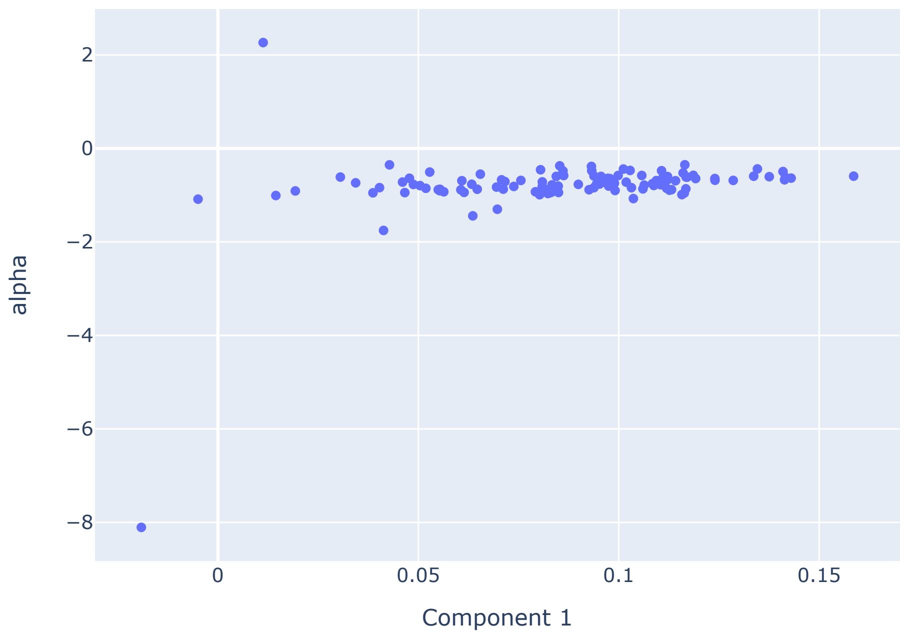


PCA Component 1 and amp

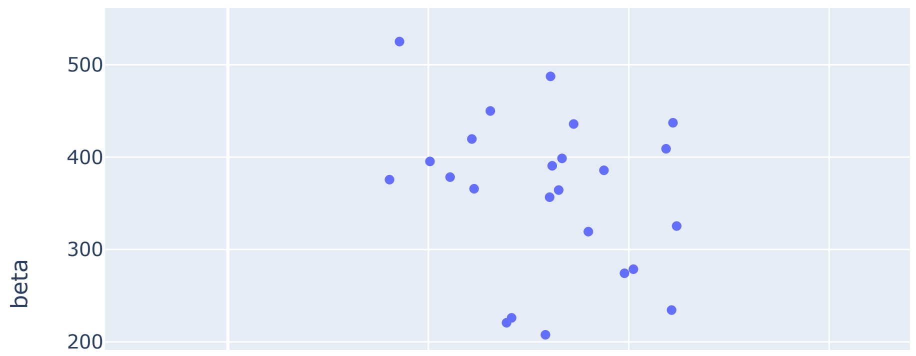


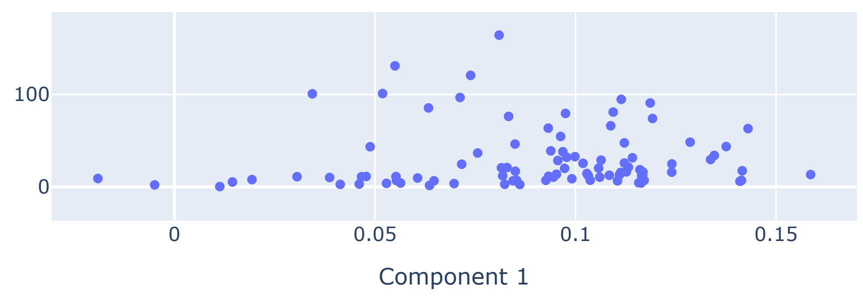


PCA Component 1 and alpha

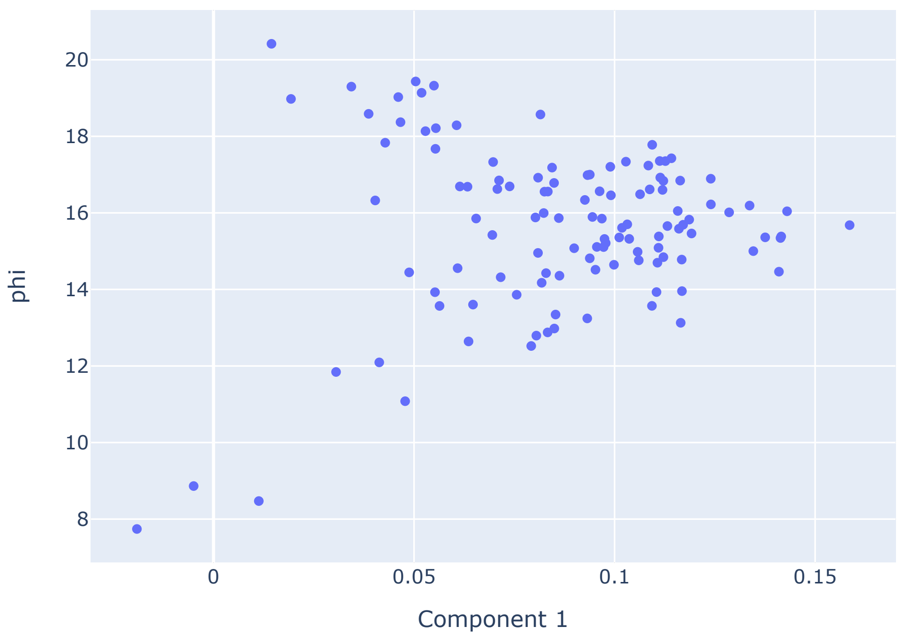


PCA Component 1 and beta

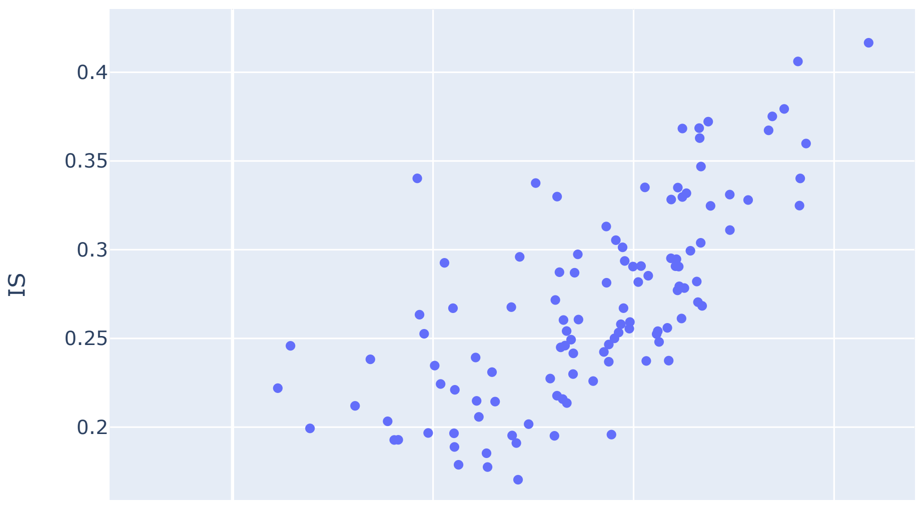


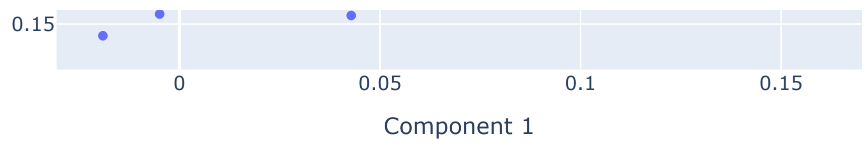


PCA Component 1 and phi

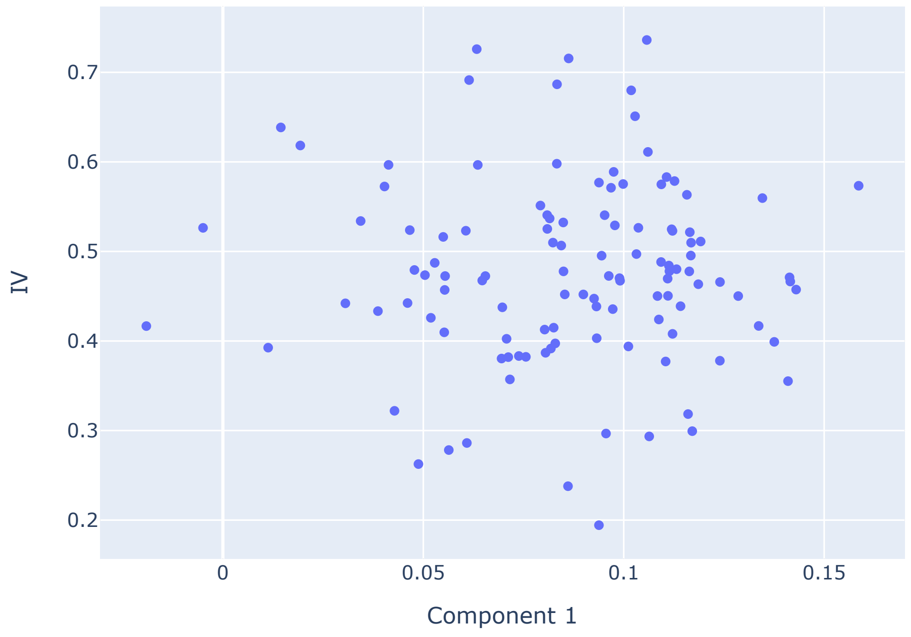


