In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy
import statsmodels as stm

from sklearn.datasets import load_breast_cancer, load_iris
from sklearn.neighbors import LocalOutlierFactor
from sklearn.ensemble import IsolationForest

%matplotlib inline
```

In [2]:
```
pd.set_option('display.max_rows',151)
```

Datasets

In [3]:
```
cancer_dataset, iris_dataset = load_breast_cancer(), load_iris()
```

1. Cancer Dataset

Segregating Features and Labels

In [4]:
```
X_cancer_df = pd.DataFrame(cancer_dataset.data, columns=cancer_dataset.feature_names
y_cancer_df = pd.DataFrame(cancer_dataset.target, columns=['Label'])
```

In [5]:
```
X_cancer_df.shape
```

Out[5]:
```
(569, 30)
```

In [6]:
```
cancer_dataset.target_names
```

Out[6]:
```
array(['malignant', 'benign'], dtype='<U9')
```

In [7]:
```
y_cancer_df.shape, y_cancer_df.value_counts()
```

Out[7]:
```
((569, 1),
 Label
 1        357
 0        212
 dtype: int64)
```

2. Iris Dataset

Segregating Features and Labels

In [8]:
```
X_iris_df = pd.DataFrame(iris_dataset.data,columns=iris_dataset.feature_names)
y_iris_df = pd.DataFrame(iris_dataset.target,columns=['Label'])
```

In [9]:
```
X_iris_df.shape, X_iris_df.head()
```

Out[9]:
```
((150, 4),
    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
 0                5.1               3.5                1.4               0.2
 1                4.9               3.0                1.4               0.2
 2                4.7               3.2                1.3               0.2
 3                4.6               3.1                1.5               0.2
 4                5.0               3.6                1.4               0.2)
```

In [10]:
```
iris_dataset.target.shape, iris_dataset.target_names
```

```
((150,), array(['setosa', 'versicolor', 'virginica'], dtype='<U10'))
```

Out[10]:

In [11]:
```
y_iris_df.shape, y_iris_df.value_counts()
```

Out[11]:
```
((150, 1),
 Label
 2        50
 1        50
 0        50
 dtype: int64)
```

Understanding Isolation Forest

In [12]:
```
from IPython.display import Image
from IPython.core.display import HTML

# Setting Images Path
# Images Source :: Extended Isolation Forest Github :: https://github.com/sahandha/e
PATH = "E:\STUDY\PROJECTS\AAIC_Practice\MODULES\Module_3\Module_3_Outliers_Detection
```

How Isolation Forest Works?

1. It is an unsupervised machine learning algorithm which is fundamentally based on the decision trees and build the ensemble forest.

2. The base estimator in Isolation Forest is Randomized Tree that subsamples the dataset in both row and feature wise.

3. It selects one feature randomly at a time to make a split or cut the branches and uses the random threshold value that exists b/w the range of min(feature value) & max(feature value).

4. The concept of impurity reduction totally followed here just like the normal decision trees with the intent to eliminate the outliers or anomaly very early in the split.

5. The assumption that it takes is that outliers or anomalies are the points which are away from the normal data points and usually isolated easily thus in very few initial splits we can find the outliers with the randomized approach at row, column and threshold level.

In [13]:
```
Image(filename = PATH + "1_IF_Random_Splits.jpg", width=700, height=700)
```

Out[13]:

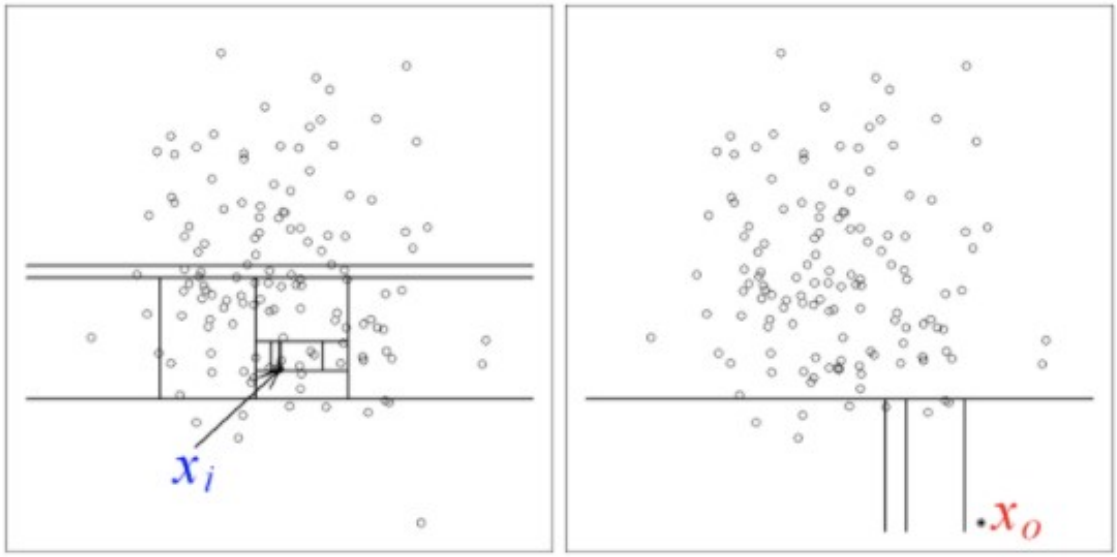


Figure 1 Identifying normal vs. abnormal observations

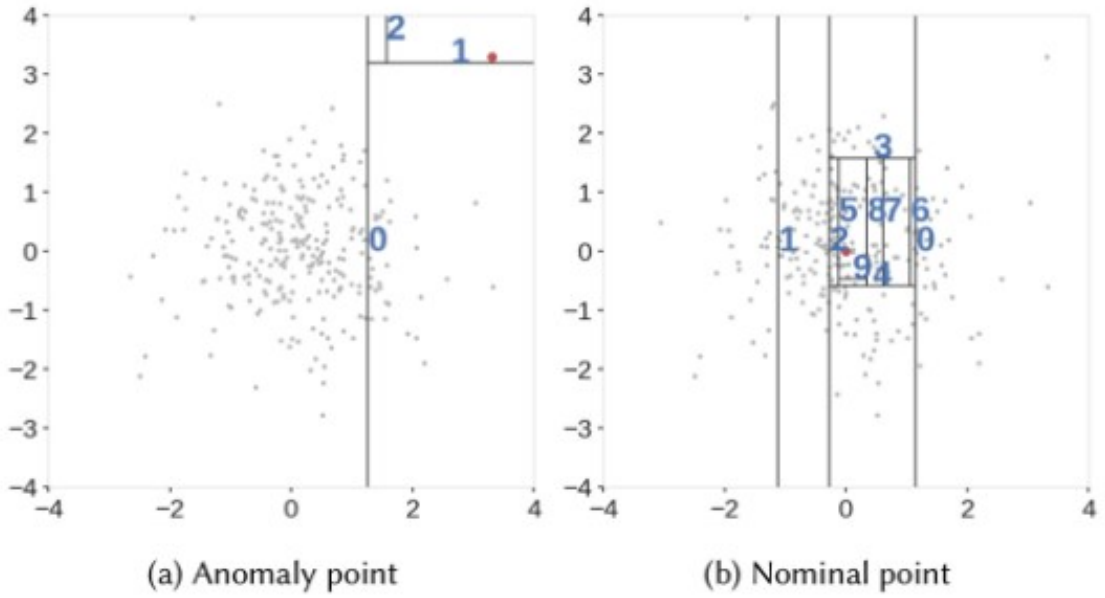
As shown in the above figure:

- **In the left image many splits are required to segregate the normal data point or,**
- **In other words we can say that we need traverse till the end of tree depth to find this element or,**
- **We can say that this Xi was common in so many nodes(from root to last decision node).**
- **On the other hand, in the right image the point Xo is an outlier which was segregated with only initial 3 splits.**

The concept of Isolation Forest is that if we are going deep into the tree to find an element then such an element has very less chance of being an Outlier, because its shortest distance or path in the tree is large thus difficult to segregate from other points. Hence, it be declared as Inlier.

