In [7]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pandas as pd
from collections import Counter
from sklearn.preprocessing import StandardScaler
import numpy as np
import seaborn as sns
from sklearn.preprocessing import QuantileTransformer
from scipy import stats
from scipy.stats import zscore
import scipy.stats as stats
from scipy.stats import boxcox

import seaborn as sns
import matplotlib
import matplotlib.dates as mdates
import matplotlib.pyplot as plt
import plotly.express as px
import holoviews as hv
from holoviews import opts
hv.extension('bokeh')
from bokeh.models import HoverTool
from IPython.display import HTML, display
from sklearn.ensemble import IsolationForest
import warnings
warnings.filterwarnings("ignore")
```

Load datasets and check data types / shape

In [8]:

```
dummy_path = r'C:\Users\dps\Documents\FLEET PERFORMANCE\2.Abnormal Values Detection\dummy_
df = pd.read_csv(dummy_path, delimiter = ';',  dayfirst=True, parse_dates = ['date'])
```

In [9]:

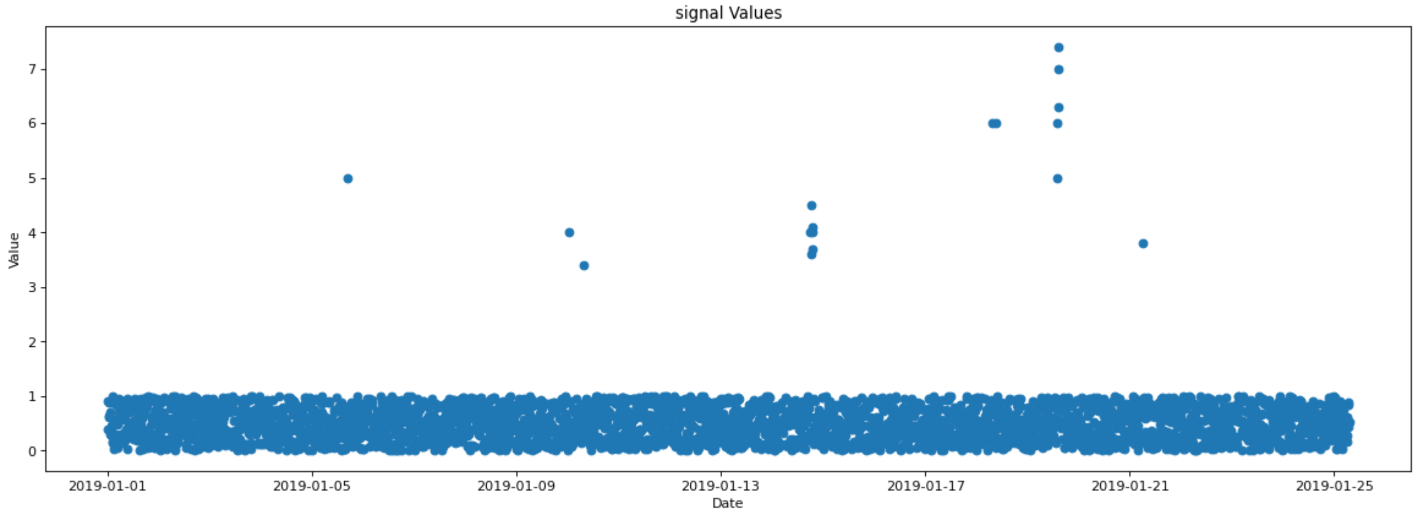
```
df.head()
```

Out[9]:

	date	signal_value
0	2019-01-01 00:00:00	0.903482
1	2019-01-01 00:10:00	0.393081
2	2019-01-01 00:20:00	0.623970
3	2019-01-01 00:30:00	0.637877
4	2019-01-01 00:40:00	0.880499

In [10]:

```
#df.set_index('dates', inplace=True)
plt.figure(figsize=(18, 6), dpi=80)
plt.scatter(df['date'],df['signal_value'])
plt.ylabel('Value')
plt.xlabel('Date')
plt.title('signal Values')
plt.show()
```



In [11]:
```
print('Variable:', '\n','\n',  'first date:', df.date.min(), '\n', 'last date:', df.date.r
```

```
Variable:

