Time Series Analisys on Geoespatial Data with Python

Author: João Otavio Nascimento Firigato

email: joaootavionf007@gmail.com

LinkedIn: https://www.linkedin.com/in/jo%C3%A3o-otavio-firigato-4876b3aa/

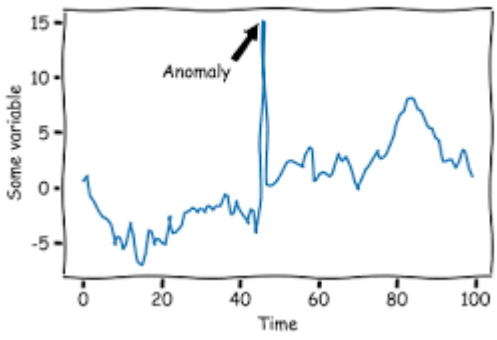
First instructions:

✅ Access the link to join our private WhatsApp community for students: https://chat.whatsapp.com/EPn27ZgR07lF3e1vnj8FiI

❗ It is important to access the Whatsapp Group to get the Colab Notebooks, as the PDF files are protected from text copying.

Chapter 12 - Anomaly and Outlier Detection

Anomaly detection is a data science application that combines several tasks such as classification, regression, and clustering. Anomaly detection is an essential tool for improving your ability to make better business decisions. It is a process in machine learning that identifies data points, events, and observations that deviate from the normal behavior of a data set. Time series anomaly detection is a way to gain insights into specific business strategies. Time series outlier detection warrants extensive exploration, given its potential to significantly influence any predictions we make.

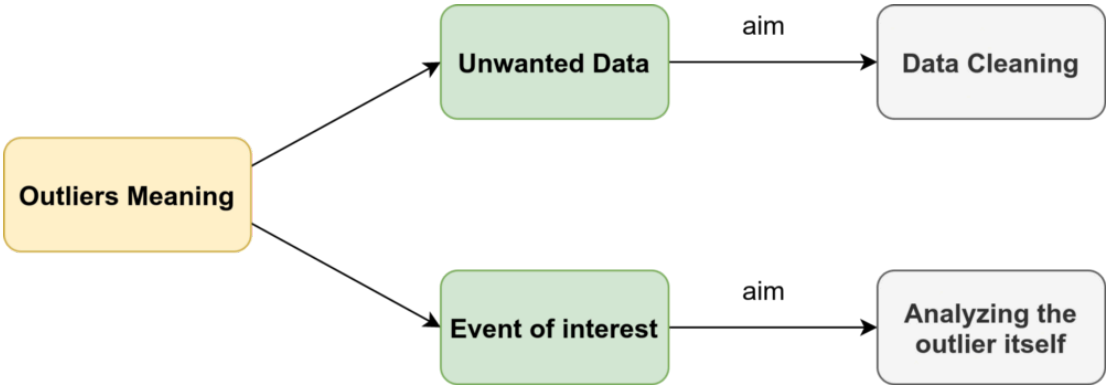


What is an Outlier: An observation that deviates so much from other observations that it raises suspicions that it was generated by a different mechanism.

Therefore, you can think of outliers as observations that do not follow the expected behavior.

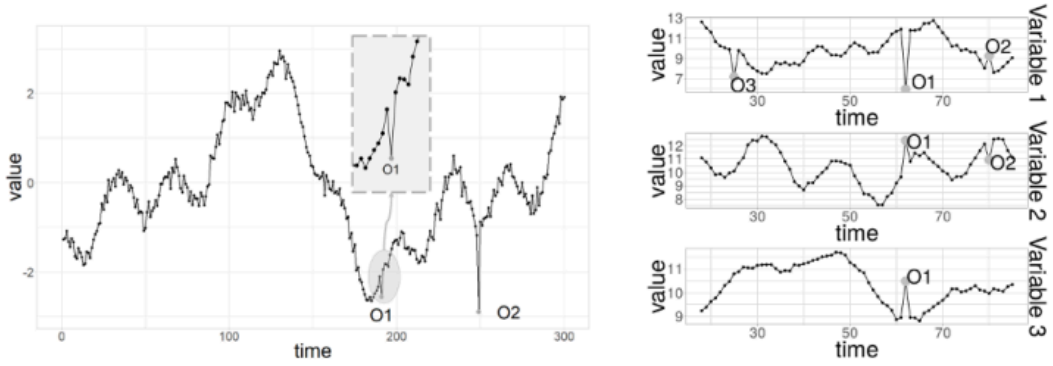
Types of outliers

Outliers in time series data can be classified into three main types. First, point outliers are individual data points that deviate significantly from the expected pattern in the data set. They usually arise due to measurement error or extreme but legitimate data variances. Second, contextual outliers, as the name suggests, are anomalous within a specific context. While they may appear normal in general circumstances, they become apparent when considering the temporal nature of the data. For example, high air conditioning sales in December may constitute a contextual outlier. Finally, collective outliers refer to a collection of data points that, as a whole, deviate from the expected behavior of the entire data set. They may not comprise individual outliers, but they can exhibit anomalous behavior when considered collectively, often violating the dynamics of the system. Identifying these outliers accurately is critical for robust time series analysis. In this article, we will focus on the first type of outliers.



Point outlier

A point outlier is a piece of data that behaves unusually at a specific time instance when compared to other values in the time series (global outlier) or to its neighboring points (local outlier).

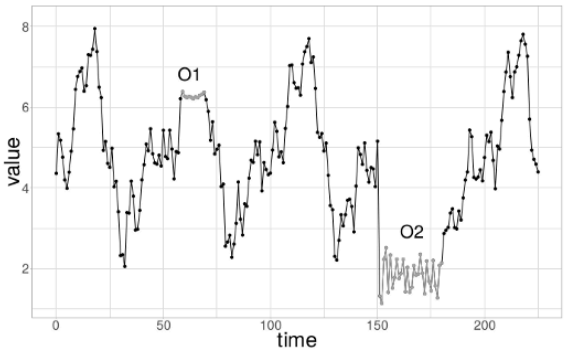


(a) Univariate time series.