PCA Component 1 and IS

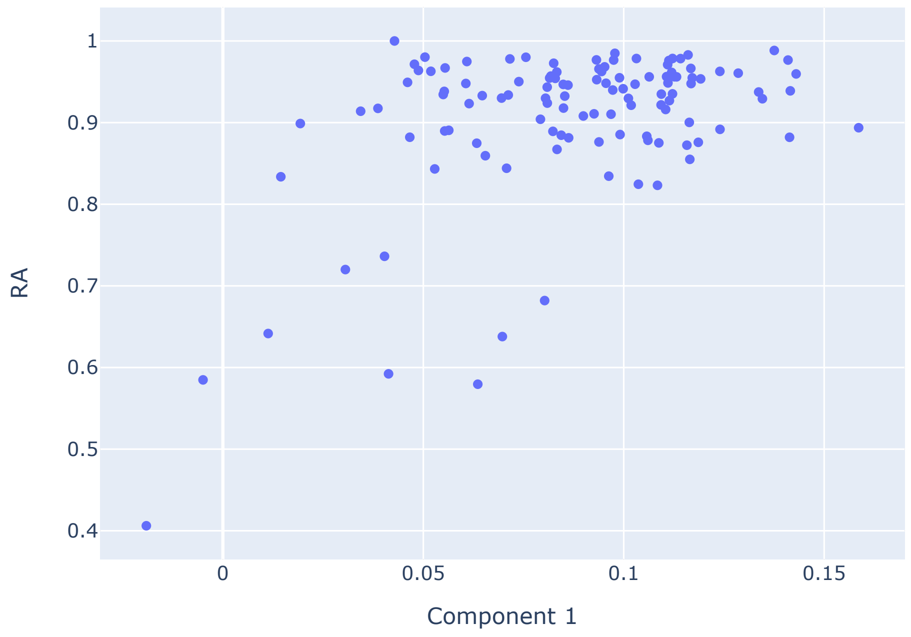




PCA Component 1 and IV



PCA Component 1 and RA



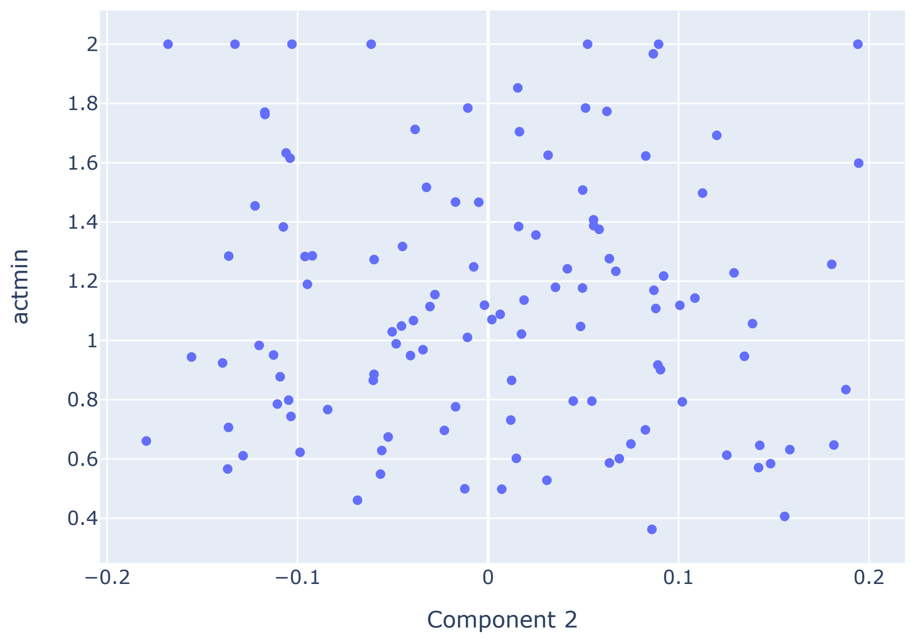
```
<Figure size 720x720 with 0 Axes>
```

Component 2

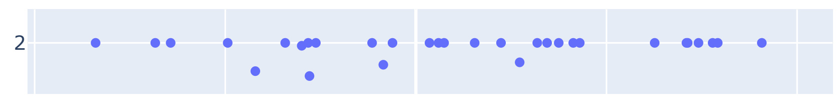
Shows some correspondance with acrophase (phi), which is an indicator of rest-activity rhythm phase and corresponds to the modelled time of peak activity

