MACHINE LEARNING
FINAL ASSIGNMENT
COURSE 4

The jupyter notebook with the code is at the end of the report

I. INTRODUCTION

i. BACKGROUND

The data I want to analyze for this last assignment is the CERN electron collision data. This data will be used to see if it is possible to cluster it. The dataframe I will describe in the next section is comprehensive of one hundred thousand entries. The final objective of this little project is to be able to efficiently cluster the various collision experiments, hence we will focus on clustering. I will try to cluster based on the energy of one of the 2 electrons.

ii. MAIN OBJECTIVE

The main objective of this exercise is to test how different clustering algorithms performs in a given dataset. We will probe models such as using the K-means model, Agglomerative Clustering and Mean Shift.

II. THE DATA

i. DESCRIPTION OF THE DATASET

The dataset I will use is comprehensive of 100000 entries collected from the CERN open database. This data is in a csv format and contains 19 features:

1. **Run:** The run number of the event
2. **Event:** The event number
3. **E1:** The total energy of the electron 1 (GeV)
4. **px1:** First component of the momemtum of the electron 1 (GeV)
5. **py1:** Second component of the momemtum of the electron 1 (GeV)
6. **pz1:** Third component of the momemtum of the electron 1 (GeV)
7. **pt1:** The transverse momentum of the electron 1 (GeV).
8. **eta1:** The pseudorapidity of the electron 1
9. **phi1:** The phi angle of the electron 1 (rad)
10. **Q1:** The charge of the electron 1
11. **E2:** The total energy of the electron 2 (GeV)
12. **px2:** First component of the momemtum of the electron 2 (GeV).
13. **py2:** Second component of the momemtum of the electron 2 (GeV)
14. **pz2:** Third component of the momemtum of the electron 2 (GeV)
15. **pt2:** The transverse momentum of the electron 2 (GeV)
16. **eta2:** The pseudorapidity of the electron 2
17. **phi2:** The phi angle of the electron 2 (rad)
18. **Q2:** The charge of the electron 2
19. **M:** The invariant mass of two electrons (GeV)

ii. DATA CLEANING & FEATURES ENGINEERING

The dataset is well presented. Reasonably, at first glance, all the features present can be important. However the '*Run*' and '*Event*' features are too specific and are not going to help in the clustering methods. I will remove these features from the dataset. Out of the 19 features, 4 are of type int and the rest are of type float. Running the dataset.info() showed that there are 85 missing values in feature '*M*'. Since 85 rows are not much compared to the 100k of the dataset I proceeded to eliminate them. Next thing I converted the *int* variables in type float. Subsequently, I found that there are 5 skewed features, *E1*, *E2*, *pt2*, *pt1* and *M*.

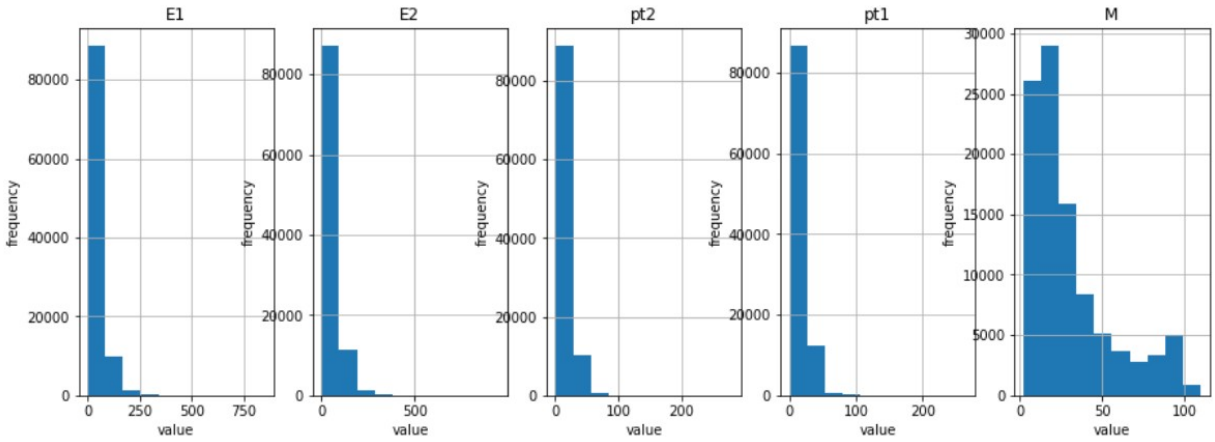


Figure 1. Histogram of the skewed features

To normalize their distribution I used the log transformation.

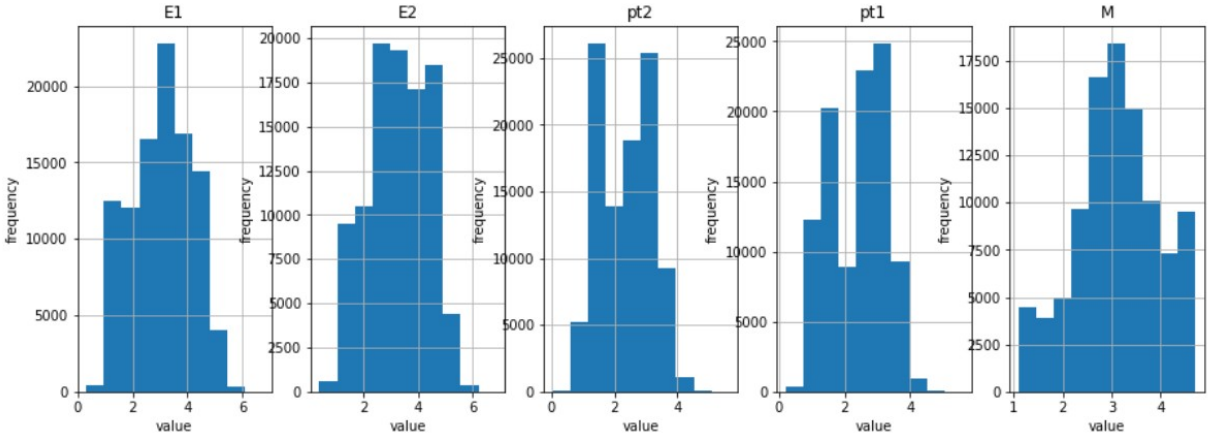


Figure 2. Histogram of the normalized features

The data was subsequently scaled using StandardScaler.

iii. DATA EXPLORATION

In order to analyze the data I calculate the correlation matrix and set to zero all the diagonal elements. After taking the absolute correlation coefficient I found the features that where there was the highest correlation. However, looking at the dataset, there is no feature that I can take to use a the clustering method on. I tried to to see the if I could use as target feature the energy of the electron 1. The E1 feature has as very similar frequency of negative and positive values. Therefore, I decided to calculate the correlation of every feature respect to E1.