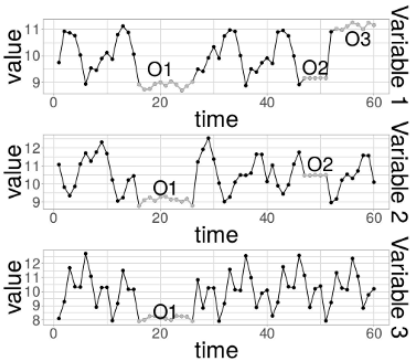
(b) Multivariate time series.

Subsequence outlier

This means consecutive points in time whose joint behavior is unusual, although each observation individually is not necessarily a point outlier. Subsequence outliers can also be global or local, and can affect one (univariate subsequence outlier) or more (multivariate subsequence outlier) time-dependent variables.



(a) Univariate time series.



(b) Multivariate time series.

Outlier detection methods

Let's access our example dataset in Drive to use in our use case:

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

Link to Dataset:
https://drive.google.com/file/d/10omy6A3pjEAbQzISjglvSkpiCpBpx3A_/view?usp=sharing

In [ ]:
```
path = '/content/drive/MyDrive/Datasets_TS/Global Temperature.csv'
```

In [ ]:
```
df = pd.read_csv(path)
```

In [ ]:
```
df.head()
```

Out[ ]:

	Year	Month	Monthly Anomaly	Monthly Unc.	Annual Anomaly	Annual Unc	Five-Year Anomaly	Five-Year Unc.	Ten-Year Anomaly	Ten Yea Unc
0	1850	1	-0.801	0.482	NaN	NaN	NaN	NaN	NaN	Nal
1	1850	2	-0.102	0.592	NaN	NaN	NaN	NaN	NaN	Nal
2	1850	3	-0.119	0.819	NaN	NaN	NaN	NaN	NaN	Nal
3	1850	4	-0.485	0.575	NaN	NaN	NaN	NaN	NaN	Nal
4	1850	5	-0.351	0.549	NaN	NaN	NaN	NaN	NaN	Nal

Let's adjust our dataframe to a time series:

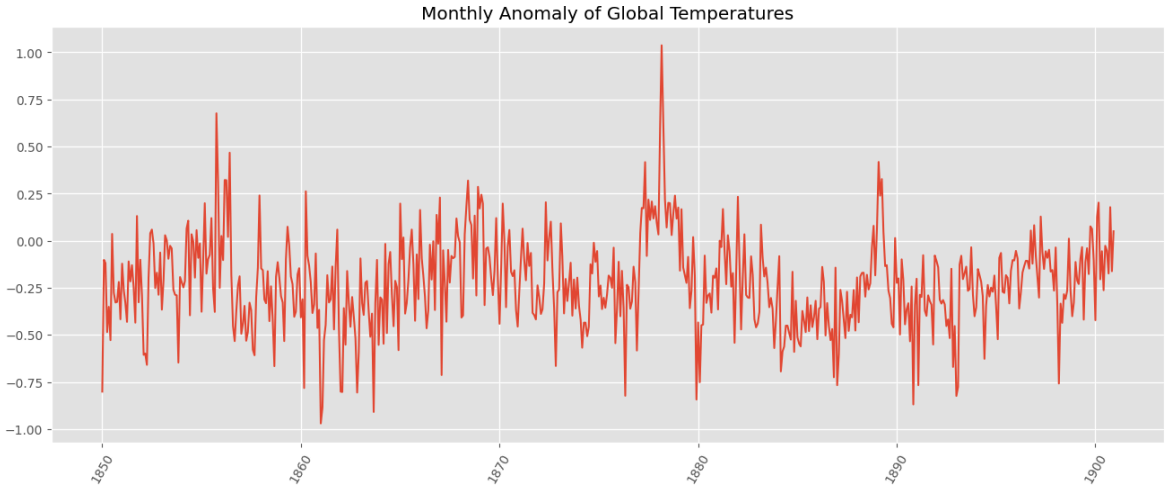
In [ ]:
```
df.rename(columns={' Month': 'Month'}, inplace=True)
```

In [ ]:
```
df['Date'] = pd.to_datetime(df.Year.astype(str) + '/' + df.Month.astype(str) + '
```

In [ ]:
```
df.set_index('Date', inplace=True)
```

In [ ]:
```
df_1900 = df[:'1900'].copy()
```

In [ ]:
```
plt.figure(figsize=(16,6))
plt.style.use("ggplot")
plt.plot(df_1900.index, df_1900['Monthly Anomaly'])
plt.title('Monthly Anomaly of Global Temperatures')
plt.xticks(rotation=60)
plt.show()
```



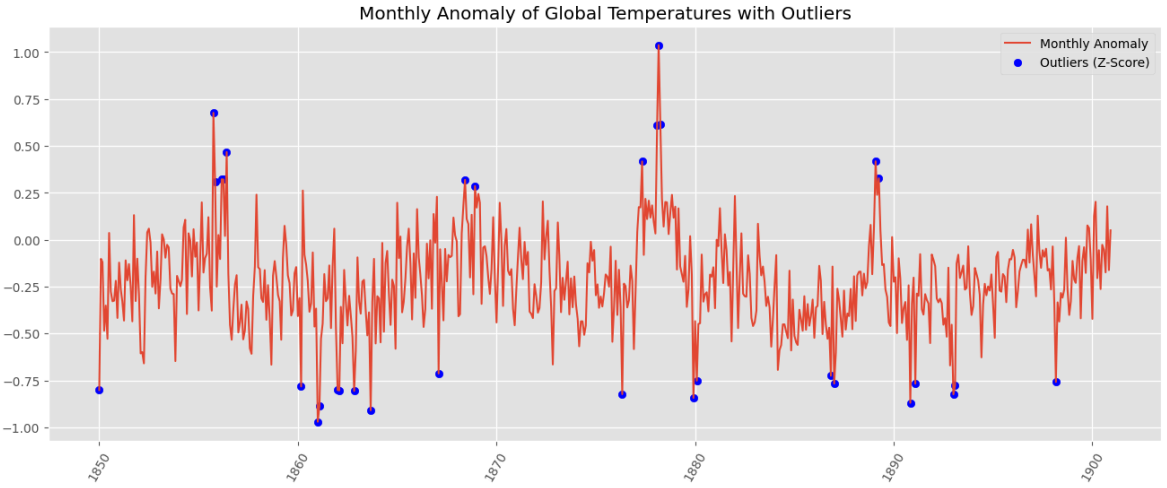
Z-Score

The z-score is a statistical measure that indicates the number of standard deviations that a data point is above or below the mean. Typically, data points that have z-scores that fall beyond a certain threshold (e.g., 2 or 3 standard deviations from the mean) are considered outliers. Note that this method is highly susceptible to extreme values in the data, as they will significantly impact the mean and standard deviation, leading to

misleading z-scores for other data points. Additionally, it assumes normal distribution, which may not be true for small sample sizes.

In [ ]:
```
import numpy as np


df_1900['Z-Score'] = np.abs((df_1900['Monthly Anomaly'] - df_1900['Monthly Anoma
outliers_z_score = df_1900[df_1900['Z-Score'] > 2]
```

In [ ]:
```
plt.figure(figsize=(16, 6))
plt.style.use("ggplot")
plt.plot(df_1900.index, df_1900['Monthly Anomaly'], label='Monthly Anomaly')
plt.scatter(outliers_z_score.index, outliers_z_score['Monthly Anomaly'], color='
plt.title('Monthly Anomaly of Global Temperatures with Outliers')
plt.xticks(rotation=60)
plt.legend()
plt.show()
```



In [ ]:
```
outliers_z_score
```

Out[ ]:

Date	Year	Month	Monthly Anomaly	Monthly Unc.	Annual Anomaly	Annual Unc	Five-Year Anomaly	Five-Year Unc.	Ten-Year Anomaly
1850-01-01	1850	1	-0.801	0.482	NaN	NaN	NaN	NaN	NaN
1855-10-01	1855	10	0.676	0.586	0.051	0.307	-0.16	0.163	-0.2
1855-11-01	1855	11	0.307	0.459	0.061	0.337	-0.161	0.161	-0.202
1856-03-01	1856	3	0.322	0.696	0.052	0.375	-0.18	0.157	-0.216
1856-04-01	1856	4	0.321	0.93	-0.035	0.354	-0.178	0.156	-0.216
1856-06-01	1856	6	0.467	0.68	-0.076	0.347	-0.17	0.153	-0.217
1860-03-01	1860	3	-0.782	0.414	-0.258	0.148	-0.309	0.116	-0.28
1861-01-01	1861	1	-0.970	0.659	-0.439	0.224	-0.344	0.105	-0.293
1861-02-01	1861	2	-0.885	0.373	-0.419	0.221	-0.343	0.108	-0.293
1862-01-01	1862	1	-0.801	0.285	-0.385	0.131	-0.363	0.108	-0.295
1862-02-01	1862	2	-0.803	0.382	-0.398	0.128	-0.364	0.109	-0.298
1862-11-01	1862	11	-0.806	0.659	-0.388	0.135	-0.359	0.126	-0.276
1863-09-01	1863	9	-0.909	0.317	-0.397	0.15	-0.307	0.119	-0.256
1867-02-01	1867	2	-0.713	0.184	-0.149	0.155	-0.113	0.095	-0.194
1868-06-01	1868	6	0.319	0.419	-0.013	0.109	-0.102	0.104	-0.168
1868-12-01	1868	12	0.286	0.708	0.026	0.165	-0.092	0.121	-0.158
1876-05-01	1876	5	-0.823	0.224	-0.312	0.104	-0.107	0.079	-0.161
1877-05-01	1877	5	0.417	0.46	0.058	0.081	-0.063	0.073	-0.154
1878-02-01	1878	2	0.611	0.316	0.299	0.152	-0.09	0.066	-0.164

Date	Year	Month	Monthly Anomaly	Monthly Unc.	Annual Anomaly	Annual Unc	Five-Year Anomaly	Five-Year Unc.	Ten-Year Anomaly
1878-03-01	1878	3	1.037	0.265	0.284	0.147	-0.092	0.066	-0.163
1878-04-01	1878	4	0.616	0.178	0.286	0.146	-0.095	0.066	-0.16
1879-12-01	1879	12	-0.843	0.378	-0.351	0.084	-0.068	0.072	-0.178
1880-02-01	1880	2	-0.751	0.222	-0.361	0.087	-0.084	0.072	-0.182
1886-11-01	1886	11	-0.725	0.128	-0.46	0.087	-0.322	0.089	-0.313
1887-01-01	1887	1	-0.767	0.269	-0.456	0.085	-0.311	0.089	-0.315
1889-02-01	1889	2	0.418	0.204	0.016	0.096	-0.283	0.068	-0.329
1889-04-01	1889	4	0.327	0.141	-0.025	0.088	-0.277	0.068	-0.322
1890-11-01	1890	11	-0.869	0.109	-0.397	0.072	-0.272	0.067	-0.293
1891-02-01	1891	2	-0.767	0.241	-0.39	0.078	-0.269	0.068	-0.285
1893-01-01	1893	1	-0.824	0.191	-0.377	0.091	-0.321	0.062	-0.232
1893-02-01	1893	2	-0.776	0.147	-0.365	0.088	-0.318	0.062	-0.231
1898-03-01	1898	3	-0.758	0.181	-0.279	0.083	-0.154	0.063	-0.185

IQR Method

The IQR (interquartile range) outlier detection method involves calculating the range between the first quartile (25th percentile) and the third quartile (75th percentile) of the data, known as the IQR. Data points that fall below the first quartile minus 1.5 times the IQR or above the third quartile plus 1.5 times the IQR are considered outliers. This method is robust to extreme values and does not assume any specific distribution of the data, making it useful in a variety of scenarios for identifying potential outliers.