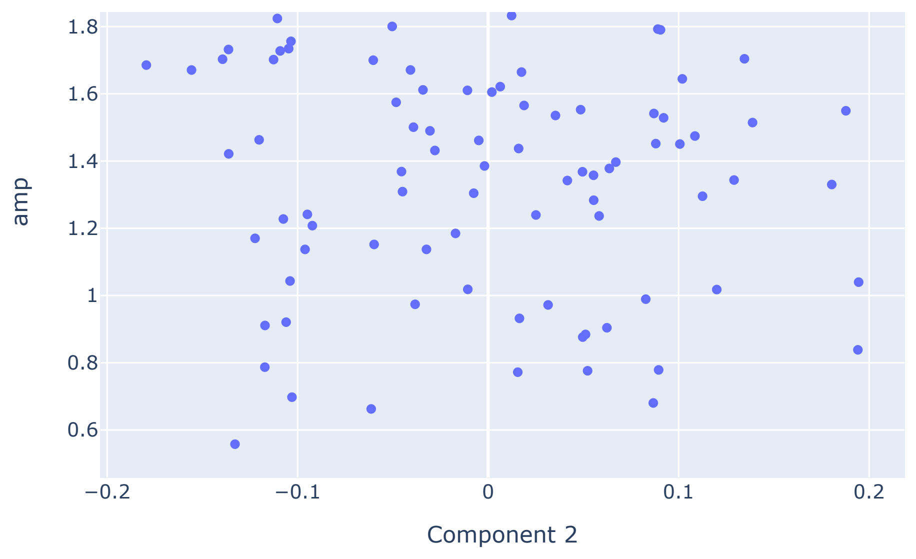
```
plt.figure(figsize=(10,10))
for col in rar2.columns:
    rarplt = px.scatter(x=pca_act.components_[1],
                y=rar2[col],
                labels = {
                       'x' : 'Component 2',
                       'y' : col
                   },
                   title='PCA Component 2 and %s' % col)

    display(rarplt)
```