In [14]:
```
Image(filename = PATH + "2_IF_Random_Partition_Algo.jpg", width=800, height=800)
```

Out[14]:



Similar things are shown in the above figure where in the left with only 3 splits outlier been found(marked as red). Whereas, in the right red point is in the center of the density of points thus requiring 10 splits which means the anamoly score will be less thus no clear signs of distinct anamolies.

In [15]:
```
Image(filename = PATH + "7_IF_Tree_Forest.jpg", width=800, height=800)
```

Out[15]:

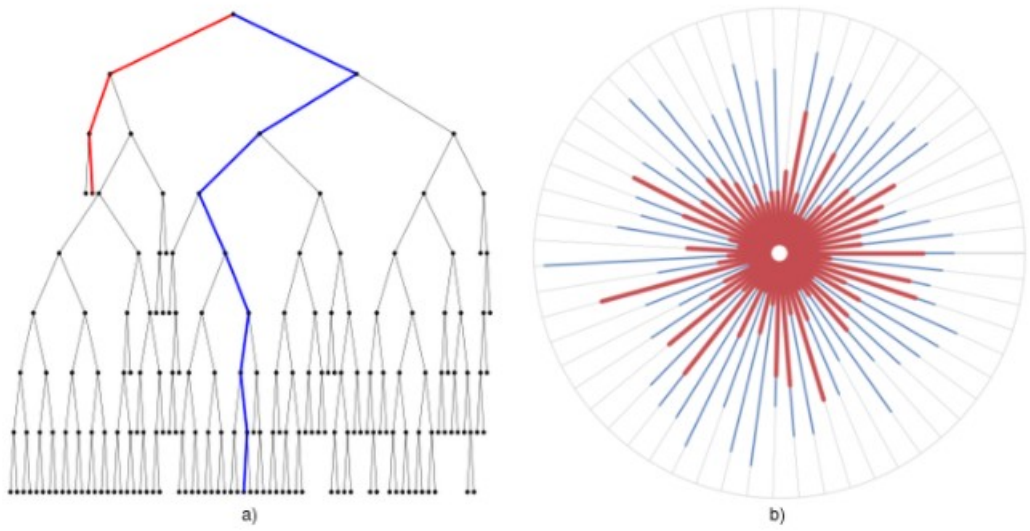


Figure 3: a) Shows an example tree formed from the example data while b) shows the forest generated where each tree is represented by a radial line from the center to the outer circle. Anomalous points (shown in red) are isolated very quickly,which means they reach shallower depths than nominal points (shown in blue).

The above image is very nice in depicting the classification of anamolies; the red color shows the shorter depth of isolated outliers whereas the blue color shows the depth of normal point.

In [16]:
```
Image(filename = PATH + "8_IF_Splitting_of_Data_while_constructing_one_tree.jpg", wi
```

Out[16]:

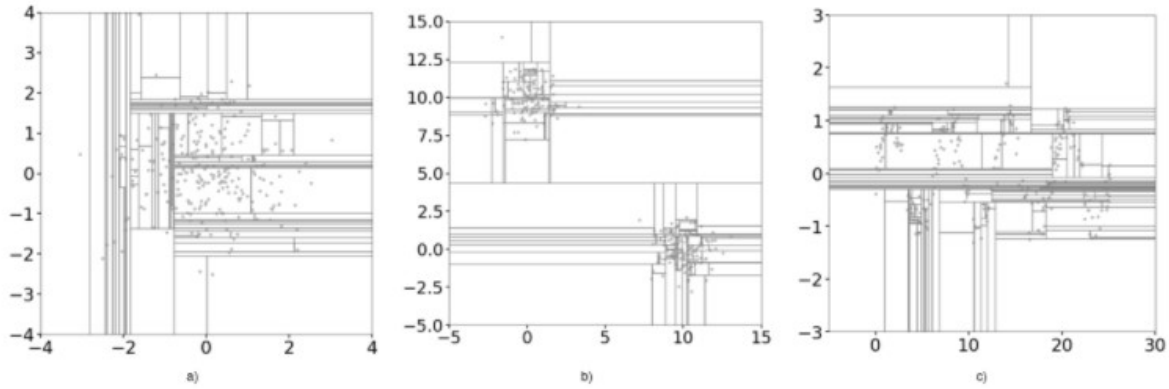


Figure 4: Splitting of data in the domain during the process of construction of one tree.

In the above image this is how the Isolation Forest splits or divides the data; as it forms the axis parallel lines for creating the branches in the trees; data values smaller than randomly selected threshold value goes into branch and others into right branch.

This random splitting which is the power of Isolation Forest is also the main of bias because of these axis parallel lines regions where threse are no points shows the artifical less anomaly scores which is not correct. (Explained in problems)

Isolation Forest Anomaly Score Formula

In [17]:
```
Image(filename = PATH + "3_IF_Anamoly_Score_Formula.jpg", width=700, height=700)
```

Out[17]:

In the case of Isolation Forest, anomaly score is defined as:

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}}$$

where $h(x)$ is the path length of observation x, $c(n)$ is the average path length of unsuccessful search in a Binary Search Tree and n is the number of external nodes. More on the anomaly score and its components can be read in [1].

Each observation is given an anomaly score and the following decision can be made on its basis:

- A score close to 1 indicates anomalies
- Score much smaller than 0.5 indicates normal observations
- If all scores are close to 0.5 then the entire sample does not seem to have clearly distinct anomalies

The anomaly score is created on the basis of

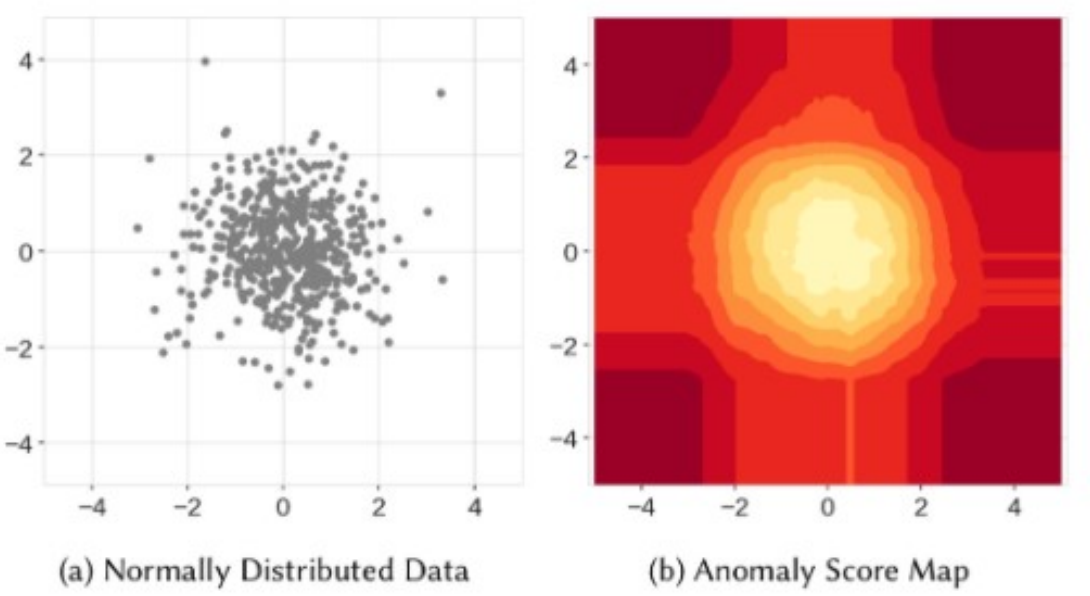
all trees in the forest and the depth the point reaches in these trees.

Problems in Isolation Forest

Here, we will talk about the problems or disadvantages or biasness of Isolation forest.

In [18]:
```
Image(filename = PATH + "4_IF_Scoremap_Gauss_Data.jpg", width=700, height=700)
```

Out[18]:



The above left image is the gaussian distributed data, now as per the understanding of normal distributions there shouldn't be any outliers and the anomaly score should increase as we move away from the center in the right image.

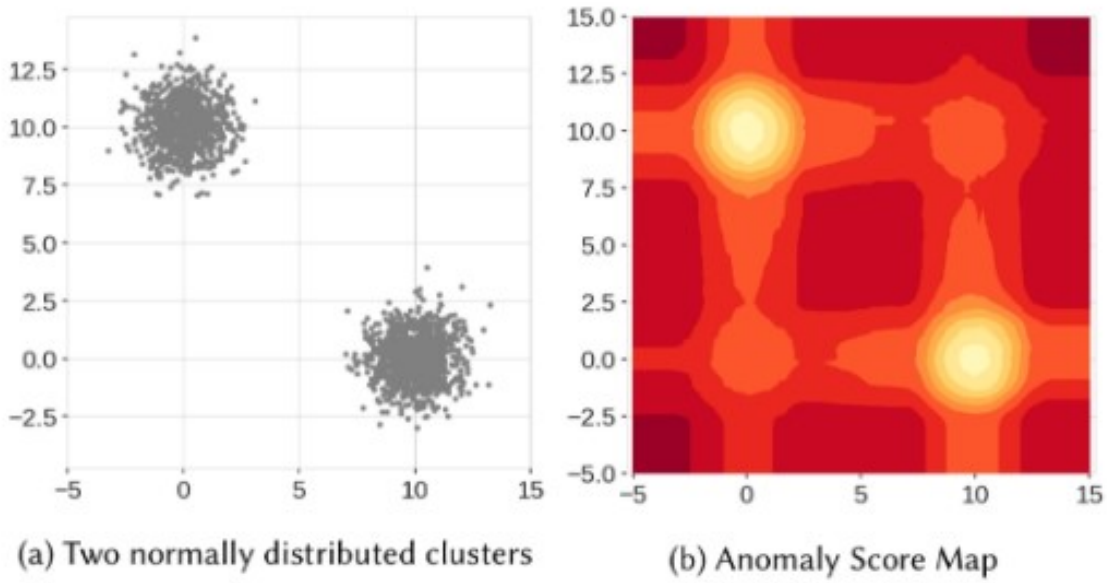
Now, Isolation Forest (right side image) is correctly showing the center [0,0] with light color region that means very low anomaly score and it is gradually increasing the strength of the color as we move away from the center [0,0]. For example, compare the color at [0,0] and [-2,0], the later has a dark shade of red as compared to the center.

But, Isolation Forest doesn't stops here, as it also created the un-wanted axis parallel lines surrounding the circle other than the 4 corners of the image which are totally(dark red corners) with higher anomaly score. These axes parallel lines should not be their logically but Isolation Forest creates them artifically which affects the overall ANOMALY SCORE. Becuase, ideally, the entire image should only have

the middle circle which means points outside the circle will be with high anomaly score.

In [19]:
```
Image(filename = PATH + "6_IF_Scoremap_Two_Gauss_Clusters.jpg", width=700, height=70
```

Out[19]:



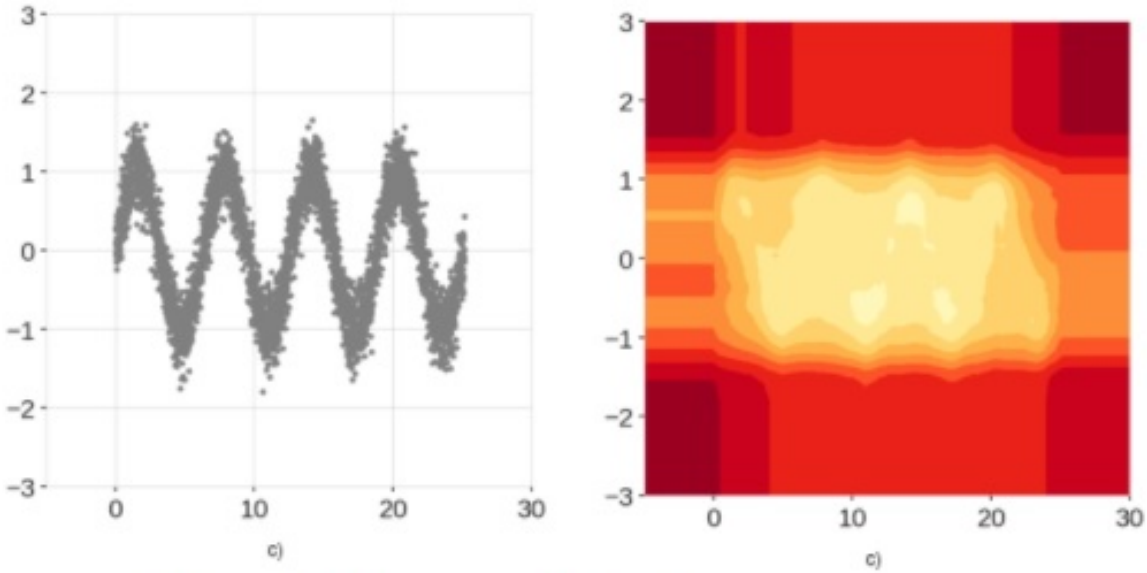
(a) Two normally distributed clusters (b) Anomaly Score Map

In the above image on the left side we have two normally distributed clusters of data and on the right we have the anomaly score map.

Now, if we closely examine the right side image then should only have 2 circles at [0,10] and [10,0]. Other than these two circles, Isolation Forest also created 2 artifical circles at [0,0] and [10,10] which were not present in the actual data. In addition to these 2 artificial circles it also created the axes parallel lines surrounding all the 4 lines. So, all of these were totally not present in the actual data and it shouldn't have been created by IF.

In [20]:
```
Image(filename = PATH + "5_IF_Scoremap_Sinusoidal_Data.jpg", width=700, height=700)
```

Out[20]:



c) Sinusoidal data points with Gaussian noise.

