In [2]:
```
#importing required libraries

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import linear_model
import statsmodels.api as sm
import seaborn as sns
from sklearn.metrics import r2_score
import math

import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
src_dest_df = pd.read_parquet('src_dest_df.parquet')
src_dest_df.timestamp = pd.to_datetime(src_dest_df.timestamp)
src_dest_df.shape
src_dest_df.dtypes
src_dest_df.head(10)
```

Out[3]:

	cloud_geo_iso1	cloud_geo_iso2	latency_ms
66	AWS.eu-west-1	AWS.ap-northeast-2	124.20
67	AWS.eu-west-1	AWS.ap-southeast-2	131.80
68	AWS.eu-west-1	AWS.ap-northeast-2	123.75
69	AWS.eu-west-1	AWS.eu-west-2	6.35
70	AWS.eu-west-1	AWS.us-west-1	72.35
71	AWS.eu-west-1	AWS.eu-north-1	23.75
72	AWS.eu-west-1	AWS.eu-central-1	14.55
73	AWS.eu-west-1	AWS.us-east-1	36.40
74	AWS.eu-west-1	AWS.ca-central-1	39.55
75	AWS.eu-west-1	AWS.eu-west-3	10.75

In [4]:
```
# Feature engineering
data = src_dest_df
data.sort_values(by='timestamp')

data = src_dest_df[['timestamp','latency_ms']]
data.head(20)
```

```
                      timestamp      cloud_geo_iso1      cloud_geo_iso2  \
0      2020-03-31 08:00:00+00:00       AWS.ap-east-1       AWS.us-west-2
1      2020-03-30 17:00:00+00:00       AWS.ap-east-1       AWS.us-west-2
2      2020-03-31 05:00:00+00:00       AWS.ap-east-1       AWS.us-west-2
3      2020-03-31 09:00:00+00:00       AWS.ap-east-1       AWS.us-west-2
4      2020-03-31 22:00:00+00:00       AWS.ap-east-1       AWS.us-west-2
...                          ...                 ...                 ...
495076 2020-03-21 10:00:00+00:00  AWS.ap-southeast-2       AWS.eu-west-2
495077 2020-03-26 17:00:00+00:00  AWS.ap-southeast-2      AWS.eu-north-1
495078 2020-03-26 02:00:00+00:00  AWS.ap-southeast-2  AWS.ap-northeast-1
```

```
495079 2020-03-31 04:00:00+00:00  AWS.ap-southeast-2       AWS.us-west-2
495080 2020-03-31 02:00:00+00:00  AWS.ap-southeast-2       AWS.sa-east-1

        latency_ms  packet_loss_percent
0           156.55                  0.0
1           154.60                  0.0
2           153.85                  0.0
3           155.60                  0.0
4           155.00                  0.0
...            ...                  ...
495076      139.25                  0.0
495077      149.95                  0.0
495078       54.75                  0.0
495079       71.50                  0.0
495080      157.15                  0.0

[495081 rows x 5 columns]
```

Out[4]:

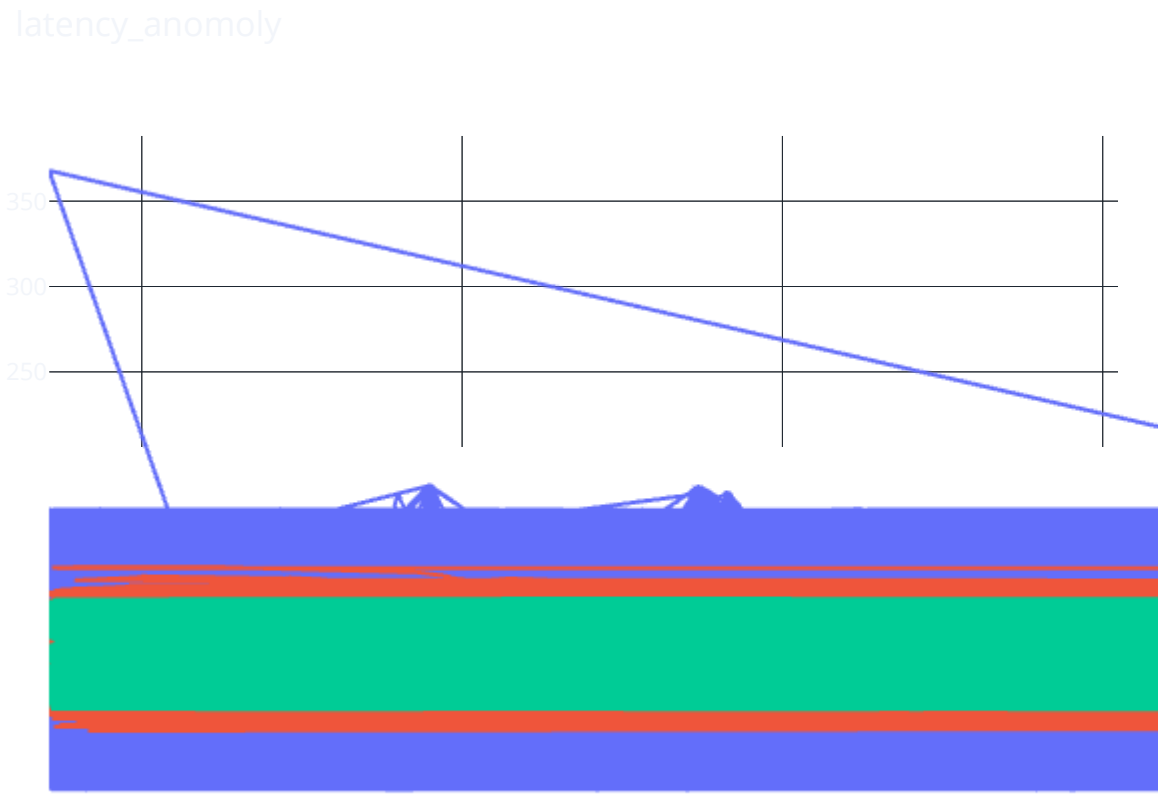
	timestamp	latency_ms
0	2020-03-31 08:00:00+00:00	156.55
1	2020-03-30 17:00:00+00:00	154.60
2	2020-03-31 05:00:00+00:00	153.85
3	2020-03-31 09:00:00+00:00	155.60
4	2020-03-31 22:00:00+00:00	155.00
5	2020-03-30 16:00:00+00:00	115.90
6	2020-03-31 15:00:00+00:00	155.40
7	2020-03-31 03:00:00+00:00	155.10
8	2020-03-31 00:00:00+00:00	154.05
9	2020-03-30 23:00:00+00:00	115.80
10	2020-03-31 03:00:00+00:00	115.85
11	2020-03-31 07:00:00+00:00	115.45
12	2020-03-31 18:00:00+00:00	115.00
13	2020-03-30 20:00:00+00:00	114.85
14	2020-03-31 23:00:00+00:00	155.45
15	2020-03-31 10:00:00+00:00	115.25
16	2020-03-31 17:00:00+00:00	115.35
17	2020-03-31 14:00:00+00:00	114.60
18	2020-03-31 09:00:00+00:00	116.45
19	2020-03-31 22:00:00+00:00	113.50

In [5]:

```
# creating moving-averages
data['MA48'] = data['latency_ms'].rolling(48).mean()
data['MA336'] = data['latency_ms'].rolling(336).mean()

# plotting the graph
```

```
import plotly.express as px
fig = px.line(data, x="timestamp", y=['latency_ms', 'MA48', 'MA336'], title='latency_an
fig.show()
```



In [6]:

```
# drop moving-average columns
data.drop(['MA48', 'MA336'], axis=1, inplace=True)
# set timestamp to index
data.set_index('timestamp', drop=True, inplace=True)
# resample timeseries to hourly
data = data.resample('H').sum()
# create features from date
data['day'] = [i.day for i in data.index]
data['day_name'] = [i.day_name() for i in data.index]
data['day_of_year'] = [i.dayofyear for i in data.index]
data['week_of_year'] = [i.weekofyear for i in data.index]
data['hour'] = [i.hour for i in data.index]
data['is_weekday'] = [i.isoweekday() for i in data.index]
data.head()
```

Out[6]:

timestamp	latency_ms	day	day_name	day_of_year	week_of_year	hour	is_weekday

timestamp	latency_ms	day	day_name	day_of_year	week_of_year	hour	is_weekday
2019-12-31 23:00:00+00:00	15320.55	31	Tuesday	365	1	23	2
2020-01-01 00:00:00+00:00	18155.50	1	Wednesday	1	1	0	3
2020-01-01 01:00:00+00:00	18187.60	1	Wednesday	1	1	1	3
2020-01-01 02:00:00+00:00	18159.30	1	Wednesday	1	1	2	3
2020-01-01 03:00:00+00:00	18161.85	1	Wednesday	1	1	3	3

In [14]:

```
from pycaret.anomaly import *
s = setup(data, session_id = 123)
```

	Description	Value
0	session_id	123
1	Original Data	(2185, 7)
2	Missing Values	False
3	Numeric Features	4
4	Categorical Features	3
5	Ordinal Features	False
6	High Cardinality Features	False
7	High Cardinality Method	None
8	Transformed Data	(2185, 32)
9	CPU Jobs	-1
10	Use GPU	False
11	Log Experiment	False
12	Experiment Name	anomaly-default-name
13	USI	bf21
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

	Description	Value
20	Unknown Categoricals Handling	least_frequent
21	Normalize	False
22	Normalize Method	None
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	Remove Perfect Collinearity	False
37	Clustering	False
38	Clustering Iteration	None
39	Polynomial Features	False
40	Polynomial Degree	None
41	Trignometry Features	False
42	Polynomial Threshold	None
43	Group Features	False
44	Feature Selection	False
45	Feature Selection Method	classic
46	Features Selection Threshold	None
47	Feature Interaction	False
48	Feature Ratio	False
49	Interaction Threshold	None

In [15]:
```
# checking of available models
models()
```

Out[15]:

	Name	Reference

ID	Name	Reference
ID		
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 [16]:
```
# training the model
iforest = create_model('iforest', fraction = 0.1)
iforest_results = assign_model(iforest)
iforest_results.head()
```

Out[16]:

timestamp	latency_ms	day	day_name	day_of_year	week_of_year	hour	is_weekday	Anomaly	A
2019-12-31 23:00:00+00:00	15320.55	31	Tuesday	365	1	23	2	0	
2020-01-01 00:00:00+00:00	18155.50	1	Wednesday	1	1	0	3	1	
2020-01-01 01:00:00+00:00	18187.60	1	Wednesday	1	1	1	3	0	
2020-01-01 02:00:00+00:00	18159.30	1	Wednesday	1	1	2	3	0	
2020-01-01 03:00:00+00:00	18161.85	1	Wednesday	1	1	3	3	0	

In [17]:
```
# checking for anomalies
iforest_results[iforest_results['Anomaly'] == 1].head()
```

Out[17]:

timestamp	latency_ms	day	day_name	day_of_year	week_of_year	hour	is_weekday	Anomaly	A

timestamp	latency_ms	day	day_name	day_of_year	week_of_year	hour	is_weekday	Anomaly
2020-01-01 00:00:00+00:00	18155.5	1	Wednesday	1	1	0	3	1
2020-01-01 20:00:00+00:00	18156.5	1	Wednesday	1	1	20	3	1
2020-01-01 22:00:00+00:00	18143.7	1	Wednesday	1	1	22	3	1
2020-01-01 23:00:00+00:00	18181.2	1	Wednesday	1	1	23	3	1
2020-01-02 00:00:00+00:00	18191.9	2	Thursday	2	1	0	4	1

In [19]:

```
# Plotting the anomolies on graph
import plotly.graph_objects as go
# plot latency_ms on y-axis and date on x-axis
fig = px.line(iforest_results, x=iforest_results.index, y="latency_ms", title='AWS- Lat
# create list of outlier_dates
outlier_dates = iforest_results[iforest_results['Anomaly'] == 1].index
# obtain latency_ms value of anomalies to plot
y_values = [iforest_results.loc[i]['latency_ms'] for i in outlier_dates]
fig.add_trace(go.Scatter(x=outlier_dates, y=y_values, mode = 'markers',
                name = 'Anomaly',
                marker=dict(color='red',size=10)))

fig.show()
```