PCA Component 2 and actmin

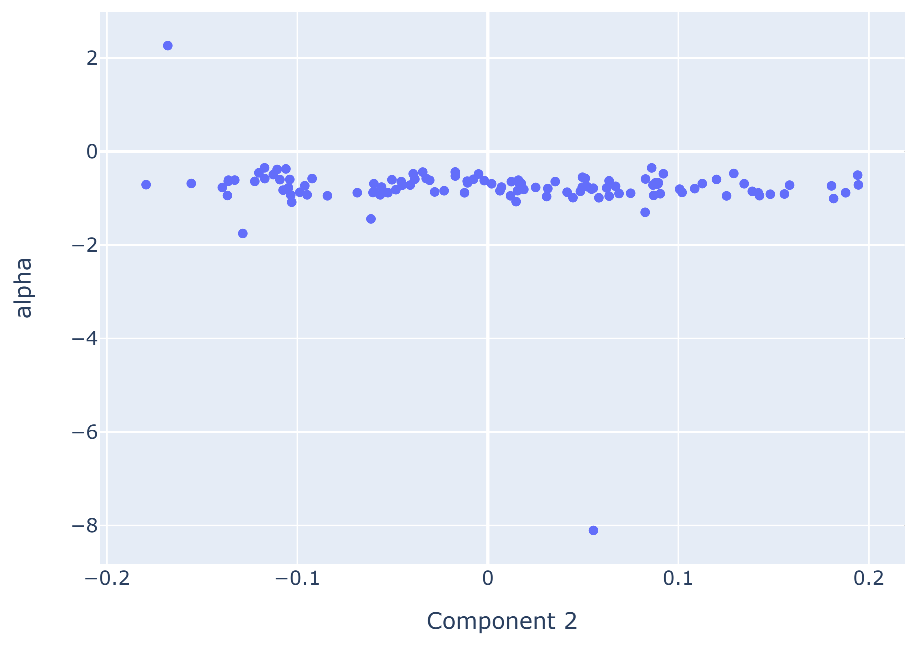


PCA Component 2 and amp

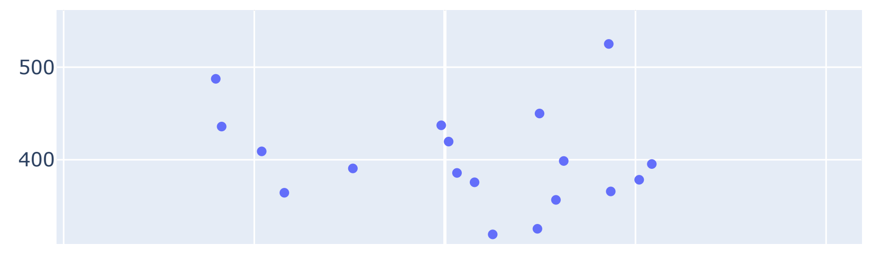


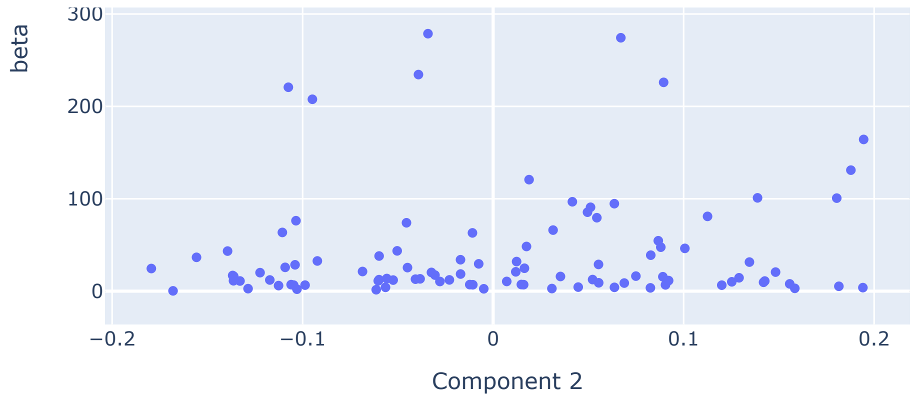


PCA Component 2 and alpha

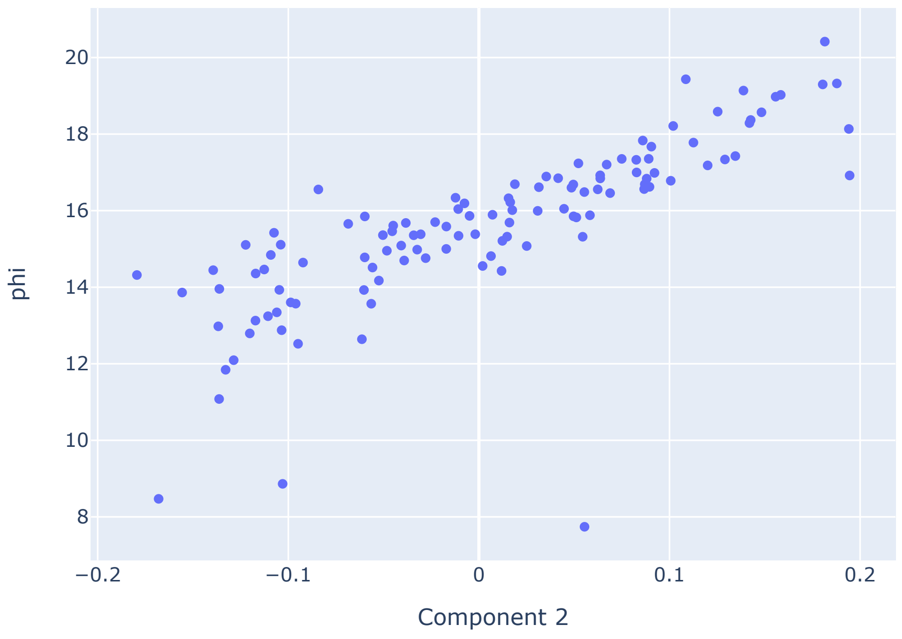


PCA Component 2 and beta

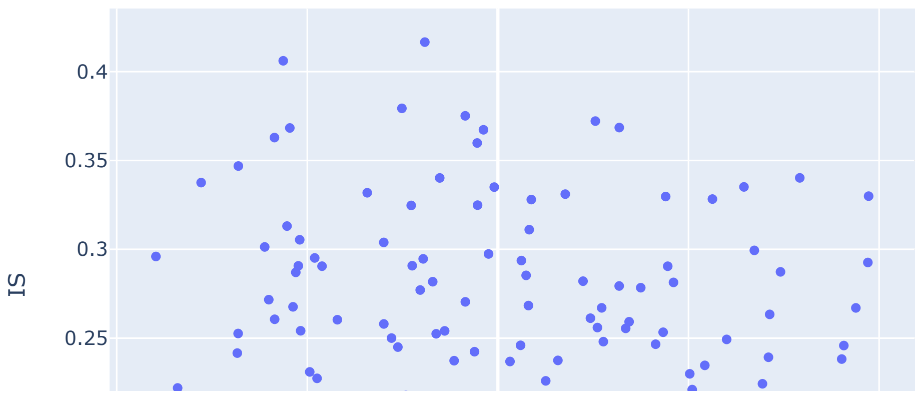


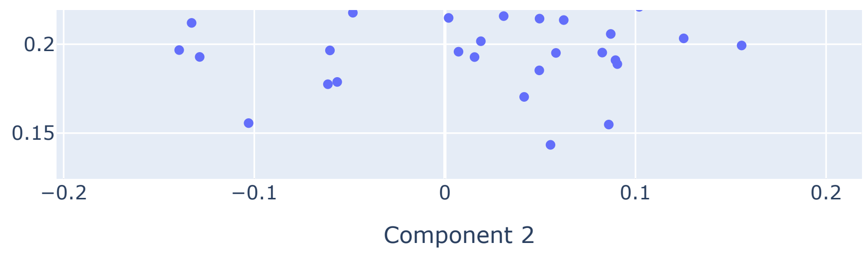


PCA Component 2 and phi

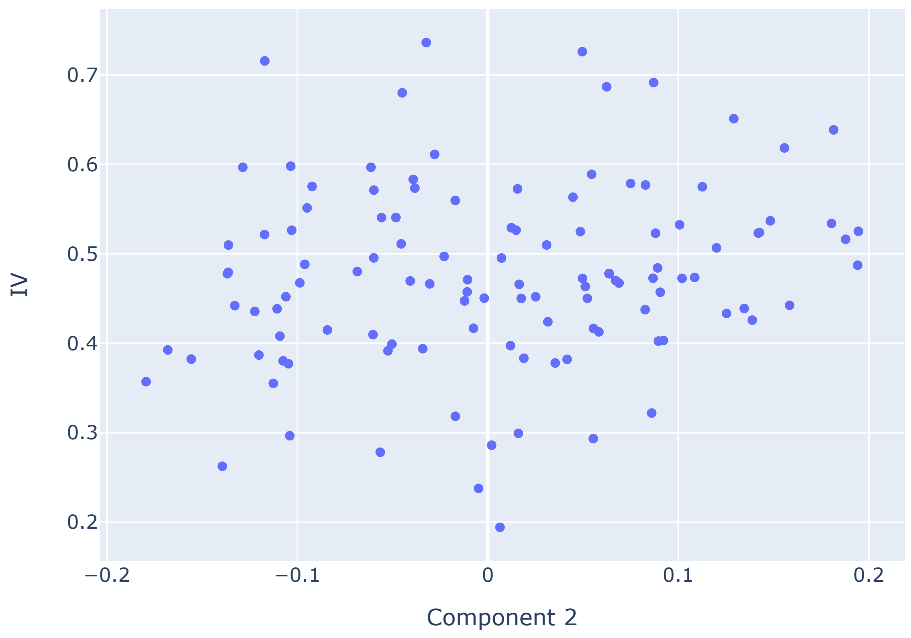


PCA Component 2 and IS

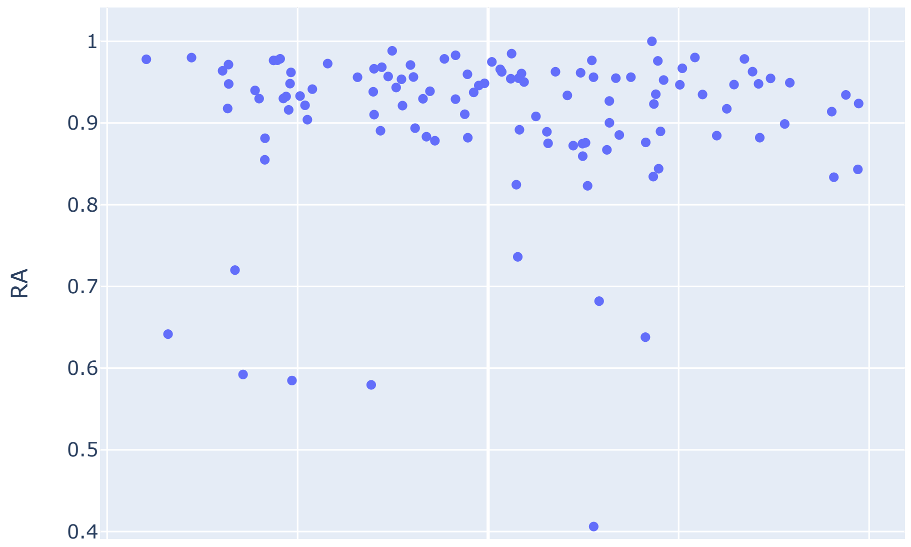


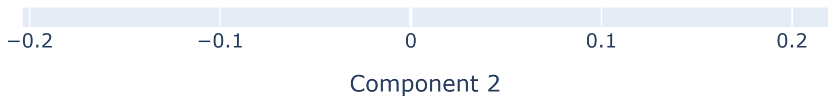


PCA Component 2 and IV



PCA Component 2 and RA





```
<Figure size 720x720 with 0 Axes>
```

LLE

```
plt.figure(figsize=(10,10))
for col in rar2.columns:
    rarplt = px.scatter(x=pca_act.components_[0],
               y=rar2[col],
               labels = {
                      'x' : 'Component 1',
                      'y' : col
                  },
                  title='PCA Component 1 and %s' % col)

    display(rarplt)
```

Classification

SVM

Datacamp

Cognition

```
targets.columns
```

```
Index(['Unnamed: 0', 'Unnamed: 0.1', 'subject', 'age', 'group', 'gender',
       'trails_b_z_score_x', 'trails_b_group', 'duration_mean_active',
       'duration_mean_daily',
       ...
       'pc_dmn_fpn_18', 'pc_dmn_fpn_20', 'pc_dmn_fpn_mean', 'pc_dmn_mean',
       'pc_fpn_mean', 'edge_0', 'edge_1', 'edge_2', 'edge_3',
       'acc_mean_test_log'],
      dtype='object', length=271)
```

```
targets['trails_b_group'] = np.where(targets['trails_b_z_score_x'] > 1, "High",
"Average")
targets['trails_b_group'] = np.where(targets['trails_b_z_score_x'] < -1, "Low",
targets['trails_b_group'])

average_idx = targets[targets['trails_b_group'] == 'Average'].index.values

# Split dataset into training set and test set
X_train, X_test, y_train, y_test =
train_test_split(np.delete(pca_act.components_[[0,1]].T, average_idx, axis=0),
                                                    targets['trails_b_group']
[targets['trails_b_group'] != 'Average'],
                                                    test_size=0.3,
random_state=100)

print('Training data points: dim_x = %s, dim_y = %s' % (X_train.shape,
y_train.shape))
print('Test data points: dim_x = %s, dim_y = %s' % (X_test.shape, y_test.shape))
```

```
Training data points: dim_x = (38, 2), dim_y = (38,)
Test data points: dim_x = (17, 2), dim_y = (17,)
```

```
#Create a svm Classifier
kernel_methods = ['linear', 'poly', 'rbf', 'sigmoid']

for kernel_method in kernel_methods:

    clf = svm.SVC(kernel=kernel_method)

    #Train the model using the training sets
    clf.fit(X_train, y_train)

    #Predict the response for test dataset
    y_pred = clf.predict(X_test)

    print(kernel_method + ", Accuracy: %.2f " % metrics.accuracy_score(y_test,
y_pred))
```

```
linear, Accuracy: 0.53
poly, Accuracy: 0.47
rbf, Accuracy: 0.53
sigmoid, Accuracy: 0.65
```

```
clf = svm.SVC(kernel='sigmoid')

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

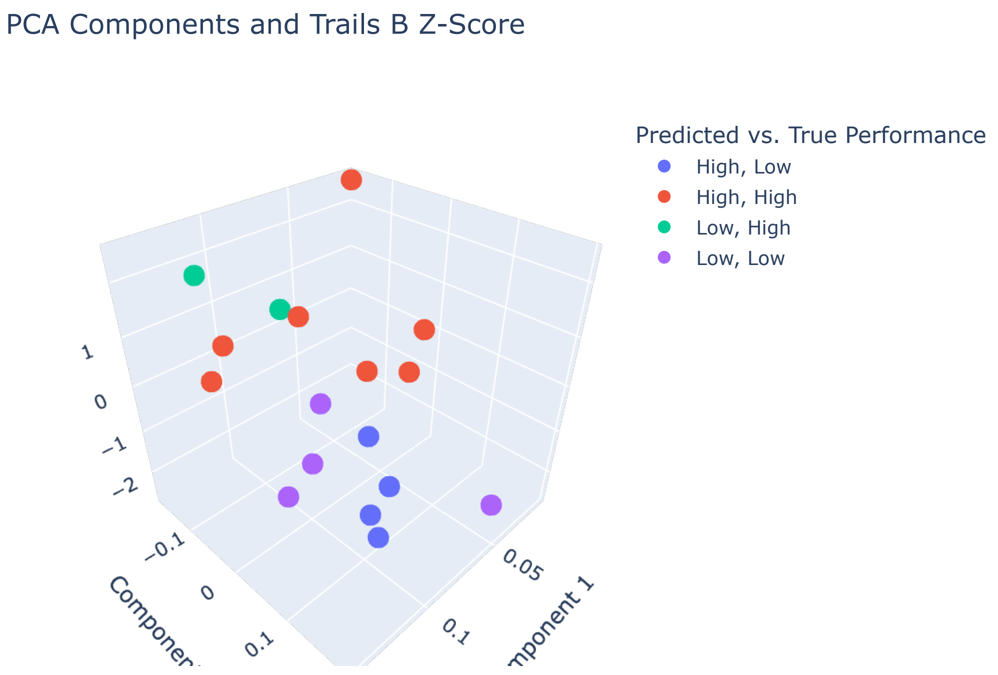
```
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision: %.2f" % metrics.precision_score(y_test, y_pred,
average='weighted'))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall: %.2f" % metrics.recall_score(y_test, y_pred, average='weighted'))
```

```
Precision: 0.65
Recall: 0.65
```

```
pca_act.components_[[0,1]].shape
```

```
(2, 116)
```

```
markers = y_pred + ', ' + y_test

px.scatter_3d(x=X_test[:,0],
              y=X_test[:,1],
              z=targets.iloc[y_test.index.values]['trails_b_z_score_x'],
              color=markers,
              labels = {
                  'x' : 'Component 1',
                  'y' : 'Component 2',
                  'z' : 'Trails B Z-Score',
                  'color': 'Predicted vs. True Performance'
              },
              title='PCA Components and Trails B Z-Score')
```



Brain

```
targets['mod_mean'].fillna(targets['mod_mean'].mean(), inplace=True)
print(targets['mod_mean'])
from sklearn.preprocessing import MinMaxScaler

scaler = StandardScaler()
targets['mod_mean_scaled'] =
scaler.fit_transform(targets['mod_mean'].values.reshape(-1,1))
targets['mod_mean_scaled']

targets['mod_mean_group'] = np.where(targets['mod_mean_scaled'] > 1, "High",
"Average")
targets['mod_mean_group'] = np.where(targets['mod_mean_scaled'] < -1, "Low",
targets['mod_mean_group'] )

average_idx = targets[targets['mod_mean_group'] == 'Average'].index.values

# Split dataset into training set and test set
X_train, X_test, y_train, y_test =
train_test_split(np.delete(pca_act.components_[[0,1]].T, average_idx, axis=0),
                                                    targets['mod_mean_group']
[targets['mod_mean_group'] != 'Average'],
                                                    test_size=0.3,
random_state=100)

print('Training data points: dim_x = %s, dim_y = %s' % (X_train.shape,
y_train.shape))
print('Test data points: dim_x = %s, dim_y = %s' % (X_test.shape, y_test.shape))
```

```
0      0.204444
1      0.180552
2      0.182655
3      0.123560
4      0.165057
         ...
111    0.178232
112    0.123560
113    0.139451
114    0.123560
115    0.123560
Name: mod_mean, Length: 116, dtype: float64
Training data points: dim_x = (25, 2), dim_y = (25,)
Test data points: dim_x = (11, 2), dim_y = (11,)
```

```
targets['mod_mean_group'].unique()
```

```
array(['High', 'Average', 'Low'], dtype=object)
```

```
#Create a svm Classifier
kernel_methods = ['linear', 'poly', 'rbf', 'sigmoid']

for kernel_method in kernel_methods:

    clf = svm.SVC(kernel=kernel_method)

    #Train the model using the training sets
    clf.fit(X_train, y_train)

    #Predict the response for test dataset
    y_pred = clf.predict(X_test)

    print(kernel_method + ", Accuracy: %.2f " % metrics.accuracy_score(y_test,
y_pred))
```

```
linear, Accuracy: 0.45
poly, Accuracy: 0.36
rbf, Accuracy: 0.45
sigmoid, Accuracy: 0.64
```

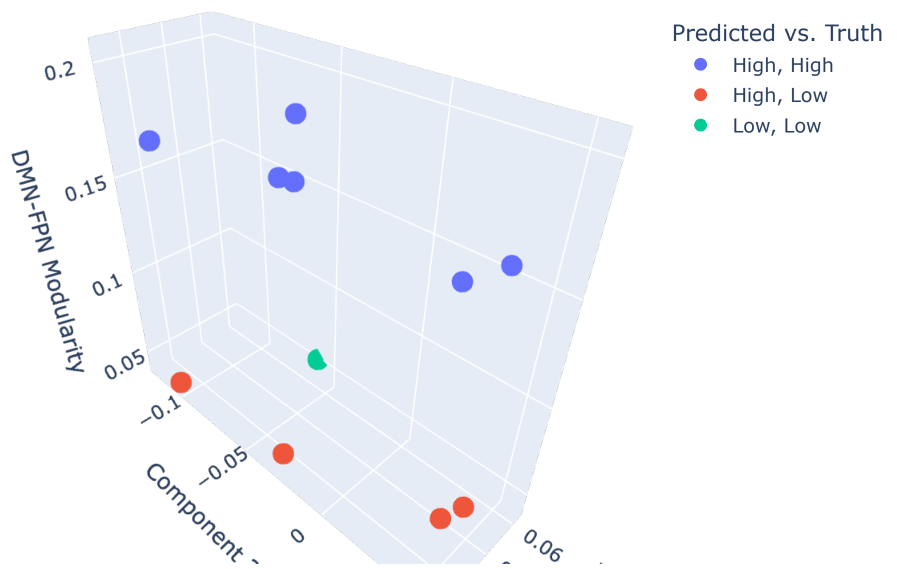
```
#Create a svm Classifier
# kernel{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}, default='rbf'
clf = svm.SVC(kernel='sigmoid')

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

```
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision: %.2f" % metrics.precision_score(y_test, y_pred,
average='weighted'))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall: %.2f" % metrics.recall_score(y_test, y_pred, average='weighted'))
```

```
Precision: 0.78
Recall: 0.64
```

```
markers = y_pred + ', ' + y_test

px.scatter_3d(x=X_test[:,0],
              y=X_test[:,1],
              z=targets.iloc[y_test.index.values]['mod_mean'],
              color=markers,
              labels = {
                  'x' : 'Component 1',
                  'y' : 'Component 2',
                  'z' : 'DMN-FPN Modularity',
                  'color': 'Predicted vs. Truth'
              },
              title='PCA Components and DMN-FPN Modularity')
```

PCA Components and DMN-FPN Modularity



```
targets['edge_mean'] = targets[[x for x in targets.columns if
x.startswith('edge_')]].mean(axis=1)
```

```
targets['edge_mean'].fillna(targets['edge_mean'].mean(), inplace=True)
print(targets['edge_mean'])
from sklearn.preprocessing import MinMaxScaler

scaler = StandardScaler()
targets['edge_mean_scaled'] =
scaler.fit_transform(targets['edge_mean'].values.reshape(-1,1))
targets['edge_mean_scaled']

targets['edge_mean_group'] = np.where(targets['edge_mean_scaled'] > 1, "High",
"Average")
targets['edge_mean_group'] = np.where(targets['edge_mean_scaled'] < -1, "Low",
targets['edge_mean_group'] )

average_idx = targets[targets['edge_mean_group'] == 'Average'].index.values

# Split dataset into training set and test set
X_train, X_test, y_train, y_test =
train_test_split(np.delete(pca_act.components_[[0,1]].T, average_idx, axis=0),
                                                    targets['edge_mean_group']
[targets['edge_mean_group'] != 'Average'],
                                                    test_size=0.3,
random_state=100)

print('Training data points: dim_x = %s, dim_y = %s' % (X_train.shape,
y_train.shape))
print('Test data points: dim_x = %s, dim_y = %s' % (X_test.shape, y_test.shape))
```

```
0      0.476839
1      0.648607
2      0.500161
3      0.438322
4      0.295476
         ...
111    0.112614
112    0.438322
113    0.373898
114    0.438322
115    0.438322
Name: edge_mean, Length: 116, dtype: float64
Training data points: dim_x = (23, 2), dim_y = (23,)
Test data points: dim_x = (10, 2), dim_y = (10,)
```

```
targets['edge_mean_group'].unique()
```

```
array(['Average', 'High', 'Low'], dtype=object)
```

```
#Create a svm Classifier
kernel_methods = ['linear', 'poly', 'rbf', 'sigmoid']

for kernel_method in kernel_methods:

    clf = svm.SVC(kernel=kernel_method)

    #Train the model using the training sets
    clf.fit(X_train, y_train)

    #Predict the response for test dataset
    y_pred = clf.predict(X_test)

    print(kernel_method + ", Accuracy: %.2f " % metrics.accuracy_score(y_test,
y_pred))
```

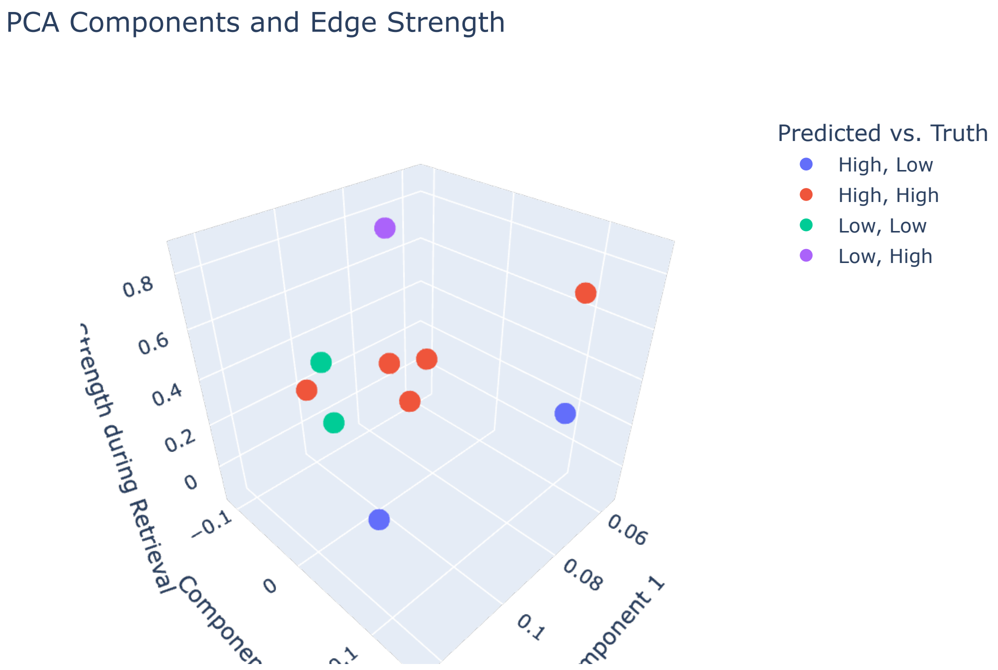
```
linear, Accuracy: 0.40
poly, Accuracy: 0.70
rbf, Accuracy: 0.70
sigmoid, Accuracy: 0.40
```

```
#Create a svm Classifier
# kernel{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}, default='rbf'
clf = svm.SVC(kernel='poly')

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

```
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision: %.2f" % metrics.precision_score(y_test, y_pred,
average='weighted'))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall: %.2f" % metrics.recall_score(y_test, y_pred, average='weighted'))
```

```
Precision: 0.70
Recall: 0.70
```

```
markers = y_pred + ', ' + y_test

px.scatter_3d(x=X_test[:,0],
              y=X_test[:,1],
              z=targets.iloc[y_test.index.values]['edge_mean'],
              color=markers,
              labels = {
                  'x' : 'Component 1',
                  'y' : 'Component 2',
                  'z' : 'Edge Strength during Retrieval',
                  'color': 'Predicted vs. Truth'
              },
              title='PCA Components and Edge Strength')
```


PCA Components and Edge Strength
Predicted vs. Truth
● High, Low
● High, High
● Low, Low
● Low, High
trength during Retrieval
Componen
mponent 1