Anomaly Detection using TRAFAIR Air Quality Dataset

1.0 Anomaly Detection

Anomaly Detection is the task of identifying the rare items, events or observations which raise suspicions by differing significantly from the majority of the data. Typically the anomalous items will translate to some kind of problem such as bank fraud, a structural defect, medical problems or errors in a text. There are three broad categories of anomaly detection techniques exist:

- **Unsupervised anomaly detection:** Unsupervised anomaly detection techniques detect anomalies in an unlabeled test data set under the assumption that the majority of the instances in the dataset are normal by looking for instances that seem to fit least to the remainder of the data set.

- **Supervised anomaly detection:** This technique requires a dataset that has been labeled as "normal" and "abnormal" and involves training a classifier.

- **Semi-supervised anomaly detection:** This technique constructs a model representing normal behavior from a given normal training dataset, and then tests the likelihood of a test instance to be generated by the learnt model.

Learn More about Anomaly Detection (https://en.wikipedia.org/wiki/Anomaly_detection)

2.0 TRAFAIR project Air Quality Dataset Details

Datasets have been generated by the TRAFAIR project (www.trafair.eu) and the data are available on the Italian open data portal (https://www.dati.gov.it/view-dataset?Cerca=&tags_set=trafair&tags=trafair&ordinamento=&sort=Invia)

3.0 Import the Dataset

You can download the data from the original source found here (https://archive.ics.uci.edu/ml/datasets/Mice+Protein+Expression) and load it using pandas (Learn How) (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html) or you can use PyCaret's data respository to load the data using `get_data()` function (This will require internet connection).

In [1]:
```
import pycaret
import pandas as pd
```

In [2]:
```
dataset = pd.read_csv('4005_20200101_20200331.csv')
```

In [3]:
```
#check the shape of data
dataset.shape
```

Out[3]:
```
(58388, 13)
```

In order to demonstrate the `predict_model()` function on unseen data, a sample of 5% (54 samples) are taken out from original dataset to be used for predictions at the end of experiment. This should not be confused with train/test split. This particular split is performed to simulate real life scenario. Another way to think about this is that these 54 samples are not available at the time when this experiment was performed.

In [4]:
```
data = dataset.sample(frac=0.95, random_state=786)
data_unseen = dataset.drop(data.index)

data.reset_index(drop=True, inplace=True)
data_unseen.reset_index(drop=True, inplace=True)

print('Data for Modeling: ' + str(data.shape))
print('Unseen Data For Predictions: ' + str(data_unseen.shape))
```

```
Data for Modeling: (55469, 13)
Unseen Data For Predictions: (2919, 13)
```

4.0 Setting up Environment in PyCaret

`setup()` function initializes the environment in PyCaret and creates the transformation pipeline to prepare the data for modeling and deployment. `setup()` must be called before executing any other function in PyCaret. It takes only one mandatory parameter: pandas dataframe. All other parameters are optional and are used to customize pre-processing pipeline (we will see them in later tutorials).

When `setup()` is executed, PyCaret's inference algorithm will automatically infer the data types for all features based on certain properties. Although, most of the times the data type is inferred correctly but it's not always the case. Therefore, after `setup()` is executed, PyCaret displays a table containing features and their inferred data types. At which stage, you can inspect and press `enter` to continue if all data types are correctly inferred or type `quit` to end the experiment. Identifying data types correctly is of fundamental importance in PyCaret as it automatically performs few pre-processing tasks which are imperative to perform any machine learning experiment. These pre-processing tasks are performed differently for each data type. As such, it is very important that data types are correctly configured.

In later tutorials we will learn how to overwrite PyCaret's inferred data types using `numeric_features` and `categorical_features` parameter in `setup()`.

In [5]:

```
from pycaret.anomaly import *

exp_ano101 = setup(data, normalize = True,
                   session_id = 123)
```

	Description	Value
0	session_id	123
1	Original Data	(55469, 13)
2	Missing Values	False
3	Numeric Features	11
4	Categorical Features	1
5	Ordinal Features	False
6	High Cardinality Features	False
7	High Cardinality Method	None
8	Transformed Data	(55469, 50)
9	CPU Jobs	-1
10	Use GPU	False
11	Log Experiment	False
12	Experiment Name	anomaly-default-name
13	USI	37de
14	Imputation Type	simple
15	Iterative Imputation Iteration	None
16	Numeric Imputer	mean
17	Iterative Imputation Numeric Model	None
18	Categorical Imputer	mode
19	Iterative Imputation Categorical Model	None
20	Unknown Categoricals Handling	least_frequent
21	Normalize	True
22	Normalize Method	zscore
23	Transformation	False
24	Transformation Method	None
25	PCA	False
26	PCA Method	None
27	PCA Components	None
28	Ignore Low Variance	False
29	Combine Rare Levels	False
30	Rare Level Threshold	None
31	Numeric Binning	False
32	Remove Outliers	False
33	Outliers Threshold	None
34	Remove Multicollinearity	False
35	Multicollinearity Threshold	None
36	Clustering	False
37	Clustering Iteration	None
38	Polynomial Features	False
39	Polynomial Degree	None
40	Trignometry Features	False

	Description	Value
41	Polynomial Threshold	None
42	Group Features	False
43	Feature Selection	False
44	Feature Selection Method	classic
45	Features Selection Threshold	None
46	Feature Interaction	False
47	Feature Ratio	False
48	Interaction Threshold	None

Once the setup is succesfully executed it prints the information grid that contains few important information. Much of the information is related to pre-processing pipeline which is constructed when `setup()` is executed. Much of these features are out of scope for the purpose of this tutorial. However, few important things to note at this stage are:

- **session_id :** A pseudo-random number distributed as a seed in all functions for later reproducibility. If no `session_id` is passed, a random number is automatically generated that is distributed to all functions. In this experiment session_id is set as `123` for later reproducibility.

- **Missing Values :** When there are missing values in original data it will show as `True` . Notice that `Missing Values` in the information grid above is `True` as the data contains missing values which are automatically imputed using `mean` for numeric features and `constant` for categorical features. The method of imputation can be changed using `numeric_imputation` and `categorical_imputation` parameter in `setup()` .

- **Original Data :** Displays the original shape of dataset. In this experiment (1026, 82) means 1026 samples and 82 features.

- **Transformed Data :** Displays the shape of transformed dataset. Notice that the shape of original dataset (1026, 82) is transformed into (1026, 91). The number of features has increased due to encoding of categorical features in the dataset.

- **Numeric Features :** Number of features inferred as numeric. In this dataset, 77 out of 82 features are inferred as numeric.

- **Categorical Features :** Number of features inferred as categorical. In this dataset, 5 out of 82 features are inferred as categorical. Also notice, we have ignored one categorical feature i.e. `MouseID` using `ignore_feature` parameter.

Notice that how few tasks such as missing value imputation and categorical encoding that are imperative to perform modeling are automatically handled. Most of the other parameters in `setup()` are optional and used for customizing pre-processing pipeline. These parameters are out of scope for this tutorial but as you progress to intermediate and expert level, we will cover them in much detail.

5.0 Create a Model

Creating an anomaly detection model in PyCaret is simple and similar to how you would have created a model in supervised modules of PyCaret. The anomaly detection model is created using `create_model()` function which takes one mandatory parameter i.e. name of the model as a string. This function returns a trained model object. See the example below:

In [6]:
```
models()
```

Out[6]:

ID	Name	Reference
abod	Angle-base Outlier Detection	pyod.models.abod.ABOD
cluster	Clustering-Based Local Outlier	pyod.models.cblof.CBLOF
cof	Connectivity-Based Local Outlier	pyod.models.cof.COF
iforest	Isolation Forest	pyod.models.iforest.IForest
histogram	Histogram-based Outlier Detection	pyod.models.hbos.HBOS
knn	K-Nearest Neighbors Detector	pyod.models.knn.KNN
lof	Local Outlier Factor	pyod.models.lof.LOF
svm	One-class SVM detector	pyod.models.ocsvm.OCSVM
pca	Principal Component Analysis	pyod.models.pca.PCA
mcd	Minimum Covariance Determinant	pyod.models.mcd.MCD
sod	Subspace Outlier Detection	pyod.models.sod.SOD
sos	Stochastic Outlier Selection	pyod.models.sos.SOS

In [11]:
```
iforest = create_model('iforest')
```

In [6]:
```
print(iforest)
```

```
IForest(behaviour='new', bootstrap=False, contamination=0.05,
    max_features=1.0, max_samples='auto', n_estimators=100, n_jobs=-1,
    random_state=123, verbose=0)
```

We have created Isolation Forest model using `create_model()`. Notice the `contamination` parameter is set `0.05` which is the default value when you do not pass `fraction` parameter in `create_model()`. `fraction` parameter determines the proportion of outliers in the dataset. In below example, we will create `One Class Support Vector Machine` model with `0.025` fraction.

In [12]:
```
svm = create_model('svm', fraction = 0.025)
```

In [13]:
```
print(svm)
```

```
OCSVM(cache_size=200, coef0=0.0, contamination=0.025, degree=3, gamma='auto',
   kernel='rbf', max_iter=-1, nu=0.5, shrinking=True, tol=0.001,
   verbose=False)
```

Just by replacing `iforest` with `svm` inside `create_model()` we have now created `OCSVM` anomaly detection model. There are 12 models available ready-to-use in `pycaret.anomaly` module. To see the complete list, please see docstring or use `models` function.

In [7]:
```
knn = create_model('knn')
print(knn)
```

```
KNN(algorithm='auto', contamination=0.05, leaf_size=30, method='largest',
  metric='minkowski', metric_params=None, n_jobs=-1, n_neighbors=5, p=2,
  radius=1.0)
```

In [8]:
```
cluster = create_model('cluster')
print(cluster)
```

```
CBLOF(alpha=0.9, beta=5, check_estimator=False, clustering_estimator=None,
   contamination=0.05, n_clusters=8, n_jobs=-1, random_state=123,
   use_weights=False)
```

In [24]:
```
abod = create_model('abod')
print(abod)
```

```
ABOD(contamination=0.05, method='fast', n_neighbors=5)
```

In [16]:
```
histogram = create_model('histogram')
print(histogram)
```

```
HBOS(alpha=0.1, contamination=0.05, n_bins=10, tol=0.5)
```

In [15]:
```
lof = create_model('lof')
print(lof)
```

```
LOF(algorithm='auto', contamination=0.05, leaf_size=30, metric='minkowski',
  metric_params=None, n_jobs=-1, n_neighbors=20, p=2)
```

In [17]:
```
pca = create_model('pca')
print(pca)
```

```
PCA(contamination=0.05, copy=True, iterated_power='auto', n_components=None,
  n_selected_components=None, random_state=123, standardization=True,
  svd_solver='auto', tol=0.0, weighted=True, whiten=False)
```

In [18]:
```
mcd = create_model('mcd')
print(mcd)
```

```
MCD(assume_centered=False, contamination=0.05, random_state=123,
  store_precision=True, support_fraction=None)
```

6.0 Assign a Model

Now that we have created a model, we would like to assign the anomaly labels to our dataset (1080 samples) to analyze the results. We will achieve this by using `assign_model()` function. See an example below:

In [15]:
```
iforest_results = assign_model(iforest)
iforest_results.head()
```

Out[15]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_au
0	4005	2020-02-02 05:53:25.263916	4.901	100.758118	5.027297	296.813965	-4.57763
1	4005	2020-02-08 23:47:16.844981	4.840	81.787628	1.938469	312.194824	8.69751(
2	4005	2020-01-21 07:01:46.596547	4.956	99.678558	-0.571204	347.717285	46.87500(
3	4005	2020-03-03 18:30:47.208524	4.418	88.692230	7.343918	310.089111	4.11987
4	4005	2020-02-19 14:59:04.939508	4.730	54.386658	17.945677	314.483643	0.09155

In [20]:
```
svm_results = assign_model(svm)
svm_results.head()
```

Out[20]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_au
0	4005	2020-02-02 05:53:25.263916	4.901	100.758118	5.027297	296.813965	-4.57763
1	4005	2020-02-08 23:47:16.844981	4.840	81.787628	1.938469	312.194824	8.69751(
2	4005	2020-01-21 07:01:46.596547	4.956	99.678558	-0.571204	347.717285	46.87500(
3	4005	2020-03-03 18:30:47.208524	4.418	88.692230	7.343918	310.089111	4.11987
4	4005	2020-02-19 14:59:04.939508	4.730	54.386658	17.945677	314.483643	0.09155

In [9]:
```
knn_results = assign_model(knn)
knn_results.head()
```

Out[9]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_au
0	4005	2020-02-02 05:53:25.263916	4.901	100.758118	5.027297	296.813965	-4.57763
1	4005	2020-02-08 23:47:16.844981	4.840	81.787628	1.938469	312.194824	8.69751(
2	4005	2020-01-21 07:01:46.596547	4.956	99.678558	-0.571204	347.717285	46.87500(
3	4005	2020-03-03 18:30:47.208524	4.418	88.692230	7.343918	310.089111	4.11987
4	4005	2020-02-19 14:59:04.939508	4.730	54.386658	17.945677	314.483643	0.09155

In [10]:
```
cluster_results = assign_model(cluster)
cluster_results.head()
```

Out[10]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_au
0	4005	2020-02-02 05:53:25.263916	4.901	100.758118	5.027297	296.813965	-4.57763
1	4005	2020-02-08 23:47:16.844981	4.840	81.787628	1.938469	312.194824	8.69751(
2	4005	2020-01-21 07:01:46.596547	4.956	99.678558	-0.571204	347.717285	46.87500(
3	4005	2020-03-03 18:30:47.208524	4.418	88.692230	7.343918	310.089111	4.11987
4	4005	2020-02-19 14:59:04.939508	4.730	54.386658	17.945677	314.483643	0.09155

In [42]:
```
abod_results = assign_model(abod)
abod_results.head()
```

Out[42]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_au
0	4005	2020-02-02 05:53:25.263916	4.901	100.758118	5.027297	296.813965	-4.57763
1	4005	2020-02-08 23:47:16.844981	4.840	81.787628	1.938469	312.194824	8.69751(
2	4005	2020-01-21 07:01:46.596547	4.956	99.678558	-0.571204	347.717285	46.87500(
3	4005	2020-03-03 18:30:47.208524	4.418	88.692230	7.343918	310.089111	4.11987
4	4005	2020-02-19 14:59:04.939508	4.730	54.386658	17.945677	314.483643	0.09155

In [44]:
```
histogram_results = assign_model(histogram)
histogram_results.head()
```

Out[44]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_au
0	4005	2020-02-02 05:53:25.263916	4.901	100.758118	5.027297	296.813965	-4.57763
1	4005	2020-02-08 23:47:16.844981	4.840	81.787628	1.938469	312.194824	8.69751(
2	4005	2020-01-21 07:01:46.596547	4.956	99.678558	-0.571204	347.717285	46.87500(
3	4005	2020-03-03 18:30:47.208524	4.418	88.692230	7.343918	310.089111	4.11987
4	4005	2020-02-19 14:59:04.939508	4.730	54.386658	17.945677	314.483643	0.09155

In [45]:
```
lof_results = assign_model(lof)
lof_results.head()
```

Out[45]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_au
0	4005	2020-02-02 05:53:25.263916	4.901	100.758118	5.027297	296.813965	-4.57763
1	4005	2020-02-08 23:47:16.844981	4.840	81.787628	1.938469	312.194824	8.69751(
2	4005	2020-01-21 07:01:46.596547	4.956	99.678558	-0.571204	347.717285	46.87500(
3	4005	2020-03-03 18:30:47.208524	4.418	88.692230	7.343918	310.089111	4.11987
4	4005	2020-02-19 14:59:04.939508	4.730	54.386658	17.945677	314.483643	0.09155

In [46]:
```
pca_results = assign_model(pca)
pca_results.head()
```

Out[46]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_au
0	4005	2020-02-02 05:53:25.263916	4.901	100.758118	5.027297	296.813965	-4.57763
1	4005	2020-02-08 23:47:16.844981	4.840	81.787628	1.938469	312.194824	8.69751(
2	4005	2020-01-21 07:01:46.596547	4.956	99.678558	-0.571204	347.717285	46.87500(
3	4005	2020-03-03 18:30:47.208524	4.418	88.692230	7.343918	310.089111	4.11987
4	4005	2020-02-19 14:59:04.939508	4.730	54.386658	17.945677	314.483643	0.09155

In [47]:
```
mcd_results = assign_model(mcd)
mcd_results.head()
```

Out[47]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_au
0	4005	2020-02-02 05:53:25.263916	4.901	100.758118	5.027297	296.813965	-4.57763
1	4005	2020-02-08 23:47:16.844981	4.840	81.787628	1.938469	312.194824	8.69751(
2	4005	2020-01-21 07:01:46.596547	4.956	99.678558	-0.571204	347.717285	46.87500(
3	4005	2020-03-03 18:30:47.208524	4.418	88.692230	7.343918	310.089111	4.11987
4	4005	2020-02-19 14:59:04.939508	4.730	54.386658	17.945677	314.483643	0.09155

Notice that two columns `Label` and `Score` are added towards the end. 0 stands for inliers and 1 for outliers/anomalies. Score is the values computed by the algorithm. Outliers are assigned with larger anomaly scores. Notice that `iforest_results` also includes `MouseID` feature that we have dropped during `setup()` . It wasn't used for the model and is only appended to the dataset when you use `assign_model()` . In the next section we will see how to analyze the results of anomaly detection using `plot_model()` .

7.0 Plot a Model

`plot_model()` function can be used to analyze the anomaly detection model over different aspects. This function takes a trained model object and returns a plot. See the examples below:

7.1 T-distributed Stochastic Neighbor Embedding (t-SNE)

In [ ]:
```
plot_model(iforest)
```

In [ ]:
```
plot_model(svm)
```

In [ ]:
```
plot_model(knn)
```

In [ ]:
```
plot_model(cluster)
```

In [ ]:
```
plot_model(abod)
```

In [ ]:
```
plot_model(histogram)
```

In [ ]:
```
plot_model(lof)
```

In [ ]:
```
plot_model(pca)
```

In [ ]:
```
plot_model(mcd)
```

7.2 Uniform Manifold Approximation and Projection

In [ ]:
```
plot_model(iforest, plot = 'umap')
```

In [22]:
```
plot_model(svm, plot = 'umap')
```

In [60]:
```
plot_model(cluster, plot = 'umap')
```

In [61]:
```
plot_model(knn, plot = 'umap')
```

In [63]:
```
plot_model(abod, plot = 'umap')
```

In [64]:
```
plot_model(histogram, plot = 'umap')
```

In [65]:
```
plot_model(lof, plot = 'umap')
```

In [66]:
```
plot_model(pca, plot = 'umap')
```

In [58]:
```
plot_model(mcd, plot = 'umap')
```

8.0 Predict on Unseen Data

`predict_model()` function is used to assign anomaly labels on the new unseen dataset. We will now use our `iforest` model to predict the data stored in `data_unseen`. This was created in the beginning of the experiment and it contains 54 new samples that were not exposed to PyCaret before.

In [18]:
```
unseen_predictions = predict_model(iforest, data=data_unseen)
unseen_predictions.head()
```

Out[18]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_aux
0	4005	2020-01-08 23:49:51.022621	5.010	96.962494	0.013314	301.116943	1.464844
1	4005	2020-01-08 23:33:48.032856	5.013	97.828430	0.184916	303.039551	3.204346
2	4005	2020-01-08 22:07:50.305441	5.013	98.686737	0.479856	299.743652	-0.274658
3	4005	2020-01-08 21:55:46.325789	5.017	98.831696	0.522756	301.025391	0.549316
4	4005	2020-01-08 21:47:49.86342	5.017	98.728699	0.554932	298.095703	-0.091553

`Label` column indicates the outlier (1 = outlier, 0 = inlier). `Score` is the values computed by the algorithm. Outliers are assigned with larger anomaly scores. You can also use `predict_model()` function to label the training data. See example below:

In [19]:
```
data_predictions = predict_model(knn, data = data)
data_predictions.head()
```

Out[19]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_au
0	4005	2020-02-02 05:53:25.263916	4.901	100.758118	5.027297	296.813965	-4.57763
1	4005	2020-02-08 23:47:16.844981	4.840	81.787628	1.938469	312.194824	8.69751(
2	4005	2020-01-21 07:01:46.596547	4.956	99.678558	-0.571204	347.717285	46.87500(
3	4005	2020-03-03 18:30:47.208524	4.418	88.692230	7.343918	310.089111	4.11987
4	4005	2020-02-19 14:59:04.939508	4.730	54.386658	17.945677	314.483643	0.09155

In [23]:
```
data_predictions = predict_model(svm, data = data)
data_predictions.head()
```

Out[23]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_au
0	4005	2020-02-02 05:53:25.263916	4.901	100.758118	5.027297	296.813965	-4.57763
1	4005	2020-02-08 23:47:16.844981	4.840	81.787628	1.938469	312.194824	8.69751(
2	4005	2020-01-21 07:01:46.596547	4.956	99.678558	-0.571204	347.717285	46.87500(
3	4005	2020-03-03 18:30:47.208524	4.418	88.692230	7.343918	310.089111	4.11987
4	4005	2020-02-19 14:59:04.939508	4.730	54.386658	17.945677	314.483643	0.09155

In [36]:
```
data_predictions = predict_model(cluster, data = data)
data_predictions.head()
```

Out[36]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_au
0	4005	2020-02-02 05:53:25.263916	4.901	100.758118	5.027297	296.813965	-4.57763
1	4005	2020-02-08 23:47:16.844981	4.840	81.787628	1.938469	312.194824	8.69751(
2	4005	2020-01-21 07:01:46.596547	4.956	99.678558	-0.571204	347.717285	46.87500(
3	4005	2020-03-03 18:30:47.208524	4.418	88.692230	7.343918	310.089111	4.11987
4	4005	2020-02-19 14:59:04.939508	4.730	54.386658	17.945677	314.483643	0.09155

In [37]:
```
data_predictions = predict_model(abod, data = data)
data_predictions.head()
```

Out[37]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_au
0	4005	2020-02-02 05:53:25.263916	4.901	100.758118	5.027297	296.813965	-4.57763
1	4005	2020-02-08 23:47:16.844981	4.840	81.787628	1.938469	312.194824	8.69751(
2	4005	2020-01-21 07:01:46.596547	4.956	99.678558	-0.571204	347.717285	46.87500(
3	4005	2020-03-03 18:30:47.208524	4.418	88.692230	7.343918	310.089111	4.11987
4	4005	2020-02-19 14:59:04.939508	4.730	54.386658	17.945677	314.483643	0.09155

In [38]:
```
data_predictions = predict_model(histogram, data = data)
data_predictions.head()
```

Out[38]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_au
0	4005	2020-02-02 05:53:25.263916	4.901	100.758118	5.027297	296.813965	-4.57763
1	4005	2020-02-08 23:47:16.844981	4.840	81.787628	1.938469	312.194824	8.69751(
2	4005	2020-01-21 07:01:46.596547	4.956	99.678558	-0.571204	347.717285	46.87500(
3	4005	2020-03-03 18:30:47.208524	4.418	88.692230	7.343918	310.089111	4.11987
4	4005	2020-02-19 14:59:04.939508	4.730	54.386658	17.945677	314.483643	0.09155

In [39]:
```
data_predictions = predict_model(lof, data = data)
data_predictions.head()
```

Out[39]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_au
0	4005	2020-02-02 05:53:25.263916	4.901	100.758118	5.027297	296.813965	-4.57763
1	4005	2020-02-08 23:47:16.844981	4.840	81.787628	1.938469	312.194824	8.69751(
2	4005	2020-01-21 07:01:46.596547	4.956	99.678558	-0.571204	347.717285	46.87500(
3	4005	2020-03-03 18:30:47.208524	4.418	88.692230	7.343918	310.089111	4.11987
4	4005	2020-02-19 14:59:04.939508	4.730	54.386658	17.945677	314.483643	0.09155

In [40]:
```
data_predictions = predict_model(pca, data = data)
data_predictions.head()
```

Out[40]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_au
0	4005	2020-02-02 05:53:25.263916	4.901	100.758118	5.027297	296.813965	-4.57763
1	4005	2020-02-08 23:47:16.844981	4.840	81.787628	1.938469	312.194824	8.69751(
2	4005	2020-01-21 07:01:46.596547	4.956	99.678558	-0.571204	347.717285	46.87500(
3	4005	2020-03-03 18:30:47.208524	4.418	88.692230	7.343918	310.089111	4.11987
4	4005	2020-02-19 14:59:04.939508	4.730	54.386658	17.945677	314.483643	0.09155

In [55]:
```
data_predictions = predict_model(mcd, data = data)
data_predictions.head()
```

Out[55]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_au
0	4005	2020-02-02 05:53:25.263916	4.901	100.758118	5.027297	296.813965	-4.57763
1	4005	2020-02-08 23:47:16.844981	4.840	81.787628	1.938469	312.194824	8.69751(
2	4005	2020-01-21 07:01:46.596547	4.956	99.678558	-0.571204	347.717285	46.87500(
3	4005	2020-03-03 18:30:47.208524	4.418	88.692230	7.343918	310.089111	4.11987
4	4005	2020-02-19 14:59:04.939508	4.730	54.386658	17.945677	314.483643	0.09155

9.0 Saving the Models

We have now finished the experiment by using our `iforest` model to predict outlier labels on unseen data. This brings us to the end of our experiment but one question is still to be asked. What happens when you have more new data to predict? Do you have to go through the entire experiment again? The answer is No, you don't need to rerun the entire experiment and reconstruct the pipeline to generate predictions on new data. PyCaret's inbuilt function `save_model()` allows you to save the model along with entire transformation pipeline for later use.

In [24]:
```
save_model(iforest,'Anom_detection_iforest')
```

```
Transformation Pipeline and Model Succesfully Saved
```

Out[24]:
```
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True, features_todrop=[],
                                       id_columns=[], ml_usecase='regression',
                                       numerical_features=[],
                                       target='UNSUPERVISED_DUMMY_TARGET',
                                       time_features=[])),
                 ('imputer',
                  Simple_Imputer(categorical_strategy='most frequent',
                                 fill_value_categorical=None,
                                 fill_value_numerical=None...
                 ('fix_perfect', 'passthrough'),
                 ('clean_names', Clean_Colum_Names()),
                 ('feature_select', 'passthrough'), ('fix_multi', 'passthrough'),
                 ('dfs', 'passthrough'), ('pca', 'passthrough'),
                 ['trained_model',
                  IForest(behaviour='new', bootstrap=False, contamination=0.05,
      max_features=1.0, max_samples='auto', n_estimators=100, n_jobs=-1,
      random_state=123, verbose=0)]],
          verbose=False),
 'Anom_detection_iforest.pkl')
```

In [25]:
```
save_model(svm,'Anom_detection_svm')
```

```
Transformation Pipeline and Model Succesfully Saved
```

Out[25]:
```
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True, features_todrop=[],
                                       id_columns=[], ml_usecase='regression',
                                       numerical_features=[],
                                       target='UNSUPERVISED_DUMMY_TARGET',
                                       time_features=[])),
                 ('imputer',
                  Simple_Imputer(categorical_strategy='most frequent',
                                 fill_value_categorical=None,
                                 fill_value_numerical=None...
                 ('dummy', Dummify(target='UNSUPERVISED_DUMMY_TARGET')),
                 ('fix_perfect', 'passthrough'),
                 ('clean_names', Clean_Colum_Names()),
                 ('feature_select', 'passthrough'), ('fix_multi', 'passthrough'),
                 ('dfs', 'passthrough'), ('pca', 'passthrough'),
                 ['trained_model',
                  OCSVM(cache_size=200, coef0=0.0, contamination=0.025, degree=3, ga
mma='auto',
      kernel='rbf', max_iter=-1, nu=0.5, shrinking=True, tol=0.001,
      verbose=False)]],
          verbose=False),
 'Anom_detection_svm.pkl')
```

In [12]:
```
save_model(knn,'Anom_detection_knn')
```

```
Transformation Pipeline and Model Succesfully Saved
```

Out[12]:
```
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True, features_todrop=[],
                                       id_columns=[], ml_usecase='regression',
                                       numerical_features=[],
                                       target='UNSUPERVISED_DUMMY_TARGET',
                                       time_features=[])),
                 ('imputer',
                  Simple_Imputer(categorical_strategy='most frequent',
                                 fill_value_categorical=None,
                                 fill_value_numerical=None...
                 ('fix_perfect', 'passthrough'),
                 ('clean_names', Clean_Colum_Names()),
                 ('feature_select', 'passthrough'), ('fix_multi', 'passthrough'),
                 ('dfs', 'passthrough'), ('pca', 'passthrough'),
                 ['trained_model',
                  KNN(algorithm='auto', contamination=0.05, leaf_size=30, method='la
rgest',
    metric='minkowski', metric_params=None, n_jobs=-1, n_neighbors=5, p=2,
    radius=1.0)]],
          verbose=False),
 'Anom_detection_knn.pkl')
```

In [13]:
```
save_model(cluster,'Anom_detection_cluster')
```

```
Transformation Pipeline and Model Succesfully Saved
```

Out[13]:
```
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True, features_todrop=[],
                                       id_columns=[], ml_usecase='regression',
                                       numerical_features=[],
                                       target='UNSUPERVISED_DUMMY_TARGET',
                                       time_features=[])),
                 ('imputer',
                  Simple_Imputer(categorical_strategy='most frequent',
                                 fill_value_categorical=None,
                                 fill_value_numerical=None...
                 ('fix_perfect', 'passthrough'),
                 ('clean_names', Clean_Colum_Names()),
                 ('feature_select', 'passthrough'), ('fix_multi', 'passthrough'),
                 ('dfs', 'passthrough'), ('pca', 'passthrough'),
                 ['trained_model',
                  CBLOF(alpha=0.9, beta=5, check_estimator=False, clustering_estimat
or=None,
    contamination=0.05, n_clusters=8, n_jobs=-1, random_state=123,
    use_weights=False)]],
          verbose=False),
 'Anom_detection_cluster.pkl')
```

In [26]:
```
save_model(abod,'Anom_detection_abod')
```

```
Transformation Pipeline and Model Succesfully Saved
```

Out[26]:
```
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True, features_todrop=[],
                                       id_columns=[], ml_usecase='regression',
                                       numerical_features=[],
                                       target='UNSUPERVISED_DUMMY_TARGET',
                                       time_features=[])),
                 ('imputer',
                  Simple_Imputer(categorical_strategy='most frequent',
                                 fill_value_categorical=None,
                                 fill_value_numerical=None...
                 ('rem_outliers', 'passthrough'), ('cluster_all', 'passthrough'),
                 ('dummy', Dummify(target='UNSUPERVISED_DUMMY_TARGET')),
                 ('fix_perfect', 'passthrough'),
                 ('clean_names', Clean_Colum_Names()),
                 ('feature_select', 'passthrough'), ('fix_multi', 'passthrough'),
                 ('dfs', 'passthrough'), ('pca', 'passthrough'),
                 ['trained_model',
                  ABOD(contamination=0.05, method='fast', n_neighbors=5)]],
          verbose=False),
 'Anom_detection_abod.pkl')
```

In [20]:
```
save_model(histogram,'Anom_detection_histogram')
```

```
Transformation Pipeline and Model Succesfully Saved
```

Out[20]:
```
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True, features_todrop=[],
                                       id_columns=[], ml_usecase='regression',
                                       numerical_features=[],
                                       target='UNSUPERVISED_DUMMY_TARGET',
                                       time_features=[])),
                 ('imputer',
                  Simple_Imputer(categorical_strategy='most frequent',
                                 fill_value_categorical=None,
                                 fill_value_numerical=None...
                 ('rem_outliers', 'passthrough'), ('cluster_all', 'passthrough'),
                 ('dummy', Dummify(target='UNSUPERVISED_DUMMY_TARGET')),
                 ('fix_perfect', 'passthrough'),
                 ('clean_names', Clean_Colum_Names()),
                 ('feature_select', 'passthrough'), ('fix_multi', 'passthrough'),
                 ('dfs', 'passthrough'), ('pca', 'passthrough'),
                 ['trained_model',
                  HBOS(alpha=0.1, contamination=0.05, n_bins=10, tol=0.5)]],
          verbose=False),
 'Anom_detection_histogram.pkl')
```

In [21]:
```
save_model(lof,'Anom_detection_lof')
```

```
Transformation Pipeline and Model Succesfully Saved
```

Out[21]:
```
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True, features_todrop=[],
                                       id_columns=[], ml_usecase='regression',
                                       numerical_features=[],
                                       target='UNSUPERVISED_DUMMY_TARGET',
                                       time_features=[])),
                 ('imputer',
                  Simple_Imputer(categorical_strategy='most frequent',
                                 fill_value_categorical=None,
                                 fill_value_numerical=None...
                 ('dummy', Dummify(target='UNSUPERVISED_DUMMY_TARGET')),
                 ('fix_perfect', 'passthrough'),
                 ('clean_names', Clean_Colum_Names()),
                 ('feature_select', 'passthrough'), ('fix_multi', 'passthrough'),
                 ('dfs', 'passthrough'), ('pca', 'passthrough'),
                 ['trained_model',
                  LOF(algorithm='auto', contamination=0.05, leaf_size=30, metric='mi
nkowski',
    metric_params=None, n_jobs=-1, n_neighbors=20, p=2)]],
          verbose=False),
 'Anom_detection_lof.pkl')
```

In [22]:
```
save_model(pca,'Anom_detection_pca')
```

```
Transformation Pipeline and Model Succesfully Saved
```

Out[22]:
```
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True, features_todrop=[],
                                       id_columns=[], ml_usecase='regression',
                                       numerical_features=[],
                                       target='UNSUPERVISED_DUMMY_TARGET',
                                       time_features=[])),
                 ('imputer',
                  Simple_Imputer(categorical_strategy='most frequent',
                                 fill_value_categorical=None,
                                 fill_value_numerical=None...
                 ('clean_names', Clean_Colum_Names()),
                 ('feature_select', 'passthrough'), ('fix_multi', 'passthrough'),
                 ('dfs', 'passthrough'), ('pca', 'passthrough'),
                 ['trained_model',
                  PCA(contamination=0.05, copy=True, iterated_power='auto', n_compon
ents=None,
    n_selected_components=None, random_state=123, standardization=True,
    svd_solver='auto', tol=0.0, weighted=True, whiten=False)]],
          verbose=False),
 'Anom_detection_pca.pkl')
```

In [23]:
```
save_model(mcd,'Anom_detection_mcd')
```

```
Transformation Pipeline and Model Succesfully Saved
```

Out[23]:
```
(Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True, features_todrop=[],
                                       id_columns=[], ml_usecase='regression',
                                       numerical_features=[],
                                       target='UNSUPERVISED_DUMMY_TARGET',
                                       time_features=[])),
                 ('imputer',
                  Simple_Imputer(categorical_strategy='most frequent',
                                 fill_value_categorical=None,
                                 fill_value_numerical=None...
                 ('dummy', Dummify(target='UNSUPERVISED_DUMMY_TARGET')),
                 ('fix_perfect', 'passthrough'),
                 ('clean_names', Clean_Colum_Names()),
                 ('feature_select', 'passthrough'), ('fix_multi', 'passthrough'),
                 ('dfs', 'passthrough'), ('pca', 'passthrough'),
                 ['trained_model',
                  MCD(assume_centered=False, contamination=0.05, random_state=123,
    store_precision=True, support_fraction=None)]],
          verbose=False),
 'Anom_detection_mcd.pkl')
```

10.0 Loading the Saved Model and Evaluation

To load a saved model on a future date in the same or different environment, we would use the PyCaret's `load_model()` function and then easily apply the saved model on new unseen data for prediction

In [48]:
```
saved_iforest = load_model('Anom_detection_iforest')
```

```
Transformation Pipeline and Model Successfully Loaded
```

Once the model is loaded in the environment, you can simply use it to predict on any new data using the same `predict_model()` function . Below we have applied the loaded model to predict the same `data_unseen` that we have used in section 10 above.

In [49]:
```
new_prediction = predict_model(saved_iforest, data=data_unseen)
new_prediction.head()
```

Out[49]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_aux
0	4005	2020-01-08 23:49:51.022621	5.010	96.962494	0.013314	301.116943	1.464844
1	4005	2020-01-08 23:33:48.032856	5.013	97.828430	0.184916	303.039551	3.204346
2	4005	2020-01-08 22:07:50.305441	5.013	98.686737	0.479856	299.743652	-0.274658
3	4005	2020-01-08 21:55:46.325789	5.017	98.831696	0.522756	301.025391	0.549316
4	4005	2020-01-08 21:47:49.86342	5.017	98.728699	0.554932	298.095703	-0.091553

In [50]:
```
saved_svm = load_model('Anom_detection_svm')
```

```
Transformation Pipeline and Model Successfully Loaded
```

In [51]:
```
new_prediction = predict_model(saved_svm, data=data_unseen)
new_prediction.head()
```

Out[51]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_aux
0	4005	2020-01-08 23:49:51.022621	5.010	96.962494	0.013314	301.116943	1.464844
1	4005	2020-01-08 23:33:48.032856	5.013	97.828430	0.184916	303.039551	3.204346
2	4005	2020-01-08 22:07:50.305441	5.013	98.686737	0.479856	299.743652	-0.274658
3	4005	2020-01-08 21:55:46.325789	5.017	98.831696	0.522756	301.025391	0.549316
4	4005	2020-01-08 21:47:49.86342	5.017	98.728699	0.554932	298.095703	-0.091553

Notice that results of `unseen_predictions` and `new_prediction` are identical.

In [27]:
```
saved_knn = load_model('Anom_detection_knn')
new_prediction = predict_model(saved_knn, data=data_unseen)
new_prediction.head()
```

```
Transformation Pipeline and Model Successfully Loaded
```

Out[27]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_aux
0	4005	2020-01-08 23:49:51.022621	5.010	96.962494	0.013314	301.116943	1.464844
1	4005	2020-01-08 23:33:48.032856	5.013	97.828430	0.184916	303.039551	3.204346
2	4005	2020-01-08 22:07:50.305441	5.013	98.686737	0.479856	299.743652	-0.274658
3	4005	2020-01-08 21:55:46.325789	5.017	98.831696	0.522756	301.025391	0.549316
4	4005	2020-01-08 21:47:49.86342	5.017	98.728699	0.554932	298.095703	-0.091553

In [28]:
```
saved_cluster = load_model('Anom_detection_cluster')
new_prediction = predict_model(saved_cluster, data=data_unseen)
new_prediction.head()
```

```
Transformation Pipeline and Model Successfully Loaded
```

Out[28]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_aux
0	4005	2020-01-08 23:49:51.022621	5.010	96.962494	0.013314	301.116943	1.464844
1	4005	2020-01-08 23:33:48.032856	5.013	97.828430	0.184916	303.039551	3.204346
2	4005	2020-01-08 22:07:50.305441	5.013	98.686737	0.479856	299.743652	-0.274658
3	4005	2020-01-08 21:55:46.325789	5.017	98.831696	0.522756	301.025391	0.549316
4	4005	2020-01-08 21:47:49.86342	5.017	98.728699	0.554932	298.095703	-0.091553

In [30]:
```
saved_adob = load_model('Anom_detection_abod')
new_prediction = predict_model(saved_adob, data=data_unseen)
new_prediction.head()
```

```
Transformation Pipeline and Model Successfully Loaded
```

Out[30]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_aux
0	4005	2020-01-08 23:49:51.022621	5.010	96.962494	0.013314	301.116943	1.464844
1	4005	2020-01-08 23:33:48.032856	5.013	97.828430	0.184916	303.039551	3.204346
2	4005	2020-01-08 22:07:50.305441	5.013	98.686737	0.479856	299.743652	-0.274658
3	4005	2020-01-08 21:55:46.325789	5.017	98.831696	0.522756	301.025391	0.549316
4	4005	2020-01-08 21:47:49.86342	5.017	98.728699	0.554932	298.095703	-0.091553

In [31]:
```
saved_histogram = load_model('Anom_detection_histogram')
new_prediction = predict_model(saved_histogram, data=data_unseen)
new_prediction.head()
```

```
Transformation Pipeline and Model Successfully Loaded
```

Out[31]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_aux
0	4005	2020-01-08 23:49:51.022621	5.010	96.962494	0.013314	301.116943	1.464844
1	4005	2020-01-08 23:33:48.032856	5.013	97.828430	0.184916	303.039551	3.204346
2	4005	2020-01-08 22:07:50.305441	5.013	98.686737	0.479856	299.743652	-0.274658
3	4005	2020-01-08 21:55:46.325789	5.017	98.831696	0.522756	301.025391	0.549316
4	4005	2020-01-08 21:47:49.86342	5.017	98.728699	0.554932	298.095703	-0.091553

In [32]:
```
saved_lof = load_model('Anom_detection_lof')
new_prediction = predict_model(saved_lof, data=data_unseen)
new_prediction.head()
```

```
Transformation Pipeline and Model Successfully Loaded
```

Out[32]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_aux
0	4005	2020-01-08 23:49:51.022621	5.010	96.962494	0.013314	301.116943	1.464844
1	4005	2020-01-08 23:33:48.032856	5.013	97.828430	0.184916	303.039551	3.204346
2	4005	2020-01-08 22:07:50.305441	5.013	98.686737	0.479856	299.743652	-0.274658
3	4005	2020-01-08 21:55:46.325789	5.017	98.831696	0.522756	301.025391	0.549316
4	4005	2020-01-08 21:47:49.86342	5.017	98.728699	0.554932	298.095703	-0.091553

In [33]:
```
saved_pca = load_model('Anom_detection_pca')
new_prediction = predict_model(saved_pca, data=data_unseen)
new_prediction.head()
```

```
Transformation Pipeline and Model Successfully Loaded
```

Out[33]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_aux
0	4005	2020-01-08 23:49:51.022621	5.010	96.962494	0.013314	301.116943	1.464844
1	4005	2020-01-08 23:33:48.032856	5.013	97.828430	0.184916	303.039551	3.204346
2	4005	2020-01-08 22:07:50.305441	5.013	98.686737	0.479856	299.743652	-0.274658
3	4005	2020-01-08 21:55:46.325789	5.017	98.831696	0.522756	301.025391	0.549316
4	4005	2020-01-08 21:47:49.86342	5.017	98.728699	0.554932	298.095703	-0.091553

In [34]:
```
saved_mcd = load_model('Anom_detection_mcd')
new_prediction = predict_model(saved_mcd, data=data_unseen)
new_prediction.head()
```

```
Transformation Pipeline and Model Successfully Loaded
```

Out[34]:

	id_sensor_low_cost	phenomenon_time	battery_voltage	humidity	temperature	no_we	no_aux
0	4005	2020-01-08 23:49:51.022621	5.010	96.962494	0.013314	301.116943	1.464844
1	4005	2020-01-08 23:33:48.032856	5.013	97.828430	0.184916	303.039551	3.204346
2	4005	2020-01-08 22:07:50.305441	5.013	98.686737	0.479856	299.743652	-0.274658
3	4005	2020-01-08 21:55:46.325789	5.017	98.831696	0.522756	301.025391	0.549316
4	4005	2020-01-08 21:47:49.86342	5.017	98.728699	0.554932	298.095703	-0.091553