In [ ]:
```
Q1 = df_1900['Monthly Anomaly'].quantile(0.25)
Q3 = df_1900['Monthly Anomaly'].quantile(0.75)
IQR = Q3 - Q1
```

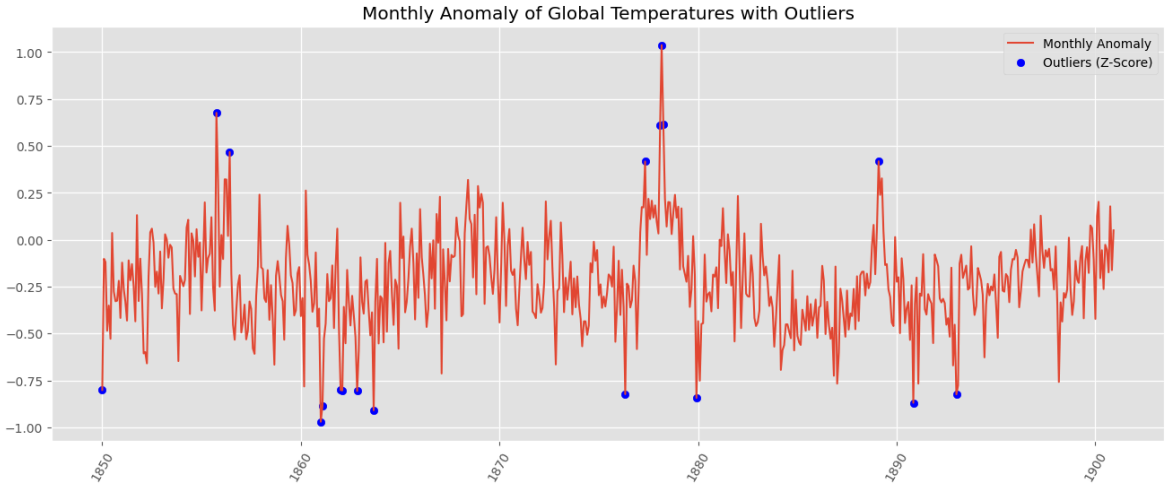
```
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers_iqr = df_1900[(df_1900['Monthly Anomaly'] < lower_bound) | (df_1900['Mo
```

In [ ]:
```
outliers_iqr
```

Out[ ]:

Date	Year	Month	Monthly Anomaly	Monthly Unc.	Annual Anomaly	Annual Unc	Five-Year Anomaly	Five-Year Unc.	Ten-Year Anomaly
1850-01-01	1850	1	-0.801	0.482	NaN	NaN	NaN	NaN	NaN
1855-10-01	1855	10	0.676	0.586	0.051	0.307	-0.16	0.163	-0.2
1856-06-01	1856	6	0.467	0.68	-0.076	0.347	-0.17	0.153	-0.217
1861-01-01	1861	1	-0.970	0.659	-0.439	0.224	-0.344	0.105	-0.293
1861-02-01	1861	2	-0.885	0.373	-0.419	0.221	-0.343	0.108	-0.293
1862-01-01	1862	1	-0.801	0.285	-0.385	0.131	-0.363	0.108	-0.295
1862-02-01	1862	2	-0.803	0.382	-0.398	0.128	-0.364	0.109	-0.298
1862-11-01	1862	11	-0.806	0.659	-0.388	0.135	-0.359	0.126	-0.276
1863-09-01	1863	9	-0.909	0.317	-0.397	0.15	-0.307	0.119	-0.256
1876-05-01	1876	5	-0.823	0.224	-0.312	0.104	-0.107	0.079	-0.161
1877-05-01	1877	5	0.417	0.46	0.058	0.081	-0.063	0.073	-0.154
1878-02-01	1878	2	0.611	0.316	0.299	0.152	-0.09	0.066	-0.164
1878-03-01	1878	3	1.037	0.265	0.284	0.147	-0.092	0.066	-0.163
1878-04-01	1878	4	0.616	0.178	0.286	0.146	-0.095	0.066	-0.16
1879-12-01	1879	12	-0.843	0.378	-0.351	0.084	-0.068	0.072	-0.178
1889-02-01	1889	2	0.418	0.204	0.016	0.096	-0.283	0.068	-0.329
1890-11-01	1890	11	-0.869	0.109	-0.397	0.072	-0.272	0.067	-0.293
1893-01-01	1893	1	-0.824	0.191	-0.377	0.091	-0.321	0.062	-0.232

In [ ]:
```
plt.figure(figsize=(16, 6))
plt.style.use("ggplot")
plt.plot(df_1900.index, df_1900['Monthly Anomaly'], label='Monthly Anomaly')
plt.scatter(outliers_iqr.index, outliers_iqr['Monthly Anomaly'], color='blue', l
plt.title('Monthly Anomaly of Global Temperatures with Outliers')
plt.xticks(rotation=60)
plt.legend()
plt.show()
```

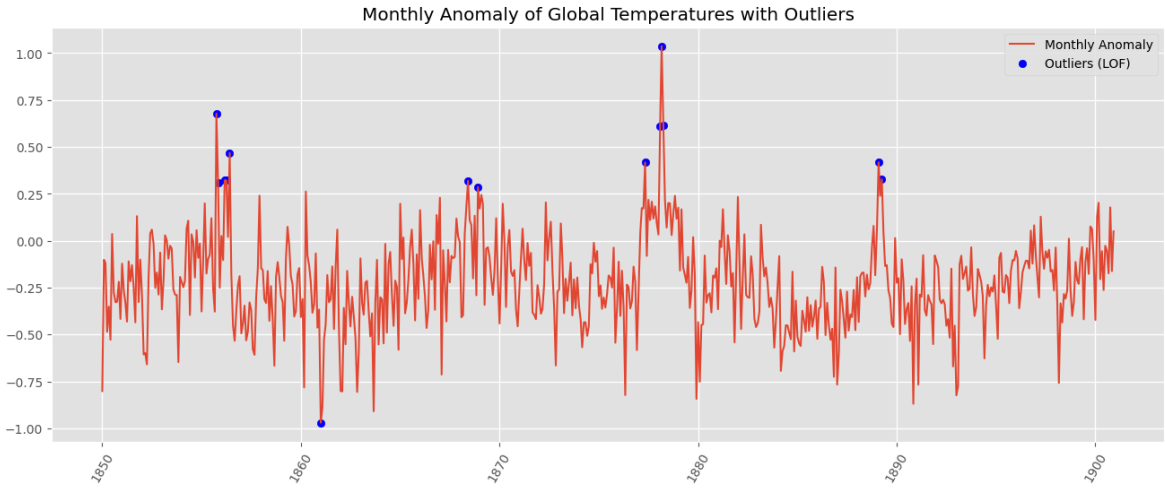


Local Outlier Factor

Local Outlier Factor (LOF) is a density-based algorithm used to detect outliers in datasets. It measures the local deviation of a given sample's density from its neighbors. It is particularly effective for datasets containing regions of varying density. LOF assigns scores to each data point in a dataset, where a higher LOF score indicates that the data point is more likely to be an outlier. The LOF score of a data point is essentially a ratio of the data point's average local density and the average local densities of its nearest neighbors. If a data point's local density is significantly lower than that of its neighbors, it is considered an outlier. This means that LOF can effectively identify both global and local outliers because it takes into account the local context of each data point.

In [ ]:
```
from sklearn.neighbors import LocalOutlierFactor

lof = LocalOutlierFactor(n_neighbors=20, contamination='auto')
df_1900['outlier'] = lof.fit_predict(df_1900[['Monthly Anomaly']])
outliers_lof = df_1900[df_1900['outlier'] == -1]
```

In [ ]:
```
plt.figure(figsize=(16, 6))
plt.style.use("ggplot")
plt.plot(df_1900.index, df_1900['Monthly Anomaly'], label='Monthly Anomaly')
plt.scatter(outliers_lof.index, outliers_lof['Monthly Anomaly'], color='blue', l
plt.title('Monthly Anomaly of Global Temperatures with Outliers')
plt.xticks(rotation=60)
plt.legend()
plt.show()
```



Isolation Forests

Isolation Forest is a tree-based anomaly detection algorithm. For the target variable, the algorithm chooses a random "split value." This split value is a specific point between the smallest (minimum) and largest (maximum) values that the chosen feature takes on in the dataset. The data is then split into two groups based on this split value: one group includes all data points with the chosen feature value less than or equal to the split value, and the other group includes all points with the feature value greater than the split value. By repeating this process recursively on each resulting subset of the data, the algorithm builds a decision tree where each split separates data points based on the randomly selected feature value relative to the random split value. Outliers are generally fewer and less numerous and distinct, so they are isolated closer to the root of the tree, resulting in shorter paths.

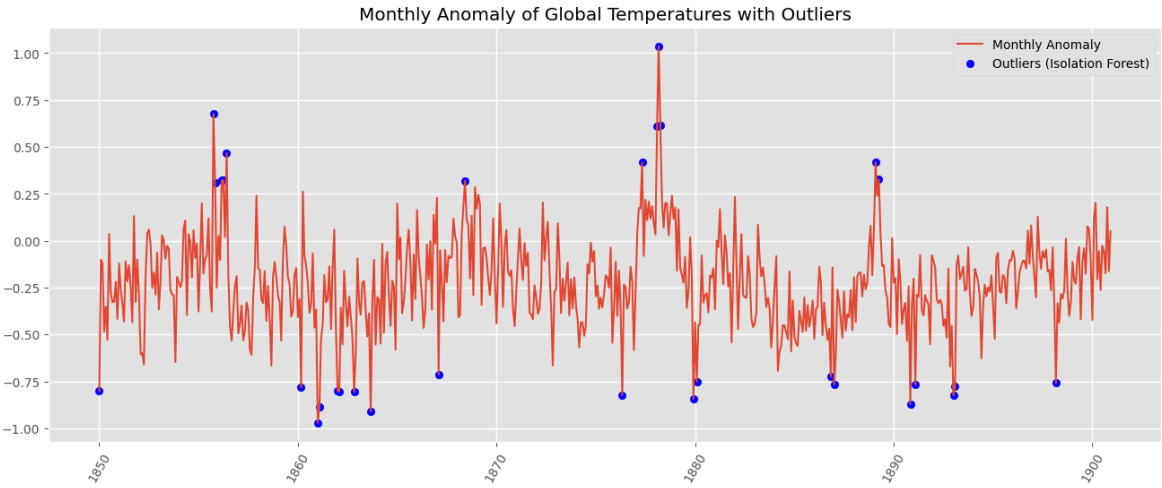
In [ ]:
```
from sklearn.ensemble import IsolationForest
```

In [ ]:
```
model = IsolationForest(contamination=0.05, random_state=42)

X = df_1900[['Monthly Anomaly']]
model.fit(X)

df_1900['Anomaly'] = model.predict(X)
```

We selected the anomalies:

In [ ]:
```
outliers_isofor = df_1900[df_1900['Anomaly'] == -1]
```

In [ ]:
```
plt.figure(figsize=(16, 6))
plt.style.use("ggplot")
plt.plot(df_1900.index, df_1900['Monthly Anomaly'], label='Monthly Anomaly')
plt.scatter(outliers_isofor.index, outliers_isofor['Monthly Anomaly'], color='bl
plt.title('Monthly Anomaly of Global Temperatures with Outliers')
plt.xticks(rotation=60)
plt.legend()
plt.show()
```



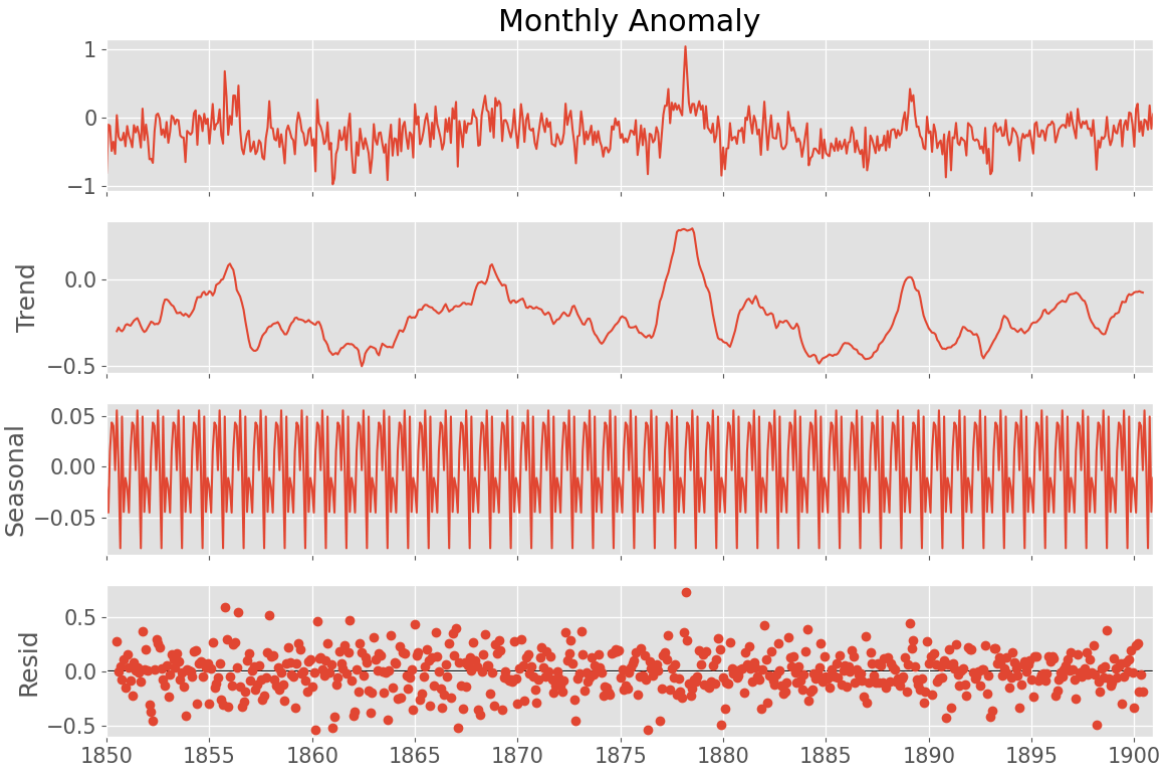
Series Decomposition

Series Decomposition is a method used in time series analysis where a time series is decomposed into trend, seasonal, and residual components. This can be achieved using methods such as additive or multiplicative decomposition, or more sophisticated methods such as STL (Seasonal and Trend Decomposition Using Loess).

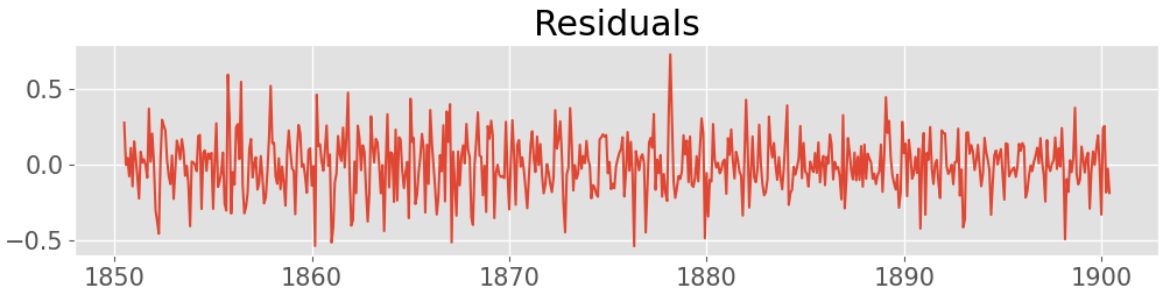
For outlier detection, we typically focus on the residual component. Since the trend and seasonal components are removed, any significant spike or dip in the residual component (which should ideally be random noise) can be considered an outlier. This makes Series Decomposition a valuable technique for time series outlier detection.

In [ ]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
```

In [ ]:
```
plt.rc('figure',figsize=(12,8))
plt.rc('font',size=15)
result = seasonal_decompose(df_1900['Monthly Anomaly'],model='additive')
fig = result.plot()
```



In [ ]:
```
plt.subplot(3,1,3)
plt.plot(result.resid)
plt.title('Residuals')

plt.show()
```



From the residuals, we calculate the mean and standard deviation to obtain the lower and upper bounds:

In [ ]:
```
res_mean = result.resid.mean()
resid_std = result.resid.std()

lower_bound = res_mean - 3*resid_std
upper_bound = res_mean + 3*resid_std
```

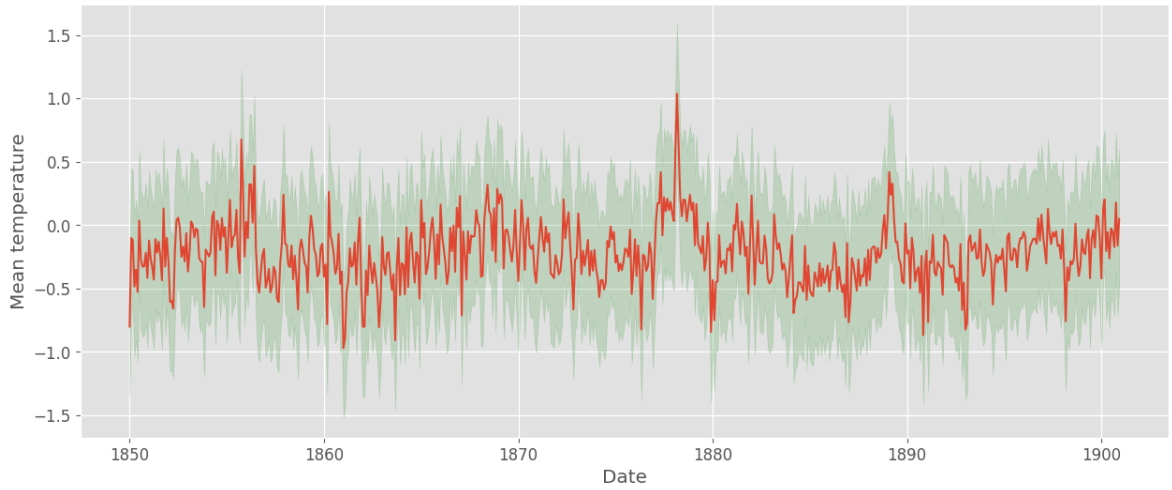
In [ ]:
```
upper_bound
```

Out[ ]:
```
np.float64(0.5519172730552877)
```

In [ ]:
```
plt.rc('font', size=12)
fig, ax = plt.subplots(figsize=(15, 6))

plt.plot(df_1900['Monthly Anomaly'])
plt.fill_between(df_1900['Monthly Anomaly'].index, df_1900['Monthly Anomaly'] -
```

```
ax.set_xlabel('Date')
ax.set_ylabel('Mean temperature')
```

Out[ ]:
```
Text(0, 0.5, 'Mean temperature')
```



Anomalies are outside the lower and upper limits:

In [ ]:
```
anomalies = df_1900['Monthly Anomaly'][(df_1900['Monthly Anomaly'] < lower_bound
```

In [ ]:
```
anomalies
```

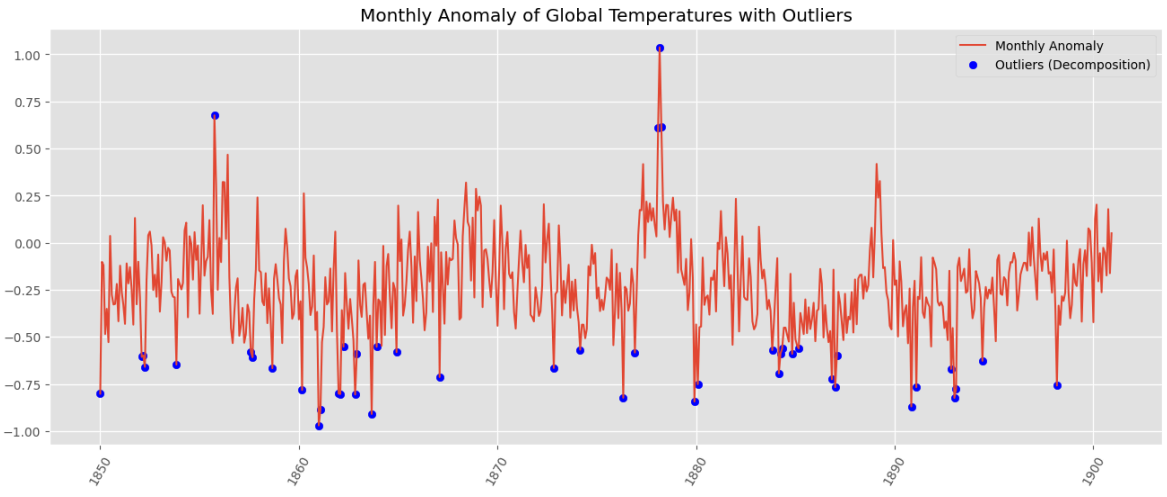
Out[ ]:

Date	Monthly Anomaly
1850-01-01	-0.801
1852-02-01	-0.606
1852-03-01	-0.599
1852-04-01	-0.659
1853-11-01	-0.647
1855-10-01	0.676
1857-08-01	-0.579
1857-09-01	-0.608
1858-09-01	-0.666
1860-03-01	-0.782
1861-01-01	-0.970
1861-02-01	-0.885
1862-01-01	-0.801
1862-02-01	-0.803
1862-04-01	-0.552
1862-11-01	-0.806
1862-12-01	-0.590
1863-09-01	-0.909
1863-12-01	-0.553
1864-12-01	-0.581
1867-02-01	-0.713
1872-11-01	-0.665
1874-03-01	-0.568
1876-05-01	-0.823
1876-12-01	-0.583
1878-02-01	0.611
1878-03-01	1.037
1878-04-01	0.616
1879-12-01	-0.843
1880-02-01	-0.751
1883-11-01	-0.570
1884-03-01	-0.694

Date	Monthly Anomaly
1884-04-01	-0.588
1884-05-01	-0.562
1884-11-01	-0.590
1885-03-01	-0.561
1886-11-01	-0.725
1887-01-01	-0.767
1887-02-01	-0.601
1890-11-01	-0.869
1891-02-01	-0.767
1892-11-01	-0.670
1893-01-01	-0.824
1893-02-01	-0.776
1894-06-01	-0.627
1898-03-01	-0.758

dtype: float64

In [ ]:
```
plt.figure(figsize=(16, 6))
plt.style.use("ggplot")
plt.plot(df_1900.index, df_1900['Monthly Anomaly'], label='Monthly Anomaly')
plt.scatter(anomalies.index, anomalies, color='blue', label='Outliers (Decomposi
plt.title('Monthly Anomaly of Global Temperatures with Outliers')
plt.xticks(rotation=60)
plt.legend()
plt.show()
```



Facebook's Prophet Model

Facebook Prophet is an open-source library developed by Facebook's internal data science team to solve time series-based forecasting problems. It is a quick and easy way to create forecasts. It can make predictions about future events based on historical data.

In [ ]:
```
!pip install prophet
```

```
Requirement already satisfied: prophet in /usr/local/lib/python3.11/dist-packages
(1.1.6)
Requirement already satisfied: cmdstanpy>=1.0.4 in /usr/local/lib/python3.11/dist
-packages (from prophet) (1.2.5)
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.11/dist-pa
ckages (from prophet) (2.0.2)
Requirement already satisfied: matplotlib>=2.0.0 in /usr/local/lib/python3.11/dis
t-packages (from prophet) (3.10.0)
Requirement already satisfied: pandas>=1.0.4 in /usr/local/lib/python3.11/dist-pa
ckages (from prophet) (2.2.2)
Requirement already satisfied: holidays<1,>=0.25 in /usr/local/lib/python3.11/dis
t-packages (from prophet) (0.73)
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.11/dist-pac
kages (from prophet) (4.67.1)
Requirement already satisfied: importlib-resources in /usr/local/lib/python3.11/d
ist-packages (from prophet) (6.5.2)
Requirement already satisfied: stanio<2.0.0,>=0.4.0 in /usr/local/lib/python3.11/
dist-packages (from cmdstanpy>=1.0.4->prophet) (0.5.1)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.11/dist-
packages (from holidays<1,>=0.25->prophet) (2.9.0.post0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist
-packages (from matplotlib>=2.0.0->prophet) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-pac
kages (from matplotlib>=2.0.0->prophet) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dis
t-packages (from matplotlib>=2.0.0->prophet) (4.58.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dis
t-packages (from matplotlib>=2.0.0->prophet) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-
packages (from matplotlib>=2.0.0->prophet) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packag
es (from matplotlib>=2.0.0->prophet) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist
-packages (from matplotlib>=2.0.0->prophet) (3.2.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-pac
kages (from pandas>=1.0.4->prophet) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-p
ackages (from pandas>=1.0.4->prophet) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-package
s (from python-dateutil->holidays<1,>=0.25->prophet) (1.17.0)
```

In [ ]:
```
from prophet import Prophet
```

We prepare our dataset to apply Prophet:

In [ ]:
```
df = pd.DataFrame()
df['ds'] = df_1900.index
df['y'] = df_1900['Monthly Anomaly'].values
```

In [ ]:
```
def fit_predict_model(dataframe, interval_width = 0.99, changepoint_range = 0.8)
    m = Prophet(daily_seasonality = False, yearly_seasonality = False, weekly_se
                seasonality_mode = 'additive',
```

```
                interval_width = interval_width,
                changepoint_range = changepoint_range)
    m = m.fit(dataframe)
    forecast = m.predict(dataframe)
    forecast['fact'] = dataframe['y'].reset_index(drop = True)
    return forecast

pred = fit_predict_model(df)
```

```
DEBUG:cmdstanpy:input tempfile: /tmp/tmp8e0e74q2/7649meoc.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmp8e0e74q2/_8446wlz.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/s
tan_model/prophet_model.bin', 'random', 'seed=3187', 'data', 'file=/tmp/tmp8e0e74
q2/7649meoc.json', 'init=/tmp/tmp8e0e74q2/_8446wlz.json', 'output', 'file=/tmp/tm
p8e0e74q2/prophet_modeligrcmvdz/prophet_model-20250527234426.csv', 'method=optimi
ze', 'algorithm=lbfgs', 'iter=10000']
23:44:26 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
23:44:26 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```

After applying Prophet, we will obtain the anomalies that are outside the lower and upper limits:

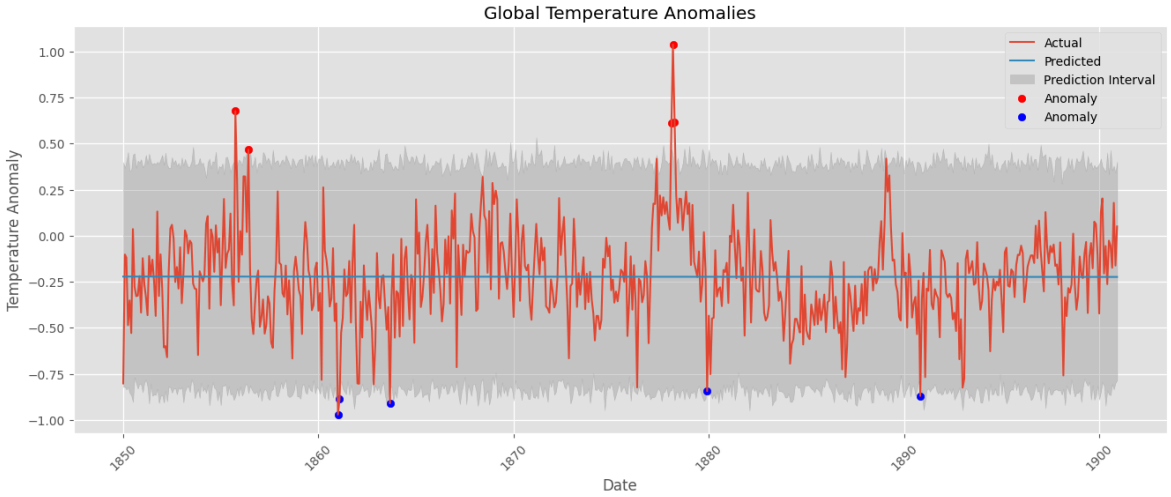
In [ ]:
```
def detect_anomalies(forecast):
    forecasted = forecast[['ds','trend', 'yhat', 'yhat_lower', 'yhat_upper', 'fa

    forecasted['anomaly'] = 0
    forecasted.loc[forecasted['fact'] > forecasted['yhat_upper'], 'anomaly'] = 1
    forecasted.loc[forecasted['fact'] < forecasted['yhat_lower'], 'anomaly'] = -

    #anomaly importances
    forecasted['importance'] = 0
    forecasted.loc[forecasted['anomaly'] ==1, 'importance'] = (forecasted['fact'
    forecasted.loc[forecasted['anomaly'] ==-1, 'importance'] = (forecasted['yhat

    return forecasted

pred = detect_anomalies(pred)
```

```
<ipython-input-42-18df92a84b25>:10: FutureWarning: Setting an item of incompatibl
e dtype is deprecated and will raise an error in a future version of pandas. Valu
e '[0.46397406 0.04028923 0.23315508 0.58450224 0.45702309]' has dtype incompatib
le with int64, please explicitly cast to a compatible dtype first.
  forecasted.loc[forecasted['anomaly'] ==1, 'importance'] = (forecasted['fact'] -
forecasted['yhat_upper'])/forecast['fact']
```

In [ ]:
```
pred.set_index('ds', inplace=True)
```

In [ ]:
```
# prompt: plot pred with lower and upper values

plt.figure(figsize=(16, 6))
plt.plot(pred.index, pred['fact'], label='Actual')
plt.plot(pred.index, pred['yhat'], label='Predicted')
plt.fill_between(pred.index, pred['yhat_lower'], pred['yhat_upper'], color='gray
plt.scatter(pred[pred['anomaly'] == 1].index, pred[pred['anomaly'] == 1]['fact']
plt.scatter(pred[pred['anomaly'] == -1].index, pred[pred['anomaly'] == -1]['fact
```

```
plt.title('Global Temperature Anomalies')
plt.xlabel('Date')
plt.ylabel('Temperature Anomaly')
plt.xticks(rotation=45)
plt.legend()
plt.show()
```



Thank you! See you in the next Chapter!

References:

https://medium.com/@alex.eslava96/outlier-detection-techniques-for-time-series-9868db2875c2

https://www.kaggle.com/code/kmkarakaya/anomaly-detection-in-time-series

https://neptune.ai/blog/anomaly-detection-in-time-series

https://medium.com/@reza.rajabi/outlier-and-anomaly-detection-using-facebook-prophet-in-python-3a83d58b1bdf

https://www.kaggle.com/code/pythonafroz/anomaly-detection-using-isolation-forest

https://www.kaggle.com/code/mehanat96/anomaly-detection-using-stl