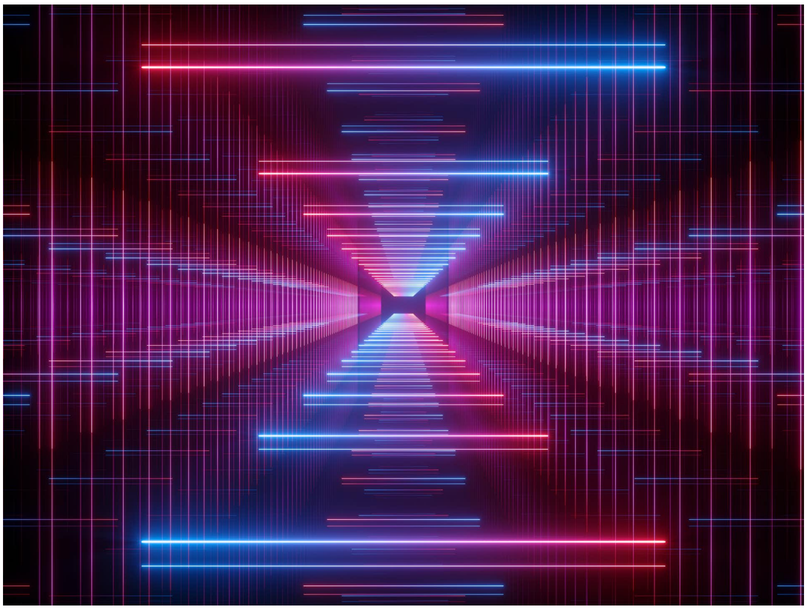


Dimensionality reduction and DBSCAN

-Further analysis on the MAGIC gamma project- ¶

-See appendix for full code-

Section 1: Data

Note, this is all taken from my previous IBM project. I am inserting the info, so you do not have find the other. Otherwise, it is available here: [https://github.com/C21AI/AI/tree/main/ibm-course/Supervised%20Machine%20Learning%20-%20Classification (https://github.com/C21AI/AI/tree/main/ibm-course/Supervised%20Machine%20Learning%20-%20Classification)](https://github.com/C21AI/AI/tree/main/ibm-course/Supervised%20Machine%20Learning%20-%20Classification)

Citation start:

"

I will use an UCI Machine Learning Repository called 'MAGIC Gamma Telescope Data Set' found on: [https://archive.ics.uci.edu/ml/datasets/MAGIC+Gamma+Telescope (https://archive.ics.uci.edu/ml/datasets/MAGIC+Gamma+Telescope)](https://archive.ics.uci.edu/ml/datasets/MAGIC+Gamma+Telescope)

Quote from data set description on how it was created: "The data set was generated by a Monte Carlo program, Corsika, described in D. Heck et al., CORSIKA, A Monte Carlo code to simulate extensive air showers, Forschungszentrum Karlsruhe FZKA 6019 (1998)".

It simulates "registration of high energy gamma particles in a ground-based atmospheric Cherenkov gamma telescope using the imaging technique." This is used to determine whether a specific particle is gamma rays, which is what we are interested in detecting, or whether it is from hadronic showers.

Shape of data: The data consists of 19,020 rows and 11 columns. The target column is "Particle". The features are:

- major axis of ellipse [mm]
- minor axis of ellipse [mm]
- 10-log of sum of content of all pixels [in #phot]
- ratio of sum of two highest pixels over fSize [ratio]
- ratio of highest pixel over fSize [ratio]
- distance from highest pixel to center, projected onto major axis [mm]
- 3rd root of third moment along major axis [mm]
- 3rd root of third moment along minor axis [mm]
- angle of major axis with vector to origin [deg]
- distance from origin to center of ellipse [mm]

Missing data: Luckily, no missing data.

Data types: All features are of float64 dtype, however the target variable "Particle" was at first an object. This was encoded to int64, as 1's and 0's.

Data cleaning: No cleaning required.

Feature engineering: Target column was binary encoded.

Other findings: The proportions of particles is 65% gamma particles/35% hadronic particles. I also included a correlation matrix to look for correlations

"

Citation end

Comment: Target column has not been encoded as it is not neccesary for this project

In [9]:

```python
display(data.sample(10))
print('')
print('Rows:', data.shape[0], '\nColumns:', data.shape[1])
print('')
data.info()
print('')
print(data['Particle'].value_counts(normalize=True))

plt.figure(figsize=(12,10))
ax = sn.heatmap(data.corr(), annot=True, linewidth=0.5)
ax.set_xlim([11,0])
ax.set_ylim([0,11])
plt.show()
```

	fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Trans	fAlpha	
9698	106.5540	31.4188	3.3434	0.1397	0.0755	-113.2060	53.6824	24.3952	7.1160	216
1466	40.0572	16.2294	2.4893	0.3468	0.1799	-10.8714	26.2948	9.4323	17.8954	205
2175	23.1207	14.0498	2.6479	0.4522	0.2396	3.9820	14.7351	-6.5502	22.2080	143
5731	42.2999	15.2507	2.4764	0.3472	0.1753	-33.2646	23.1308	-8.3025	11.1493	163
15142	195.0339	120.6847	3.7992	0.1320	0.0636	-236.7155	86.7975	-79.4557	83.2553	170
2732	27.2726	12.6129	2.7288	0.3978	0.2512	-10.4679	-17.5545	-9.9008	3.7890	185
9214	18.8375	11.9932	2.4720	0.5565	0.3558	-0.4388	-8.7199	-9.8301	32.2560	242
16246	35.7492	13.0825	2.6832	0.3689	0.1697	20.8657	-24.2911	-7.1984	80.2599	260
5608	45.9832	21.4123	2.9325	0.2734	0.1408	-4.4598	40.5153	18.8747	15.1333	187
11127	29.7465	11.6710	2.3531	0.4967	0.3171	35.0728	13.8927	6.3463	25.4410	192

```
Rows: 19020 
Columns: 11

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19020 entries, 0 to 19019
Data columns (total 11 columns):
fLength     19020 non-null float64
fWidth      19020 non-null float64
fSize       19020 non-null float64
fConc       19020 non-null float64
fConc1      19020 non-null float64
fAsym       19020 non-null float64
fM3Long     19020 non-null float64
fM3Trans    19020 non-null float64
fAlpha      19020 non-null float64
fDist       19020 non-null float64
Particle    19020 non-null object
dtypes: float64(10), object(1)
memory usage: 1.6+ MB

g    0.64837
h    0.35163
Name: Particle, dtype: float64
```

Section 2: Main objectives of analysis

The goal of this project is to try and reduce the feature dimensions of the MAGIC gamma dataset, then classifying, and observing how this affects the performance. In the end, I am trying out DBSCAN to see how that performs.

The dimensionality reduction will serve to provide insight into how many features really are important to understand the data. With low numbers of observations, it can actually worsen the performace of the model, if there are too many features; the curse of dimensionality.

DBSCAN will show whether there is a natural boundary between Hadron and Gamma particle observations. I just found this interesting. :)

Section 3: Model

First, I fitted the old model to have some kind of guiding principles for how well the reduced model worked.

I scaled the data with a MinMax classifier to keep the shape of the data. I had to to this as it was not required for the Random Forest Classifier from earlier.

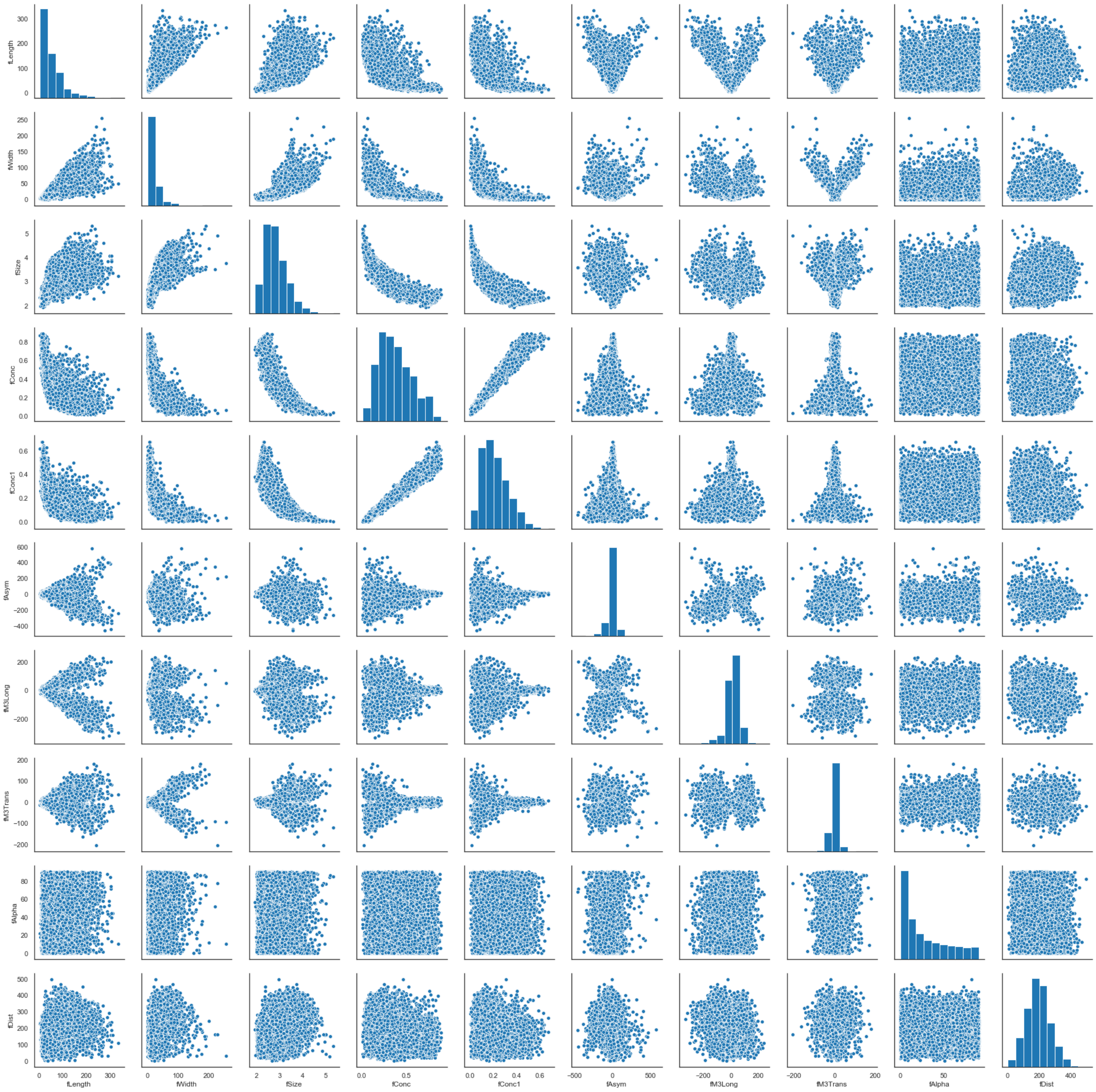
I plotted the scaled features in a pairplot to see if there were any correlations visible for me.

I then used sklearns PCA class, and plotted a graph with dimensions and explained variance

Lastly, I fitted the DBSCAN model.

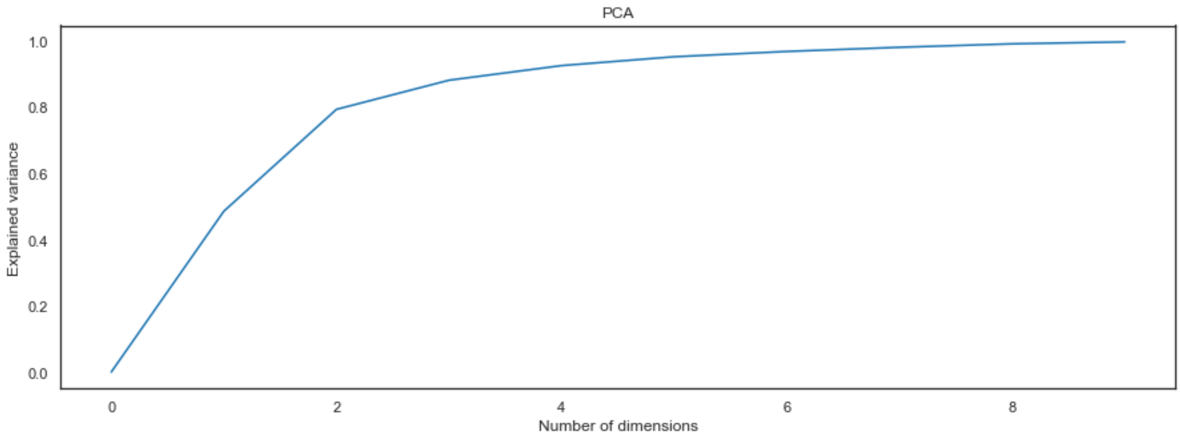
In [144]:

```
# Appendix
```



In [147]:

```
# Appendix
```



In [231]:

```
# Appendix
```

Out[231]:

dbscan	Particle	number
-1	g	6561
	h	5957
0	g	4794
	h	543
1	g	73
...	...	...
117	h	5
118	g	5
	h	2
119	g	1
	h	3