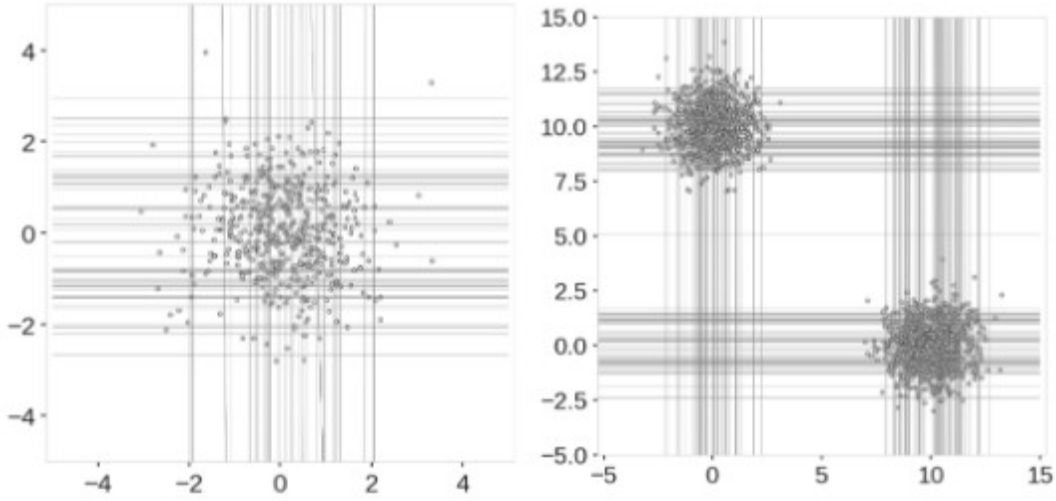
In the above image on the left side we have sinusoidal data and on the right we have the anomaly score map.

Because Isolation Forest can only have the axes parallel lines thus we lose the shape of the data(no wave in the anomaly scores), this is again not the right thing. Along with this the lines are also surrounding the central region that is again unwanted.

Why Isolation Forest does that?

In [21]:
```
Image(filename = PATH + "11_IF_Problems.jpg", width=850, height=850)
```

Out[21]:



(a) Single blob (b) Multiple Blobs

Point to note here is that the area in the images where the density of points is high also has higher

number of axis paralled lines splitting the branches. And, if we closely observe the images then we can also found that area where there are actually no points originally also been included unwantedly only due to the nature of lines.

For example, in image (b) look at the point [0,0] and [10,10] so many lines are passing across these regions, but as the lines can only be parallel to the axes, these are regions that contain many branch cuts and only a few/single or no observations exists, which results in improper anomaly scores calculation for some of the observations.

The solution to above problems of Isolation Forest is the Extended Isolation Forest or also known as Isolation Forest Extension.

EXTENDED Isolation Forest

The only difference b/w IF and EIF is that the latter allows the slicing of data using hyperplanes with random slopes which results in improved score maps. Instead of selecting a random feature and then random value within the range of data, it selects:

- **the random slope for the branch cut**
- **random intercept chosen from the range of available values from the training data**

In [22]:
```
Image(filename = PATH + "9_EIF_Hyperplanes_splitting_the_space.jpg", width=850, heig
```

Out[22]:

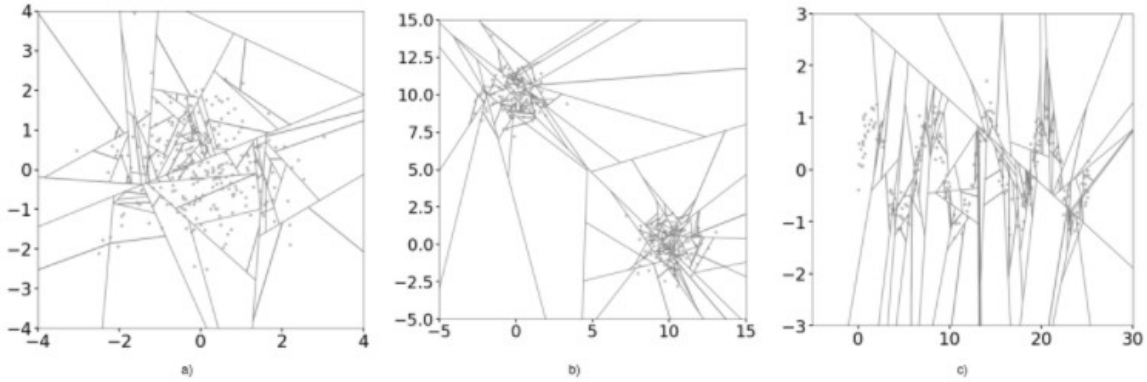


Figure 5: Same as Figure 4 but using Extended Isolation Forest

The inclusion of random hyperplanes no more creating the unwanted axis

parallel line cuts where the density of points is high.

In [23]:
```
Image(filename = PATH + "10_EIF_Score_Maps.jpg", width=850, height=850)
```

Out[23]:

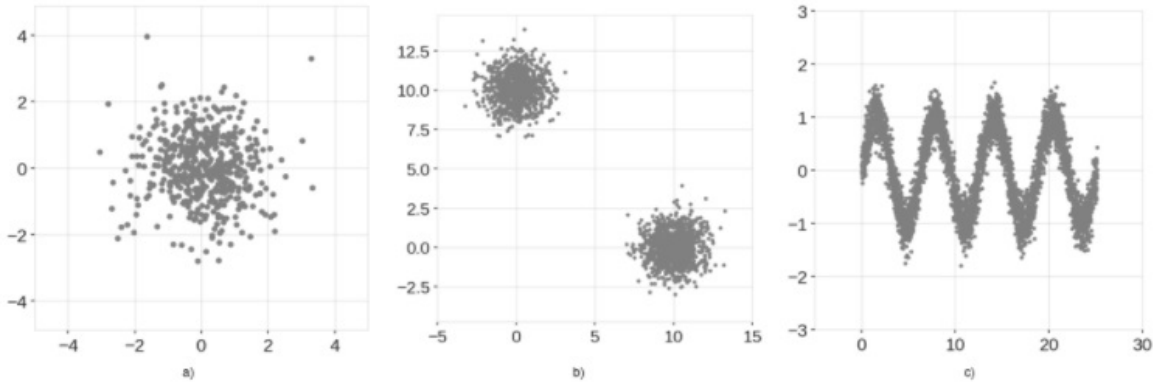


Figure 1: Example training data. a) Normally distributed cluster. b) Two normally distributed clusters. c) Sinusoidal data points with Gaussian noise.

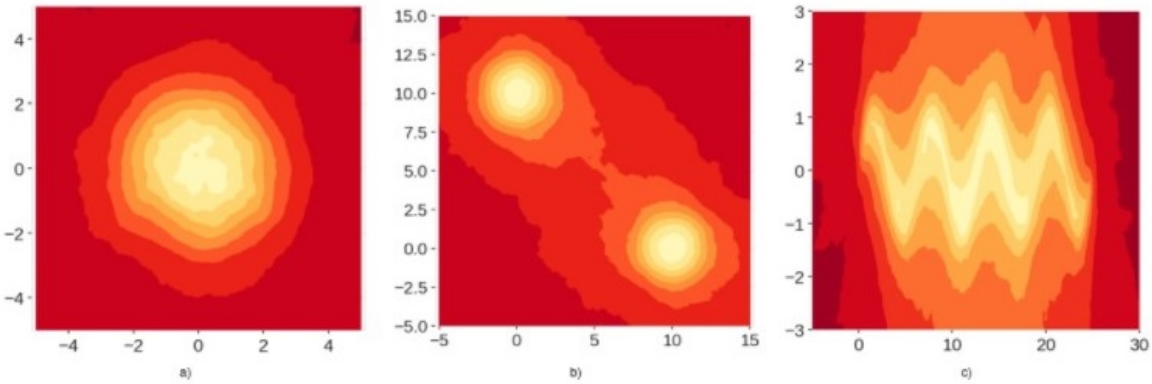


Figure 6: Score maps using the Extended Isolation Forest.

As shown in the above images EIF gives us better anomaly score maps also it doesn't lose the shape of the original data. In the (b) image the region where the two circles are linking with each other is the point where the anomaly is score is high because that point or area is far away from both the circles.

Sklearn IF Implementation

In [43]:
```
iso_for = IsolationForest(n_estimators=15,max_samples=25,contamination=0.05,max_feat
```

In [44]:
```
iso_for.fit(X_iris_df)
```

Out[44]:
```
IsolationForest(bootstrap=True, contamination=0.05, max_features=2,
                max_samples=25, n_estimators=15, random_state=41)
```

In [45]:
```
iso_for.estimators_
```

Out[45]:
```
[ExtraTreeRegressor(max_depth=5, max_features=1, random_state=716905170),
 ExtraTreeRegressor(max_depth=5, max_features=1, random_state=180789943),
 ExtraTreeRegressor(max_depth=5, max_features=1, random_state=1315178973),
 ExtraTreeRegressor(max_depth=5, max_features=1, random_state=1681872075),
 ExtraTreeRegressor(max_depth=5, max_features=1, random_state=280069627),
 ExtraTreeRegressor(max_depth=5, max_features=1, random_state=1055549073),
 ExtraTreeRegressor(max_depth=5, max_features=1, random_state=1764538871),
 ExtraTreeRegressor(max_depth=5, max_features=1, random_state=405848271),
 ExtraTreeRegressor(max_depth=5, max_features=1, random_state=1224622560),
 ExtraTreeRegressor(max_depth=5, max_features=1, random_state=2088199739),
```

```
 ExtraTreeRegressor(max_depth=5, max_features=1, random_state=1220789893),
 ExtraTreeRegressor(max_depth=5, max_features=1, random_state=271136312),
 ExtraTreeRegressor(max_depth=5, max_features=1, random_state=1321442735),
 ExtraTreeRegressor(max_depth=5, max_features=1, random_state=454001309),
 ExtraTreeRegressor(max_depth=5, max_features=1, random_state=2135786185)]
```

Using ExtraTreeRegressor as the base estimator

In [46]:
```
iso_for.estimators_features_
```

Out[46]:
```
[array([1, 0]),
 array([0, 1]),
 array([0, 3]),
 array([1, 0]),
 array([2, 0]),
 array([2, 3]),
 array([1, 2]),
 array([2, 1]),
 array([2, 3]),
 array([0, 2]),
 array([3, 2]),
 array([2, 3]),
 array([0, 2]),
 array([2, 3]),
 array([3, 0])]
```

Features randomly selected fro every tree

In [47]:
```
iso_for.estimators_samples_
```

Out[47]:
```
[array([ 88,  29, 146,  21,  26,  26,  52,  90,  36, 141,  47, 141,  77,
         67, 115, 120, 133, 120,   3, 121,  41, 118, 136, 149, 130]),
 array([  8, 113,  84,  71,  13, 108,  54, 101,  37, 102,   3,  73, 115,
         23, 104,  98,  95,  93,  69,  53,  93,  44, 135,  55,   5]),
 array([  4,  15,  36,  27,  25,  40, 138, 116, 127,  25,  64,  86, 120,
        116,  33,  31, 125,  80, 102, 102, 145, 120,  43,  97,  50]),
 array([ 95, 122, 144,  20, 101,   5,  39, 109,  30, 144, 136,  39, 143,
         40,  38, 127, 107,  98,  39,   3,  98,  31,  88, 140,  14]),
 array([ 67,  82,  87,  56,  18,  14,  73,  47,  40, 118,  23,  35,  35,
        142, 139,  48, 125, 132, 112,  56,  60,  73, 110, 107,  41]),
 array([ 62,  42, 112, 142, 130,   2,  91,  41,  76,   4,  74,  62,  90,
         31, 145, 102, 147,  10,  41,  39,   6, 131,  25, 136, 127]),
 array([ 53, 102,  45, 116, 123,  78, 115,  22, 148,  22,  93,  12,  72,
        139,  59,  98,  63,  77,  72,  31, 104,  59,  60, 129,  52]),
 array([  1, 111,  35, 125, 100,  73,  74, 118,   2, 116,  85, 102,  31,
         51,  53, 131, 118,  14, 135, 136, 113,  69,  97,  15,  83]),
 array([  3, 147,  30,  70,  42,  20,  64,  75,  40, 120, 125,  81,  46,
        134,  52,  15,  95,  78, 148, 114, 147, 124,  83,  13,  97]),
 array([148,  61, 136,  14,  38, 124,  22,   8,  18,  57,  70,  91, 140,
        137,  27,  88, 130,  74,  24,  92,  29, 128,  60,  78,  21]),
 array([126,  97,  64,  12, 142, 115, 142, 106,  96,  77,  87,   8,  18,
        107, 124,   5,  28, 122, 108,  61,   4,  62,  31,  42, 109]),
 array([ 52, 106,  41,  38, 144, 144,  97, 138,  62,  19,  19,  84,  83,
         28, 110,  60, 104, 102, 130, 128,  66,   9,  83,  21,  60]),
 array([102,  90,  29, 100, 101, 131,  81,  53,  44,  56,  85,  62,  93,
         32,   9,   9, 148,  43,  75, 129, 105, 137,  17,  18, 114]),
 array([ 68, 144, 143,  23,  75, 103,  46,  72, 142,  56, 119,  26,  97,
         97,  48,  77,  47,  43, 126,  27,  35,  82, 142,  27,  14]),
 array([ 86, 139,  71,  69, 137,  39, 108,  60,  60, 127,  24,  81,  60,
         79,  14, 102,  48,  46,  97,  20, 144,  92,   7,  94,  97])]
```

Observations or samples using to build the trees.

In [29]:
```
iso_for.estimators_samples_[0].shape
```

Out[29]:
```
(25,)
```

In [65]:
```
iso_for.offset_
```

Out[65]:
```
-0.6031896746731866
```

In [55]:
```
X_iris_df['Avg_Anomaly_Scr'] = iso_for.decision_function(X_iris_df)
```

In [57]:
```
X_iris_df['Opp_Anomaly_Scr'] = iso_for.score_samples(X_iris_df.iloc[:,0:4])
```

Decision_Function gives the average anomaly score of an observation based on its score from every tree. If it is less than 0 then it means or indicates the abnormality.

Score_Samples gives the opposite of average anomaly score of an observation based on its score from every tree. If it is less than offset then it means or indicates the abnormality.

In [59]:
```
X_iris_df['Pred'] = iso_for.fit_predict(X_iris_df.iloc[:,0:4])
```

In [60]:
```
X_iris_df[np.sign(X_iris_df['Pred']) != 1]
```

Out[60]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Avg_Anomaly_Scr	Opp_Anomaly_Scr	Pred
41	4.5	2.3	1.3	0.3	-0.015337	-0.618527	-1
105	7.6	3.0	6.6	2.1	-0.006849	-0.610039	-1
109	7.2	3.6	6.1	2.5	-0.031503	-0.634693	-1
117	7.7	3.8	6.7	2.2	-0.074891	-0.678081	-1
118	7.7	2.6	6.9	2.3	-0.052838	-0.656028	-1
122	7.7	2.8	6.7	2.0	-0.011912	-0.615102	-1
131	7.9	3.8	6.4	2.0	-0.059095	-0.662285	-1
135	7.7	3.0	6.1	2.3	-0.005982	-0.609172	-1

Understanding the relationship b/w decision_function, score_samples and offset

`decision_function = score_samples - offset_`

- `offset_` **is defined as follows:**
  - **When the contamination parameter is set to "auto", the offset is equal to -0.5 as the scores of inliers are close to 0 and the scores of outliers are close to -1.**
  - **When a contamination parameter different than "auto" is provided, the offset is defined in such a way we obtain the expected number of outliers (samples with decision function < 0) in training.**

In [61]:
```
X_iris_df[np.sign(X_iris_df['Avg_Anomaly_Scr']) != 1]
```

Out[61]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Avg_Anomaly_Scr	Opp_Anomaly_Scr	Pred
41	4.5	2.3	1.3	0.3	-0.015337	-0.618527	-1
105	7.6	3.0	6.6	2.1	-0.006849	-0.610039	-1
109	7.2	3.6	6.1	2.5	-0.031503	-0.634693	-1
117	7.7	3.8	6.7	2.2	-0.074891	-0.678081	-1
118	7.7	2.6	6.9	2.3	-0.052838	-0.656028	-1
122	7.7	2.8	6.7	2.0	-0.011912	-0.615102	-1
131	7.9	3.8	6.4	2.0	-0.059095	-0.662285	-1
135	7.7	3.0	6.1	2.3	-0.005982	-0.609172	-1

If we compare the above results of IF with LOF then some of these are marked as Outliers from both the techniques.

In [78]:
```
-1 * np.unique(X_iris_df['Avg_Anomaly_Scr'] - X_iris_df['Opp_Anomaly_Scr'])
```

Out[78]:
```
array([-0.60318967])
```

Calculated the Offset value manually

In [73]:
```
X_iris_df[X_iris_df['Avg_Anomaly_Scr'] < 0]
```

Out[73]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Avg_Anomaly_Scr	Opp_Anomaly_Scr	Pred
41	4.5	2.3	1.3	0.3	-0.015337	-0.618527	-1
105	7.6	3.0	6.6	2.1	-0.006849	-0.610039	-1
109	7.2	3.6	6.1	2.5	-0.031503	-0.634693	-1
117	7.7	3.8	6.7	2.2	-0.074891	-0.678081	-1
118	7.7	2.6	6.9	2.3	-0.052838	-0.656028	-1
122	7.7	2.8	6.7	2.0	-0.011912	-0.615102	-1
131	7.9	3.8	6.4	2.0	-0.059095	-0.662285	-1
135	7.7	3.0	6.1	2.3	-0.005982	-0.609172	-1

Outliers are the points whose Anomaly score is less than 0 or -ve.

In [77]:
```
X_iris_df[X_iris_df['Opp_Anomaly_Scr'] < iso_for.offset_]
```

Out[77]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Avg_Anomaly_Scr	Opp_Anomaly_Scr	Pred
41	4.5	2.3	1.3	0.3	-0.015337	-0.618527	-1
105	7.6	3.0	6.6	2.1	-0.006849	-0.610039	-1
109	7.2	3.6	6.1	2.5	-0.031503	-0.634693	-1

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Avg_Anomaly_Scr	Opp_Anomaly_Scr	Pred
117	7.7	3.8	6.7	2.2	-0.074891	-0.678081	-1
118	7.7	2.6	6.9	2.3	-0.052838	-0.656028	-1
122	7.7	2.8	6.7	2.0	-0.011912	-0.615102	-1
131	7.9	3.8	6.4	2.0	-0.059095	-0.662285	-1
135	7.7	3.0	6.1	2.3	-0.005982	-0.609172	-1

Outliers are the points whose Opposite Anomaly score is less than the offset value.

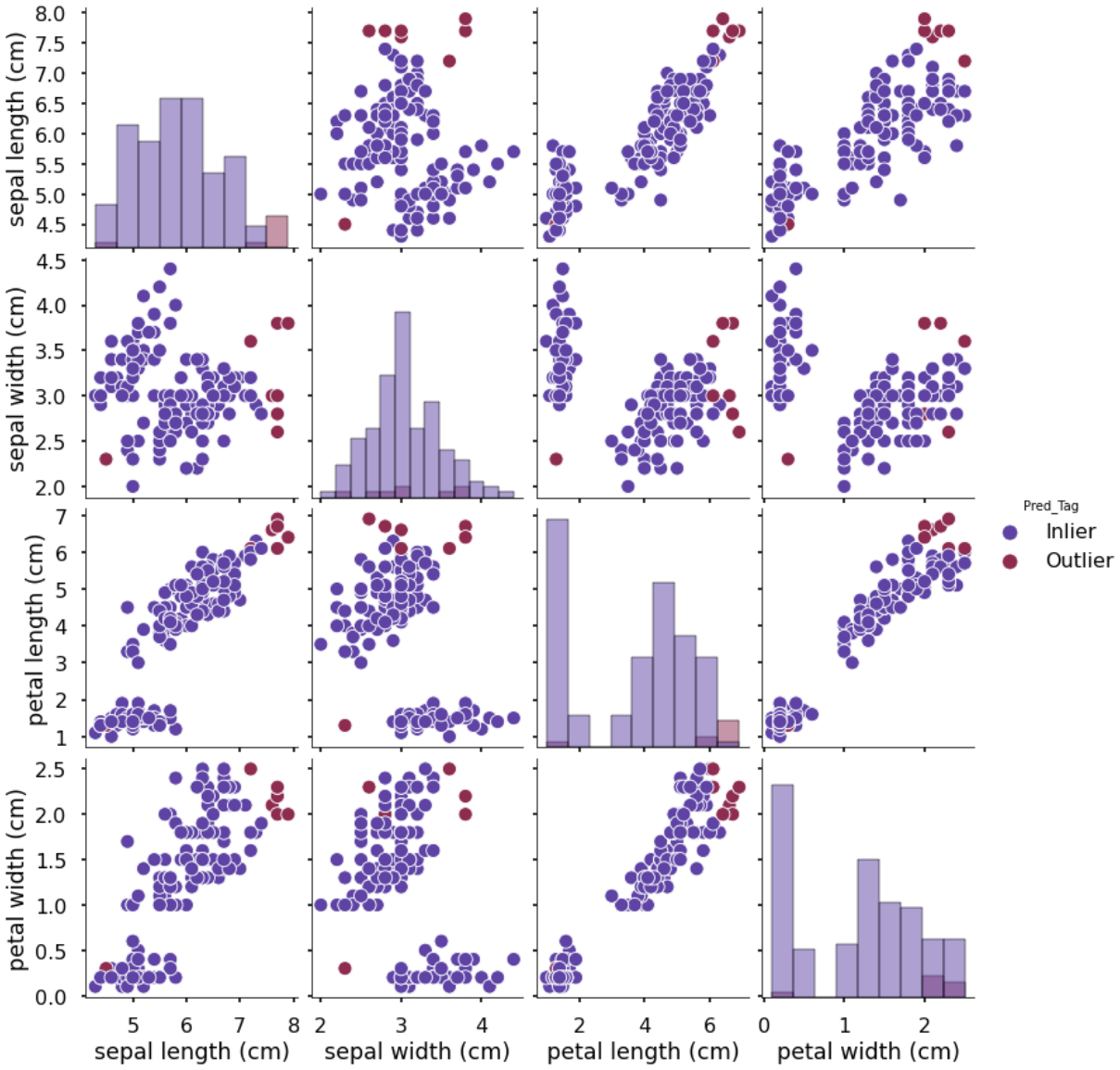
Let's visualize some results

In [84]:
```
if_results = pd.concat([X_iris_df.copy(deep=True),y_iris_df.copy(deep=True)],axis=1)
tag_dict = {1:'Inlier',-1:'Outlier'}
if_results['Pred_Tag'] = if_results['Pred'].apply(lambda val: tag_dict.get(val))
if_results.head()
```

Out[84]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Avg_Anomaly_Scr	Opp_Anomaly_Scr	Pred	Label	Pred_Tag
0	5.1	3.5	1.4	0.2	0.112295	-0.490895	1	0	Inlier
1	4.9	3.0	1.4	0.2	0.130553	-0.472637	1	0	Inlier
2	4.7	3.2	1.3	0.2	0.060184	-0.543006	1	0	Inlier
3	4.6	3.1	1.5	0.2	0.095554	-0.507636	1	0	Inlier
4	5.0	3.6	1.4	0.2	0.108220	-0.494969	1	0	Inlier

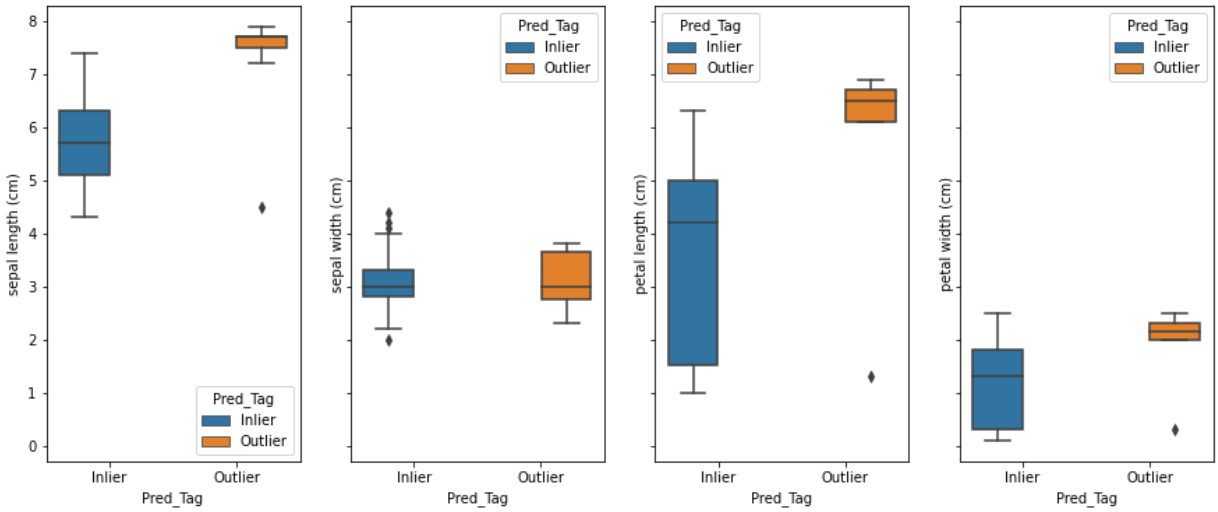
In [85]:
```
with plt.style.context('seaborn-poster'):
    g = sns.pairplot(data=if_results[['sepal length (cm)','sepal width (cm)','petal
                 hue='Pred_Tag',palette='twilight',height=3,aspect=0.9,diag_kind='hi
```



The first look gave me an impression of slightly better results are compared to LOF.

Let's do some more plotting

In [86]:
```
fig, ax = plt.subplots(1,4,sharex=True,sharey=True,squeeze=True,figsize=(15,6))
sns.boxplot(data=if_results,x='Pred_Tag',y='sepal length (cm)',hue='Pred_Tag',ax=ax[
sns.boxplot(data=if_results,x='Pred_Tag',y='sepal width (cm)',hue='Pred_Tag',ax=ax[1
sns.boxplot(data=if_results,x='Pred_Tag',y='petal length (cm)',hue='Pred_Tag',ax=ax[
sns.boxplot(data=if_results,x='Pred_Tag',y='petal width (cm)',hue='Pred_Tag',ax=ax[3
plt.show()
```



In [87]:
```
if_results.groupby(['Pred_Tag'])[['sepal length (cm)', 'sepal width (cm)', 'petal le
```

Out[87]:

Pred_Tag	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
Inlier	4.3	2.0	1.0	0.1
Outlier	4.5	2.3	1.3	0.3

In [89]:
```
if_results.groupby(['Pred_Tag'])[['sepal length (cm)', 'sepal width (cm)', 'petal le
```

Out[89]:

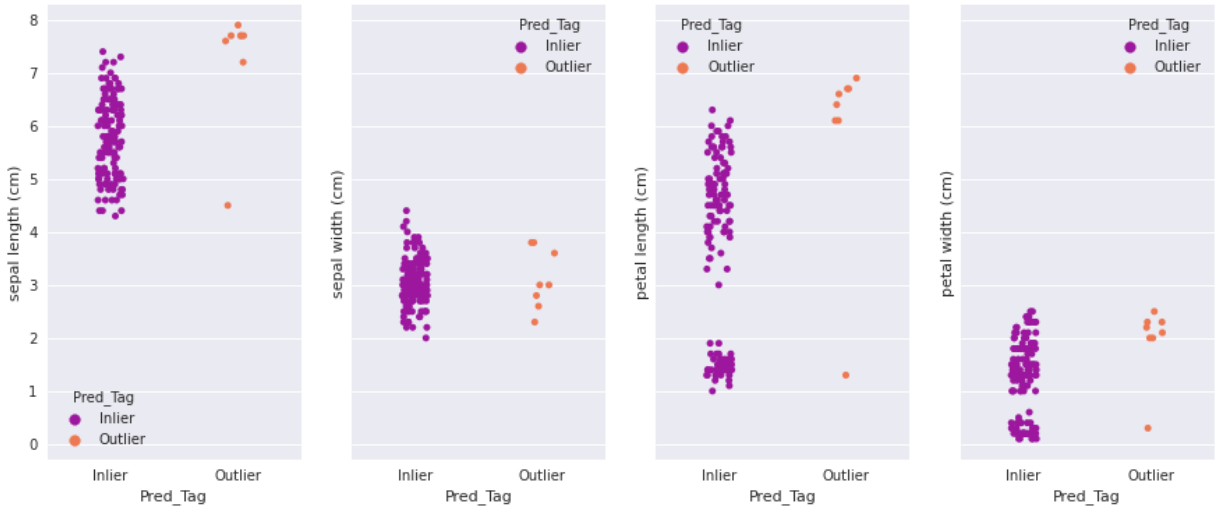
Pred_Tag	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
Inlier	5.7	3.0	4.2	1.30
Outlier	7.7	3.0	6.5	2.15

In [88]:
```
if_results.groupby(['Pred_Tag'])[['sepal length (cm)', 'sepal width (cm)', 'petal le
```

Out[88]:

Pred_Tag	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
Inlier	7.4	4.4	6.3	2.5
Outlier	7.9	3.8	6.9	2.5

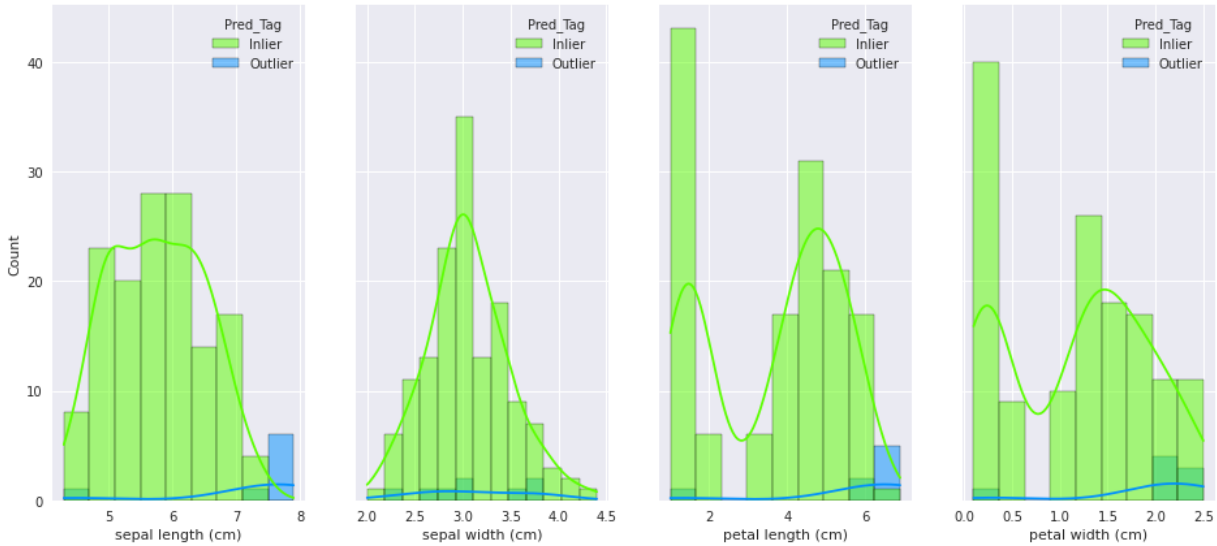
In [90]:
```
with plt.style.context('seaborn'):
    fig, ax = plt.subplots(1,4,figsize=(15,6),sharex=True,sharey=True)
    sns.stripplot(data=if_results,x='Pred_Tag',y='sepal length (cm)',hue='Pred_Tag',
    sns.stripplot(data=if_results,x='Pred_Tag',y='sepal width (cm)',hue='Pred_Tag',p
    sns.stripplot(data=if_results,x='Pred_Tag',y='petal length (cm)',hue='Pred_Tag',
    sns.stripplot(data=if_results,x='Pred_Tag',y='petal width (cm)',hue='Pred_Tag',p
```



If we look at the outliers then they are clearly the extreme points from Sepal length, Petal Length and Petal Width which are either at the edges or boundaries of the clusters.

In [95]:
```
with plt.style.context('seaborn'):
    fig, ax = plt.subplots(1,4,figsize=(16,7),sharex=False,sharey=True)
    sns.histplot(data=if_results,x='sepal length (cm)',hue='Pred_Tag',palette='gist_
    sns.histplot(data=if_results,x='sepal width (cm)',hue='Pred_Tag',palette='gist_r
```

```
    sns.histplot(data=if_results,x='petal length (cm)',hue='Pred_Tag',palette='gist_
    sns.histplot(data=if_results,x='petal width (cm)',hue='Pred_Tag',palette='gist_r
```



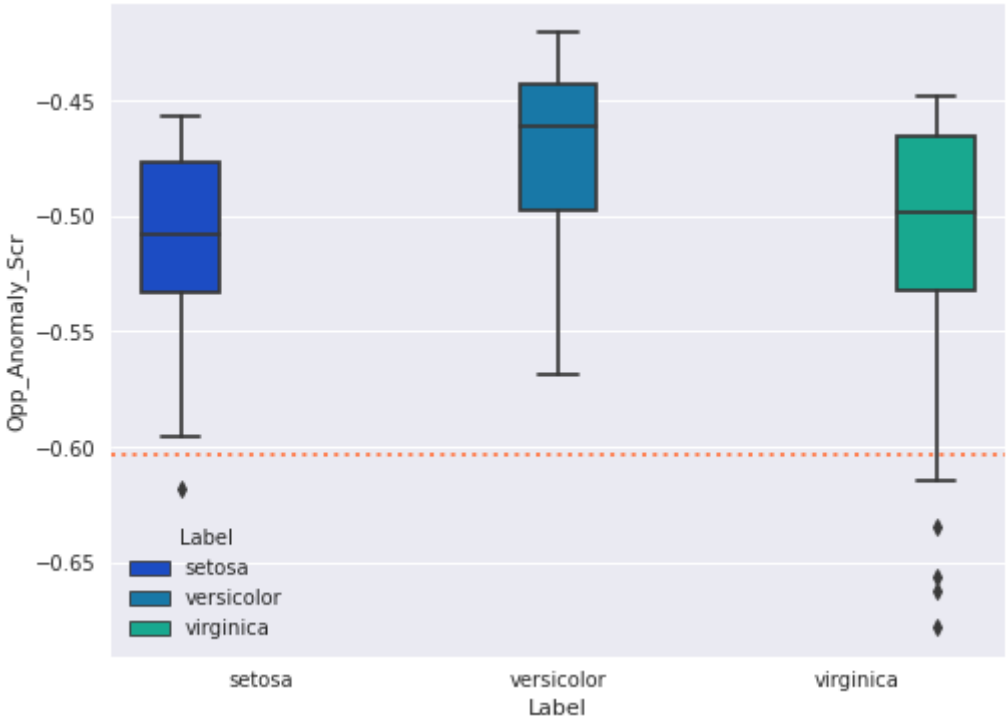
Some gaps are quite evident in the above plots and point to mention here is that majority of the outliers are from the extreme or higher values of features.

In [96]:
```
iris_classes = {0:'setosa',1:'versicolor',2:'virginica'}
if_results['Label'] = if_results['Label'].apply(lambda val : iris_classes.get(val))
```

In [97]:
```
with plt.style.context('seaborn'):
    fig, ax = plt.subplots(1,4,figsize=(15,6),sharex=True,sharey=True)
    sns.stripplot(data=if_results,x='Pred_Tag',y='sepal length (cm)',hue='Label',pal
    sns.stripplot(data=if_results,x='Pred_Tag',y='sepal width (cm)',hue='Label',pale
    sns.stripplot(data=if_results,x='Pred_Tag',y='petal length (cm)',hue='Label',pal
    sns.stripplot(data=if_results,x='Pred_Tag',y='petal width (cm)',hue='Label',pale
```

No outliers been selected from Versicolor class. Same result was also generated by LOF which means both of these techniques quite have behaved in similar fashion in this dataset.

In [131…
```
with plt.style.context('seaborn'):
    plt.figure(figsize=(8,6))
    sns.boxplot(data=if_results,x='Label',y='Opp_Anomaly_Scr',hue='Label',palette='w
    plt.axhline(iso_for.offset_,color='coral',linestyle=':',linewidth=2)
```



The Anomaly score in Versicolor is less than the offset value thus no outliers in this class.

In the above analysis, I have encountered a bit of similar results between LOF and IF. I enojyed learning about Isolation Forest, its problems and its extension, however EIF is not available in Sklearn. And, its implementation in EIF library currently doesn't provide much flexibility to the user.

As Isolation Forest works in a manner that easily isolated observations are outliers by building the binary trees this makes it powerful in identifying the golbal outliers but it kind of fails to identify or capture the local outlier which is near to the normal points but abnormal in nature. LOF doesn't really suffer from this problem. However, the time complexity of IF is better than LOF and the addition of hyperplanes in EIF makes it very suitable for higher dimensional data.