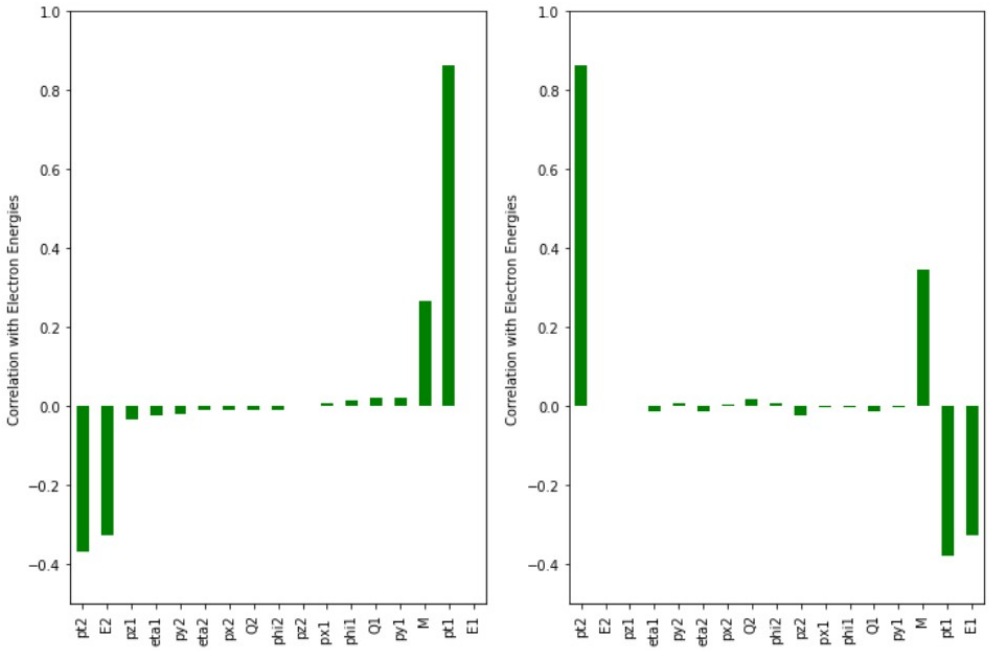


Figure 3. Plot of the correlations of E1 vs all other features

Subsequently, I created a new binary column with values either '*N*' or '*P*'. This column is based on the E1 features which is the energy of the electron 1 which, after scaling, goes from ~ -3 to +3. If the value of the E1 is below 0 then the value assigned is N for negative and if the values is above 0 the value assigned is P for positive. Third, I visualized the data with a *pairplot.* Now it is possible to see that applying a clustering method might be useful.

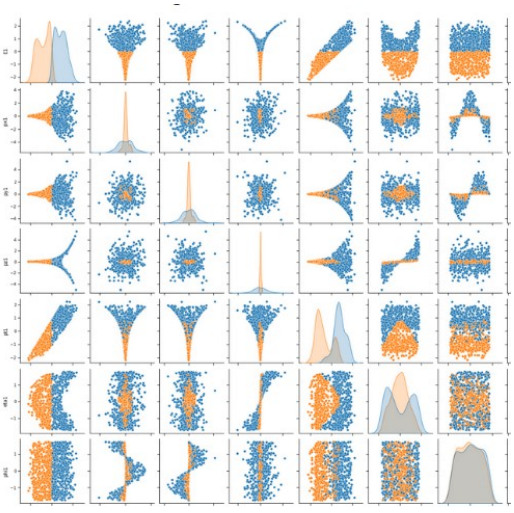


Figure 4. This is only the upper-right corner of the PairPlot between all the features because the full plot is too big. Yellow – E1<0, Blue - E1>0

III. MACHINE LEARNING

1. K-Means, Mini Batch

The first clustering method I will use is the K-means method. However, since the dataset was pretty big (100k rows and 19 columns) for the computational power I have at my disposal I decided to implement the Mini Bach Kmeans algorigthm. The computation time of K-means increases as the dataset size increases because it needs to store the whole dataset in memory. To bypass this issue, the Mini Bach Kmeans can reduce both computation time and memory usage. The main idea of this method is to use small random batches of data of fixed size. In this way it is possible to avoid the memory intensive usage of the standard K-means algorithm. Every iteration of the algorithm takes a new batch which is used to update the clusters. The process will be repeated until convergence is reached. In my case while the K-means algorithm would take tens of minutes to converge while Mini Batch K-means methods would take only a couple of seconds. I run the latter algorithm with a *for* loop updating every time the number of cluster considered and proceeded to calculate the inertia of all the implementations.

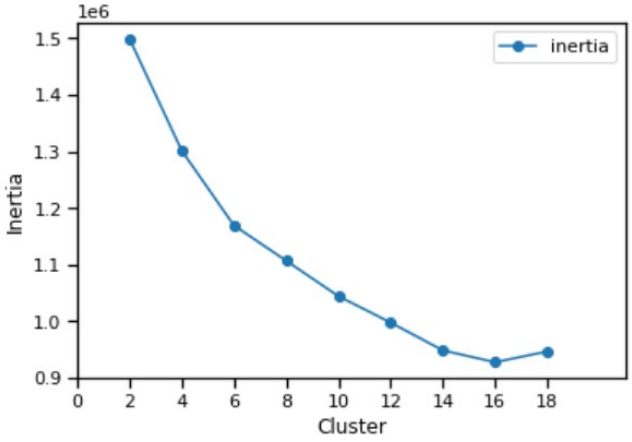


Figure 5. Plot of the inertia vs cluster number

In *figure 5* we can see the change in the inertia value respect to the cluster number. As we see, it is not very clear where is the elbow point in this graph. We can argue that maybe it is at 14 or 16 but after that point something unexpected happens. The inertia goes up again when it is supposed to get lower and lower as the number of clusters increases. However, I did a check on the side and after 18 the inertial goes down again. The Mini Batch algorithm will be implemented with 2 and with 16 clusters.

- **Mini Batch K-means** with **cluster=2, 16**

Here we can see that this method is creating 2 clusters each one of them has a higher component on one of the two possible classes ‘N’ or ‘P’. The cluster 0 is made of 81% of P and 19% of N while cluster 1 is made of 82% of N and 18% of P (Table 1).

CLUSTER	N/P RATIONS
0	81% P , 19% N
1	18% P, 82% N
AVERAGE	81.5%

Table 1. Cluster purity evaluation.

Lets see if we can do better by using 16 clusters. Using a higher number of clusters should improve the selectivity between N & P. Among the 16 clusters there are two clusters that performs poorly.

CLUSTER	N/P RATIONS
0	98% P, 2% N
1	90% P, 10% N
2	18% P, 82% N
3	75% P, 25% N
4	93% P, 7% N
5	18% P, 82% N
6	98% P, 2% N
7	17% P, 83% N
8	94% P, 6% N
9	98% P, 2% N
10	63% P, 37% N
11	45% P, 55% N
12	88% P, 12% N
13	14% P, 86% N
14	99% P, 1% N
15	4% P, 96% N
AVERAGE	~ 86.25%

Table 2. Cluster purity evaluation for the Mini batch method with 16 clusters.

In *table 2* are reported all relative purity of all the 16 clusters obtained with the Mini Batch k-means method. Among them cluster 10 and 11 are not very good while the rest are definitely better. The average cluster purity is of 86% compared to the 81% of the same method but with just 2 clusters. As expected, we see that considering more clusters does a better job. In the *table 3* below is reported the inertia value and best core of both implementations

n_clusters	INERTIA	GRID BEST SCORE
2	1499015	-300793.84
16	924176	-185269.94

Table 3. Inertia and Grid Score for both algorithms, with 2 and 16 clustres

- **Hierarchical clustering, WARD**

Now I want to see if there is difference using the agglomerative clustering which is a hierarchical method. For this kind of clustering, running it with different number of cluster is not necessary. However I just want to see what happens if when I choose 2 clusters or 16. Furthermore, I will use a sub-sample of the dataset comprehensive of 5000 randomly chosen entries from the main dataset for computational costs limitations.

Setting the number of cluster to 2 results in a overall higher cluster quality compared the previous K-means method. The cluster 0 is made of 80% of P and 20% of N while cluster 1 is made of 89% of N and 11% of P.

CLUSTER	N/P RATIONS
0	80% P , 20% N
1	11% P, 89% N
AVERAGE	85%

Table 4. Cluster segmentation evaluation.

The average cluster purity has an increment of 3.5% passing from 81.5 to 85%. Now lets see if using 16 cluster has the same kind of advantage or if there is any difference.

CLUSTER	N/P RATIONS
0	81% P, 19% N
1	84% P, 16% N
2	4% P, 96% N
3	63% P, 27% N
4	98% P, 2% N
5	30% P, 70% N
6	99% P, 1% N
7	8% P, 92% N
8	74% P, 26% N
9	11% P, 89% N
10	93% P, 7% N
11	18% P, 82% N

12	**81% P, 19% N**
13	**70% P, 30% N**
14	**98% P, 2% N**
15	**9% P, 91% N**
AVERAGE	**~ 85%**

Table 4. Cluster purity evaluation for the ward method with 16 clusters.

As we see in the table above, using 16 clusters doesn't have the same beneficial effect it had on the K-means algorithm. This is in line with was previously said that for this method doesn't matter the number of cluster specified. This method benefit of more flexibility when combining clusters of different shapes but it can not deal with nose very well. However, judging from the result it seems that there shouldn't be much noise in the data since the results are similar to the K-means.

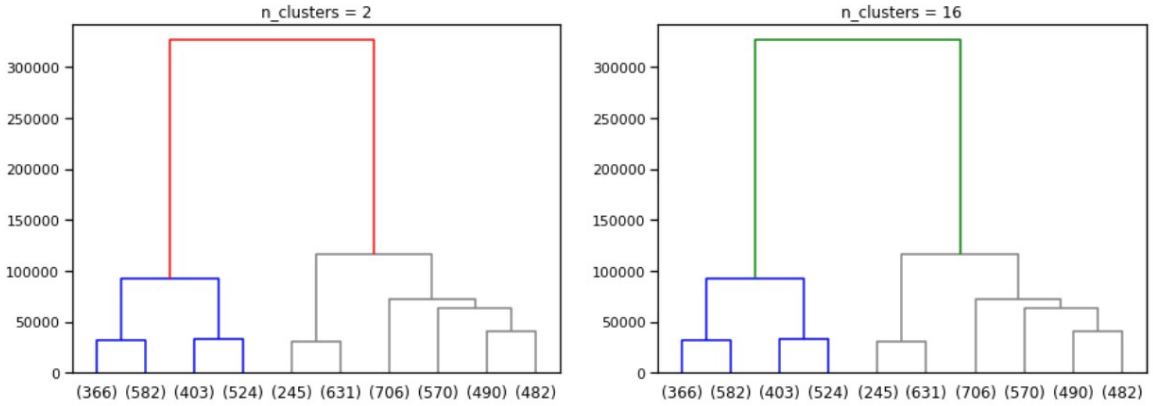


Figure 6. Dendograms of the same sub-sample of the dataset with the same random_state with number of cluster equal to 2 and 16

Again, looking at *figure 6*, changing the number of clusters, when using the Hierarchical clustering, it is not necessary. As we can see there is no difference between using 16 or 2 clusters because the results are the same. The height is the same in both situations.

- ## Mean Shift

Finally the last algorithm used is Mean Shift. I will use the *estimate_bandwidth* function from the *sklearn.cluster* library to find the best bandwidth for the situation at hand. This method does not scale well with big datasets. It is not possible to run such model with my machine when considering the full dataset. I will use a sub-sample comprehensive of 5000 random entries from the dataset as we did for the hierarchical method. This method as the advantage that it does not require to have the number of cluster selected by the user since it will find the number of cluster on its own. As previously said, the *estimate_bandwidth* gives values between 5 and 7 as the best choices for the different quartiles chosen. However looking at the results, it seems that this algorithm is not suited to cluster our data. It always gives us poorly clustered clusters.

CLUSTER	N/P RATIOS QUARTILE=0.2	N/P RATIOS QUARTILE=0.3
0	50%P, 50%N	50%P, 50%N
1	70%P, 30%N	100%P
2	87%P, 13%N	100%P
3		100%P
4		100%P
5		85%P, 15%N
Bandwidth	5.93	6.54

Table 5. Cluster segmentation evaluation. Quantile=0.2, 0.3, 0.4

Even though, it looks like there are many pure clusters in the case of the bandwidth equal to 6.54, these clusters are very small, comprehensive of a couple of points at most. The 99% of the total points are in the cluster 0. As we see this clustering method has failed. The under-performance of this method can be due to the fact that, even though it is a good algorithm to find a lot of clusters, it is not efficient when weird cluster shapes are present with a predilection of spherical shapes. Moreover, the only distance metric available is the euclidean distance which might not be a good metric to use in this case.

IV. SUMMARY, KEY FINDINGS & RECOMMENDED METHODS

Lets have a look at the comparison between all the method we have used in this assignment. However, it is not really possible to compare these method with one another since these are all unsupervised learning methods which means that we don't know what the result should have been, making it hard to define a score. Anyways, in order to have a general idea on the viability of these models, I decided to consider cluster purity as an accuracy metric.

Proceeding with the methods; we used the K-Means, this method has been the first used in order to try to find the best number of clusters for this dataset by relying on the elbow method. Unfortunately, in this case, the elbow method did not help us find unequivocally the best number of clusters. The values of the inertia calculated with 2 to 16 clusters goes down progressively without any major change in the slope. It is interesting to see that the value of the inertia after the number of cluster equal to 16, briefly goes up. This is an unexpected behavior since the inertia is supposed to go down with the increasing of the number of clusters no matter what. This method is the only one where I implemented GridSearch which I used, not to find the best parameters but to look at the best scores and make sure of two things. 1) That the behavior of the inertia increasing after 18 was just a fluke which was the case. 2) That the best found model was indeed the one with more clusters. The average cluster purity archived with the Mini Batch K-means algorithm when the number of cluster was equal to 2 is ~81.5%. When the number of cluster had been increased to 16 the average cluster purity increased by 3.5% from 81.5 to 85%. Even though the increased cluster selectivity is small, it shows that increasing the number of cluster in this method does indeed increase the clustering accuracy. The next algorithm that was used is the agglomerative clustering implemented with the '*ward*'

linkage. I would have loved to run a GridSearch for this method since it would have been nice to test all the different linkages available ('ward', 'complete', 'average', 'single') and find the best one. However, it is not possible since this model is missing a score attribute. Anyways, this method should be invariant to the choice of the number of clusters as *figure 6* shows. For the sake of experimentation I tried this method with both the number of cluster equal to 2 and to 16. Besides finding that increasing the number of clusters does not affect the outcome, we can see that this model has a similar average cluster purity to the Mini Batch K-means method indicating that these methods are both good to cluster this data. The third and last method used is the Mean Shift algorithm. This method had performed far worse then any other method used in this exercise. It identified a number of cluster ranging from 3 to 6 depending on the bandwidth used. In both cases the cluster with high purity accounted for single digit percentages of the total data. As previously said, even though this is an algorithm that preforms well when a lot of cluster can be found, it is also inefficient when weird cluster shapes are present and it can use only the euclidean distance.

V. SUGGESTIONS

Finding the best clustering method for this project was not obvious. Among the used method, the best one seems to be the K-means method. In future studies would be interesting to analyze more in dept which is the best number of cluster for the K-means algorithm and probing different distance metrics for the Hierarchical method. Also it would be interesting to see if DBSCAN would be a smart choice in this case since it can detect outliers can work with different densities and is good at finding different size clusters.

Final_course4

February 14, 2021

0.1 MACHINE LEARNING FINAL EXERCISE - COURSE 4

0.1.1 CERN Electron Collision Data

I will work with the CERN electron collision data. I think it will be interesting to see if we can cluster the results. I do not know how many cluster will be the optimum choice since it is not then obvious.

IMPORTING LIBRARIES

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import math
import matplotlib.pyplot as plt
```

LOADING THE DATA

[2]:
```
filepath = 'dielectron.csv'
data_main = pd.read_csv(filepath)
data_main.head()
```

[2]:

	Run	Event	E1	px1	py1	pz1	pt1	\
0	147115	366639895	58.71410	-7.31132	10.531000	-57.29740	12.82020	
1	147115	366704169	6.61188	-4.15213	-0.579855	-5.11278	4.19242	
2	147115	367112316	25.54190	-11.48090	2.041680	22.72460	11.66100	
3	147115	366952149	65.39590	7.51214	11.887100	63.86620	14.06190	
4	147115	366523212	61.45040	2.95284	-14.622700	-59.61210	14.91790	

	eta1	phi1	Q1	E2	px2	py2	pz2	pt2	\
0	-2.20267	2.17766	1	11.2836	-1.032340	-1.88066	-11.0778	2.14537	
1	-1.02842	-3.00284	-1	17.1492	-11.713500	5.04474	11.4647	12.75360	
2	1.42048	2.96560	1	15.8203	-1.472800	2.25895	-15.5888	2.69667	
3	2.21838	1.00721	1	25.1273	4.087860	2.59641	24.6563	4.84272	
4	-2.09375	-1.37154	-1	13.8871	-0.277757	-2.42560	-13.6708	2.44145	

	eta2	phi2	Q2	M
0	-2.344030	-2.072810	-1	8.94841
1	0.808077	2.734920	1	15.89300
2	-2.455080	2.148570	1	38.38770

```
3  2.330210  0.565865  -1   3.72862
4 -2.423700 -1.684810  -1   2.74718
```

[3]:
```
data_main.shape
```

[3]:
```
(100000, 19)
```

[4]:
```
data_main.dtypes
```

[4]:
```
Run        int64
Event      int64
E1       float64
px1      float64
py1      float64
pz1      float64
pt1      float64
eta1     float64
phi1     float64
Q1         int64
E2       float64
px2      float64
py2      float64
pz2      float64
pt2      float64
eta2     float64
phi2     float64
Q2         int64
M        float64
dtype: object
```

[5]:
```
data_main.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 19 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Run     100000 non-null  int64
 1   Event   100000 non-null  int64
 2   E1      100000 non-null  float64
 3   px1     100000 non-null  float64
 4   py1     100000 non-null  float64
 5   pz1     100000 non-null  float64
 6   pt1     100000 non-null  float64
 7   eta1    100000 non-null  float64
 8   phi1    100000 non-null  float64
 9   Q1      100000 non-null  int64
```

```
 10  E2      100000 non-null  float64
 11  px2     100000 non-null  float64
 12  py2     100000 non-null  float64
 13  pz2     100000 non-null  float64
 14  pt2     100000 non-null  float64
 15  eta2    100000 non-null  float64
 16  phi2    100000 non-null  float64
 17  Q2      100000 non-null  int64
 18  M       99915 non-null   float64
dtypes: float64(15), int64(4)
memory usage: 14.5 MB
```

0.2 Data Cleaning & Features Engineering

Revoming not useful columns: Run and Event

[6]:
```
data_main=data_main.drop(['Run', 'Event'], axis=1)
```

Trasforming all the int columns to float

[7]:
```
data_main[['Q1', 'Q2']]=data_main[['Q1', 'Q2']].astype(np.float)
```

Make a copy of the original dataframe

[8]:
```
data_clean=data_main.copy()
```

Lets see if we got NaN values

[9]:
```
k=0
cols_nan=[]
for x in data_clean:
    for e in data_clean[x]:
        if math.isnan(e):
            cols_nan.append(x)
            k=k+1
print(k)
print(cols_nan)
```

```
85
['M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M']
```

There are 85 NaN values all in the M column. Since 85 is small respect to the data size I will procees to remove these rows.

[10]:
```
for val, idx in zip(data_clean.M, data_clean.index):
    if math.isnan(val):
        #print('empty', 'index=', idx)
        data_clean=data_clean.drop(idx)
```

[11]:
```
data_clean=data_clean.reset_index(drop=True)
```

[12]:
```
data_clean.head()
```

[12]:

	E1	px1	py1	pz1	pt1	eta1	phi1	Q1	\
0	58.71410	-7.31132	10.531000	-57.29740	12.82020	-2.20267	2.17766	1.0	
1	6.61188	-4.15213	-0.579855	-5.11278	4.19242	-1.02842	-3.00284	-1.0	
2	25.54190	-11.48090	2.041680	22.72460	11.66100	1.42048	2.96560	1.0	
3	65.39590	7.51214	11.887100	63.86620	14.06190	2.21838	1.00721	1.0	
4	61.45040	2.95284	-14.622700	-59.61210	14.91790	-2.09375	-1.37154	-1.0	

	E2	px2	py2	pz2	pt2	eta2	phi2	Q2	\
0	11.2836	-1.032340	-1.88066	-11.0778	2.14537	-2.344030	-2.072810	-1.0	
1	17.1492	-11.713500	5.04474	11.4647	12.75360	0.808077	2.734920	1.0	
2	15.8203	-1.472800	2.25895	-15.5888	2.69667	-2.455080	2.148570	1.0	
3	25.1273	4.087860	2.59641	24.6563	4.84272	2.330210	0.565865	-1.0	
4	13.8871	-0.277757	-2.42560	-13.6708	2.44145	-2.423700	-1.684810	-1.0	

	M
0	8.94841
1	15.89300
2	38.38770
3	3.72862
4	2.74718

Lets check for skews

[13]:
```
skew_columns = (data_clean
                .skew()
                .sort_values(ascending=False))

skew_columns = skew_columns.loc[skew_columns > 0.75]
skew_columns
```

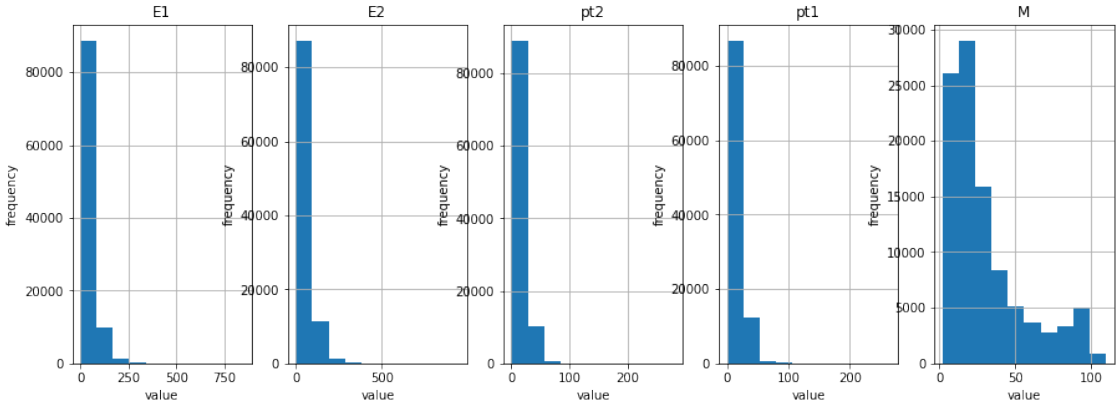
[13]:
```
E1      2.511899
E2      2.208866
pt2     1.993168
pt1     1.740526
M       1.300985
dtype: float64
```

Thesea the 5 most skewed columns, lets see them

[14]:
```
fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(1, 5, figsize=(15, 5))

cols={ax1:'E1', ax2:'E2', ax3:'pt2', ax4:'pt1', ax5:'M'}

for loc, name in cols.items():
    data_clean[name].hist(ax=loc)
    loc.set(title=name, ylabel='frequency', xlabel='value')
```



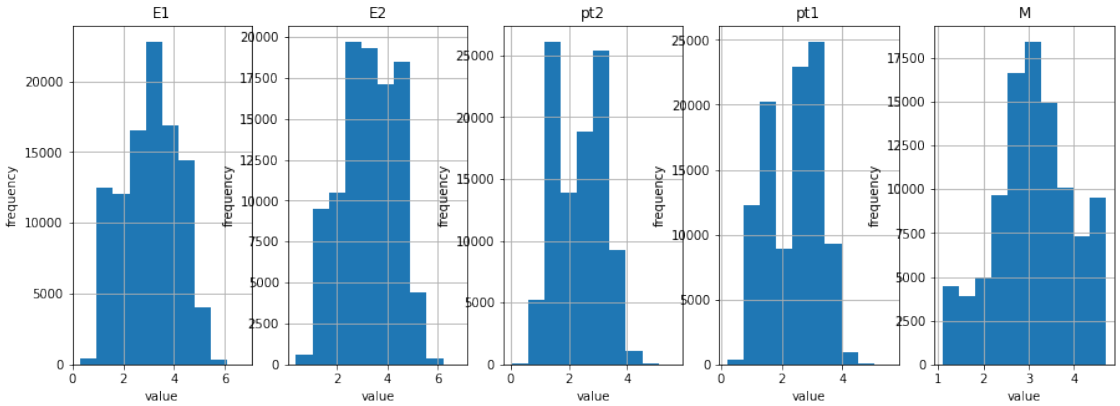
Lets try to normalize them by appliyng a log transformation

[15]:
```
for col in skew_columns.index.tolist():
    data_clean[col] = np.log1p(data_clean[col])
```

[16]:
```
fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(1, 5, figsize=(15, 5))

cols={ax1:'E1', ax2:'E2', ax3:'pt2', ax4:'pt1', ax5:'M'}

for loc, name in cols.items():
    data_clean[name].hist(ax=loc)
    loc.set(title=name, ylabel='frequency', xlabel='value')
```



[17]:
```
data=data_clean.copy()
```

0.3 Exploratory Data Analysis

Lets see the correlation matrix and also remove the diagonal values

[18]:
```
corr_mat = data.corr()

for x in range(data.shape[1]):
    corr_mat.iloc[x,x] = 0.0

corr_mat
```

[18]:

	E1	px1	py1	pz1	pt1	eta1	phi1	\
E1	0.000000	0.008008	0.021921	-0.034887	0.864005	-0.023103	0.015794	
px1	0.008008	0.000000	-0.008247	-0.012973	0.003024	-0.010436	-0.007438	
py1	0.021921	-0.008247	0.000000	-0.010181	0.012161	-0.011546	0.594070	
pz1	-0.034887	-0.012973	-0.010181	0.000000	-0.013774	0.771580	-0.011113	
pt1	0.864005	0.003024	0.012161	-0.013774	0.000000	0.012118	0.004675	
eta1	-0.023103	-0.010436	-0.011546	0.771580	0.012118	0.000000	-0.013213	
phi1	0.015794	-0.007438	0.594070	-0.011113	0.004675	-0.013213	0.000000	
Q1	0.021610	-0.005356	-0.002737	-0.002709	0.021642	-0.001716	0.000757	
E2	-0.327768	-0.003717	-0.003634	0.001873	-0.378039	-0.011625	-0.001409	
px2	-0.009627	-0.326424	0.002928	0.002605	-0.008990	-0.000781	0.003101	
py2	-0.019121	0.001333	-0.333596	-0.001770	-0.019837	-0.004052	-0.217749	
pz2	0.000811	-0.004636	0.000208	0.205665	0.010278	0.223923	0.000039	
pt2	-0.366516	-0.003720	-0.003101	0.005424	-0.361698	-0.009407	-0.002430	
eta2	-0.010455	-0.003695	-0.001602	0.229267	-0.000598	0.245638	0.000686	
phi2	-0.008054	0.002332	-0.222068	0.003797	-0.008119	0.002144	-0.135276	
Q2	-0.009018	0.005813	0.003474	0.000515	-0.010257	0.002978	-0.001807	
M	0.267581	0.006446	0.003235	0.012680	0.312226	0.018530	-0.000818	

	Q1	E2	px2	py2	pz2	pt2	eta2	\
E1	0.021610	-0.327768	-0.009627	-0.019121	0.000811	-0.366516	-0.010455	
px1	-0.005356	-0.003717	-0.326424	0.001333	-0.004636	-0.003720	-0.003695	
py1	-0.002737	-0.003634	0.002928	-0.333596	0.000208	-0.003101	-0.001602	
pz1	-0.002709	0.001873	0.002605	-0.001770	0.205665	0.005424	0.229267	
pt1	0.021642	-0.378039	-0.008990	-0.019837	0.010278	-0.361698	-0.000598	
eta1	-0.001716	-0.011625	-0.000781	-0.004052	0.223923	-0.009407	0.245638	
phi1	0.000757	-0.001409	0.003101	-0.217749	0.000039	-0.002430	0.000686	
Q1	0.000000	-0.013007	-0.003541	0.003521	-0.005276	-0.011953	-0.004422	
E2	-0.013007	0.000000	0.004552	0.006739	-0.023523	0.863053	-0.012774	
px2	-0.003541	0.004552	0.000000	0.003971	0.004169	0.000484	-0.002338	
py2	0.003521	0.006739	0.003971	0.000000	-0.024796	0.002573	-0.021905	
pz2	-0.005276	-0.023523	0.004169	-0.024796	0.000000	-0.009571	0.764610	

```
pt2  -0.011953  0.863053  0.000484  0.002573 -0.009571  0.000000  0.009858
eta2 -0.004422 -0.012774 -0.002338 -0.021905  0.764610  0.009858  0.000000
phi2 -0.001455  0.009453 -0.004872  0.579332 -0.016683  0.004546 -0.022888
Q2   -0.140376  0.016827 -0.007504 -0.001635  0.000183  0.015808 -0.002671
M     0.005291  0.346985 -0.005555 -0.008664  0.012530  0.338312  0.016564

          phi2        Q2         M
E1   -0.008054 -0.009018  0.267581
px1   0.002332  0.005813  0.006446
py1  -0.222068  0.003474  0.003235
pz1   0.003797  0.000515  0.012680
pt1  -0.008119 -0.010257  0.312226
eta1  0.002144  0.002978  0.018530
phi1 -0.135276 -0.001807 -0.000818
Q1   -0.001455 -0.140376  0.005291
E2    0.009453  0.016827  0.346985
px2  -0.004872 -0.007504 -0.005555
py2   0.579332 -0.001635 -0.008664
pz2  -0.016683  0.000183  0.012530
pt2   0.004546  0.015808  0.338312
eta2 -0.022888 -0.002671  0.016564
phi2  0.000000  0.001785 -0.003099
Q2    0.001785  0.000000  0.000348
M    -0.003099  0.000348  0.000000
```

[19]:
```
corr_mat.abs().idxmax()
```

[19]:
```
E1       pt1
px1      px2
py1     phi1
pz1     eta1
pt1       E1
eta1     pz1
phi1     py1
Q1         Q2
E2       pt2
px2     px1
py2     phi2
pz2     eta2
pt2       E2
eta2     pz2
phi2     py2
Q2         Q1
M          E2
dtype: object
```

I want to see the most important features that are correlated to the total energy of the electrons

[20]:
```
#For electron 1
y = (data['E1'])
features = list(data.loc[:, data.columns != 'E1'])
corr_E1 = data[features].corrwith(y)
corr_E1.sort_values(inplace=True)

#For electron 2
y = (data['E2'])
features = list(data.loc[:, data.columns != 'E2'])
corr_E2 = data[features].corrwith(y)
corr_E2.sort_values(inplace=True)
```

[21]:
```
corr_E1_df=pd.DataFrame(corr_E1, columns=['Corr to E1'])
corr_E2_df=pd.DataFrame(corr_E2, columns=['Corr to E2'])
```

[22]:
```
electrons_cors=pd.concat([corr_E1_df, corr_E2_df], axis=1)
```

[23]:
```
electrons_cors
```

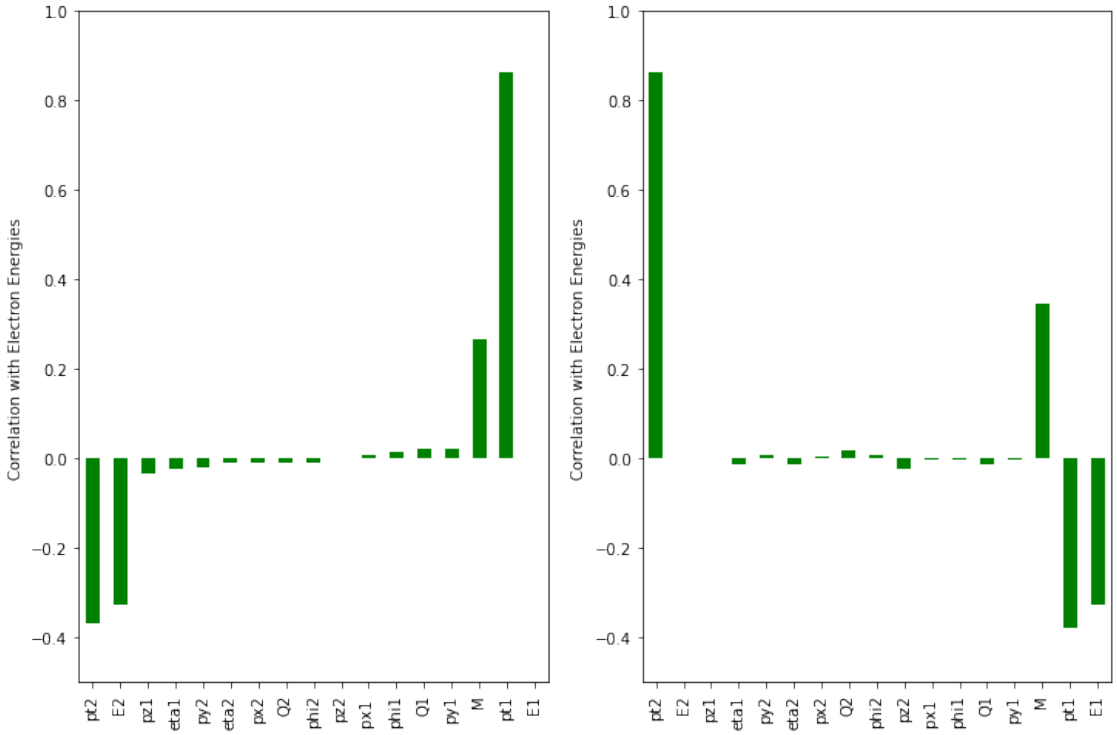
[23]:

	Corr to E1	Corr to E2
pt2	-0.366516	0.863053
E2	-0.327768	NaN
pz1	-0.034887	0.001873
eta1	-0.023103	-0.011625
py2	-0.019121	0.006739
eta2	-0.010455	-0.012774
px2	-0.009627	0.004552
Q2	-0.009018	0.016827
phi2	-0.008054	0.009453
pz2	0.000811	-0.023523
px1	0.008008	-0.003717
phi1	0.015794	-0.001409
Q1	0.021610	-0.013007
py1	0.021921	-0.003634
M	0.267581	0.346985
pt1	0.864005	-0.378039
E1	NaN	-0.327768

[24]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 15))

cols={ax1: 'Corr to E1', ax2: 'Corr to E2'}

for loc, name in cols.items():
    electrons_cors[name].plot(kind='bar', color='g', figsize=(12,8), ax=loc)
    loc.set(ylim=[-.5, 1], ylabel='Correlation with Electron Energies')
```



[25]:
```
c=0
for x in data.E1:
    if (0 < x < 3):
        c=c+1
print(c)
```

45309

Lets scale the data

[26]:
```
float_columns = [x for x in data.columns if x not in ['N or P']]
```

[27]:
```
from sklearn.preprocessing import StandardScaler
```

[28]:
```
sc = StandardScaler()
data[float_columns] = sc.fit_transform(data[float_columns])

data.head()
```

[28]:

	E1	px1	py1	pz1	pt1	eta1	phi1	\
0	0.926475	-0.555376	0.767948	-1.080643	0.260118	-1.462122	1.197923	
1	-0.963783	-0.319778	-0.056630	-0.069805	-0.889226	-0.659281	-1.680937	
2	0.182393	-0.866324	0.137924	0.469417	0.157263	1.015044	1.635789	
3	1.023809	0.550089	0.868589	1.266348	0.361132	1.560573	0.547491	

```
4  0.967590  0.210077 -1.098802 -1.125480  0.426031 -1.387652 -0.774406

        Q1        E2       px2       py2       pz2       pt2      eta2  \
0  1.00551 -0.702159 -0.078361 -0.152343 -0.154357 -1.435744 -1.306268
1 -0.99452 -0.345545 -0.891835  0.373434  0.212457  0.312955  0.506693
2  1.00551 -0.415011 -0.111906  0.161937 -0.227761 -1.244323 -1.370139
3  1.00551 -0.012690  0.311592  0.187557  0.427112 -0.701753  1.382160
4 -0.99452 -0.526548 -0.020892 -0.193715 -0.196551 -1.329116 -1.352091

       phi2        Q2         M
0 -1.152781 -0.995775 -0.967123
1  1.493217  1.004243 -0.336109
2  1.170511  1.004243  0.672769
3  0.299449 -0.995775 -1.853520
4 -0.939240 -0.995775 -2.130756
```

I want to add column where the outcome is either True or False base on the sign of the electron energy. This is just for the purpose of the exercies so I can visually see the presence of clusters.

[29]:
```
new_col=[]
for x in data.E1:
    if x < 0:
        new_col.append('N')
    else:
        new_col.append('P')
```

[30]:
```
data['N or P']=new_col
```

[31]:
```
data.head()
```

[31]:
```
          E1      px1        py1       pz1       pt1      eta1      phi1  \
0  0.926475 -0.555376  0.767948 -1.080643  0.260118 -1.462122  1.197923
1 -0.963783 -0.319778 -0.056630 -0.069805 -0.889226 -0.659281 -1.680937
2  0.182393 -0.866324  0.137924  0.469417  0.157263  1.015044  1.635789
3  1.023809  0.550089  0.868589  1.266348  0.361132  1.560573  0.547491
4  0.967590  0.210077 -1.098802 -1.125480  0.426031 -1.387652 -0.774406

        Q1        E2       px2       py2       pz2       pt2      eta2  \
0  1.00551 -0.702159 -0.078361 -0.152343 -0.154357 -1.435744 -1.306268
1 -0.99452 -0.345545 -0.891835  0.373434  0.212457  0.312955  0.506693
2  1.00551 -0.415011 -0.111906  0.161937 -0.227761 -1.244323 -1.370139
3  1.00551 -0.012690  0.311592  0.187557  0.427112 -0.701753  1.382160
4 -0.99452 -0.526548 -0.020892 -0.193715 -0.196551 -1.329116 -1.352091

       phi2        Q2         M N or P
0 -1.152781 -0.995775 -0.967123      P
1  1.493217  1.004243 -0.336109      N
```

```
2  1.170511  1.004243  0.672769      P
3  0.299449 -0.995775 -1.853520      P
4 -0.939240 -0.995775 -2.130756      P
```

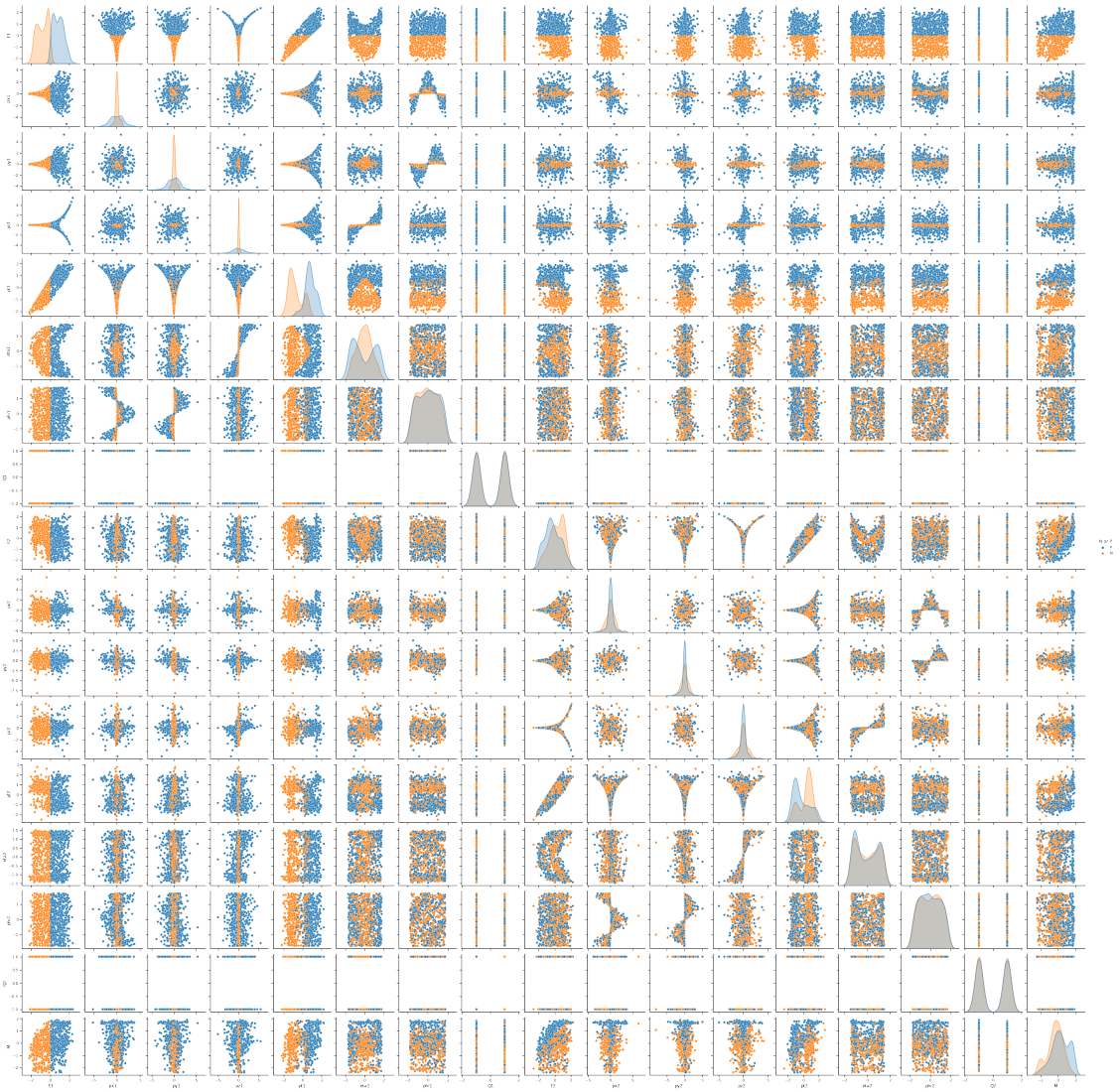
Lets see if we can find some clusters

I will define a sample 'sam' so the plotting will take less time

[32]:
```
sam=data.sample(n=1000, random_state=42)
```

[33]:
```
sns.set_context('notebook')
sns.pairplot(sam, hue='N or P')
```

[33]:
```
<seaborn.axisgrid.PairGrid at 0x7fe0365656a0>
```



0.4 Machine Learnig

[67]:
```
from sklearn.cluster import MiniBatchKMeans, MeanShift,␣
↪AgglomerativeClustering, estimate_bandwidth
from sklearn.metrics import mean_squared_error, r2_score
```

[97]:
```
data_calc=data.drop(['N or P'], axis=1)
```

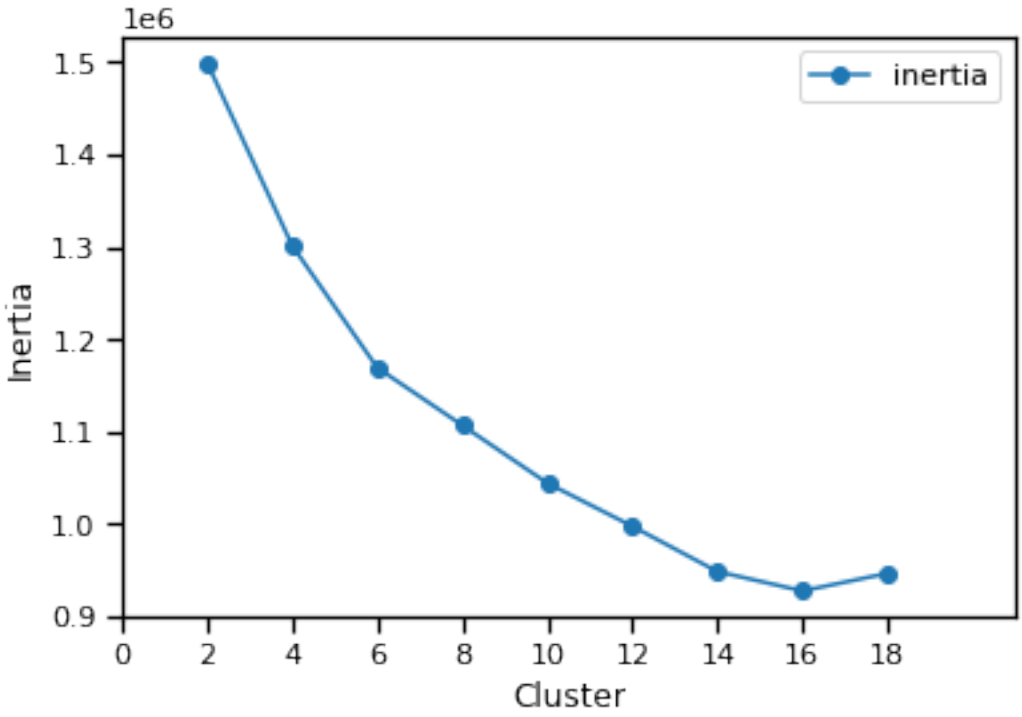
[98]:
```
mbkm_list = list()

for clust in range(2,20,2):
    mbkm = MiniBatchKMeans(n_clusters=clust, random_state=42)
    mbkm = mbkm.fit(data_calc)

    mbkm_list.append(pd.Series({'clusters': clust,
                                'inertia': mbkm.inertia_,
                                'model': mbkm}))
```

[99]:
```
plot_data = (pd.concat(mbkm_list, axis=1)
             .T
             [['clusters','inertia']]
             .set_index('clusters'))

ax = plot_data.plot(marker='o',ls='-')
ax.set_xticks(range(0,20,2))
ax.set_xlim(0,21)
ax.set(xlabel='Cluster', ylabel='Inertia');
### END SOLUTION
```



0.4.1 MiniBatchKmeans With Gridsearch

[100]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
```

[101]:
```
MiniBatchKMeans().get_params().keys()
```

[101]:
```
dict_keys(['batch_size', 'compute_labels', 'init', 'init_size', 'max_iter',
'max_no_improvement', 'n_clusters', 'n_init', 'random_state',
'reassignment_ratio', 'tol', 'verbose'])
```

[144]:
```
estimator_mbkm = Pipeline([("Kmeans", MiniBatchKMeans())])

params = {
    'Kmeans__n_clusters': [1, 2, 3, 4, 5, 6, 10, 12, 16],
    'Kmeans__batch_size': [100, 200, 300, 400, 500],
    'Kmeans__random_state' : [42]
}

grid_mbkm = GridSearchCV(estimator_mbkm, params)
grid_mbkm.fit(data_calc)
```

[144]:
```
GridSearchCV(estimator=Pipeline(steps=[('Kmeans', MiniBatchKMeans())]),
             param_grid={'Kmeans__batch_size': [100, 200, 300, 400, 500],
                         'Kmeans__n_clusters': [1, 2, 3, 4, 5, 6, 10, 12, 16],
                         'Kmeans__random_state': [42]})
```

[145]:
```
grid_mbkm.best_score_, grid_mbkm.best_params_
```

[145]:
```
(-185269.9437026138,
 {'Kmeans__batch_size': 200,
  'Kmeans__n_clusters': 16,
  'Kmeans__random_state': 42})
```

[121]:
```
grid_mbkm.best_estimator_
```

[121]:
```
Pipeline(steps=[('Kmeans',
                 MiniBatchKMeans(batch_size=400, n_clusters=2,
                                 random_state=42))])
```

[118]:
```
#predict_mbkm = grid_mbkm.predict(data_calc)
```

0.4.2 Mini Batch Kmeans with 2 clusters

[146]:
```
mbkm2 = MiniBatchKMeans(n_clusters=2, random_state=42, batch_size=100)
mbkm2 = mbkm2.fit(data_calc)

data['kmeans'] = mbkm2.predict(data_calc)
```

[147]:
```
(data[['N or P','kmeans']]
 .groupby(['kmeans','N or P'])
 .size()
 .to_frame()
 .rename(columns={0:'number'}))
```

[147]:

		number
kmeans	N or P	
0	N	9901
	P	42958
1	N	38437
	P	8619

0.4.3 Mini Batch Kmeans with 16 Clusters

[148]:
```
mbkm16 = MiniBatchKMeans(n_clusters=16, random_state=42, batch_size=200)
mbkm16 = mbkm16.fit(data_calc)

data['kmeans16'] = mbkm16.predict(data_calc)
```

[149]:
```
(data[['N or P','kmeans16']]
 .groupby(['kmeans16','N or P'])
 .size()
 .to_frame()
 .rename(columns={0:'number'}))
```

[149]:

kmeans16	N or P	number
0	N	116
	P	6164
1	N	416
	P	4046
2	N	4897
	P	1042
3	N	1592
	P	4627
4	N	7518
	P	1406
5	N	6738
	P	346
6	N	45
	P	2245
7	N	6678
	P	1384
8	N	335
	P	5613
9	N	102
	P	6434
10	N	3365
	P	5730
11	N	4678
	P	3710
12	N	311
	P	2280
13	N	5587
	P	898
14	N	10
	P	5434
15	N	5950
	P	218

I knew this was not a good method, in this situation we have clusters of different shapes and different sizes, I will try to use the agglomerative cluster method to compare it to the k-means.

[150]:
```
mbkm2.inertia_, mbkm16.inertia_
```

[150]:
```
(1499015.0420968777, 924176.8079526126)
```

0.4.4 Agglomerative Clustering WARD

[ ]:
```
from scipy.cluster import hierarchy
```

[177]:
```
sam_calc=sam.drop(['N or P'], axis=1)
```

WITH 2 CLUSTERS

[191]:
```
agc2 = AgglomerativeClustering(n_clusters=2, linkage='ward',␣
 ↪compute_full_tree=True)
#ag = ag.fit(sam_calc)
sam['agglom2'] = agc2.fit_predict(sam_calc)
```

[192]:
```
(sam[['N or P','agglom2']]
 .groupby(['agglom2','N or P'])
 .size()
 .to_frame()
 .rename(columns={0:'number'}))
```

[192]:

		number
agglom2	N or P	
0	N	1598
	P	999
1	N	847
	P	1556

WITH 16 CLUSTERS

[193]:
```
agc16 = AgglomerativeClustering(n_clusters=16, linkage='ward',␣
 ↪compute_full_tree=True)
#ag = ag.fit(sam_calc)
sam['agglom16'] = agc16.fit_predict(sam_calc)
```

[194]:
```
(sam[['N or P','agglom16']]
 .groupby(['agglom16','N or P'])
 .size()
 .to_frame()
 .rename(columns={0:'number'}))
```

[194]:

		number
agglom16	N or P	
0	N	27
	P	122
1	N	6
	P	410
2	N	38
	P	191
3	N	82

```
          P          356
4         N           99
          P           41
5         N          443
          P           39
6         N          251
          P          438
7         N            5
          P          217
8         N            1
          P          121
9         N          359
          P           44
10        N          320
          P           34
11        N          203
          P           42
12        N           72
          P          208
13        N           82
          P          193
14        N          451
          P           17
15        N            6
          P           82
```

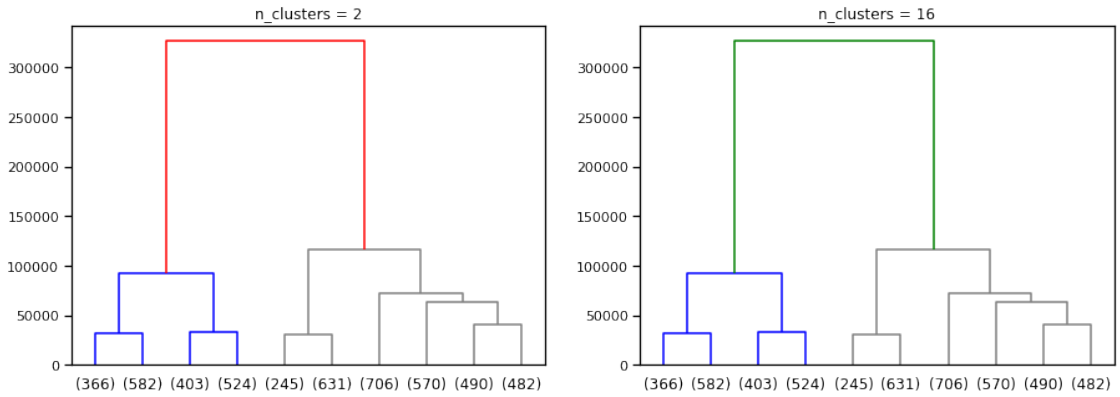
[227]:
```
Z1 = hierarchy.linkage(agc2.children_, method='ward')
Z2 = hierarchy.linkage(agc16.children_, method='ward')

fig, (ax1, ax2) = plt.subplots(1,2, figsize=(15,5))

den1 = hierarchy.dendrogram(Z1, orientation='top',
                            p=10, truncate_mode='lastp',
                            show_leaf_counts=True, ax=ax1,
                            above_threshold_color='red')

den2 = hierarchy.dendrogram(Z2, orientation='top',
                            p=10, truncate_mode='lastp',
                            show_leaf_counts=True, ax=ax2,
                            above_threshold_color='green')

ax1.set_title('n_clusters = 2');
ax2.set_title('n_clusters = 16');
```



As we can see there is no difference between using 16 or 2 cluster because the hight is the same. Also the heigh can be in a similar way used as the inertia with k-means.

0.4.5 MEANS SHIFT

Lets try with meanshift, the number of sample is the sam which has only 5000 rows and the bandwidth will found through the estimate_bandwidth function.

[231]:
```
MeanShift().get_params().keys()
```

[231]:
```
dict_keys(['bandwidth', 'bin_seeding', 'cluster_all', 'max_iter',
'min_bin_freq', 'n_jobs', 'seeds'])
```

[247]:
```
bw = estimate_bandwidth(sam_calc, quantile=0.1 , n_jobs=-1)
```

[248]:
```
bw
```

[248]:
```
5.131676115548401
```

[249]:
```
ms = MeanShift(bandwidth=bw)
ms = ms.fit(sam_calc)
sam['MeanShift'] = ms.fit_predict(sam_calc)
```

[250]:
```
(sam[['N or P','MeanShift']]
 .groupby(['MeanShift','N or P'])
 .size()
 .to_frame()
 .rename(columns={0:'number'}))
```

[250]:

		number
MeanShift	N or P	
0	N	2016
	P	2124
1	N	2

```
         P          16
2        P          19
3        P           1
4        P           6
5        N         425
         P         378
6        P           2
7        N           2
         P           9
```

[ ]: