Outlier and Novelty Detection using `Local Outlier Factor`

Notebook Contents

Package_import

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)
```

Dataset_import

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
```

Out[10]:
```
((150,), array(['setosa', 'versicolor', 'virginica'], dtype='<U10'))
```

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

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

Outlier_Detection

- ***We want to find the outliers in the given dataset or make the train dataset outliers free.***

LOF totally relies on distance calculations and below are some points which we need to consider when we are working distance calculating algorithms:

- **`Does your features exist in different scales?` For example f1 in kg and f2 in mm so on..**
    - **If yes and you are using Euclidean distance (most popular among data scientists), then without performing the features standard scaling(means all features are one mean centric) you will get the highly variant or skewed distances.**
    - **If no (means feature values are in same scale/unit) and you are using Eucildean distance then still you will be better off if perform the features standard scaling (means all features are one mean centric) else you will get higher distances of**

global outliers and these will dominate your outlier detection distances and the local outliers will get impacted.

Here, I will use below distance metrics:

- **Euclidean**
- **Manhattan**
- **Mahalanobis** (if we want to consider the correlation b/w the features)

In [12]:
```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
```

In [13]:
```
X_iris_df_st = pd.DataFrame(ss.fit_transform(X_iris_df.copy(deep=True)),columns=iris
```

In [14]:
```
X_iris_df_st.head()
```

Out[14]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	-0.900681	1.019004	-1.340227	-1.315444
1	-1.143017	-0.131979	-1.340227	-1.315444
2	-1.385353	0.328414	-1.397064	-1.315444
3	-1.506521	0.098217	-1.283389	-1.315444
4	-1.021849	1.249201	-1.340227	-1.315444

Using LOF to find outliers in IRIS Dataset

CASE-I

Neighbors = 20 & Contamination = 0.05 or 5% & Leaf_Size = 15 & Distance Metric = Manhattan

In [15]:
```
print(X_iris_df_st.shape)

iris_contam = 0.05
print(X_iris_df_st.shape[0]*iris_contam)
```

```
(150, 4)
7.5
```

In [16]:
```
lof_iris = LocalOutlierFactor(n_neighbors=20,algorithm='kd_tree',leaf_size=15,contam
                              novelty=False,metric='minkowski',p=1)
```

In [17]:
```
lof_iris_pred = lof_iris.fit_predict(X_iris_df_st)
lof_iris_pred.shape
```

Out[17]:
```
(150,)
```

In [18]:
```
pred_dict = {1:0, -1:1}
lof_iris_pred = [pred_dict.get(val) for val in lof_iris_pred]
```

Here, LOF returns the results in the form of [1, -1] where 1's are considered as inliers and -1 are considered as Outliers.

In [19]:
```
np.unique(lof_iris_pred), np.bincount(lof_iris_pred)
```

Out[19]:
```
(array([0, 1]), array([142,   8], dtype=int64))
```

In [20]:
```
lof_iris_vals = lof_iris.negative_outlier_factor_          ## -ve lof value
```

LOF will always returns the -ve factor score and the inliers generally have score close to 1 i.e. close to -1, whereas Outliers tends to have a larger score.

In [21]:
```
lof_iris_offset = lof_iris.offset_        ## Threshold Value
lof_iris_offset
```

Out[21]:
```
-1.4735496359340885
```

This is the negative threshold value generated by LOF if any value is greater than it then such a record will be considered as an Outlier.

In [22]:
```
iris_lof_result = pd.concat([X_iris_df_st,y_iris_df,
           pd.DataFrame(lof_iris_vals,columns=['LOF_Values']),
           pd.DataFrame(lof_iris_pred,columns=['LOF_Tags'])],axis=1)
```

In [23]:
```
iris_lof_result[iris_lof_result['LOF_Values'] < lof_iris_offset]
```

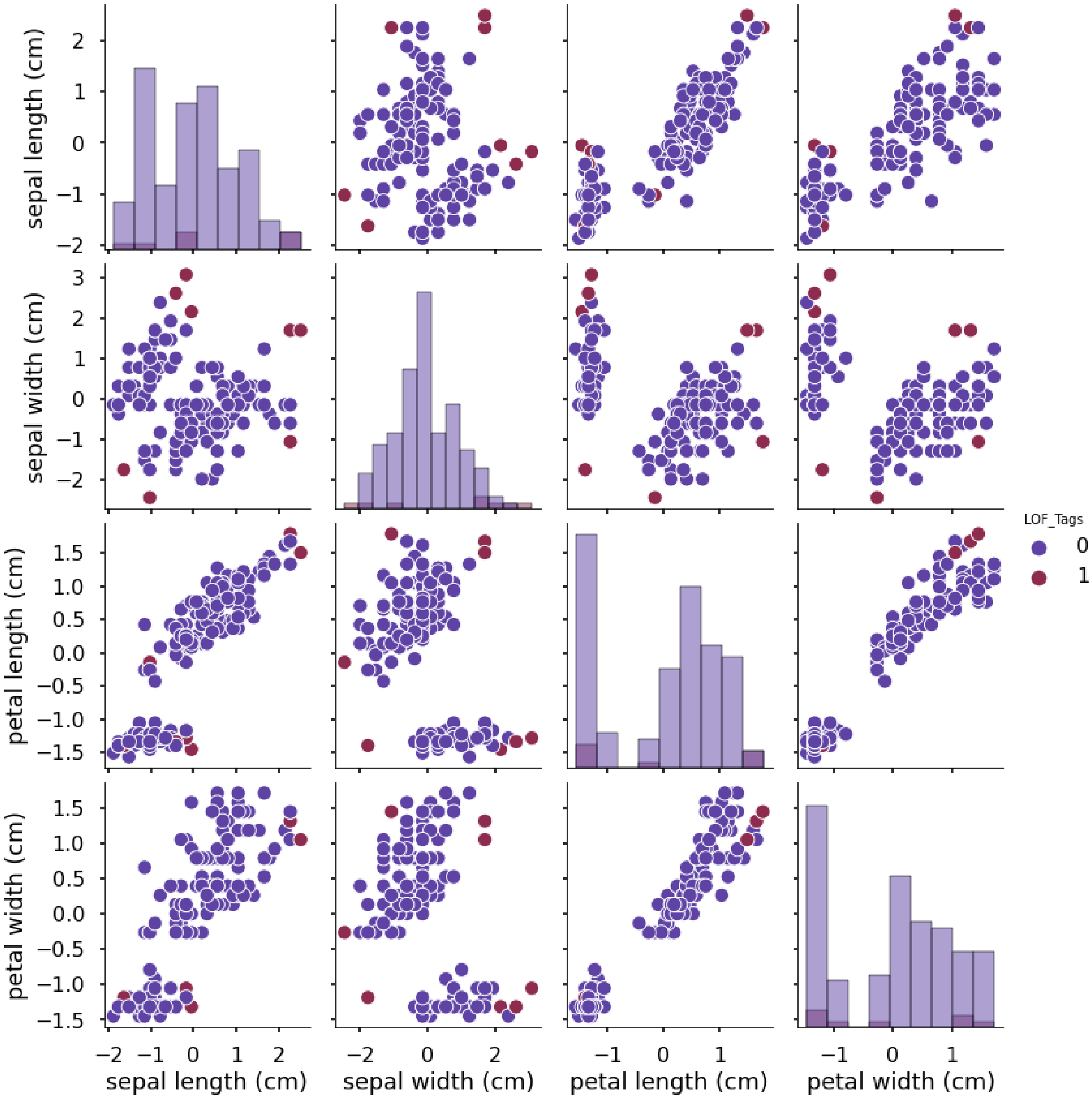
Out[23]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Label	LOF_Values	LOF_Tags
14	-0.052506	2.169988	-1.453901	-1.315444	0	-1.511115	1
15	-0.173674	3.090775	-1.283389	-1.052180	0	-1.910514	1
33	-0.416010	2.630382	-1.340227	-1.315444	0	-1.506934	1
41	-1.627688	-1.743357	-1.397064	-1.183812	0	-1.878636	1
60	-1.021849	-2.433947	-0.146641	-0.262387	1	-1.496300	1
117	2.249683	1.709595	1.672157	1.317199	2	-1.667270	1
118	2.249683	-1.052767	1.785832	1.448832	2	-1.626788	1
131	2.492019	1.709595	1.501645	1.053935	2	-1.736445	1

Now, if we see the -ve scores then they are not so highly away or larger than the offset value but if we compare these values with -1 then yes the above records are a bit away from the generally considered inliers.

- Here, the dataset was not having any labels for the outliers, so this is completely driven based on distance calculation
- Plotting the records with above generated score will tell the better story

In [24]:
```
with plt.style.context('seaborn-poster'):
    g = sns.pairplot(data=iris_lof_result[['sepal length (cm)','sepal width (cm)','p
                     hue='LOF_Tags',palette='twilight',height=3,aspect=0.9,diag_kind='hi
```



As, the plot is self-explanatory because the data points which are away(separated from others) from the clusters or at the edges/boundaries of it are labelled as Outliers. This is exactly the way LOF behaves.

More analysis

In [25]:
```
iris_lof_result.columns
```

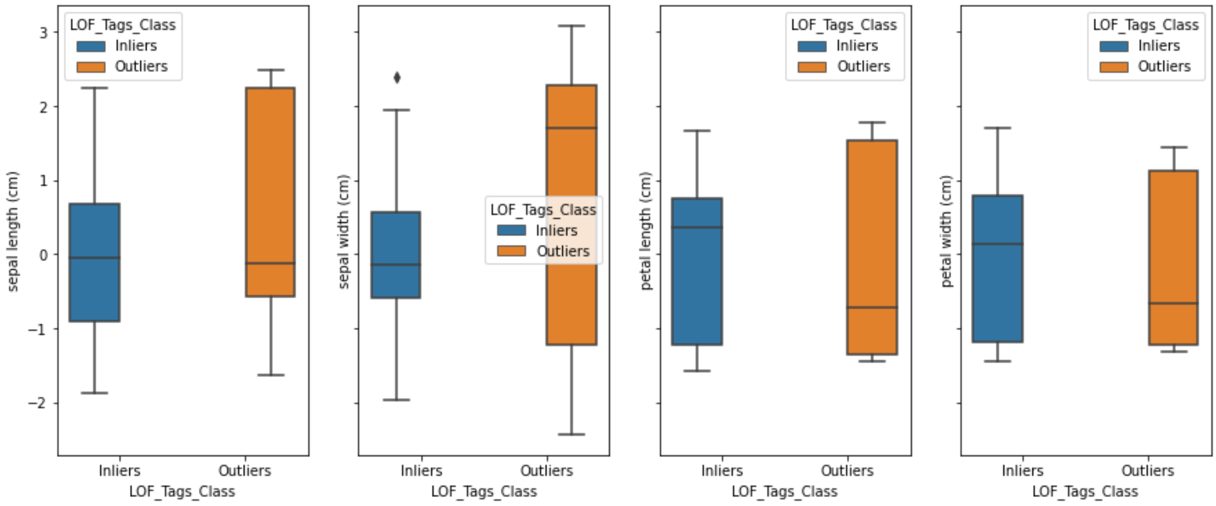
Out[25]:
```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)', 'Label', 'LOF_Values', 'LOF_Tags'],
      dtype='object')
```

In [26]:
```
def lbl(val):
    if val == 1:
        return 'Outliers'
    else:
        return 'Inliers'
```

In [27]:
```
iris_lof_result['LOF_Tags_Class'] = iris_lof_result['LOF_Tags'].apply(lambda val: lb
```

In [28]:
```
fig, ax = plt.subplots(1,4,sharex=True,sharey=True,squeeze=True,figsize=(15,6))
sns.boxplot(data=iris_lof_result,x='LOF_Tags_Class',y='sepal length (cm)',hue='LOF_T
sns.boxplot(data=iris_lof_result,x='LOF_Tags_Class',y='sepal width (cm)',hue='LOF_Ta
sns.boxplot(data=iris_lof_result,x='LOF_Tags_Class',y='petal length (cm)',hue='LOF_T
```

```
sns.boxplot(data=iris_lof_result,x='LOF_Tags_Class',y='petal width (cm)',hue='LOF_Ta
plt.show()
```



In [29]:
```
iris_lof_result.groupby(['LOF_Tags_Class'])[['sepal length (cm)', 'sepal width (cm)'
```

Out[29]:

LOF_Tags_Class	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
Inliers	-1.870024	-1.973554	-1.567576	-1.447076
Outliers	-1.627688	-2.433947	-1.453901	-1.315444

In [30]:
```
iris_lof_result.groupby(['LOF_Tags_Class'])[['sepal length (cm)', 'sepal width (cm)'
```

Out[30]:

LOF_Tags_Class	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
Inliers	-0.052506	-0.131979	0.364896	0.132510
Outliers	-0.113090	1.709595	-0.715015	-0.657283

In [31]:
```
iris_lof_result.groupby(['LOF_Tags_Class'])[['sepal length (cm)', 'sepal width (cm)'
```

Out[31]:

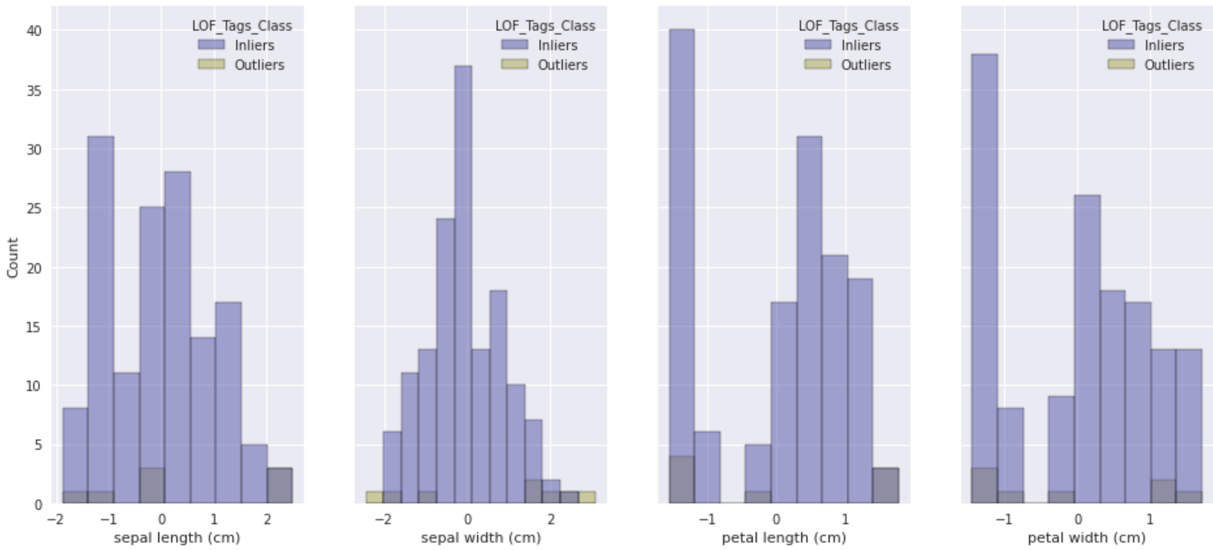
LOF_Tags_Class	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
Inliers	2.249683	2.400185	1.672157	1.712096
Outliers	2.492019	3.090775	1.785832	1.448832

So, all the features have good point variation in the 5 number summary of Inliers and Outliers.

In [32]:
```
with plt.style.context('seaborn'):
    fig, ax = plt.subplots(1,4,figsize=(15,6),sharex=True,sharey=True)
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='sepal length (cm)',hue=
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='sepal width (cm)',hue='
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='petal length (cm)',hue=
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='petal width (cm)',hue='
```

If we look at the outliers then they are clearly the extreme points which are either at the edges or boundaries of the clusters.

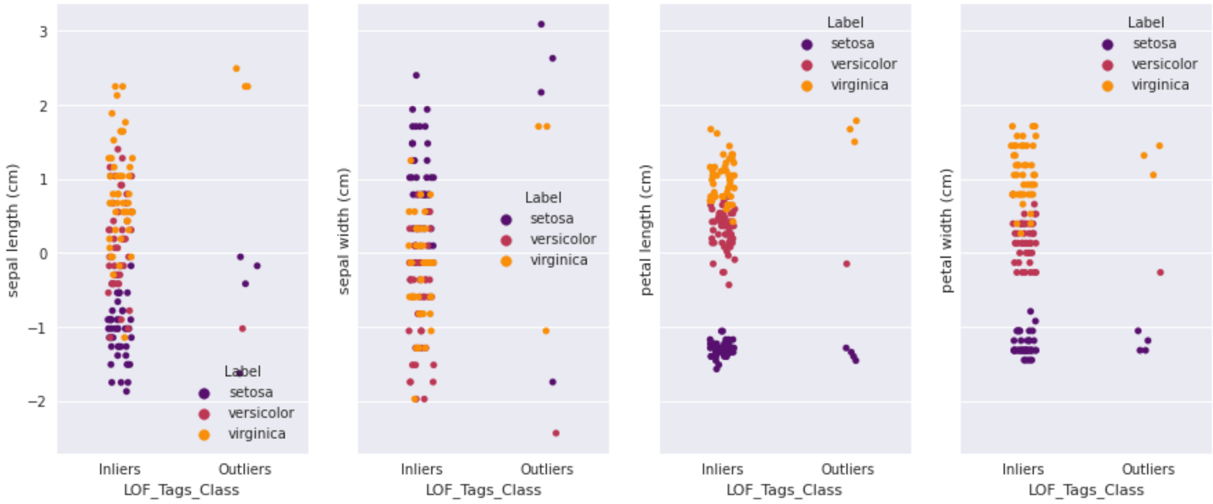
In [33]:
```
with plt.style.context('seaborn'):
    fig, ax = plt.subplots(1,4,figsize=(16,7),sharex=False,sharey=True)
    sns.histplot(data=iris_lof_result,x='sepal length (cm)',hue='LOF_Tags_Class',pal
    sns.histplot(data=iris_lof_result,x='sepal width (cm)',hue='LOF_Tags_Class',pale
    sns.histplot(data=iris_lof_result,x='petal length (cm)',hue='LOF_Tags_Class',pal
    sns.histplot(data=iris_lof_result,x='petal width (cm)',hue='LOF_Tags_Class',pale
```



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

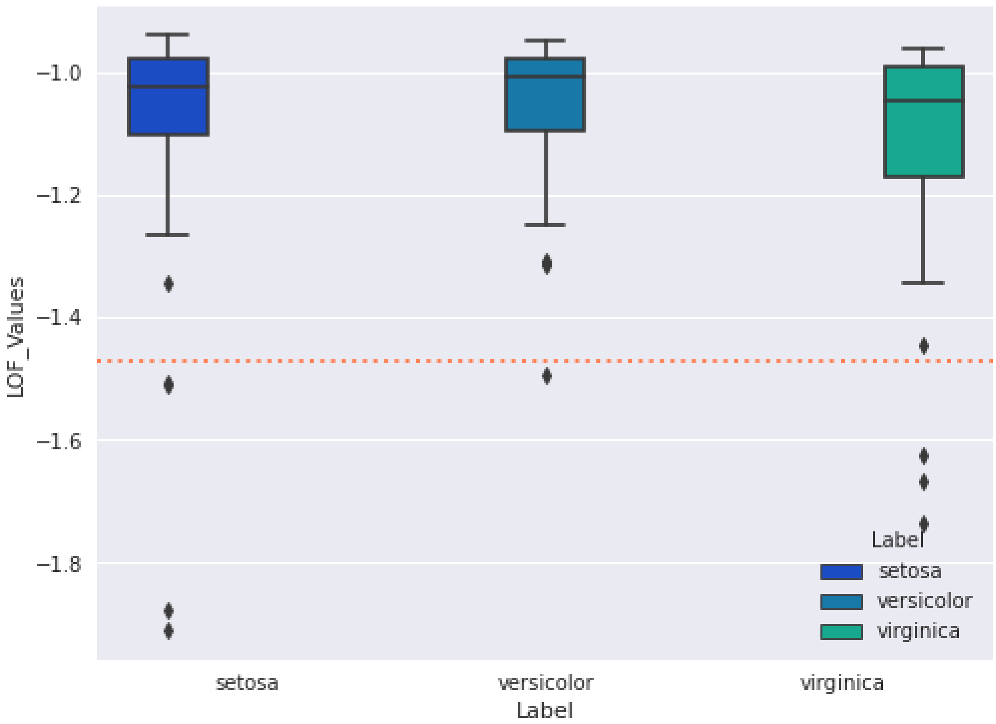
In [34]:
```
iris_classes = {0:'setosa',1:'versicolor',2:'virginica'}
iris_lof_result['Label'] = iris_lof_result['Label'].apply(lambda val : iris_classes.
```

In [35]:
```
with plt.style.context('seaborn'):
    fig, ax = plt.subplots(1,4,figsize=(15,6),sharex=True,sharey=True)
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='sepal length (cm)',hue=
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='sepal width (cm)',hue='
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='petal length (cm)',hue=
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='petal width (cm)',hue='
```



In this case with `contamination == 0.05` and `20 NN` , based on that LOF looks to work good in capturing the local and overall outliers.

In [36]:
```
with plt.style.context('seaborn'):
    plt.figure(figsize=(8,6))
    sns.boxplot(data=iris_lof_result,x='Label',y='LOF_Values',hue='Label',palette='w
    plt.axhline(lof_iris_offset,color='coral',linestyle=':',linewidth=2)
```



Above showing the outliers captured on the basis of the Manhattan Distance used by LOF, out of these only one point belongs to Versicolor class.

`CASE-II`

Neighbors = 20 & Contamination = 0.05 or 5% & Leaf_Size = 15 & Distance Metric = Euclidean

In [37]:
```
print(X_iris_df_st.shape)

iris_contam = 0.05
print(X_iris_df.shape[0]*iris_contam)
```

```
(150, 4)
7.5
```

In [38]:
```
lof_iris = LocalOutlierFactor(n_neighbors=20,algorithm='kd_tree',leaf_size=15,contam
                              novelty=False,metric='minkowski',p=2)
```

In [39]:
```
lof_iris_pred = lof_iris.fit_predict(X_iris_df_st)
lof_iris_pred.shape
```

Out[39]:
```
(150,)
```

In [40]:
```
pred_dict = {1:0, -1:1}
lof_iris_pred = [pred_dict.get(val) for val in lof_iris_pred]
```

Here, LOF returns the results in the form of [1, -1] where 1's are considered as inliers and -1 are considered as Outliers.

In [41]:
```
np.unique(lof_iris_pred), np.bincount(lof_iris_pred)
```

Out[41]:
```
(array([0, 1]), array([142,   8], dtype=int64))
```

In [42]:
```
lof_iris_vals = lof_iris.negative_outlier_factor_          ## -ve lof value
```

LOF will always returns the -ve factor score and the inliers generally have score close to 1 i.e. close to -1, whereas Outliers tends to have a larger score.

In [43]:
```
lof_iris_offset = lof_iris.offset_        ## Threshold Value
lof_iris_offset
```

Out[43]:
```
-1.4412048348442545
```

This is the negative threshold value generated by LOF if any value is greater than it then such a record will be considered as an Outlier.

In [44]:
```
iris_lof_result = pd.concat([X_iris_df_st,y_iris_df,
           pd.DataFrame(lof_iris_vals,columns=['LOF_Values']),
           pd.DataFrame(lof_iris_pred,columns=['LOF_Tags'])],axis=1)
```

In [45]:
```
iris_lof_result[iris_lof_result['LOF_Values'] < lof_iris_offset]
```

Out[45]:

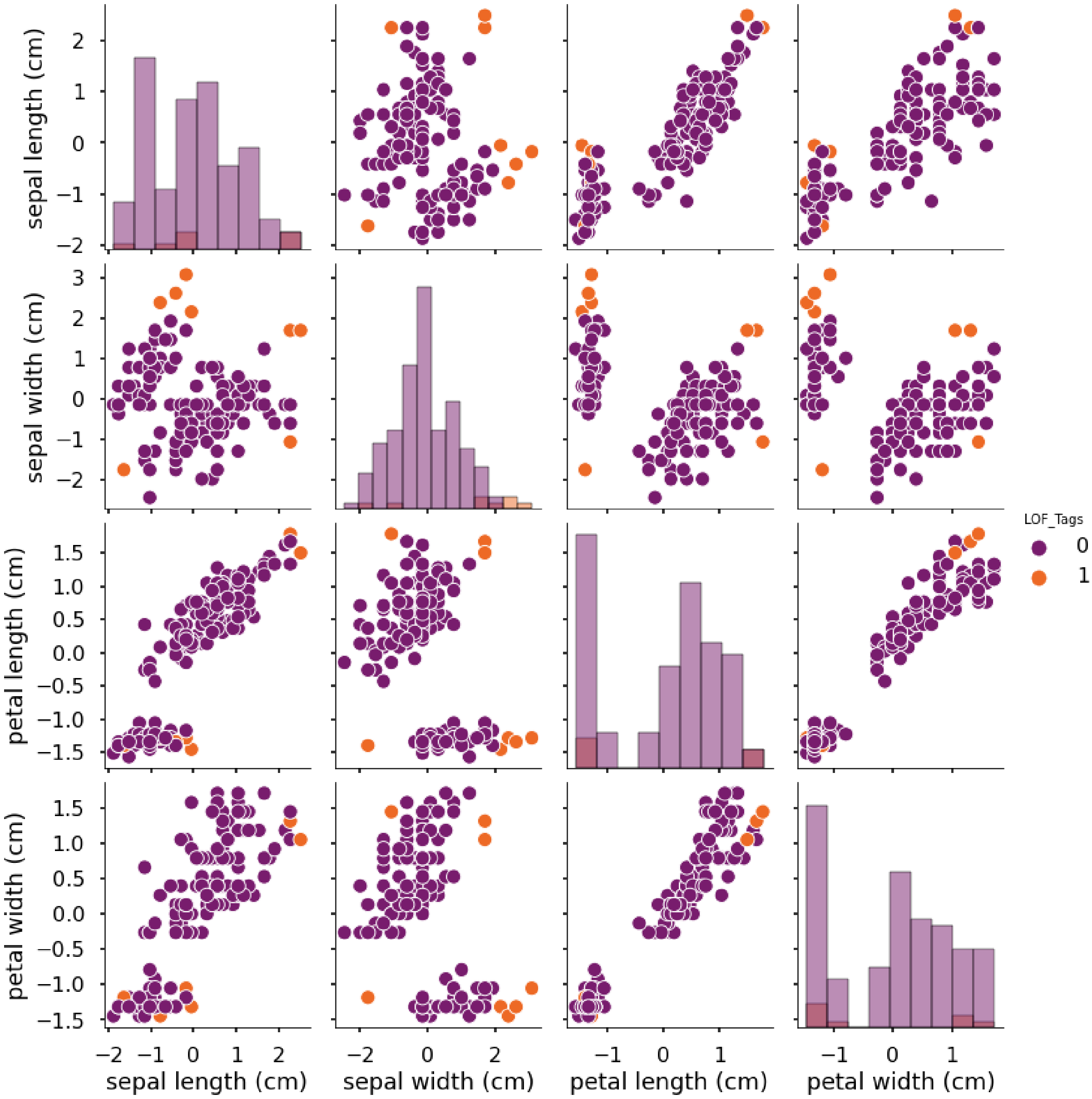
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Label	LOF_Values	LOF_Tags
14	-0.052506	2.169988	-1.453901	-1.315444	0	-1.495164	1
15	-0.173674	3.090775	-1.283389	-1.052180	0	-2.068567	1
32	-0.779513	2.400185	-1.283389	-1.447076	0	-1.456299	1
33	-0.416010	2.630382	-1.340227	-1.315444	0	-1.637883	1
41	-1.627688	-1.743357	-1.397064	-1.183812	0	-1.912040	1
117	2.249683	1.709595	1.672157	1.317199	2	-1.822850	1
118	2.249683	-1.052767	1.785832	1.448832	2	-1.602136	1
131	2.492019	1.709595	1.501645	1.053935	2	-1.883861	1

Now, if we see the -ve scores then they are not so highly away or larger than the offset value but if we compare these values with -1 then yes the

above records are a bit away from the generally considered inliers.

- Here, the dataset was not having any labels for the outliers, so this is completely driven based on distance calculation
- Plotting the records with above generated score will tell the better story

In [46]:
```
with plt.style.context('seaborn-poster'):
    g = sns.pairplot(data=iris_lof_result[['sepal length (cm)','sepal width (cm)','p
                     hue='LOF_Tags',palette='inferno',height=3,aspect=0.9,diag_kind='his
```



As, the plot is self-explanatory because the data points which are away(separated from others) from the clusters or at the edges/boundaries of it are labelled as Outliers. This is exactly the way LOF behaves.

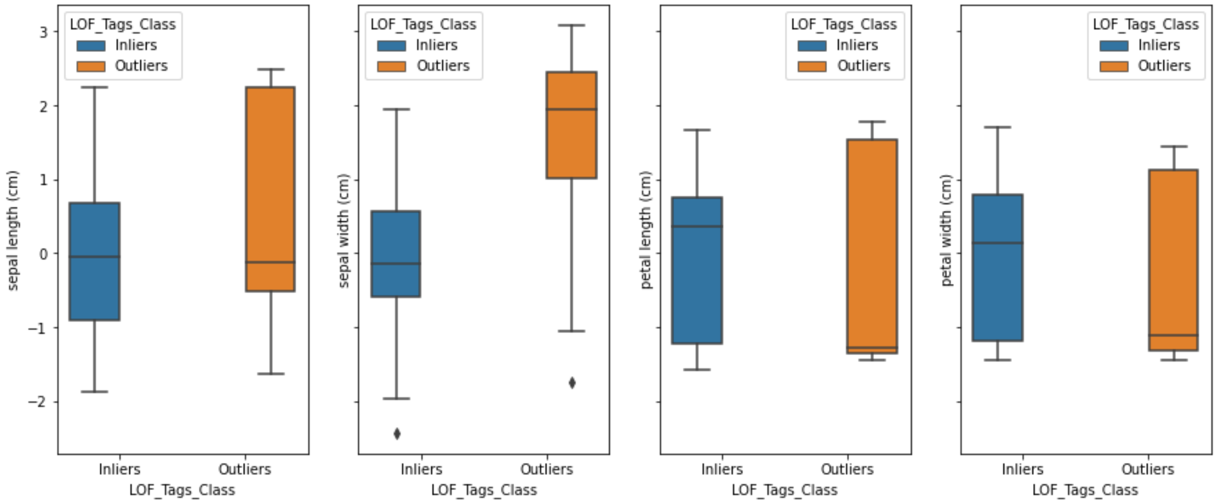
More analysis

In [47]:
```
iris_lof_result.columns
```

Out[47]:
```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)', 'Label', 'LOF_Values', 'LOF_Tags'],
      dtype='object')
```

In [48]:
```
iris_lof_result['LOF_Tags_Class'] = iris_lof_result['LOF_Tags'].apply(lambda val: lb
```

In [49]:
```
fig, ax = plt.subplots(1,4,sharex=True,sharey=True,squeeze=True,figsize=(15,6))
sns.boxplot(data=iris_lof_result,x='LOF_Tags_Class',y='sepal length (cm)',hue='LOF_T
sns.boxplot(data=iris_lof_result,x='LOF_Tags_Class',y='sepal width (cm)',hue='LOF_Ta
sns.boxplot(data=iris_lof_result,x='LOF_Tags_Class',y='petal length (cm)',hue='LOF_T
sns.boxplot(data=iris_lof_result,x='LOF_Tags_Class',y='petal width (cm)',hue='LOF_Ta
plt.show()
```



In [50]:
```
iris_lof_result.groupby(['LOF_Tags_Class'])[['sepal length (cm)', 'sepal width (cm)'
```

Out[50]:

LOF_Tags_Class	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
Inliers	-1.870024	-2.433947	-1.567576	-1.447076
Outliers	-1.627688	-1.743357	-1.453901	-1.447076

In [51]:
```
iris_lof_result.groupby(['LOF_Tags_Class'])[['sepal length (cm)', 'sepal width (cm)'
```

Out[51]:

LOF_Tags_Class	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
Inliers	-0.052506	-0.131979	0.364896	0.132510
Outliers	-0.113090	1.939791	-1.283389	-1.117996

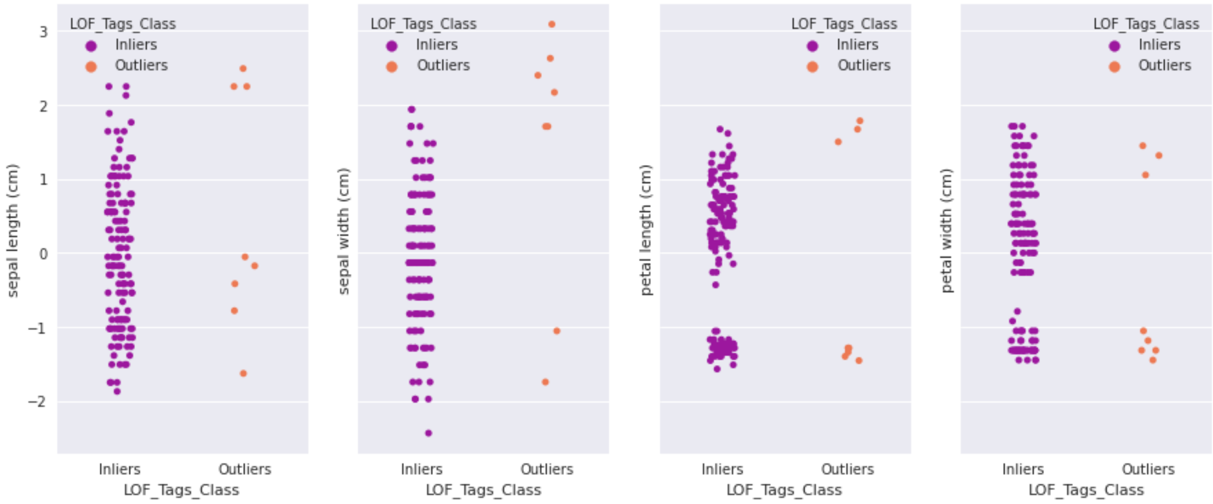
In [52]:
```
iris_lof_result.groupby(['LOF_Tags_Class'])[['sepal length (cm)', 'sepal width (cm)'
```

Out[52]:

LOF_Tags_Class	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
Inliers	2.249683	1.939791	1.672157	1.712096
Outliers	2.492019	3.090775	1.785832	1.448832

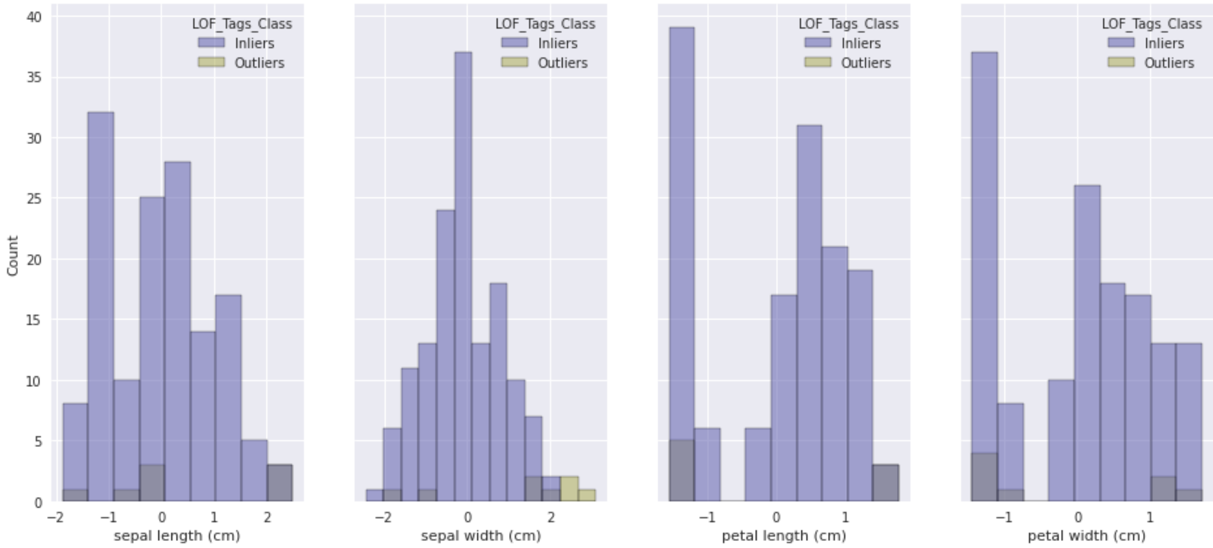
So, all the features have good point variations in the 5 number summary of Inliers and Outliers.

In [53]:
```
with plt.style.context('seaborn'):
    fig, ax = plt.subplots(1,4,figsize=(15,6),sharex=True,sharey=True)
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='sepal length (cm)',hue=
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='sepal width (cm)',hue='
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='petal length (cm)',hue=
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='petal width (cm)',hue='
```



If we look at the outliers then they are clearly the extreme points which are either at the edges or boundaries of the clusters.

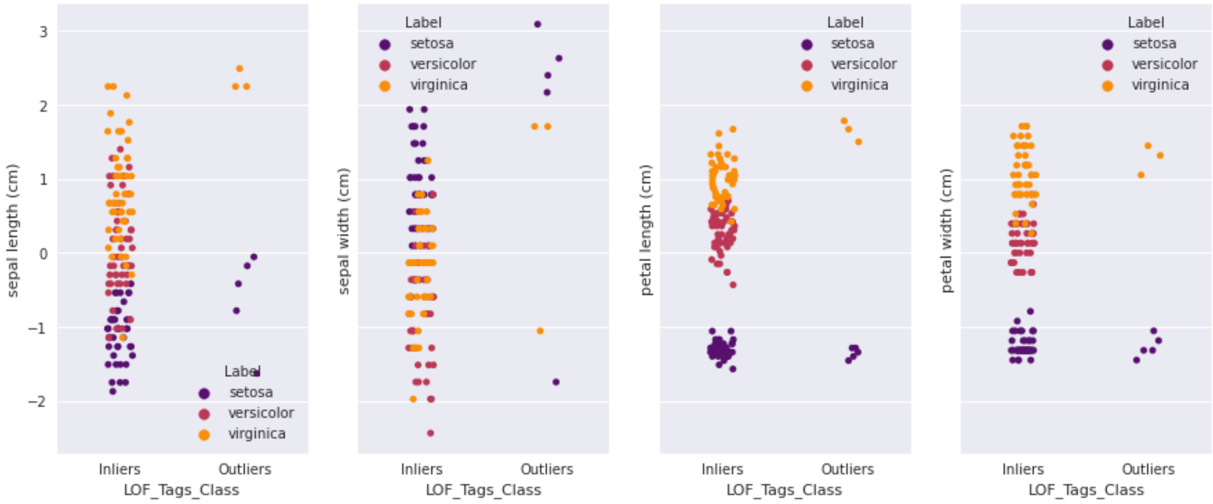
In [54]:
```
with plt.style.context('seaborn'):
    fig, ax = plt.subplots(1,4,figsize=(16,7),sharex=False,sharey=True)
    sns.histplot(data=iris_lof_result,x='sepal length (cm)',hue='LOF_Tags_Class',pal
    sns.histplot(data=iris_lof_result,x='sepal width (cm)',hue='LOF_Tags_Class',pale
    sns.histplot(data=iris_lof_result,x='petal length (cm)',hue='LOF_Tags_Class',pal
    sns.histplot(data=iris_lof_result,x='petal width (cm)',hue='LOF_Tags_Class',pale
```



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

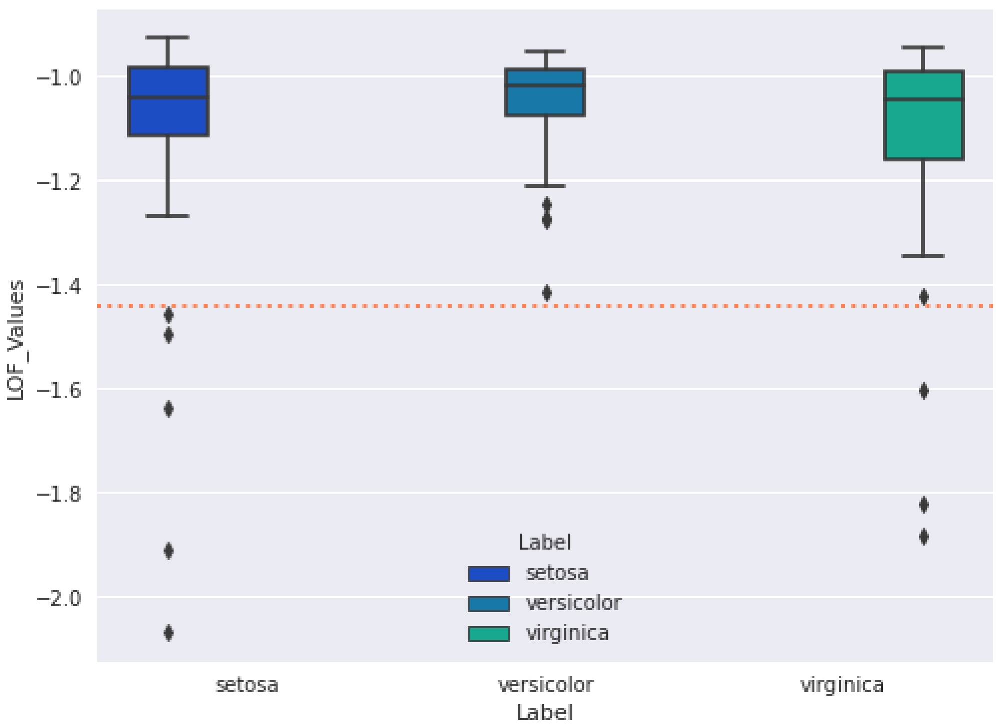
In [55]:
```
iris_classes = {0:'setosa',1:'versicolor',2:'virginica'}
iris_lof_result['Label'] = iris_lof_result['Label'].apply(lambda val : iris_classes.
```

In [56]:
```
with plt.style.context('seaborn'):
    fig, ax = plt.subplots(1,4,figsize=(15,6),sharex=True,sharey=True)
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='sepal length (cm)',hue=
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='sepal width (cm)',hue='
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='petal length (cm)',hue=
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='petal width (cm)',hue='
```



In this case with `contamination == 0.05` and `20 NN` , based on that LOF looks to work good in capturing the local and overall outliers.

In [57]:
```
with plt.style.context('seaborn'):
    plt.figure(figsize=(8,6))
    sns.boxplot(data=iris_lof_result,x='Label',y='LOF_Values',hue='Label',palette='w
    plt.axhline(lof_iris_offset,color='coral',linestyle=':',linewidth=2)
```



Above showing the outliers captured on the basis of the Manhattan Distance used by LOF, out of these no point belongs to Versicolor class.

`CASE-III`

Neighbors = 20 & Contamination = 0.05 or 5% & Leaf_Size = 15 & `Distance Metric = Precomputed Mahalanobis Distance Metric` with shape nxn where n is the number of observations in the dataset

`NOTE :: MAHALANOBIS is scale-invariant`

In [58]:
```
X_iris_df.head()
```

Out[58]:

	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

Generating Co-variance Symmetric Matrix of the given dataset

In [59]:
```
X_iris_df_st.head()
```

Out[59]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	-0.900681	1.019004	-1.340227	-1.315444
1	-1.143017	-0.131979	-1.340227	-1.315444
2	-1.385353	0.328414	-1.397064	-1.315444
3	-1.506521	0.098217	-1.283389	-1.315444
4	-1.021849	1.249201	-1.340227	-1.315444

Biased Co-variance Symmetric Matrix

In [60]:
```
pd.DataFrame((1/X_iris_df_st.shape[0]) * (X_iris_df_st.T @ X_iris_df_st))
```

Out[60]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
sepal length (cm)	1.000000	-0.117570	0.871754	0.817941
sepal width (cm)	-0.117570	1.000000	-0.428440	-0.366126
petal length (cm)	0.871754	-0.428440	1.000000	0.962865
petal width (cm)	0.817941	-0.366126	0.962865	1.000000

In [61]:
```
pd.DataFrame(np.cov(X_iris_df_st,rowvar=False,bias=True))
```

Out[61]:

	0	1	2	3
0	1.000000	-0.117570	0.871754	0.817941
1	-0.117570	1.000000	-0.428440	-0.366126
2	0.871754	-0.428440	1.000000	0.962865
3	0.817941	-0.366126	0.962865	1.000000

Un-Biased Co-variance Symmetric Matrix

In [62]:
```
pd.DataFrame((1/(X_iris_df_st.shape[0]-1)) * (X_iris_df_st.T @ X_iris_df_st))
```

Out[62]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
sepal length (cm)	1.006711	-0.118359	0.877604	0.823431

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
sepal width (cm)	-0.118359	1.006711	-0.431316	-0.368583
petal length (cm)	0.877604	-0.431316	1.006711	0.969328
petal width (cm)	0.823431	-0.368583	0.969328	1.006711

In [63]:

```
pd.DataFrame(np.cov(X_iris_df_st,rowvar=False,bias=False))
```

Out[63]:

	0	1	2	3
0	1.006711	-0.118359	0.877604	0.823431
1	-0.118359	1.006711	-0.431316	-0.368583
2	0.877604	-0.431316	1.006711	0.969328
3	0.823431	-0.368583	0.969328	1.006711

Computing Mahalanobis Distance Metric

In [64]:

```
precomp_dist = pd.DataFrame(((X_iris_df - np.mean(X_iris_df)).values @ scipy.linalg.
             @ ((X_iris_df - np.mean(X_iris_df)).T).values)

precomp_dist_sq = np.square(precomp_dist.copy(deep=True))  # Squaring the distances;
precomp_dist_sq.head()
```

Out[64]:

	0	1	2	3	4	5	6	7	8	9
0	4.555953	2.479101	2.685287	1.704051	4.512709	5.574929	2.484211	3.555688	1.126132	2.598991
1	2.479101	8.117477	3.979952	2.545627	1.032946	0.136103	0.963860	2.065696	4.442087	4.39571
2	2.685287	3.979952	4.331971	4.039356	2.684836	1.285841	3.997162	2.735031	5.289667	3.24630
3	1.704051	2.545627	4.039356	6.014176	2.476593	0.426890	4.863778	2.642637	7.430052	3.933751
4	4.512709	1.032946	2.684836	2.476593	6.062205	7.434547	4.496747	4.024086	1.384612	2.21633

5 rows × 150 columns

In [65]:

```
precomp_dist_sq.shape
```

Out[65]:

```
(150, 150)
```

In [66]:

```
lof_precomp_iris = LocalOutlierFactor(n_neighbors=20,leaf_size=15,contamination=iris
                                      metric='precomputed',algorithm='auto',n_jobs=N
```

In [67]:

```
lof_precomp_iris.fit_predict(precomp_dist_sq)
```

Out[67]:

```
array([ 1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,
        1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1, -1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,
        1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1])
```

In [68]:
```
lof_precomp_iris.offset_
```

Out[68]:
```
-1.407113636384807
```

In [69]:
```
X_mahala_results = pd.concat([X_iris_df.copy(deep=True),y_iris_df],axis=1)
X_mahala_results.head()
```

Out[69]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Label
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [70]:
```
X_mahala_results['LOF_Values'] = lof_precomp_iris.negative_outlier_factor_
X_mahala_results['Outlier_Status'] = lof_precomp_iris.fit_predict(precomp_dist_sq)
X_mahala_results['Outlier_Status'] = X_mahala_results['Outlier_Status'].apply(lambda
```

In [71]:
```
X_mahala_results.head()
```

Out[71]:

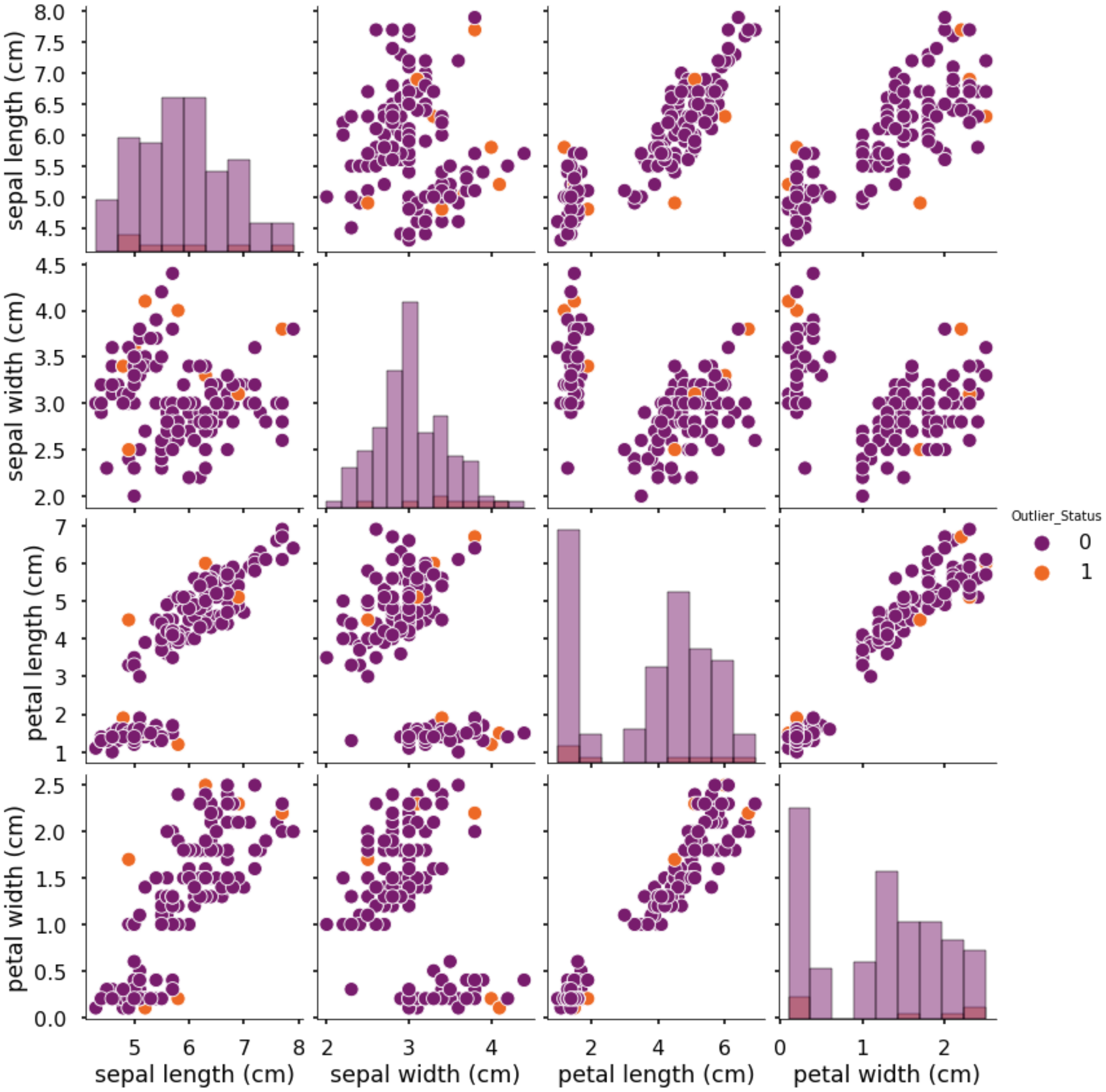
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Label	LOF_Values	Outlier_Status
0	5.1	3.5	1.4	0.2	0	-1.090641	0
1	4.9	3.0	1.4	0.2	0	-1.151355	0
2	4.7	3.2	1.3	0.2	0	-1.047989	0
3	4.6	3.1	1.5	0.2	0	-0.957528	0
4	5.0	3.6	1.4	0.2	0	-1.423120	1

In [72]:
```
X_mahala_results[X_mahala_results['LOF_Values'] < lof_precomp_iris.offset_]
```

Out[72]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Label	LOF_Values	Outlier_Status
4	5.0	3.6	1.4	0.2	0	-1.423120	1
14	5.8	4.0	1.2	0.2	0	-1.440691	1
24	4.8	3.4	1.9	0.2	0	-1.457974	1
32	5.2	4.1	1.5	0.1	0	-1.750358	1
100	6.3	3.3	6.0	2.5	2	-1.450683	1
106	4.9	2.5	4.5	1.7	2	-1.778370	1
117	7.7	3.8	6.7	2.2	2	-1.938577	1
141	6.9	3.1	5.1	2.3	2	-1.883941	1

In [73]:
```
with plt.style.context('seaborn-poster'):
    g = sns.pairplot(data=X_mahala_results[['sepal length (cm)','sepal width (cm)','
                     hue='Outlier_Status',palette='inferno',height=3,aspect=0.9,diag_kin
```



With Mahalanobis, I have observed quite a bit different results as compared to Euclidean and Manhattan. Some of points in above plot are surrounded by inliers but still they are labelled as 1(means outliers).

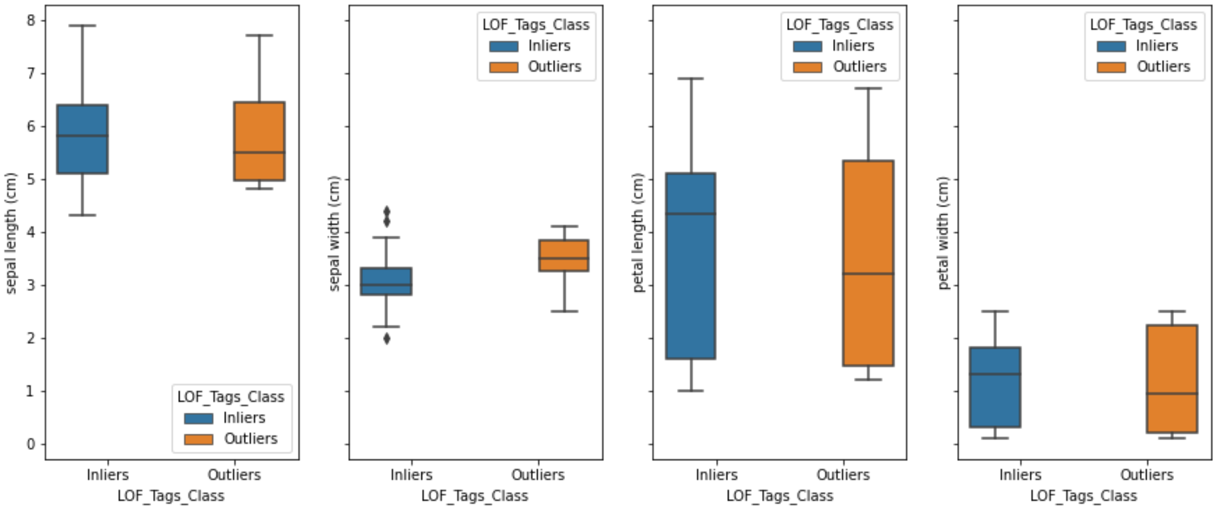
More analysis

In [74]:
```
X_mahala_results.columns
```

Out[74]:
```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)', 'Label', 'LOF_Values', 'Outlier_Status'],
      dtype='object')
```

In [75]:
```
X_mahala_results['LOF_Tags_Class'] = X_mahala_results['Outlier_Status'].apply(lambda
```

In [76]:
```
fig, ax = plt.subplots(1,4,sharex=True,sharey=True,squeeze=True,figsize=(15,6))
sns.boxplot(data=X_mahala_results,x='LOF_Tags_Class',y='sepal length (cm)',hue='LOF_
sns.boxplot(data=X_mahala_results,x='LOF_Tags_Class',y='sepal width (cm)',hue='LOF_T
sns.boxplot(data=X_mahala_results,x='LOF_Tags_Class',y='petal length (cm)',hue='LOF_
sns.boxplot(data=X_mahala_results,x='LOF_Tags_Class',y='petal width (cm)',hue='LOF_T
plt.show()
```



In [77]:
```
X_mahala_results.groupby(['LOF_Tags_Class'])[['sepal length (cm)', 'sepal width (cm)
```

Out[77]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
LOF_Tags_Class				
Inliers	4.3	2.0	1.0	0.1
Outliers	4.8	2.5	1.2	0.1

In [78]:
```
X_mahala_results.groupby(['LOF_Tags_Class'])[['sepal length (cm)', 'sepal width (cm)
```

Out[78]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
LOF_Tags_Class				
Inliers	5.8	3.0	4.35	1.30
Outliers	5.5	3.5	3.20	0.95

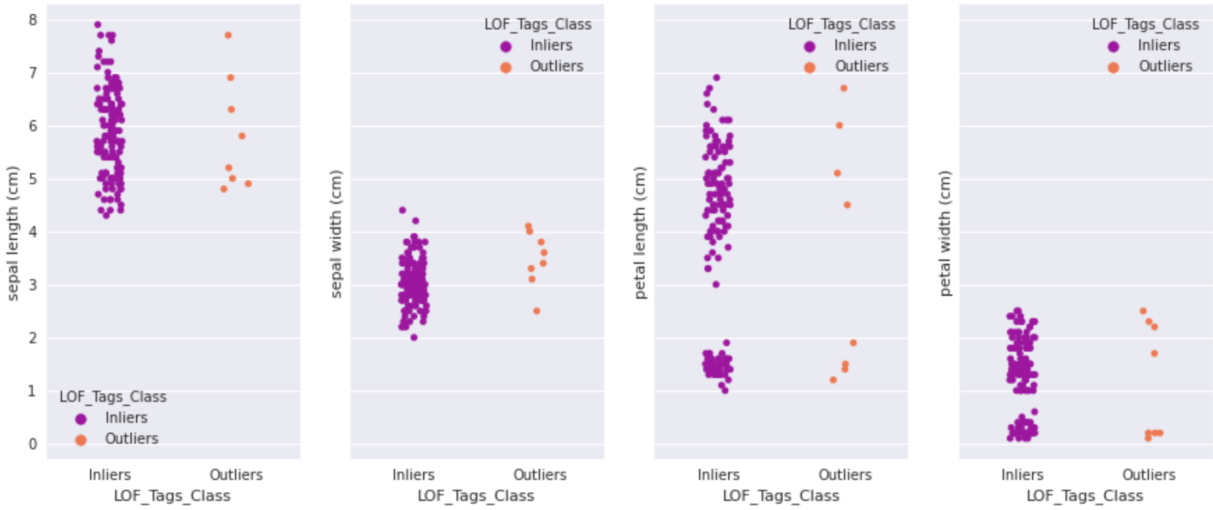
In [79]:
```
X_mahala_results.groupby(['LOF_Tags_Class'])[['sepal length (cm)', 'sepal width (cm)
```

Out[79]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
LOF_Tags_Class				
Inliers	7.9	4.4	6.9	2.5
Outliers	7.7	4.1	6.7	2.5

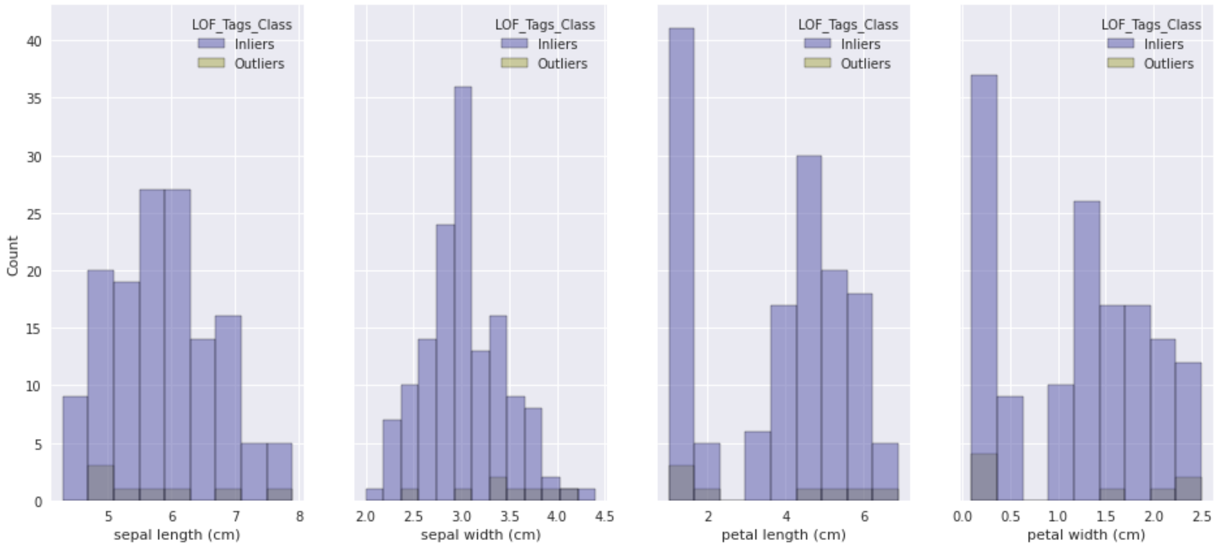
So, all the features have good point variations in the 5 number summary of Inliers and Outliers.

In [80]:
```
with plt.style.context('seaborn'):
    fig, ax = plt.subplots(1,4,figsize=(15,6),sharex=True,sharey=True)
    sns.stripplot(data=X_mahala_results,x='LOF_Tags_Class',y='sepal length (cm)',hue
    sns.stripplot(data=X_mahala_results,x='LOF_Tags_Class',y='sepal width (cm)',hue=
    sns.stripplot(data=X_mahala_results,x='LOF_Tags_Class',y='petal length (cm)',hue
    sns.stripplot(data=X_mahala_results,x='LOF_Tags_Class',y='petal width (cm)',hue=
```



If we look at the outliers then some of the oultiers are even from the area where we have good amount of inliers.

In [81]:
```
with plt.style.context('seaborn'):
    fig, ax = plt.subplots(1,4,figsize=(16,7),sharex=False,sharey=True)
    sns.histplot(data=X_mahala_results,x='sepal length (cm)',hue='LOF_Tags_Class',pa
    sns.histplot(data=X_mahala_results,x='sepal width (cm)',hue='LOF_Tags_Class',pal
    sns.histplot(data=X_mahala_results,x='petal length (cm)',hue='LOF_Tags_Class',pa
    sns.histplot(data=X_mahala_results,x='petal width (cm)',hue='LOF_Tags_Class',pal
```

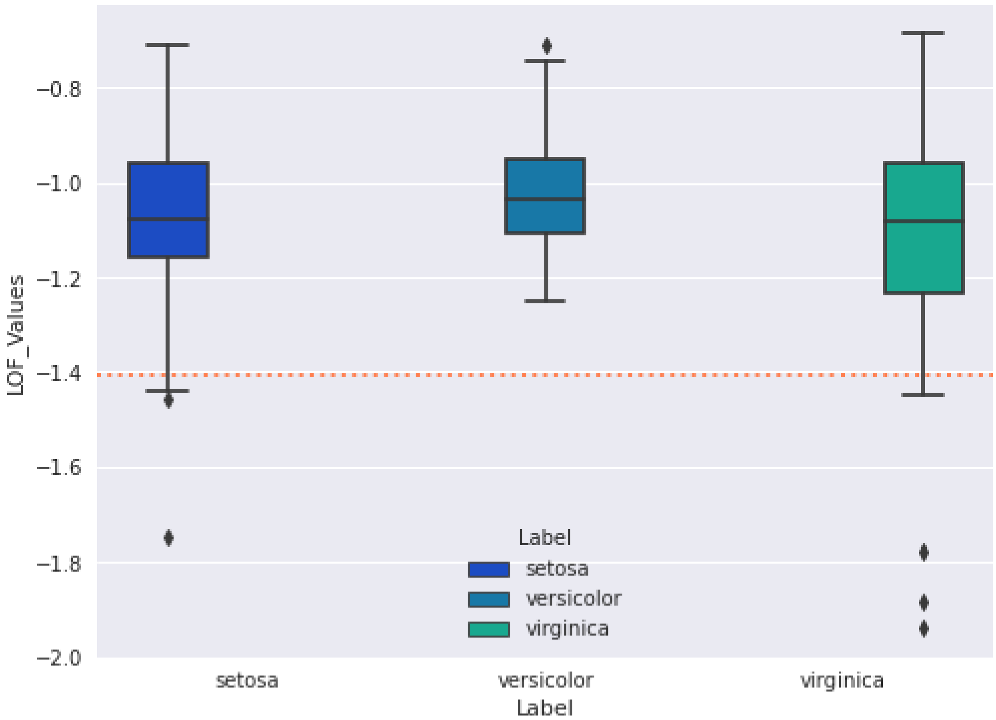


Here, the results are quite different as compared to Euclidean and Manhattan. Outliers here are not the extreme points of the dataset whereas some of them are even from the area where we have good amount of inliers.**

In [82]:
```
iris_classes = {0:'setosa',1:'versicolor',2:'virginica'}
X_mahala_results['Label'] = X_mahala_results['Label'].apply(lambda val : iris_classe
```

In [83]:
```
with plt.style.context('seaborn'):
    fig, ax = plt.subplots(1,4,figsize=(15,6),sharex=True,sharey=True)
    sns.stripplot(data=X_mahala_results,x='LOF_Tags_Class',y='sepal length (cm)',hue
    sns.stripplot(data=X_mahala_results,x='LOF_Tags_Class',y='sepal width (cm)',hue=
    sns.stripplot(data=X_mahala_results,x='LOF_Tags_Class',y='petal length (cm)',hue
    sns.stripplot(data=X_mahala_results,x='LOF_Tags_Class',y='petal width (cm)',hue=
```

In [84]:
```
with plt.style.context('seaborn'):
    plt.figure(figsize=(8,6))
    sns.boxplot(data=X_mahala_results,x='Label',y='LOF_Values',hue='Label',palette='
    plt.axhline(lof_precomp_iris.offset_,color='coral',linestyle=':',linewidth=2)
```



Above showing the outliers captured on the basis of the Mahalanobis Distance used by LOF, out of these no point belongs to Versicolor class.

CASE-IV

Neighbors = 20 & Contamination = 0.1 or 1% & Leaf_Size = 15 & Euclidean as Distance Metric

In [85]:
```
iris_contam = 0.1
lof_iris = LocalOutlierFactor(n_neighbors=20,algorithm='kd_tree',leaf_size=15,contam
```

In [86]:
```
lof_iris_pred = lof_iris.fit_predict(X_iris_df)
lof_iris_pred.shape
```

Out[86]: (150,)

In [87]:
```
pred_dict = {1:0, -1:1}          # LOF returns the results in form of [1, -1] ; 1's
lof_iris_pred = [pred_dict.get(val) for val in lof_iris_pred]
```

In [88]:
```
np.unique(lof_iris_pred), np.bincount(lof_iris_pred)
```

Out[88]:
```
(array([0, 1]), array([135,  15], dtype=int64))
```

In [89]:
```
lof_iris_vals = lof_iris.negative_outlier_factor_          ## -ve lof value
```

In [90]:
```
lof_iris_offset = lof_iris.offset_        ## Threshold Value
lof_iris_offset
```

Out[90]:
```
-1.3060565525969516
```

The new offset value with higher percentage of contamination has increased quite significantly.

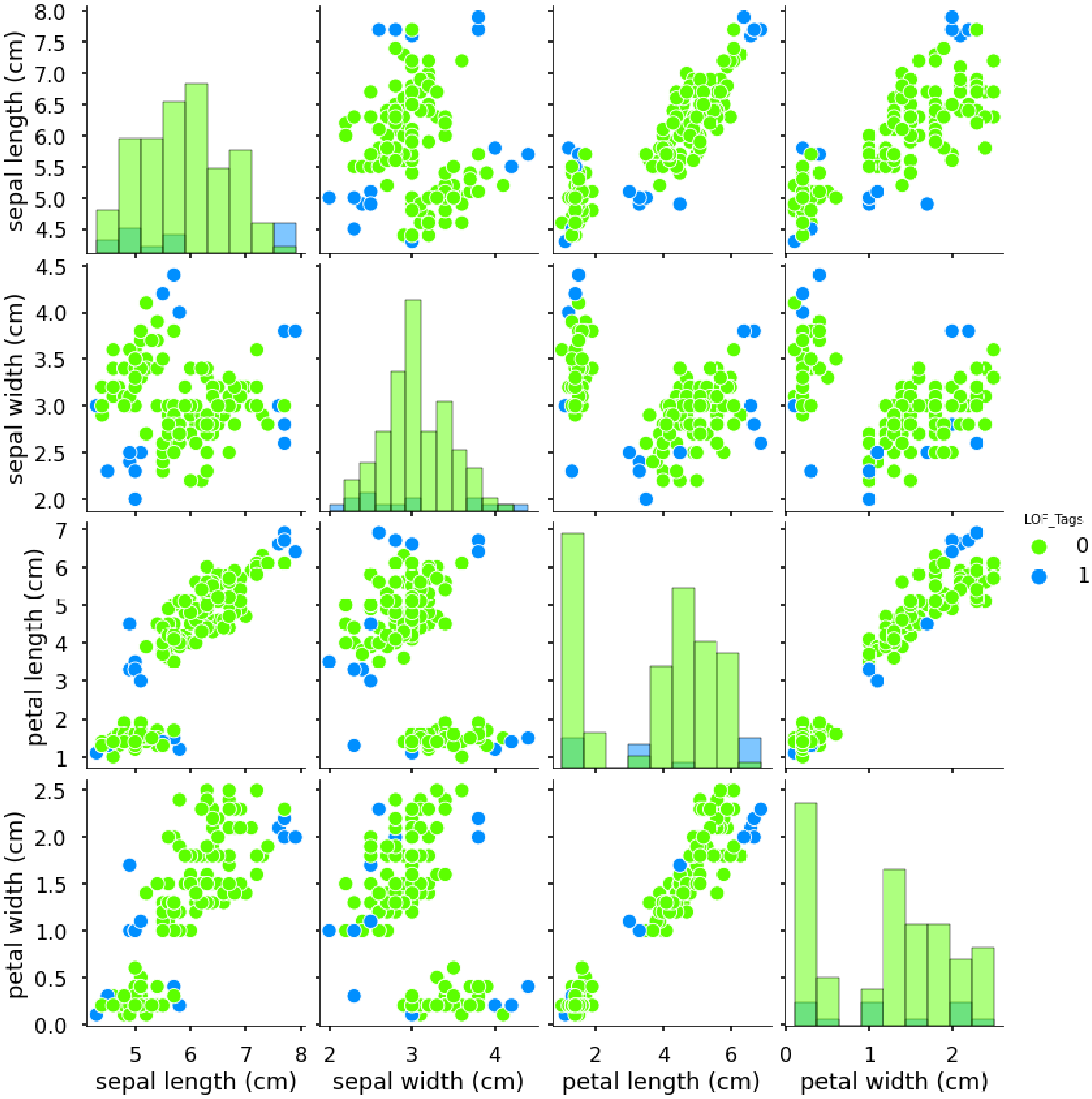
In [91]:
```
iris_lof_result = pd.concat([X_iris_df,y_iris_df,
           pd.DataFrame(lof_iris_vals,columns=['LOF_Values']),
           pd.DataFrame(lof_iris_pred,columns=['LOF_Tags'])],axis=1)
```

In [92]:
```
iris_lof_result[iris_lof_result['LOF_Values'] < lof_iris_offset]
```

Out[92]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Label	LOF_Values	LOF_Tags
13	4.3	3.0	1.1	0.1	0	-1.325687	1
14	5.8	4.0	1.2	0.2	0	-1.480203	1
15	5.7	4.4	1.5	0.4	0	-1.671872	1
33	5.5	4.2	1.4	0.2	0	-1.365882	1
41	4.5	2.3	1.3	0.3	0	-1.777582	1
57	4.9	2.4	3.3	1.0	1	-1.432891	1
60	5.0	2.0	3.5	1.0	1	-1.395791	1
93	5.0	2.3	3.3	1.0	1	-1.401342	1
98	5.1	2.5	3.0	1.1	1	-1.511868	1
105	7.6	3.0	6.6	2.1	2	-1.399554	1
106	4.9	2.5	4.5	1.7	2	-1.373468	1
117	7.7	3.8	6.7	2.2	2	-1.572990	1
118	7.7	2.6	6.9	2.3	2	-1.624653	1
122	7.7	2.8	6.7	2.0	2	-1.491365	1
131	7.9	3.8	6.4	2.0	2	-1.547288	1

In [93]:
```
with plt.style.context('seaborn-poster'):
    g = sns.pairplot(data=iris_lof_result[['sepal length (cm)','sepal width (cm)','p
                 hue='LOF_Tags',palette='gist_rainbow',height=3,aspect=0.9,diag_kind
```



Outliers marking has become a bit aggressive in this case as some of the points near to the clusters are also marked as outliers.

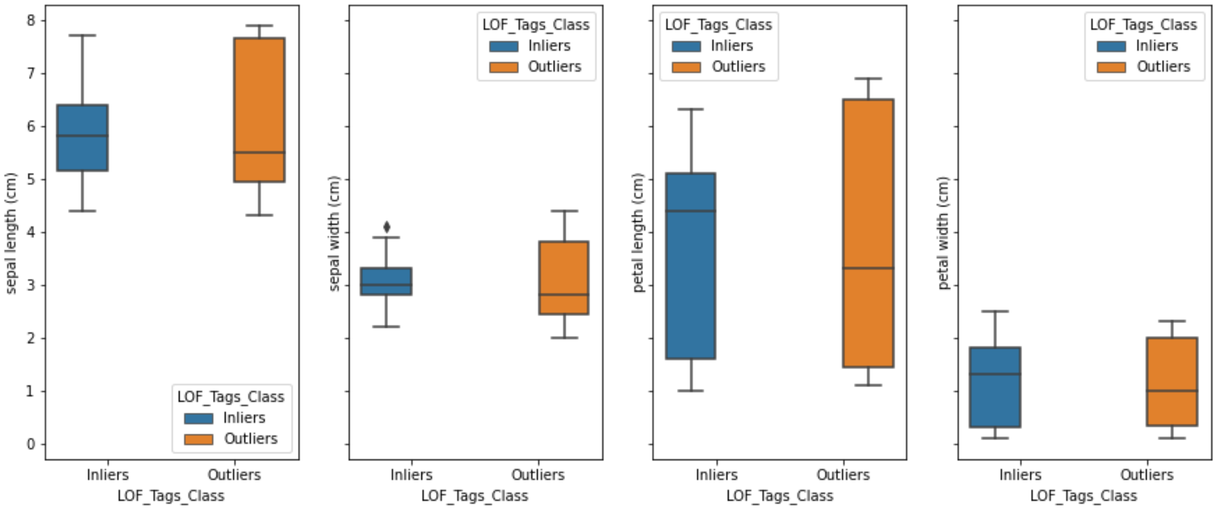
More analysis

In [94]:
```
iris_lof_result.columns
```

Out[94]:
```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)', 'Label', 'LOF_Values', 'LOF_Tags'],
      dtype='object')
```

In [95]:
```
iris_lof_result['LOF_Tags_Class'] = iris_lof_result['LOF_Tags'].apply(lambda val: lb
```

In [96]:
```
fig, ax = plt.subplots(1,4,sharex=True,sharey=True,squeeze=True,figsize=(15,6))
sns.boxplot(data=iris_lof_result,x='LOF_Tags_Class',y='sepal length (cm)',hue='LOF_T
sns.boxplot(data=iris_lof_result,x='LOF_Tags_Class',y='sepal width (cm)',hue='LOF_Ta
sns.boxplot(data=iris_lof_result,x='LOF_Tags_Class',y='petal length (cm)',hue='LOF_T
sns.boxplot(data=iris_lof_result,x='LOF_Tags_Class',y='petal width (cm)',hue='LOF_Ta
plt.show()
```



In [97]:
```
iris_lof_result.groupby(['LOF_Tags_Class'])[['sepal length (cm)', 'sepal width (cm)'
```

Out[97]:

LOF_Tags_Class	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
Inliers	4.4	2.2	1.0	0.1
Outliers	4.3	2.0	1.1	0.1

In [98]:
```
iris_lof_result.groupby(['LOF_Tags_Class'])[['sepal length (cm)', 'sepal width (cm)'
```

Out[98]:

LOF_Tags_Class	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
Inliers	5.8	3.0	4.4	1.3
Outliers	5.5	2.8	3.3	1.0

In [99]:
```
iris_lof_result.groupby(['LOF_Tags_Class'])[['sepal length (cm)', 'sepal width (cm)'
```

Out[99]:

LOF_Tags_Class	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
Inliers	7.7	4.1	6.3	2.5
Outliers	7.9	4.4	6.9	2.3

Above boxplots shows us some good amount of variations in features other than Petal Width.

In [100…
```
with plt.style.context('seaborn'):
    fig, ax = plt.subplots(1,4,figsize=(15,6),sharex=True,sharey=True)
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='sepal length (cm)',hue=
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='sepal width (cm)',hue='
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='petal length (cm)',hue=
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='petal width (cm)',hue='
```

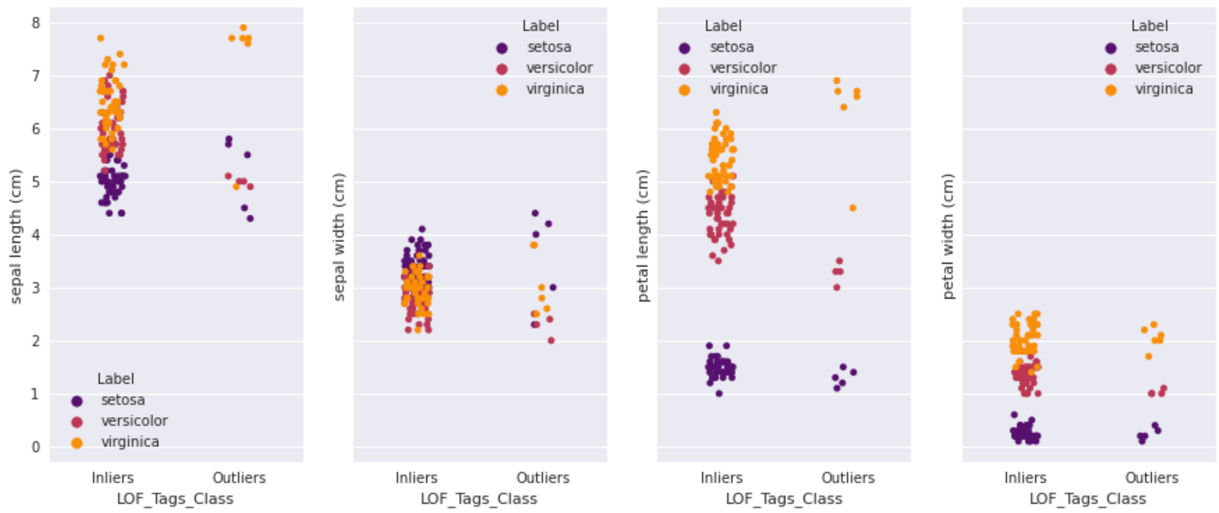
One thing to mention here is that as seen in the plots with contamination == 0.05, the outliers are picked from the same or nearby region with contamination == 0.1

In [101…

```
iris_classes = {0:'setosa',1:'versicolor',2:'virginica'}
iris_lof_result['Label'] = iris_lof_result['Label'].apply(lambda val : iris_classes.
```

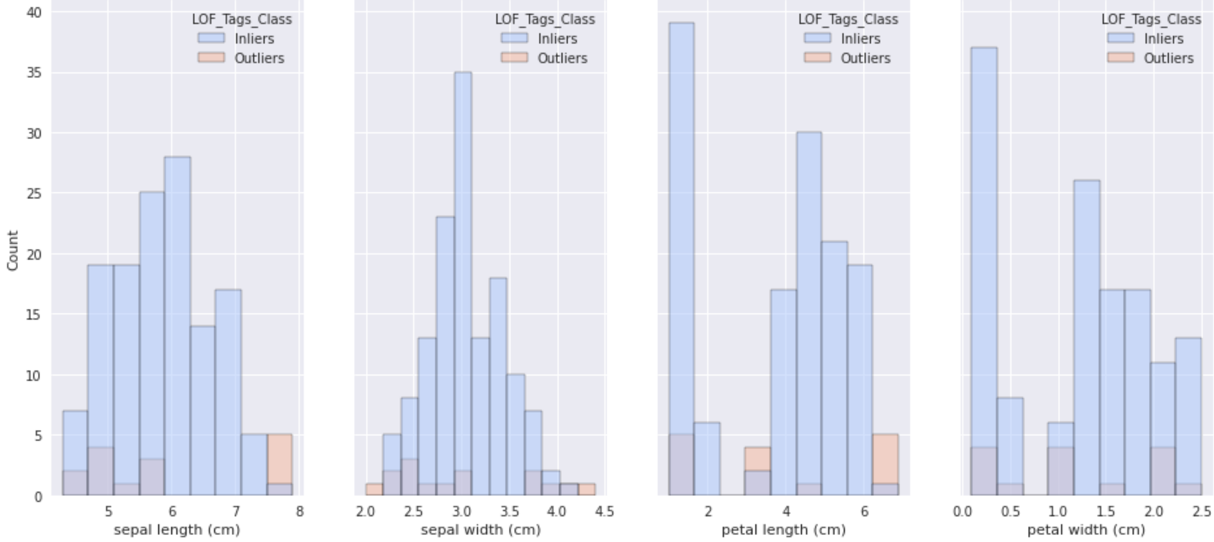
In [102…

```
with plt.style.context('seaborn'):
    fig, ax = plt.subplots(1,4,figsize=(15,6),sharex=True,sharey=True)
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='sepal length (cm)',hue=
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='sepal width (cm)',hue='
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='petal length (cm)',hue=
    sns.stripplot(data=iris_lof_result,x='LOF_Tags_Class',y='petal width (cm)',hue='
```



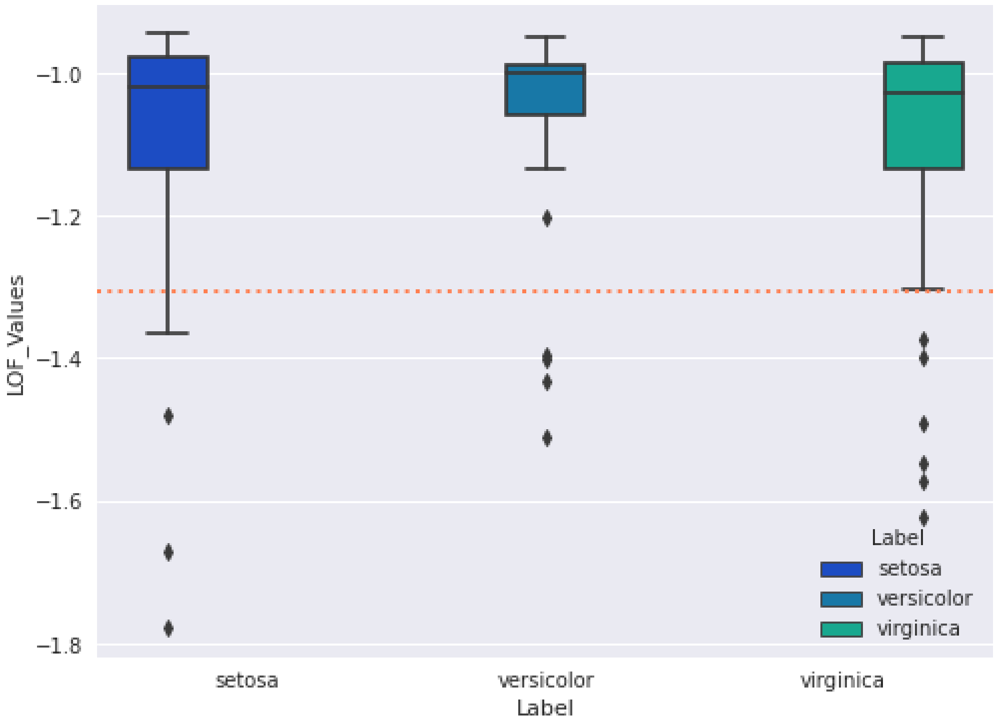
In [103…

```
with plt.style.context('seaborn'):
    fig, ax = plt.subplots(1,4,figsize=(16,7),sharex=False,sharey=True)
    sns.histplot(data=iris_lof_result,x='sepal length (cm)',hue='LOF_Tags_Class',pal
    sns.histplot(data=iris_lof_result,x='sepal width (cm)',hue='LOF_Tags_Class',pale
    sns.histplot(data=iris_lof_result,x='petal length (cm)',hue='LOF_Tags_Class',pal
    sns.histplot(data=iris_lof_result,x='petal width (cm)',hue='LOF_Tags_Class',pale
```



In [104…

```
with plt.style.context('seaborn'):
    plt.figure(figsize=(8,6))
    sns.boxplot(data=iris_lof_result,x='Label',y='LOF_Values',hue='Label',palette='w
    plt.axhline(lof_iris_offset,color='coral',linestyle=':',linewidth=2)
```



As, we can see here the offset value shifted a bit upwards thus few of the data points of Versicolor are also been labelled as Outliers.

Results_Analysis

In the above analysis across all 4 cases, I have observed that LOF is good in capturing the local and global outliers, however there are few things which we need to take into account before using it:

- **As it relies on distances between the points so we need to be sure with the distance metric that we are using**

  - **For example, when I used `Mahalanobis Distance` then I observed some of points which were surrounded by others points labelled as Outliers, however, this behavior is not observed with `E.D and M.D.`**
  - **Another example, if dataset has categorical variables then these should be handled before and converted into numerical values**
  - **Similarly, the scale has to be same across features otherwise distances will gets skewed**
- **With a small dataset this can be a good approach but if we have a high dimensional data then it suffers with all the drawbacks of KNN.**

Novelty_Detection

There is a slight difference between the Outlier and Novelty Detection.

- ***Outliers are also known as Novelty, in outlier detection we provide the dataset features in the LOF then based on it we identify the potential outliers. However, in the Novelty Detection we train the model on Outliers free dataset then fed the unseen data into it to find the outliers in the provided unseen dataset.***

In [132…
```
X = iris_lof_result[iris_lof_result['LOF_Tags'] == 0][['sepal length (cm)','sepal wi
y = iris_lof_result[iris_lof_result['LOF_Tags'] == 0][['Label']].copy(deep=True)
```

In [133…
```
X.shape, y.shape
```

Out[133…
```
((135, 4), (135, 1))
```

In [134…
```
X_test = iris_lof_result[iris_lof_result['LOF_Tags'] == 1][['sepal length (cm)','sep
y_test = iris_lof_result[iris_lof_result['LOF_Tags'] == 1][['Label']].copy(deep=True
```

In [135…
```
X_test.shape, y_test.shape
```

Out[135…
```
((15, 4), (15, 1))
```

In [136…
```
novl_lof = LocalOutlierFactor(n_neighbors=25,algorithm='kd_tree',novelty=True,contam
```

In [137…
```
novl_lof.fit(X)
```

Out[137…
```
LocalOutlierFactor(algorithm='kd_tree', contamination=0.1, leaf_size=15,
                   n_neighbors=25, novelty=True)
```

I have used the dataset cleaned in the above Outlier Detection, so I'm using the outliers free features as a train data and the observations which were declared Outliers in the above steps as a test set.

In [138…
```
novl_lof.offset_
```

Out[138…
```
-1.1395064780485147
```

When novelty is set to True be aware that you must only use predict , decision_function and score_samples on new unseen data and not on the

training samples as this would lead to wrong results.

In [139…
```
novl_lof.decision_function(X_test.iloc[0:2,:])
```
Out[139…
```
array([-0.21279232, -0.39126247])
```

In [140…
```
novl_lof.decision_function(X_test.iloc[:,:])
```
Out[140…
```
array([-0.21279232, -0.39126247, -0.62380956, -0.25358871, -0.67333104,
       -0.40624575, -0.37870996, -0.37254652, -0.52122239, -0.39445628,
       -0.13998717, -0.64729715, -0.69816836, -0.5362388 , -0.62497454])
```

For the majority of test records the `shifted opposite LOF scores` are away from 0 and all are negatives thus all are classified as Outliers. If the `shifted opposite score` is a large and positive number then it is an inlier.

In [141…
```
novl_lof.predict(X_test)
```
Out[141…
```
array([-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1])
```

Random Test Input

In [142…
```
novl_lof.decision_function([[1.33717756,  1.00979142, 1.16971425, 1.05217993]])
```
Out[142…
```
array([-6.18355273])
```

In [143…
```
novl_lof.predict([[1.33717756,  1.00979142, 1.16971425, 1.05217993]])
```
Out[143…
```
array([-1])
```

The shifted opposite LOF score is +ve thus Random Test Input is labelled as Inlier.

In [144…
```
novl_lof.score_samples(X_test)
```
Out[144…
```
array([-1.35229879, -1.53076895, -1.76331604, -1.39309518, -1.81283752,
       -1.54575223, -1.51821644, -1.512053  , -1.66072887, -1.53396276,
       -1.27949365, -1.78680363, -1.83767484, -1.67574527, -1.76448102])
```

This is the `opposite of LOF score` returned by score_samples the higher this score means observation is an inlier. In our case, all the scores are -ve thus all are labelled as Outliers.

- **The offset from training data is -1.16 and the opposite LOF score for the test data is greater than the offset value thus all are marked as Outliers.**

In [145…
```
novl_lof.score_samples([[1.33717756,  1.00979142, 1.16971425, 1.05217993]])
```
Out[145…
```
array([-7.32305921])
```

- **The opposite LOF score of the random test input is less than -1.16 thus it is marked as Inlier.**

In [146…
```
X['-ve_lof'] = novl_lof.negative_outlier_factor_
```

In [147…
```
X[X['-ve_lof'] <= novl_lof.offset_].shape
```

Out[147…
```
(14, 5)
```

In [148…
```
X[X['-ve_lof'] <= novl_lof.offset_]
```

Out[148…

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	-ve_lof
5	5.4	3.9	1.7	0.4	-1.152862
8	4.4	2.9	1.4	0.2	-1.221901
18	5.7	3.8	1.7	0.3	-1.290834
22	4.6	3.6	1.0	0.2	-1.209243
32	5.2	4.1	1.5	0.1	-1.219220
38	4.4	3.0	1.3	0.2	-1.193754
50	7.0	3.2	4.7	1.4	-1.170177
79	5.7	2.6	3.5	1.0	-1.157464
81	5.5	2.4	3.7	1.0	-1.140011
107	7.3	2.9	6.3	1.8	-1.305762
109	7.2	3.6	6.1	2.5	-1.266638
125	7.2	3.2	6.0	1.8	-1.157932
130	7.4	2.8	6.1	1.9	-1.272495
135	7.7	3.0	6.1	2.3	-1.402265

In [149…
```
X_test['Oppo_LOF_score'] = novl_lof.score_samples(X_test)
X_test['shif_opp_LOF_score'] = novl_lof.decision_function(X_test.iloc[:,0:-1])
```

In [150…
```
X_test[X_test['Oppo_LOF_score'] < novl_lof.offset_]
```

Out[150…

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Oppo_LOF_score	shif_opp_LOF_score
13	4.3	3.0	1.1	0.1	-1.352299	-0.212792
14	5.8	4.0	1.2	0.2	-1.530769	-0.391262
15	5.7	4.4	1.5	0.4	-1.763316	-0.623810
33	5.5	4.2	1.4	0.2	-1.393095	-0.253589
41	4.5	2.3	1.3	0.3	-1.812838	-0.673331
57	4.9	2.4	3.3	1.0	-1.545752	-0.406246
60	5.0	2.0	3.5	1.0	-1.518216	-0.378710
93	5.0	2.3	3.3	1.0	-1.512053	-0.372547
98	5.1	2.5	3.0	1.1	-1.660729	-0.521222
105	7.6	3.0	6.6	2.1	-1.533963	-0.394456
106	4.9	2.5	4.5	1.7	-1.279494	-0.139987
117	7.7	3.8	6.7	2.2	-1.786804	-0.647297
118	7.7	2.6	6.9	2.3	-1.837675	-0.698168
122	7.7	2.8	6.7	2.0	-1.675745	-0.536239

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Oppo_LOF_score	shif_opp_LOF_score
131	7.9	3.8	6.4	2.0	-1.764481	-0.624975

In [151…

```
X_test['Oppo_LOF_score'] - X_test['shif_opp_LOF_score']
```

Out[151…

```
13     -1.139506
14     -1.139506
15     -1.139506
33     -1.139506
41     -1.139506
57     -1.139506
60     -1.139506
93     -1.139506
98     -1.139506
105    -1.139506
106    -1.139506
117    -1.139506
118    -1.139506
122    -1.139506
131    -1.139506
dtype: float64
```

In [159…

```
novl_lof.offset_
```

Out[159…

```
-1.1395064780485147
```

In [158…

```
np.round(np.unique(X_test['Oppo_LOF_score'] - X_test['shif_opp_LOF_score'])[0],4) ==
```

Out[158…

```
True
```

Relationship_bw_DF_SC_and_OFF

This means `Decision_Function values = Score_Samples values - Offset`

`Points_to_remember_when_doing_novelty_detection`

In my analysis I found that LOF works good with a small dimensional dataset to find the local and global outliers. Few points we need to consider while working with NOVELTY DETECTION using LOF:

- **SciKit-Learn implements two "modes" for LocalOutlierFactor, where one is unsupervised(NOVELTY=FALSE) and one is semi-supervised(NOVELTY=True).**
- **Novelty : boolean, default False By default, LocalOutlierFactor is only meant to be used for outlier detection (novelty=False). Set novelty to True if you want to use LocalOutlierFactor for novelty detection. In this case be aware that that you should only use predict, decision_function and score_samples on new unseen data and not on the training set.**