188 rows × 1 columns

Section 4: Findings

From training the models, it seemed the data can be reduced to 4 dimensions and still contain a lot of information

I then tried fitting the transformed data with 4 and 6 dimensions. The out-of-bag error increased by 7% for the model with 4 dimensions, but only 4. Considering it was respectively 40% and 60% of the entire data, that was great results.

Then, I continued with the DBSCAN. Here, not much was of interest. The model did a poor job, perhaps because I did not tweak it enough or perhaps there are no natural division.

Section 5: Possible flaws and plan to revisit

The PCA was quite interesting and could be inspected further. Of course, different classifications methods could be tried out.

In terms of the cluster, for further analysis, it would prove interesting to try out K-Means with 2 centroids. Also, the data were divided with 65% of observations being gamma. This surely has some influence on its tendency to assign variables to that cluster. By some kind of sampling technique, oversampling, undersampling SMOTE, this could be negated to some extent.

Appendix A: Workbook

DATA

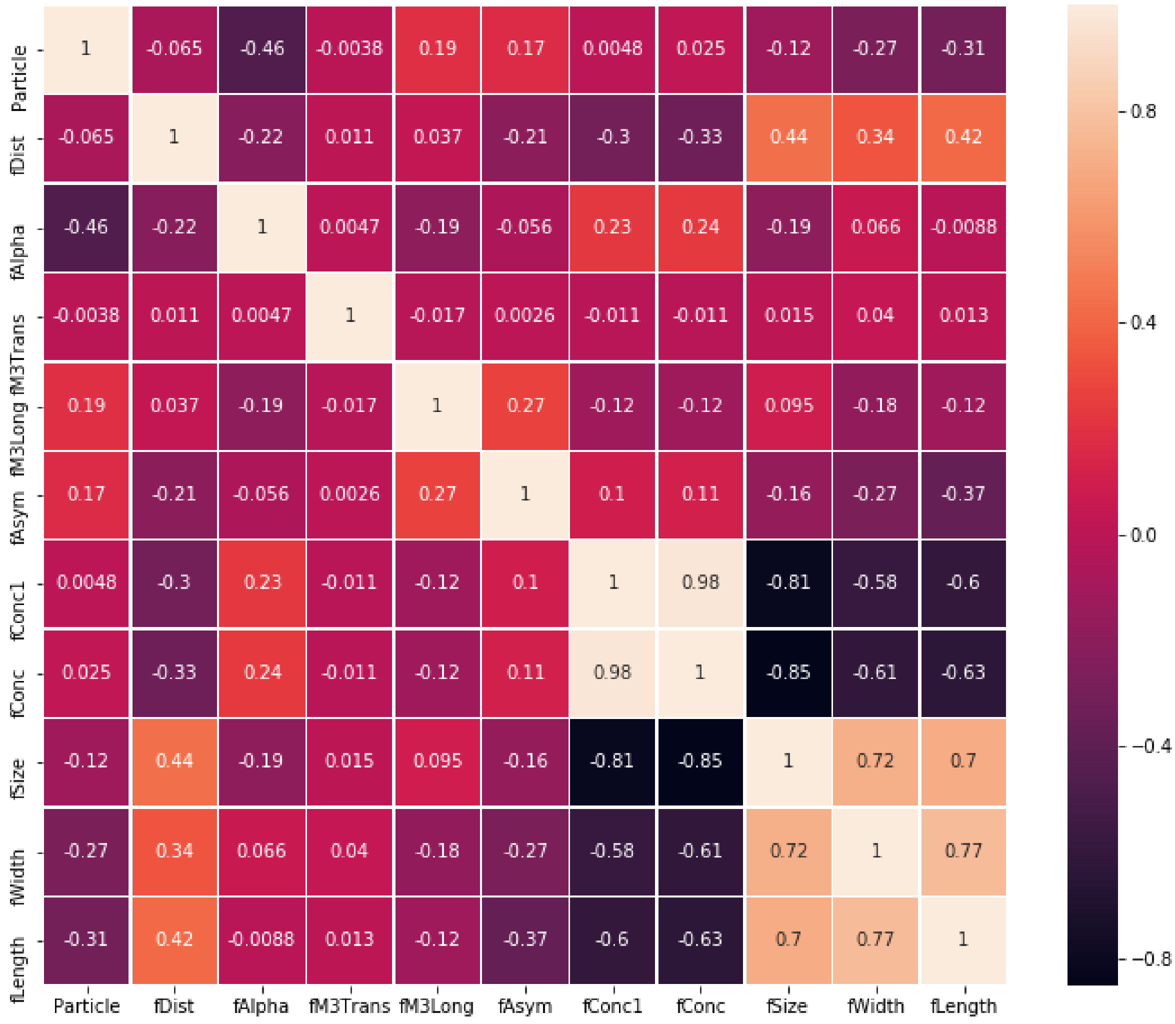
In [2]:

```
# OLD CODE FROM PREVIOUS ASSIGMENT

# Import
# Data wrangling
import pandas as pd
import numpy as np

# Import dataset
index_name_list = ['fLength', 'fWidth', 'fSize', 'fConc', 'fConc1', 'fAsym', 'fM3Long', 'fM
data = pd.read_csv('magic04.data', header = None, names = index_name_list)

display(data.sample(10))
print('')
print('Rows:', data.shape[0], '\nColumns:', data.shape[1])
print('')
data.info()
print('')
print(data['Particle'].value_counts(normalize=True))

df = data.copy()
df['Particle'] = df['Particle'].replace('h', 0).replace('g',1)

import pandas
import seaborn as sn
import matplotlib.pyplot as plt

plt.figure(figsize=(12,10))
ax = sn.heatmap(df.corr(), annot=True, linewidth=0.5)
ax.set_xlim([11,0])
ax.set_ylim([0,11])
plt.show()
```

	fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Trans	fAlpha	fI
14058	102.7604	21.2461	3.1720	0.1784	0.1251	76.1983	62.9071	21.5339	69.6356	168.4
10554	52.5594	16.3333	2.5826	0.3791	0.1922	31.9170	-22.3346	10.7765	2.8080	199.0
4827	31.8536	15.1511	2.5347	0.3679	0.2088	32.1326	21.1783	6.4599	2.0040	242.2
6875	21.8910	6.8045	2.0881	0.7102	0.3633	-25.5749	-6.6720	6.6325	35.1373	96.1
5652	52.7350	12.6817	2.6561	0.5563	0.3609	-64.5213	32.0105	11.6103	8.9282	277.7
6743	75.5657	16.5339	2.9004	0.2642	0.1365	-81.1110	52.6888	-16.4860	4.9959	258.8
2849	31.5349	12.1830	2.7206	0.4681	0.2445	40.4118	22.6149	-9.3787	5.6246	178.2
6794	78.9039	26.6129	3.4931	0.1497	0.0750	65.5807	66.1459	13.7387	9.8964	197.3
15318	28.2691	19.8662	2.6762	0.3804	0.2212	27.8963	12.3808	12.0986	38.1192	64.8
18253	159.0650	95.6256	4.1360	0.0763	0.0387	-8.7170	113.1460	70.2285	85.2700	374.0

```
Rows: 19020
Columns: 11
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19020 entries, 0 to 19019
Data columns (total 11 columns):
fLength     19020 non-null float64
fWidth      19020 non-null float64
fSize       19020 non-null float64
fConc       19020 non-null float64
fConc1      19020 non-null float64
fAsym       19020 non-null float64
fM3Long     19020 non-null float64
fM3Trans    19020 non-null float64
fAlpha      19020 non-null float64
fDist       19020 non-null float64
Particle    19020 non-null object
dtypes: float64(10), object(1)
memory usage: 1.6+ MB

g    0.64837
h    0.35163
Name: Particle, dtype: float64
```



MODEL

Last time, I found Random Forest Classifier to be the best model. I will continue with this model. After runnnig, I get the same result as last time

In [10]:

```
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, f1_scc

# X and y
y = data['Particle'].replace('h', 0).replace('g',1)
X = data[[x for x in data.columns if x not in 'Particle']]

# Initialize object
sss = StratifiedShuffleSplit(n_splits=1, test_size = 0.3)

# Create indexes
train_index, test_index = next(sss.split(X, y))

# Assign set
X_train, X_test = X.loc[train_index], X.loc[test_index]
y_train, y_test = y.loc[train_index], y.loc[test_index]

# Initialize classifier
RF = RandomForestClassifier(n_estimators = 100, oob_score = True, warm_start = True, n_jobs

# Fitting
RF = RF.fit(X_train, y_train)

y_pred = RF.predict(X_test)

# OOB
oob_error = 1- RF.oob_score_

# Performance
print('Out-of-bag error:', oob_error)
print('')
print(classification_report(y_test, y_pred))
```

```
Out-of-bag error: 0.12490611386510442

              precision    recall  f1-score   support

           0       0.87      0.77      0.82      2006
           1       0.88      0.94      0.91      3700

    accuracy                           0.88      5706
   macro avg       0.88      0.85      0.86      5706
weighted avg       0.88      0.88      0.88      5706
```

Prinicpal Component Analysis

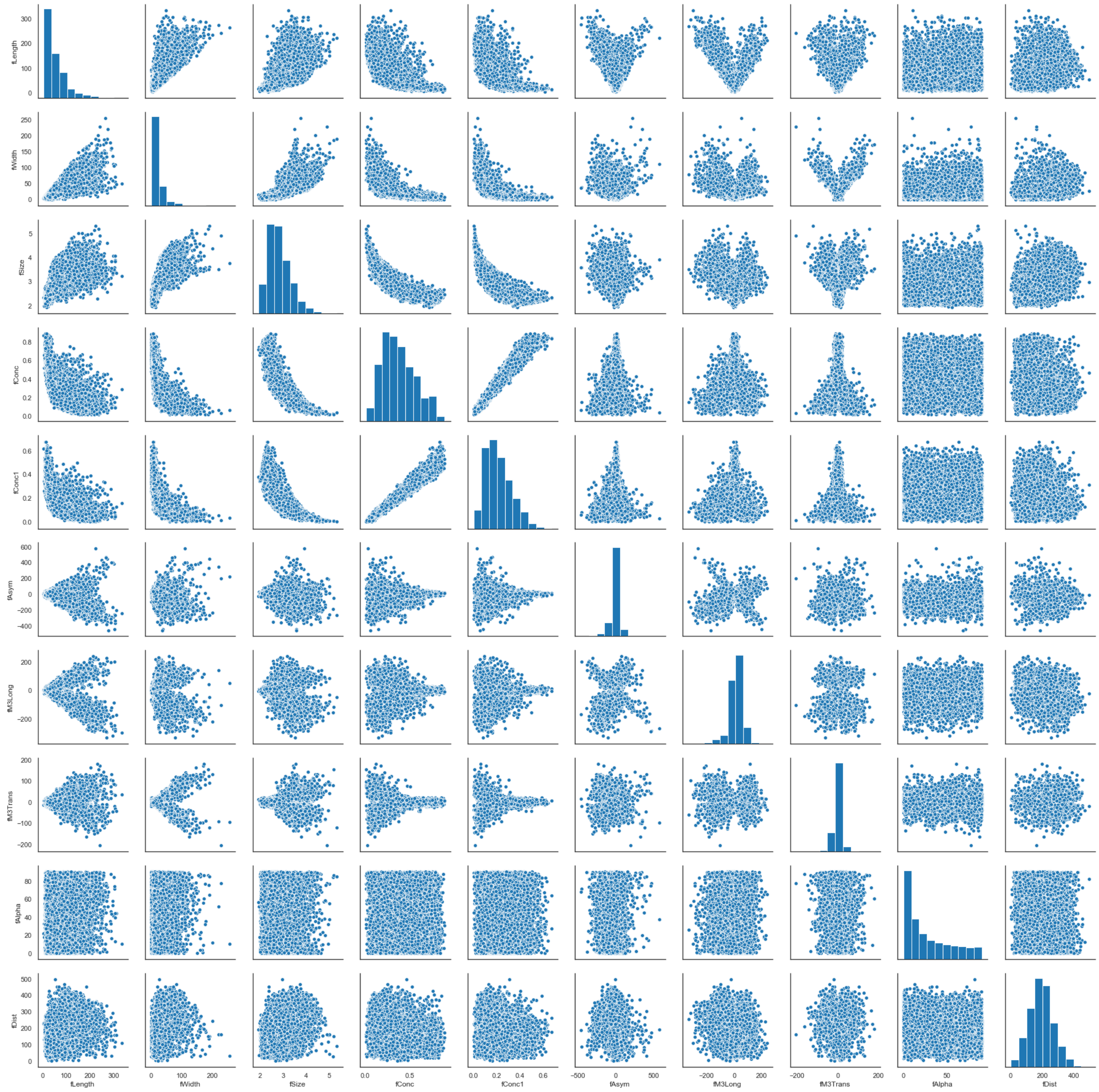
Firstly, the data must be scaled. This was not neccesary for the forest classifier, but it is for PCA

In [143]:

```
from sklearn.preprocessing import MinMaxScaler

MMscaler = MinMaxScaler().fit(X)
X_scaled = MMscaler.transform(X)
```

In [144]:

```
sns.set_context('notebook')
sns.set_style('white')
sns.pairplot(X);
```

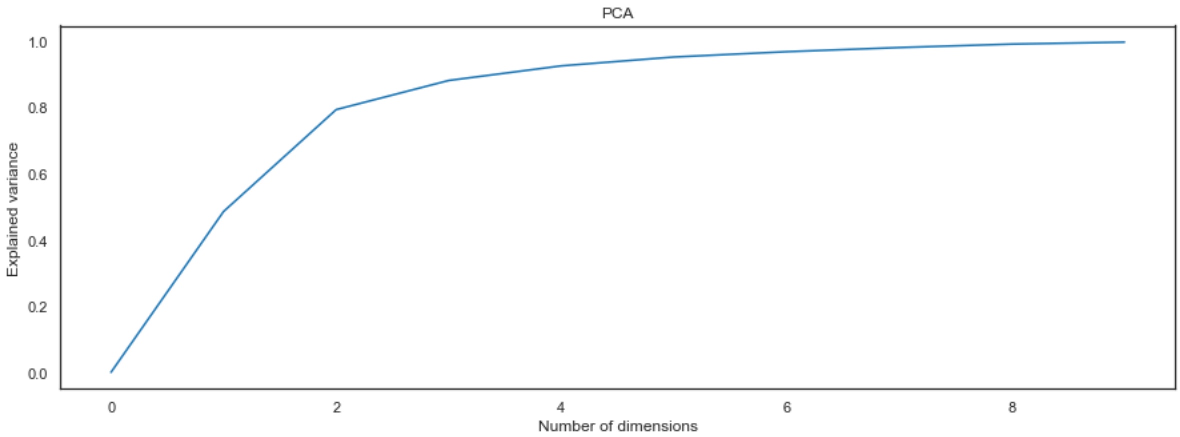


In [146]:

```
from sklearn.decomposition import PCA

expl_var_list = list()

# Initialize object
for i in list(range(10)):
    PCA_scaled = PCA(n_components = i)
    PCA_scaled = PCA_scaled.fit(X_scaled)
    expl_var_list.append(PCA_scaled.explained_variance_ratio_.sum())
```

In [147]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(15,5))
plt.plot(range(10), expl_var_list)
plt.title('PCA')
plt.xlabel('Number of dimensions')
plt.ylabel('Explained variance')
plt.show()
```



It seems the data can be reduced to 4 dimensions and still contain a lot of information

In [89]:

```
# PCA Transforming data
PCA_transform = PCA(n_components = 4)
X_PCA = PCA_transform.fit_transform(X_scaled)
```

In [115]:

```
# Trying Random Forest
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, f1_scc

# X and y
y = data['Particle'].replace('h', 0).replace('g',1)
X_PCA = pd.DataFrame(X_PCA)
X = X_PCA


# Initialize object
sss = StratifiedShuffleSplit(n_splits=1, test_size = 0.3)

# Create indexes
train_index, test_index = next(sss.split(X, y))

# Assign set
X_train, X_test = X.iloc[train_index], X.loc[test_index]
y_train, y_test = y.iloc[train_index], y.loc[test_index]

# Initialize classifier
RF = RandomForestClassifier(n_estimators = 100, oob_score = True, warm_start = True, n_jobs

# Fitting
RF = RF.fit(X_train, y_train)

y_pred = RF.predict(X_test)

# OOB
oob_error = 1- RF.oob_score_

# Performance
print(f'Out-of-bag error: {oob_error:.2f}')
print('')
print(classification_report(y_test, y_pred))
```

```
Out-of-bag error: 0.19

              precision    recall  f1-score   support

           0       0.76      0.65      0.70      2006
           1       0.82      0.89      0.86      3700

    accuracy                           0.81      5706
   macro avg       0.79      0.77      0.78      5706
weighted avg       0.80      0.81      0.80      5706
```

Considering it is only 40% of the dimensions and the Out-of-bag error increased by only 7%, this is pretty great

In [116]:

```
# PCA Transforming data
PCA_transform = PCA(n_components = 6)
X_PCA = PCA_transform.fit_transform(X_scaled)
```

In [117]:

```
# Trying Random Forest
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, f1_scc

# X and y
y = data['Particle'].replace('h', 0).replace('g',1)
X_PCA = pd.DataFrame(X_PCA)
X = X_PCA


# Initialize object
sss = StratifiedShuffleSplit(n_splits=1, test_size = 0.3)

# Create indexes
train_index, test_index = next(sss.split(X, y))

# Assign set
X_train, X_test = X.iloc[train_index], X.loc[test_index]
y_train, y_test = y.iloc[train_index], y.loc[test_index]

# Initialize classifier
RF = RandomForestClassifier(n_estimators = 100, oob_score = True, warm_start = True, n_jobs

# Fitting
RF = RF.fit(X_train, y_train)

y_pred = RF.predict(X_test)

# OOB
oob_error = 1- RF.oob_score_

# Performance
print(f'Out-of-bag error: {oob_error:.2f}')
print('')
print(classification_report(y_test, y_pred))
```

```
Out-of-bag error: 0.16

              precision    recall  f1-score   support

           0       0.81      0.73      0.77      2006
           1       0.86      0.91      0.88      3700

    accuracy                           0.84      5706
   macro avg       0.84      0.82      0.83      5706
weighted avg       0.84      0.84      0.84      5706
```

Pretty good results considering the dimensions are reduced by 40%

Clustering

It would be interesting to see whether a cluster algorithm with no pre-determined clusters could figure out that there exists 2 classes. Lets see

In [221]:

```
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [222]:

```
# Load data
index_name_list = ['fLength', 'fWidth', 'fSize', 'fConc', 'fConc1', 'fAsym', 'fM3Long', 'fM
data = pd.read_csv('magic04.data', header = None, names = index_name_list)
X = data[[x for x in data.columns if x not in 'Particle']]
```

In [223]:

```
# Scaling
SS = StandardScaler()
X_SS = SS.fit_transform(X)

# Cluster
dbscan = DBSCAN()
dbscan_model = dbscan.fit(X_SS)
clusters = pd.DataFrame(dbscan_model.fit_predict(X_SS))
data['dbscan'] = clusters
```

In [230]:

```
# Looking at the values
data['dbscan'].value_counts(normalize=True)
```

Out[230]:

```
-1     0.658149
 0     0.280599
 7     0.008675
 1     0.005100
 4     0.004942
         ...
 50    0.000158
 84    0.000158
 95    0.000158
 71    0.000158
 76    0.000105
Name: dbscan, Length: 121, dtype: float64
```

In [231]:

```
# Inspecting
(data[['Particle','dbscan']]
 .groupby(['dbscan','Particle'])
 .size()
 .to_frame()
 .rename(columns={0:'number'}))
```

Out[231]:

dbscan	Particle	number
-1	g	6561
	h	5957
0	g	4794
	h	543
1	g	73
...	...	...
117	h	5
118	g	5
	h	2
119	g	1
	h	3

188 rows × 1 columns

The results were quite poor. Perhaps there are no natural division. For further analysis, it would prove interesting to try out K-Means. Also, the data were divided with 65% of observations being gamma. This surely has some influence on its tendency to assign variables to that cluster. By some kind of sampling technique, oversampling, undersampling SMOTE, this could be negated to some extent.

End of notebook