 first date: 2019-01-01 00:00:00
 last date: 2019-01-25 07:10:00
```

Introduction

In [12]:
```
# When dealing with abnormal values we must select the strategy of detection and if needed
# the method to handle them (like exclude them replace them, etc.). For the detection
# we need to understand what distribution our data follow. This is an essential part of ou
# analysis because it will determine the strategy, as there are different strategies for p
# (data that follow Gaussian distribution) and other strategies for data that do not follo
# When we know about the distribution, we select the data, the transformation models the r
# we code the algorithm to detect the abnormal values
```

EDA

In [13]:
```
stats = df.describe()
(stats.style.set_caption('Variable A: Statistics').format({'Signal':"{:,.2f}"}))
```

Out[13]:

Variable A: Statistics

	signal_value
count	3500.000000
mean	0.512763
std	0.430490
min	0.000030
25%	0.233767
50%	0.491465
75%	0.743393
max	7.400000

In [14]:
```
# From the above table we get a general description of our data. This will be useful in th
```

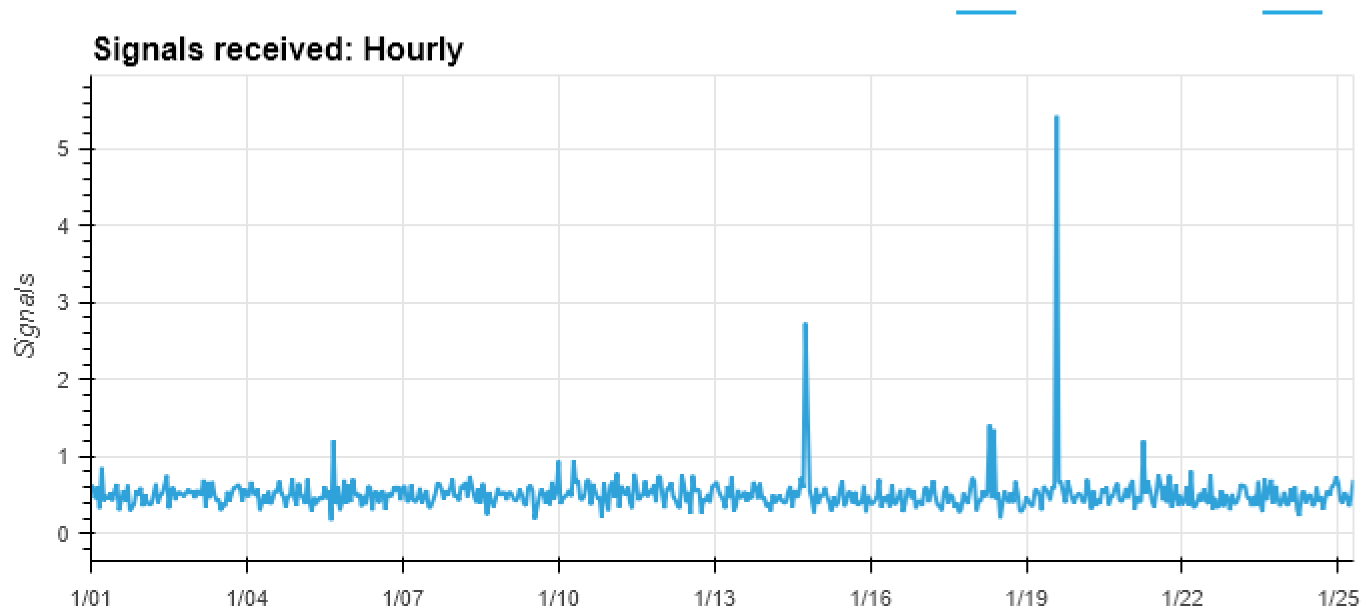
```
# drill down into more details.
```

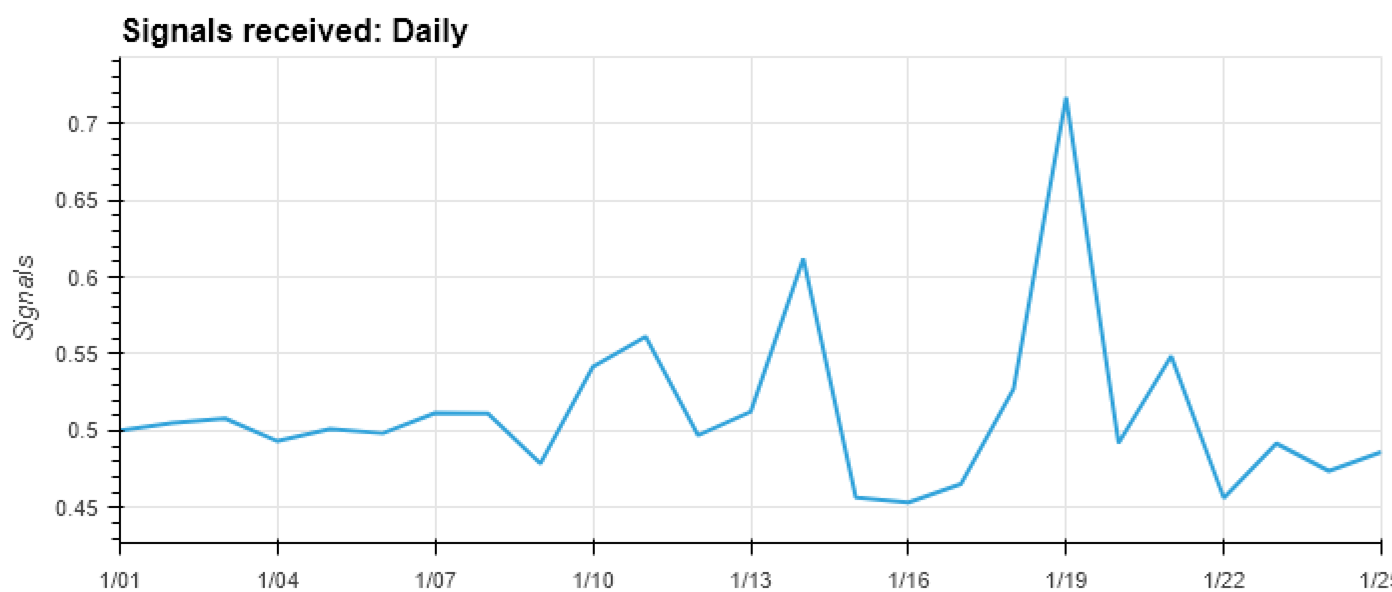
Parametric data – Distribution tests

In [15]:

```
Hourly = hv.Curve(df.set_index('date').resample('H').mean()).opts(
    opts.Curve(title="Signals received: Hourly", xlabel="", ylabel="Signals",
               width=700, height=300,tools=['hover'],show_grid=True))

Daily = hv.Curve(df.set_index('date').resample('D').mean()).opts(
    opts.Curve(title="Signals received: Daily", xlabel="", ylabel="Signals",
               width=700, height=300,tools=['hover'],show_grid=True))


(Hourly + Daily).opts(shared_axes=False).cols(1)
```

Out[15]:



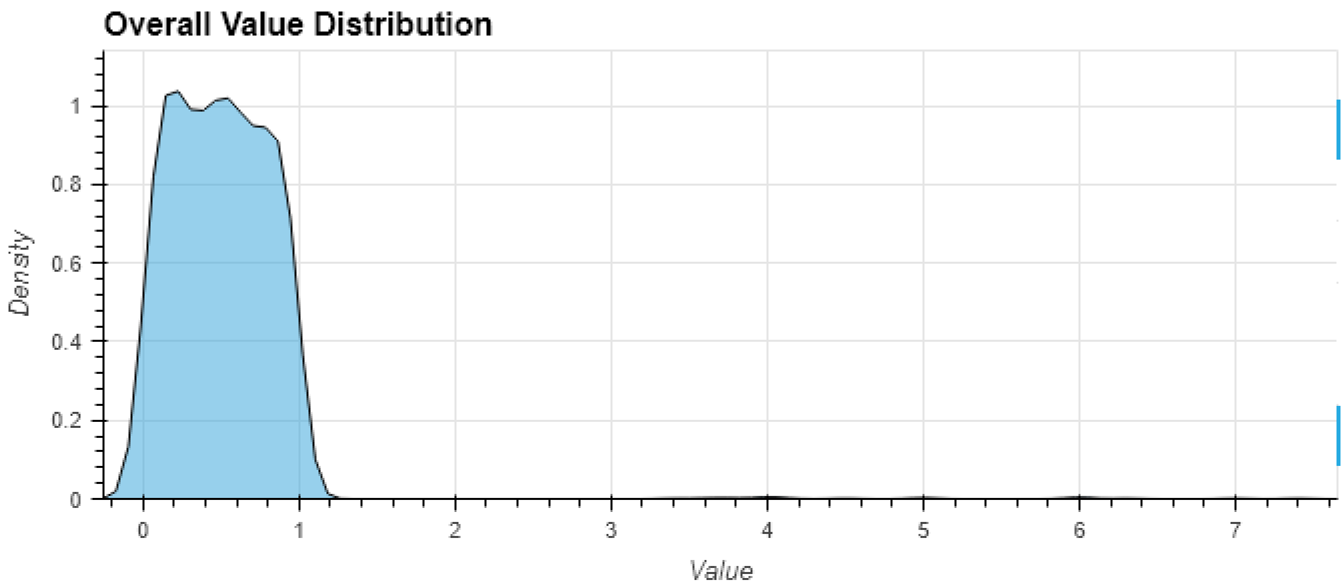


In [16]:

```
# To identify an observation as an outlier abnormal value, we start with the underlying di
# and we limit our research to univariate data that are assumed to follow an approximately
# We are allowed to do that because our data consists of observations of only a single cha
# So, we create a probability plot of the data before applying an outlier test, to check i
```

In [17]:
```
(hv.Distribution(df['signal_value'])
.opts(opts.Distribution(title="Overall Value Distribution",
                        xlabel="Value",
                        ylabel="Density",
                        width=700, height=300,
                        tools=['hover'],show_grid=True)
    ))
```

Out[17]:

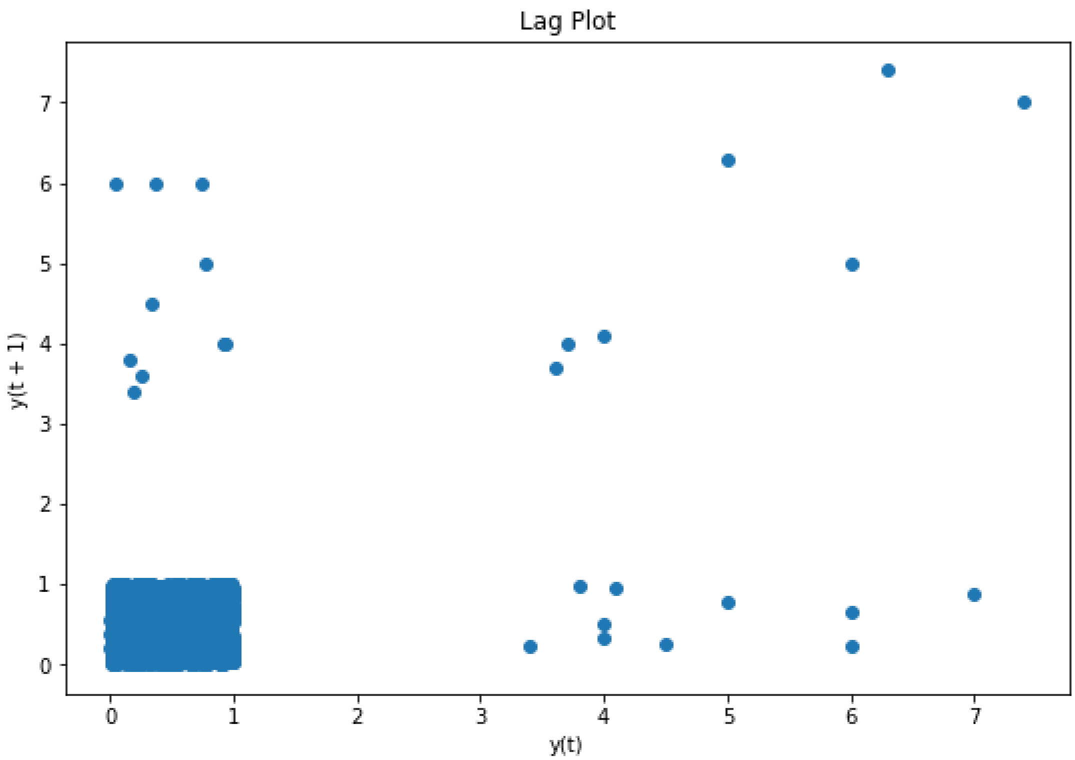


In [18]:
```
# Our assumption is that the variable A follows the bimodal distribution.
# This means that the sample data have two local maximums, hence two modes
# (the term "mode" refers to the most common number) this usually indicates that we have t
```

Lag Plot

In [19]:
```
# Next, we use a lag plot to check for patterns, randomness, and seasonality of the data.
# A lag plot is a special type of scatter plot when the two variables (X,Y) are "lagged."
# With the term lagged we mean a fixed amount of passing time.

# A plot of lag 1 is a plot of the values of Yi versus Yi-1
#     •Vertical axis: Yi for all i
#     •Horizontal axis: Yi-1 for
```

In [20]:
```
plt.figure(figsize=(9, 6))
plt.title('Lag Plot')
figure = pd.plotting.lag_plot(df['signal_value'], lag=1)
```



In [21]:

```
# This shows that the data are strongly non-random and further suggests that an autoregres
```

Bimodal distribution - transformation to Normal

In [22]:

```
# Generally, when we investigate the distribution of a dataset we must keep in mind that t
# data of variable might follow normal distribution, but we can't see because we have a sm
# Maybe, if we had data of a longer period, we would conclude that the data follow normal
# However sometimes the distribution of the data may be normal, but the data may require a

# We will transform our data to normal distribution to use parametric metrics and to run t
# In this way we will test the algorithm in a safe environment and then we will test the a
# see if we get some different results.
```

Transformation method: Quantile Transformation

In [23]:

```
# For the transformation, I tried several techniques (of Box-Cox method, power transforma
# and I concluded with the quantile fractionation as I got the best results.
# This method is centering the values on the mean value of 0 and a standard deviation of 1
# standardized result.
```

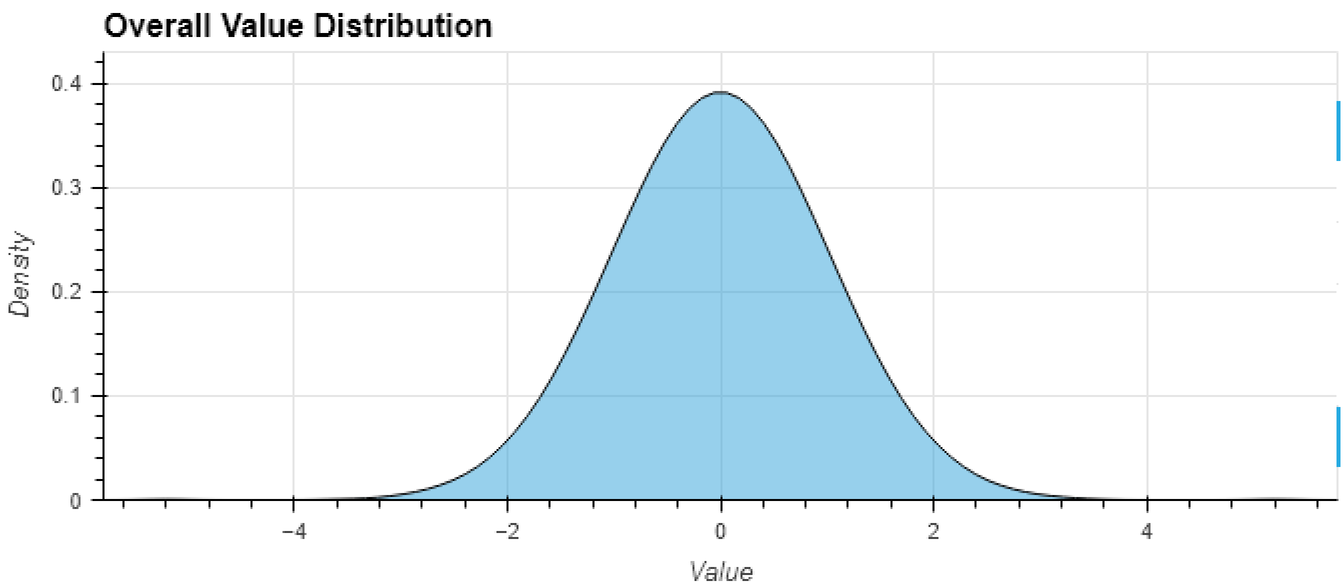
In [24]:

```
quantile = QuantileTransformer(output_distribution='normal') #n_quantiles=500

data = df['signal_value']
data.to_numpy()
print(type(data.to_numpy()))
data_to_array = data.values.reshape(-1,1)
quantile = QuantileTransformer(output_distribution='normal')
data_trans = quantile.fit_transform(data_to_array)
#pyplot.hist(data_trans)

(hv.Distribution(data_trans)
.opts(opts.Distribution(title="Overall Value Distribution",
```

```
                                xlabel="Value",
                                ylabel="Density",
                                width=700, height=300,
                                tools=['hover'],show_grid=True)
      ))
```

```
<class 'numpy.ndarray'>
```

Out[24]:



In [25]:

```
# From the above charts, we understand that the underlying data are parametrical data, he
# Z-score is a parametric method that calculates the distance between observations with t
# standard deviation.
```

Abnormal values detection

Method: Z- Score

dataset: transformed dataset to Normal Distribution

In [26]:

```
# I performed the model on the transposed data and the outcome is reflected in the below 
```

In [27]:

```
df_quartile = pd.DataFrame(data_to_array, columns = ['signal_value_quantile'])
df_quantile =  pd.concat([df, df_quartile], axis=1)
df_quantile = df_quantile.drop(['signal_value'], axis=1)
df_quantile.head()

# A variety of resamples which I may or may not use
df_hourly = df_quantile.set_index('date').resample('H').mean().reset_index()
df_daily = df_quantile.set_index('date').resample('D').mean().reset_index()

# New features
# Loop to cycle through both DataFrames
for DataFrame in [df_hourly, df_daily]:
    DataFrame['Weekday'] = pd.Categorical(DataFrame['date'].dt.strftime('%A'), categories=
    DataFrame['Hour'] = DataFrame['date'].dt.hour
    DataFrame['Day'] = DataFrame['date'].dt.weekday
    DataFrame['Month'] = DataFrame['date'].dt.month
```

```
    DataFrame['Year'] = DataFrame['date'].dt.year
    DataFrame['Month_day'] = DataFrame['date'].dt.day
    DataFrame['Lag'] = DataFrame['signal_value_quantile'].shift(1)
    DataFrame['Rolling_Mean'] = DataFrame['signal_value_quantile'].rolling(7).mean()

df_daily = df_daily.join(df_daily.groupby(['Hour','Weekday'])['signal_value_quantile'].mea
on = ['Hour','Weekday'], rsuffix='_Average')

df_daily = df_daily.dropna()
df_hourly = df_hourly.dropna()
df_hourly.head()

df_daily = df_daily.dropna()
df_hourly = df_hourly.dropna()
df_hourly.head(2)

# Daily
df_daily_model_data = df_daily[['signal_value_quantile', 'Hour', 'Day',  'Month','Month_da

# Hourly
model_data = df_hourly[['signal_value_quantile', 'Hour', 'Day', 'Month_day', 'Month','Rol
model_data.head(2)
```

Out[27]:

	signal_value_quantile	Hour	Day	Month_day	Month	Rolling_Mean	Lag
6	0.424960	6	1	1	1	0.555477	0.857504
7	0.484986	7	1	1	1	0.535759	0.424960

In [28]:

```
import scipy.stats as stats
```

In [29]:

```
model_data['Score'] = stats.zscore(model_data['signal_value_quantile'])
model_data['Outliers'] = model_data['Score'].apply(lambda x: -1 if x > 0.5 else 1)
model_data.head(2)
```

Out[29]:

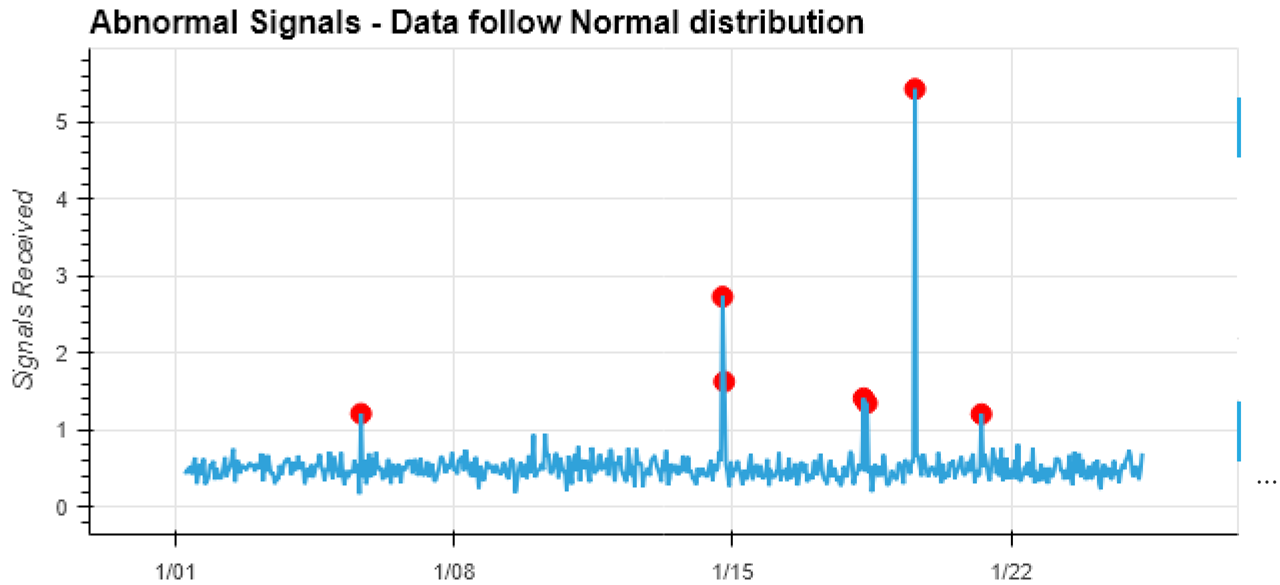
	signal_value_quantile	Hour	Day	Month_day	Month	Rolling_Mean	Lag	Score	Outliers
6	0.424960	6	1	1	1	0.555477	0.857504	-0.326805	1
7	0.484986	7	1	1	1	0.535759	0.424960	-0.102218	1

In [30]:

```
# New Anomaly Score column
df_hourly['Score'] = stats.zscore(df_hourly['signal_value_quantile'])

# Get Anomaly Score
df_hourly['Outliers'] = df_hourly['Score'].apply(lambda x: -1 if x > 2 else 1)
df_hourly.head(2)

def outliers(thresh):
    print(f'Number of Outliers below Anomaly Score Threshold {thresh}:')
    print(len(df_Z_hourly.query(f"Outliers == -1 & Score <= {thresh}")))

tooltips = [
    ('Weekday', '@Weekday'),
    ('Day', '@Month_day'),
    ('Month', '@Month'),
    ('Value', '@signal_value_quantile'),
    ('Average Vale', '@signal_value_quantile_Average'),
    ('Outliers', '@Outliers')
]
hover = HoverTool(tooltips=tooltips)
```

```
hv.Points(df_hourly.query("Outliers == -1")).opts(size=10, color='#ff0000') * hv.Curve(df_
```

Out[30]:



In [31]:

```
# The above chart, shows that the solutions is able to detect all abnormal values
```

In [32]:

```
# Below we will also try one other way to detect abnormal values.
# Specifically we will use the Isolation Forest algorithm on the origianl data to compare
# the performance with the previous solution (z-score)
```

Abnormal values detection

Method: IsolationForest

dataset: origial dataset

In [39]:

```
dummy_path = r'C:\Users\dps\Documents\FLEET PERFORMANCE\2.Abnormal Values Detection\dummy_
df = pd.read_csv(dummy_path, delimiter = ';',  dayfirst=True, parse_dates = ['date'])
```

In [40]:

```
# A variety of resamples which I may or may not use
df_hourly = df.set_index('date').resample('H').mean().reset_index()
df_daily = df.set_index('date').resample('D').mean().reset_index()
```

In [41]:

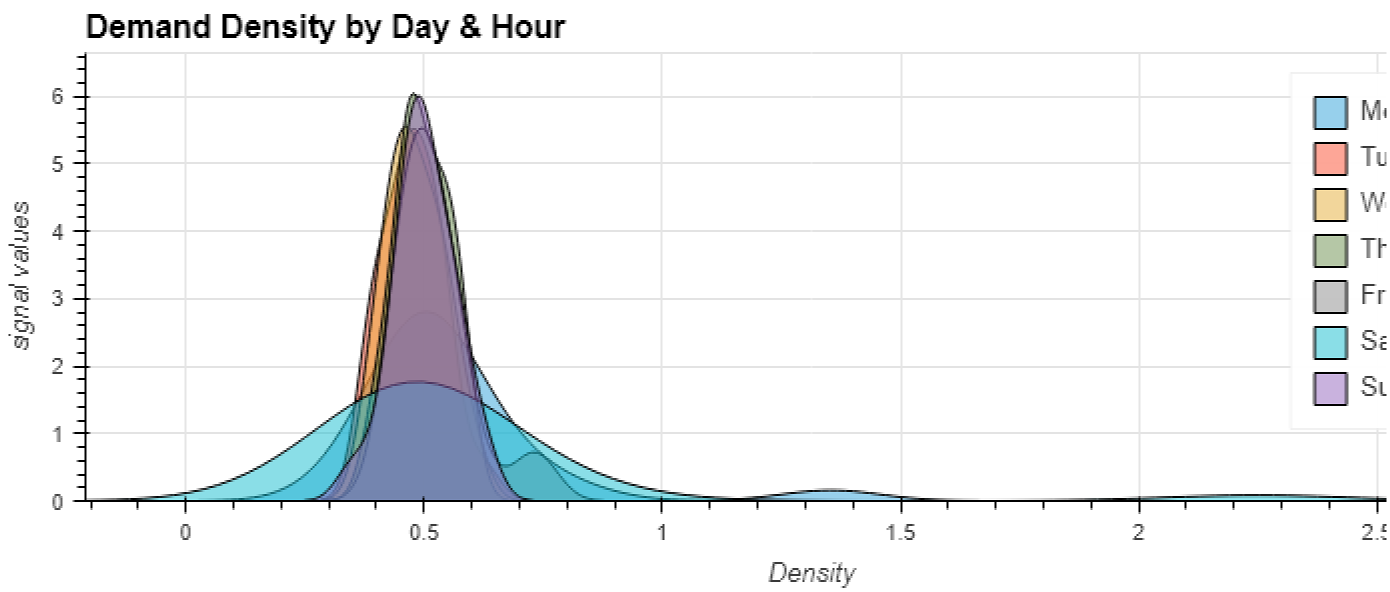
```
# New features
# Loop to cycle through both DataFrames
for DataFrame in [df_hourly, df_daily]:
    DataFrame['Weekday'] = pd.Categorical(DataFrame['date'].dt.strftime('%A'), categories=
    DataFrame['Hour'] = DataFrame['date'].dt.hour
    DataFrame['Day'] = DataFrame['date'].dt.weekday
    DataFrame['Month'] = DataFrame['date'].dt.month
    DataFrame['Year'] = DataFrame['date'].dt.year
    DataFrame['Month_day'] = DataFrame['date'].dt.day
```

```
    DataFrame['Lag'] = DataFrame['signal_value'].shift(1)
    DataFrame['Rolling_Mean'] = DataFrame['signal_value'].rolling(7).mean()
```

In [50]:

```
by_weekday = df_hourly.groupby(['Hour','Weekday']).mean()['signal_value'].unstack()
plot = hv.Distribution(by_weekday['Monday'], label='Monday') * hv.Distribution(by_weekday
plot.opts(opts.Distribution(width=800, height=300,tools=['hover'],show_grid=True, ylabel='
```

Out[50]:



In [51]:

```
df_hourly = df_hourly.join(df_hourly.groupby(['Hour','Weekday'])['signal_value'].mean(),
on = ['Hour', 'Weekday'], rsuffix='_Average')
```

In [52]:

```
df_daily = df_daily.join(df_daily.groupby(['Hour','Weekday'])['signal_value'].mean(),
on = ['Hour', 'Weekday'], rsuffix='_Average')
```

In [53]:

```
df_daily = df_daily.dropna()
df_hourly = df_hourly.dropna()
```

In [54]:

```
# Daily
df_daily_model_data = df_daily[['signal_value', 'Hour', 'Day',  'Month','Month_day','Rolli

# Hourly
model_data = df_hourly[['signal_value', 'Hour', 'Day', 'Month_day', 'Month','Rolling_Mean'
model_data.head()
```

Out[54]:

	signal_value	Hour	Day	Month_day	Month	Rolling_Mean	Lag
6	0.424960	6	1	1	1	0.555477	0.857504
7	0.484986	7	1	1	1	0.535759	0.424960
8	0.450070	8	1	1	1	0.514054	0.484986
9	0.530868	9	1	1	1	0.524821	0.450070
10	0.425466	10	1	1	1	0.499544	0.530868

In [89]:

```
IF = IsolationForest(random_state=0, contamination=0.03, n_estimators=150, max_samples=0.
IF.fit(model_data)
```

```python
# New Outliers Column
df_hourly['Outliers'] = pd.Series(IF.predict(model_data)).apply(lambda x: 1 if x == -1 els

# Get Anomaly Score
score = IF.decision_function(model_data)

# New Anomaly Score column
df_hourly['Score'] = score
df_hourly.head(2)
```

Out[89]:

	date	signal_value	Weekday	Hour	Day	Month	Year	Month_day	Lag	Rolling_Mean	Score	Outlie
6	2019-01-01 06:00:00	0.424960	Tuesday	6	1	1	2019	1	0.857504	0.555477	0.028273	0
7	2019-01-01 07:00:00	0.484986	Tuesday	7	1	1	2019	1	0.424960	0.535759	0.098571	0

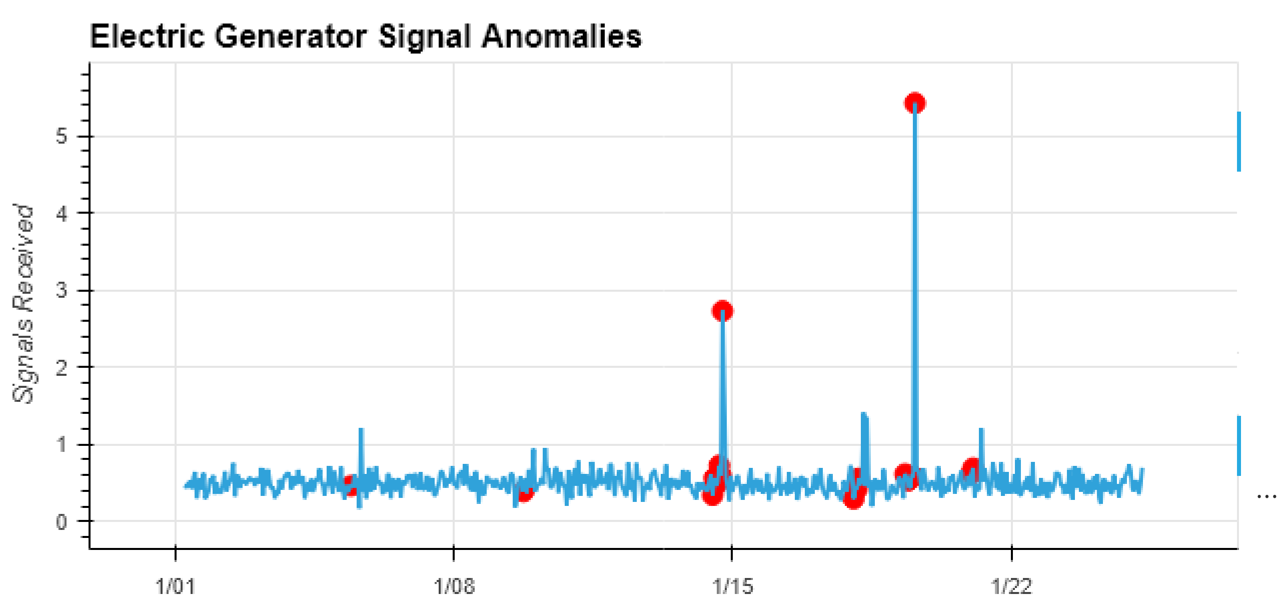
In [90]:

```python
def outliers(thresh):
    print(f'Number of Outliers below Anomaly Score Threshold {thresh}:')
    print(len(df_hourly.query(f"Outliers == 1 & Score <= {thresh}")))
```

In [91]:

```python
tooltips = [
    ('Weekday', '@Weekday'),
    ('Day', '@Month_day'),
    ('Month', '@Month'),
    ('Value', '@signal_value'),
    ('Average Vale', '@signal_value_Average'),
    ('Outliers', '@Outliers')
]
hover = HoverTool(tooltips=tooltips)

hv.Points(df_hourly.query("Outliers == 1")).opts(size=10, color='#ff0000') * hv.Curve(df_h
```

Out[91]:



In [92]:

```python
# A we can observe from the the above chart, the results of this algorith are not quite go
```

In [93]:
```
# In the next step we will use to z-score in the original data
```

Abnormal values detection

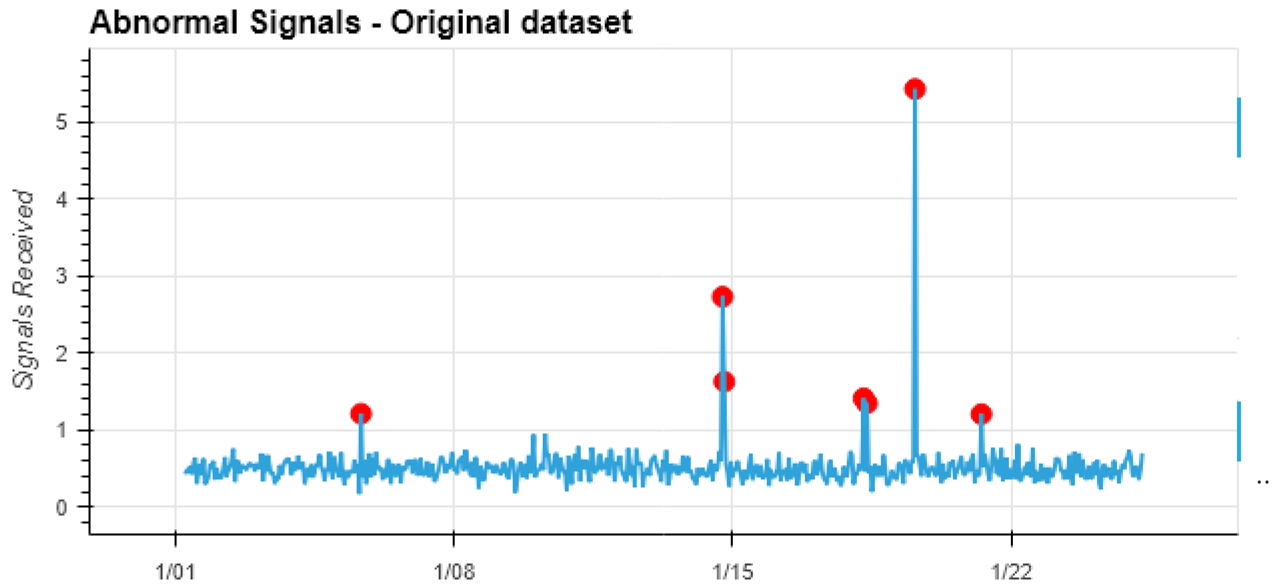
Method: Z score

dataset: origial dataset

In [94]:
```
# A variety of resamples which I may or may not use
df_hourly = df.set_index('date').resample('H').mean().reset_index()
df_daily = df.set_index('date').resample('D').mean().reset_index()

# New features
# Loop to cycle through both DataFrames
for DataFrame in [df_hourly, df_daily]:
    DataFrame['Weekday'] = pd.Categorical(DataFrame['date'].dt.strftime('%A'), categories=
    DataFrame['Hour'] = DataFrame['date'].dt.hour
    DataFrame['Day'] = DataFrame['date'].dt.weekday
    DataFrame['Month'] = DataFrame['date'].dt.month
    DataFrame['Year'] = DataFrame['date'].dt.year
    DataFrame['Month_day'] = DataFrame['date'].dt.day
    DataFrame['Lag'] = DataFrame['signal_value'].shift(1)
    DataFrame['Rolling_Mean'] = DataFrame['signal_value'].rolling(7).mean()

df_daily = df_daily.join(df_daily.groupby(['Hour','Weekday'])['signal_value'].mean(),
on = ['Hour', 'Weekday'], rsuffix='_Average')

df_daily = df_daily.dropna()
df_hourly = df_hourly.dropna()

# Daily
df_daily_model_data = df_daily[['signal_value', 'Hour', 'Day',  'Month','Month_day','Rolli

# Hourly
model_data = df_hourly[['signal_value', 'Hour', 'Day', 'Month_day', 'Month','Rolling_Mean'

model_data['Score'] = stats.zscore(model_data['signal_value'])
model_data['Outliers'] = model_data['Score'].apply(lambda x: -1 if x > 2 else 1)

# New Anomaly Score column
df_hourly['Score'] = stats.zscore(df_hourly['signal_value'])

# Get Anomaly Score
df_hourly['Outliers'] = df_hourly['Score'].apply(lambda x: -1 if x > 2 else 1)

def outliers(thresh):
    print(f'Number of Outliers below Anomaly Score Threshold {thresh}:')
    print(len(df_hourly.query(f"Outliers == -1 & Score <= {thresh}")))

tooltips = [
    ('Weekday', '@Weekday'),
    ('Day', '@Month_day'),
    ('Month', '@Month'),
    ('Value', '@signal_value'),
    ('Average Vale', '@signal_value_Average'),
    ('Outliers', '@Outliers')
]
hover = HoverTool(tooltips=tooltips)
```

```
hv.Points(df_hourly.query("Outliers == -1")).opts(size=10, color='#ff0000') * hv.Curve(df_
```

Out[94]:



In [95]:

```
# As we can see the z-score performs very well in the original dataset as well.
```

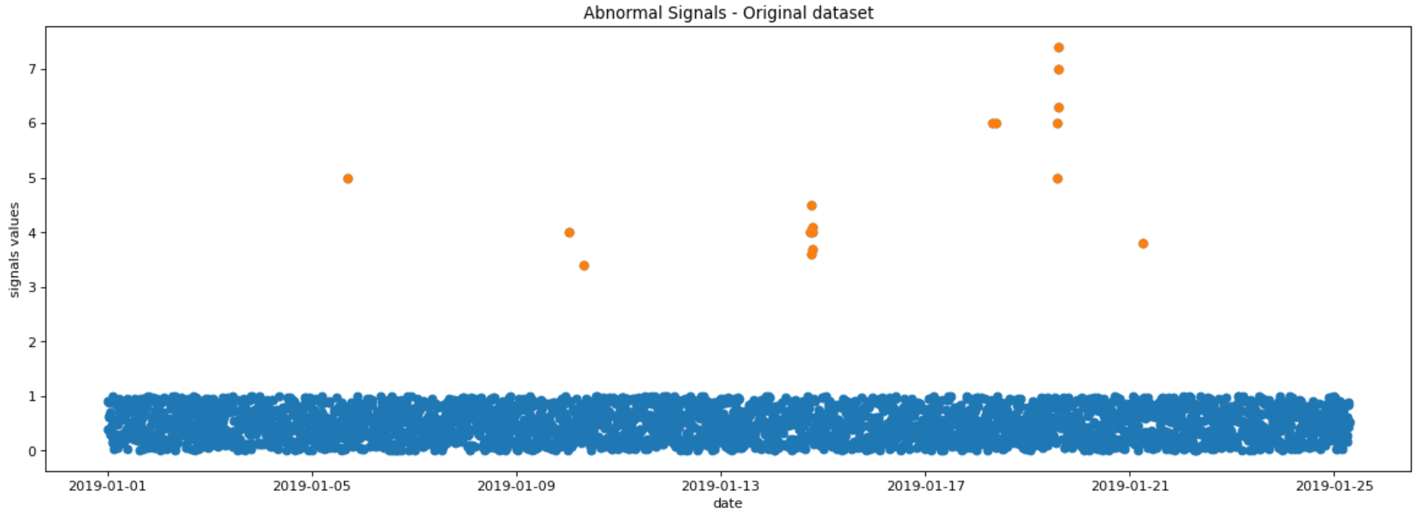
Simple way of detection and presentation of the results

In [96]:

```
# We may also perform the same detection model using a more simple way of a calcualtion ar
# matplotlib library for the charts. This way is more fast but not scalable as the previo
# It is presented for a quick solution.
```

In [97]:

```
df=df.dropna()
df.index=[i for i in range(0,len(df))]#reindexing | change accordingle to reset index of c
d = pd.DataFrame(stats.zscore(df['signal_value']))
d.columns = ['z_score']
d=d[(d['z_score']>2) | (d['z_score']<-2)]

signal_value = []
date = []
for i in df.index:
    if( i in d.index):
        signal_value.append(df.loc[i]['signal_value'])
        date.append(df.loc[i]['date'])

#df.plot(x = 'date', y = 'signal_value', figsize = (16,6), kind = 'scatter', style = 'o' ,

# import matplotlib.pyplot as plt
plt.figure(figsize=(18, 6), dpi=80)
plt.scatter(df['date'],df['signal_value'])
plt.scatter(date,signal_value)
plt.title('Abnormal Signals - Original dataset')
plt.ylabel('signals values')
plt.xlabel('date')
plt.show()
```


Abnormal Signals - Original dataset
signals values
date
0
1
2
3
4
5
6
7
2019-01-01
2019-01-05
2019-01-09
2019-01-13
2019-01-17
2019-01-21
2019-01-25

In [ ]: