In [3]:
```python
import matplotlib.pyplot as plt
from matplotlib import pyplot
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from statsmodels.tsa.vector_ar.var_model import VAR
from statsmodels.tsa.stattools import grangercausalitytests
```

In [4]:
```python
#Assume the time series frequency is daily
price_data=pd.read_csv(r'D:\2023\XJCP Trader Quant\xjcp.csv')
df=pd.DataFrame(price_data)
print(df)
```

```
             1        2        3        4        5       6        7       8  \
0     74.23525  124.000   23.000  149.187    7.459   7.872  257.347  77.510
1     74.17525   33.105  280.280  133.749    0.709  31.305   87.454  51.044
2     74.18325  375.086  323.644  170.037    3.999  25.476  168.794  72.876
3     74.17625   48.775   25.853   93.927   39.872  14.148   72.699  65.654
4     74.17125   48.774  301.886   90.637   30.003  20.829  201.224  24.241
...        ...      ...      ...      ...      ...     ...      ...     ...
8683  65.77675  497.010  578.836   67.163  152.204   7.749   99.030  47.483
8684  65.76575  384.242  257.090   90.231    9.971  61.501   30.629   8.503
8685  65.80125  540.689  237.266   51.585    6.927  15.746   36.759   6.136
8686  65.83125  309.421  182.045   60.519   18.490   4.964   27.717  21.620
8687  65.79675  288.086  269.588    8.500    4.464  13.414   15.549  20.285

            9  10  ...     47      48     49      50      51      52     53  \
0      86.753   0  ...  15582  6.8649     44     533  201481   85873  42474
1     130.774   2  ...  13398  6.8589  45012   63355  230732   84898  45959
2     270.396   2  ...  12777  6.8595  27710   55091  247450   98765  43705
3     352.091   0  ...  14498  6.8595  32364   39759  237211   95343  35553
4      96.640   0  ...  14704  6.8583  17667   52192  236698  102647  44195
...       ...  ..  ...    ...     ...    ...     ...     ...     ...    ...
8683   84.494   2  ...   5742  6.2493  55771  203724  183714   10791  10770
8684   33.369   2  ...   6359  6.2415  63770   59750  104302   32718  19958
8685    9.268   1  ...   5693  6.2427  68479   26975   55850   32381   6030
8686   16.830   0  ...   6330  6.2418  49254   49670   52459   33657   3869
8687    7.768   2  ...   6798  6.2445  21444   30830   69699   26630   4612

          54     55     56
0     237989  69445  33461
1     155027  45111  44533
2     206995  74398  42840
3     217658  75365  53509
4     196868  74185  19152
...      ...    ...    ...
8683   48708  10358   7681
8684  155624  56757  13380
8685  123323  56775   6422
8686  124079  67859   4275
8687  112075  66791  13109

[8688 rows x 56 columns]
```

In [5]:
```python
# Replace zero in df with previous number
df = df.replace(0, method='ffill')
df
```

Out[5]:

	1	2	3	4	5	6	7	8	9	10	...	47
0	74.23525	124.000	23.000	149.187	7.459	7.872	257.347	77.510	86.753	0	...	1558
1	74.17525	33.105	280.280	133.749	0.709	31.305	87.454	51.044	130.774	2	...	1339
2	74.18325	375.086	323.644	170.037	3.999	25.476	168.794	72.876	270.396	2	...	1277
3	74.17625	48.775	25.853	93.927	39.872	14.148	72.699	65.654	352.091	2	...	1449
4	74.17125	48.774	301.886	90.637	30.003	20.829	201.224	24.241	96.640	2	...	1470
...	...	...	...	...	...	...	...	...	...	...	...	.
8683	65.77675	497.010	578.836	67.163	152.204	7.749	99.030	47.483	84.494	2	...	574
8684	65.76575	384.242	257.090	90.231	9.971	61.501	30.629	8.503	33.369	2	...	635
8685	65.80125	540.689	237.266	51.585	6.927	15.746	36.759	6.136	9.268	1	...	569
8686	65.83125	309.421	182.045	60.519	18.490	4.964	27.717	21.620	16.830	1	...	633
8687	65.79675	288.086	269.588	8.500	4.464	13.414	15.549	20.285	7.768	2	...	679

8688 rows × 56 columns

In [6]:

```
#Replace negatgive numbers with NaN
df[df < 0] = pd.np.nan
```

```
C:\Users\sigma\AppData\Local\Temp\ipykernel_3352\546709128.py:2: FutureWarning: Th
e pandas.np module is deprecated and will be removed from pandas in a future versi
on. Import numpy directly instead.
  df[df < 0] = pd.np.nan
```

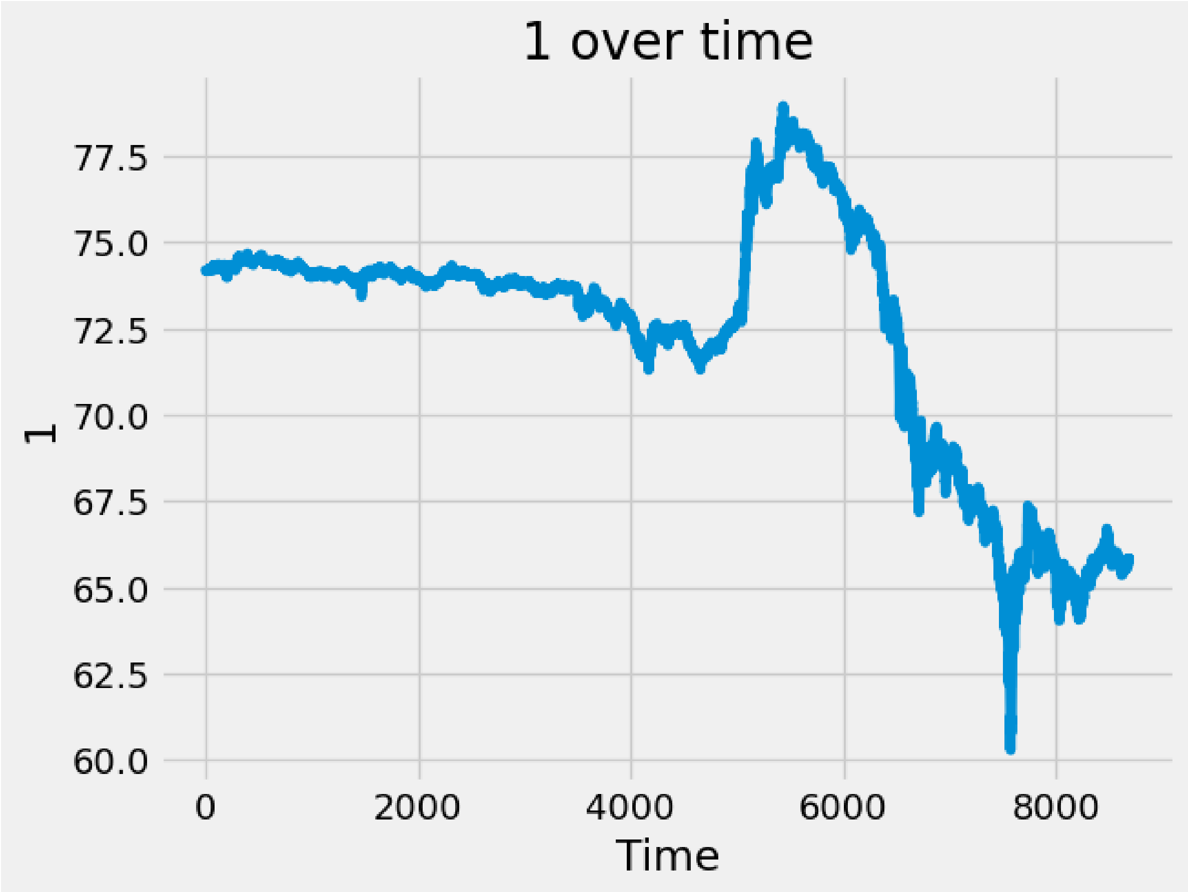
In [7]:

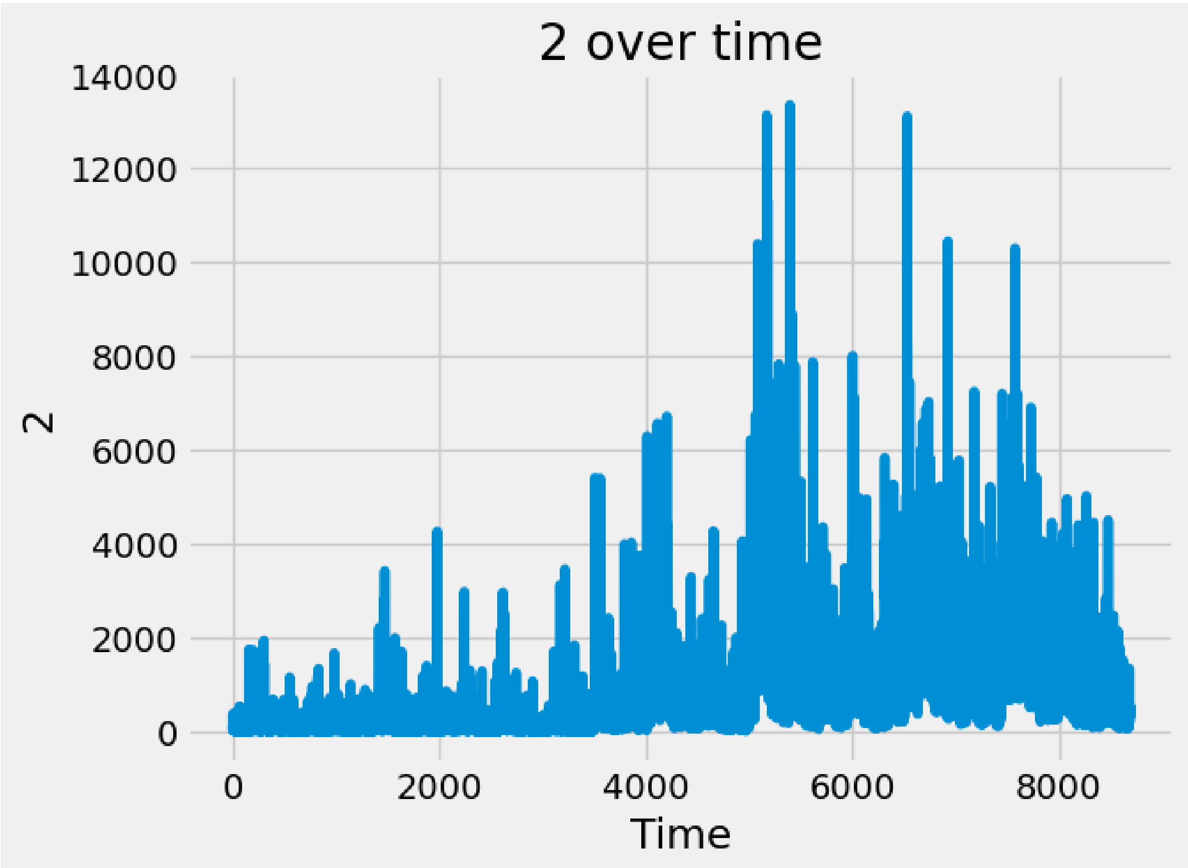
```
#Then forward fill Nan values
df = df.fillna(method='ffill')
```

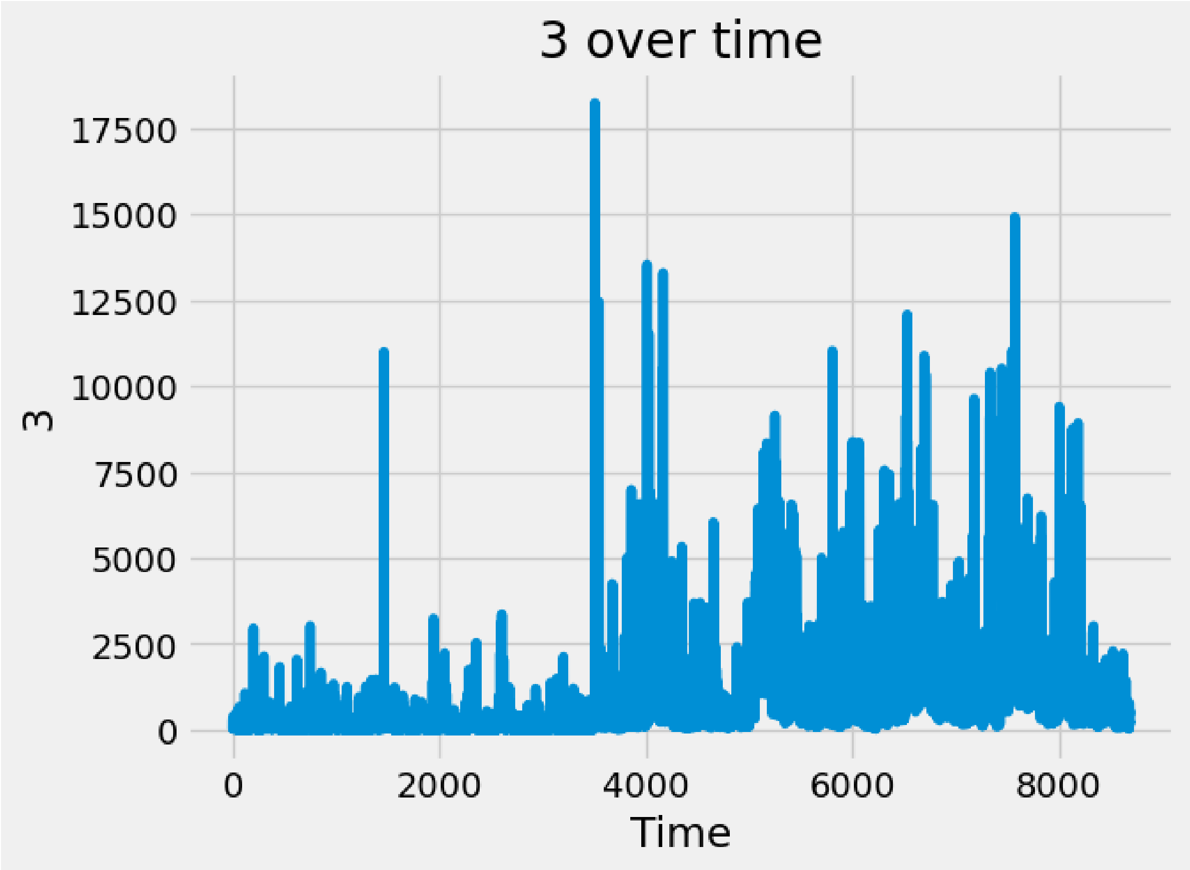
In [48]:

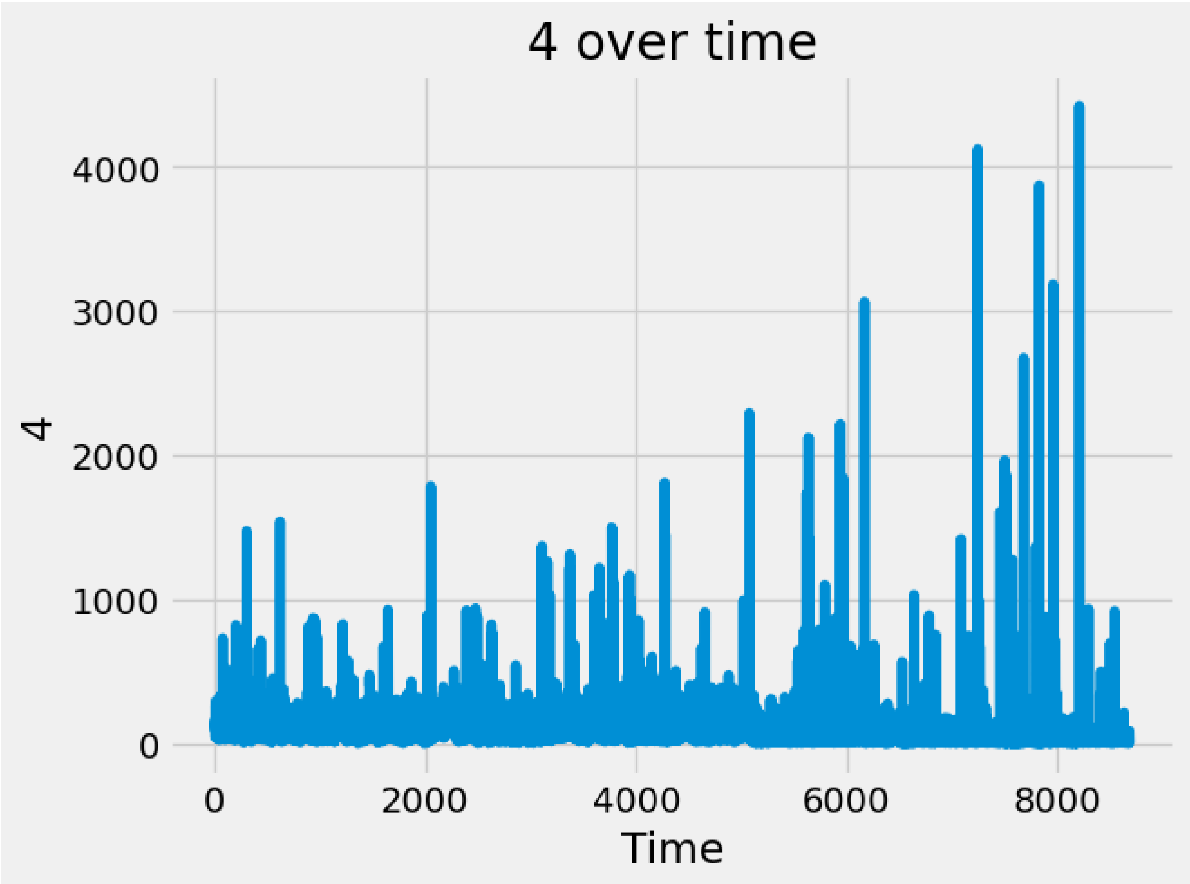
```
#Look up each column in the dataframe python
for column in df.columns:
    print(f"{column}: {df[column].values}")

# Plot time series by price for each asset
for col in df.columns[0:]:
    plt.plot(df[col])
    plt.xlabel('Time')
    plt.ylabel(col.capitalize())
    plt.title(f'{col.capitalize()} over time')
    plt.show()
```

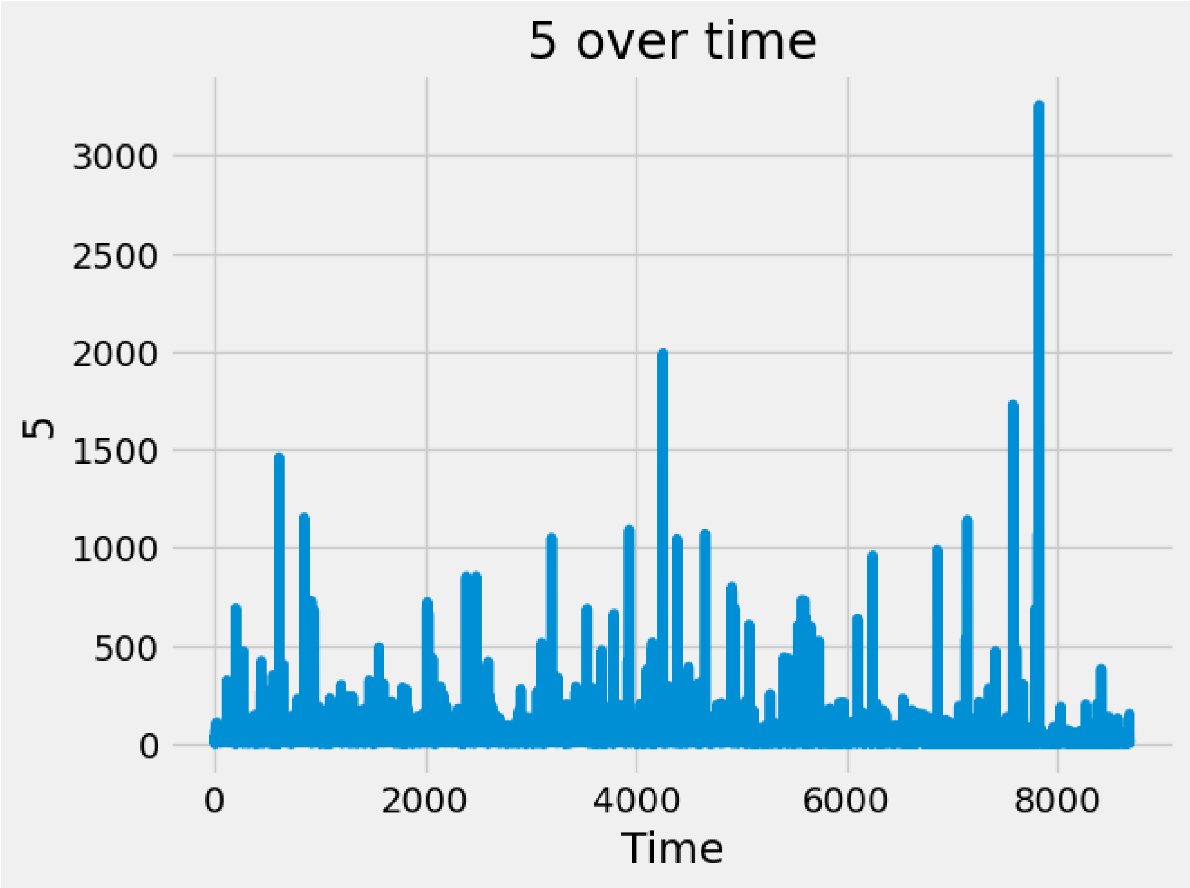
```
1: [74.23525 74.17525 74.18325 ... 65.80125 65.83125 65.79675]
2: [124.     33.105 375.086 ... 540.689 309.421 288.086]
3: [ 23.    280.28  323.644 ... 237.266 182.045 269.588]
4: [149.187 133.749 170.037 ...  51.585  60.519   8.5  ]
5: [ 7.459  0.709  3.999 ...  6.927 18.49   4.464]
6: [ 7.872 31.305 25.476 ... 15.746  4.964 13.414]
7: [257.347  87.454 168.794 ...  36.759  27.717  15.549]
8: [77.51  51.044 72.876 ...  6.136 21.62  20.285]
9: [ 86.753 130.774 270.396 ...   9.268  16.83    7.768]
10: [0 2 2 ... 1 1 2]
11: [3.94307 3.94129 3.94109 ... 3.65355 3.65519 3.65567]
12: [123.      1.441   1.905 ...  18.384  25.461  27.73 ]
13: [655.     51.152  11.772 ...  16.83   49.347  28.344]
14: [7.234 7.788 8.995 ... 2.632 5.738 6.701]
15: [8.504 1.76  4.74  ... 0.383 1.368 3.194]
16: [ 1.253  6.395 10.691 ...  1.792  2.454  3.823]
17: [ 4.271 16.373  4.66  ...  6.189  7.371 12.44 ]
18: [ 0.609 12.321  0.544 ...  2.958  0.934  0.696]
19: [43.883 68.778  6.678 ...  1.736  8.554 11.297]
20: [0.43767664 0.31891556 0.2524555  ... 0.21786751 0.63032706 0.39775   ]
21: [  0.   151.17 361.97 ... 369.73  14.24 152.59]
22: [  0.   311.18  42.35 ... 164.63 151.25 372.53]
23: [223.12 179.52 304.59 ... 107.99 164.5  160.95]
24: [ 87.06 115.23  92.94 ...  44.72  44.63 150.32]
25: [ 25.37 118.88  47.01 ...  68.81   9.18  42.43]
26: [216.93 237.69 352.14 ... 174.   128.49  86.63]
27: [904.57  65.48 163.57 ...  28.98  55.26  23.5 ]
28: [ 96.88 805.45 145.25 ...  19.42  34.17   8.17]
29: [2.18838322 1.5945778  1.26227752 ... 1.08933754 3.1516353  1.98875   ]
30: [2.19075 2.18975 2.19075 ... 1.98825 1.98875 1.98875]
31: [3.200000e+01 5.624300e+04 1.142636e+05 ... 3.070110e+04 5.855530e+04
 4.562630e+04]
32: [4443234.   209798.8   22740.7 ...   21578.    76190.5   72928.2]
33: [3208895.8 2494573.2 3425565.5 ... 1111958.2 1398825.7 1361060.8]
34: [1533511.6 1822260.6 1470097.7 ...  591803.2  640501.4  749529.7]
35: [1405822.  1372616.1 1386080.4 ...  326009.9  350554.   414867.6]
36: [5316877.3 6118601.3 5282302.4 ... 1578809.3 1685472.1 1879834.2]
37: [1933595.4  924577.4 1800512.5 ...  800793.5  629612.9  490004.7]
38: [1014822.8 1051055.3  954954.1 ...  440471.   564092.4  498247.4]
39: [56.39 56.39 56.39 ... 46.43 46.45 46.45]
40: [  234 10014   195 ...  1821  1903  1377]
41: [ 544 6591 1069 ...  987 1314 2889]
42: [68666 64082 65066 ... 24894 25015 25447]
43: [29961 36002 38917 ... 15138 15852 16506]
44: [20993 21560 23052 ... 12368 12967 12885]
45: [56618 50693 54047 ... 14577 17502 17935]
46: [22913 20197 22575 ...  9847  9270  9858]
47: [15582 13398 12777 ...  5693  6330  6798]
48: [6.8649 6.8589 6.8595 ... 6.2427 6.2418 6.2445]
49: [   44 45012 27710 ... 68479 49254 21444]
50: [  533 63355 55091 ... 26975 49670 30830]
51: [201481 230732 247450 ...  55850  52459  69699]
52: [85873 84898 98765 ... 32381 33657 26630]
53: [42474 45959 43705 ...  6030  3869  4612]
54: [237989 155027 206995 ... 123323 124079 112075]
55: [69445 45111 74398 ... 56775 67859 66791]
56: [33461 44533 42840 ...  6422  4275 13109]
```

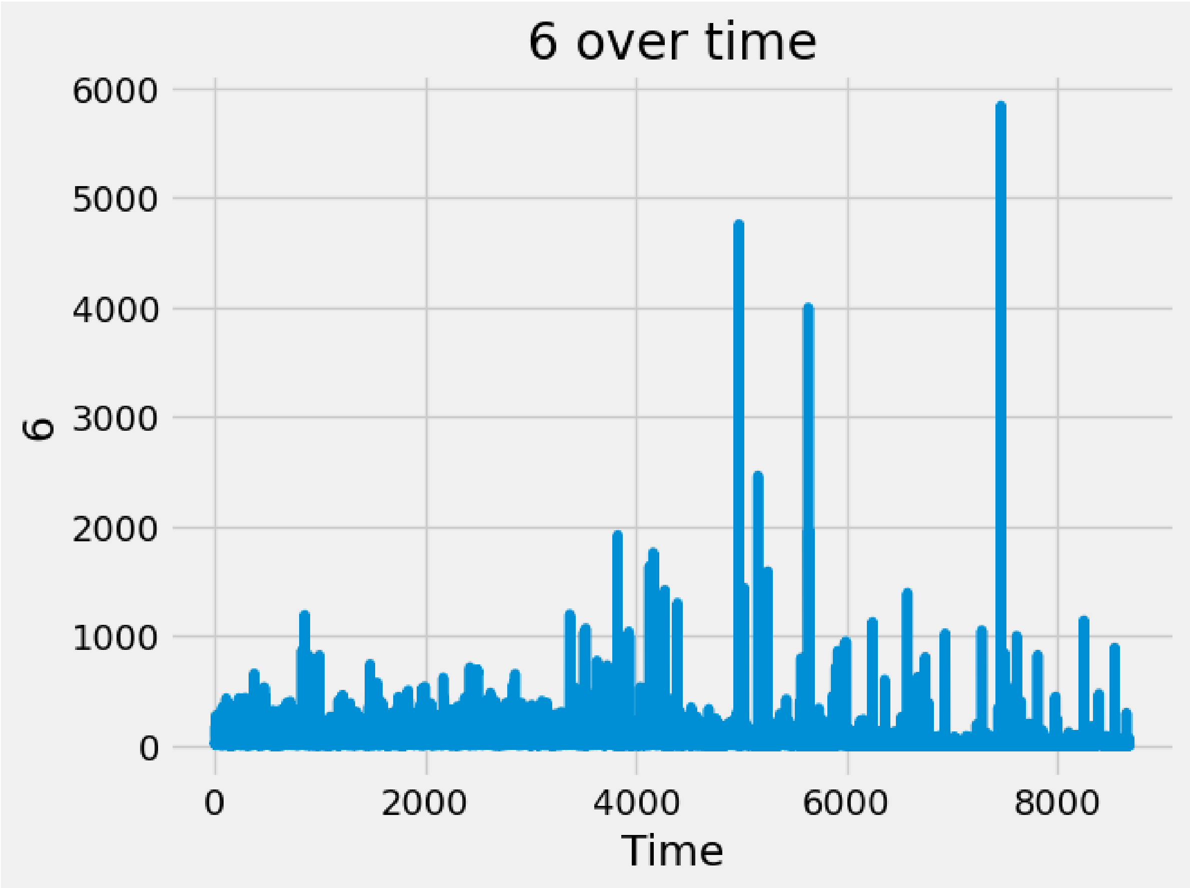

1 over time
77.5
75.0
72.5
70.0
67.5
65.0
62.5
60.0
1
0
2000
4000
6000
8000
Time

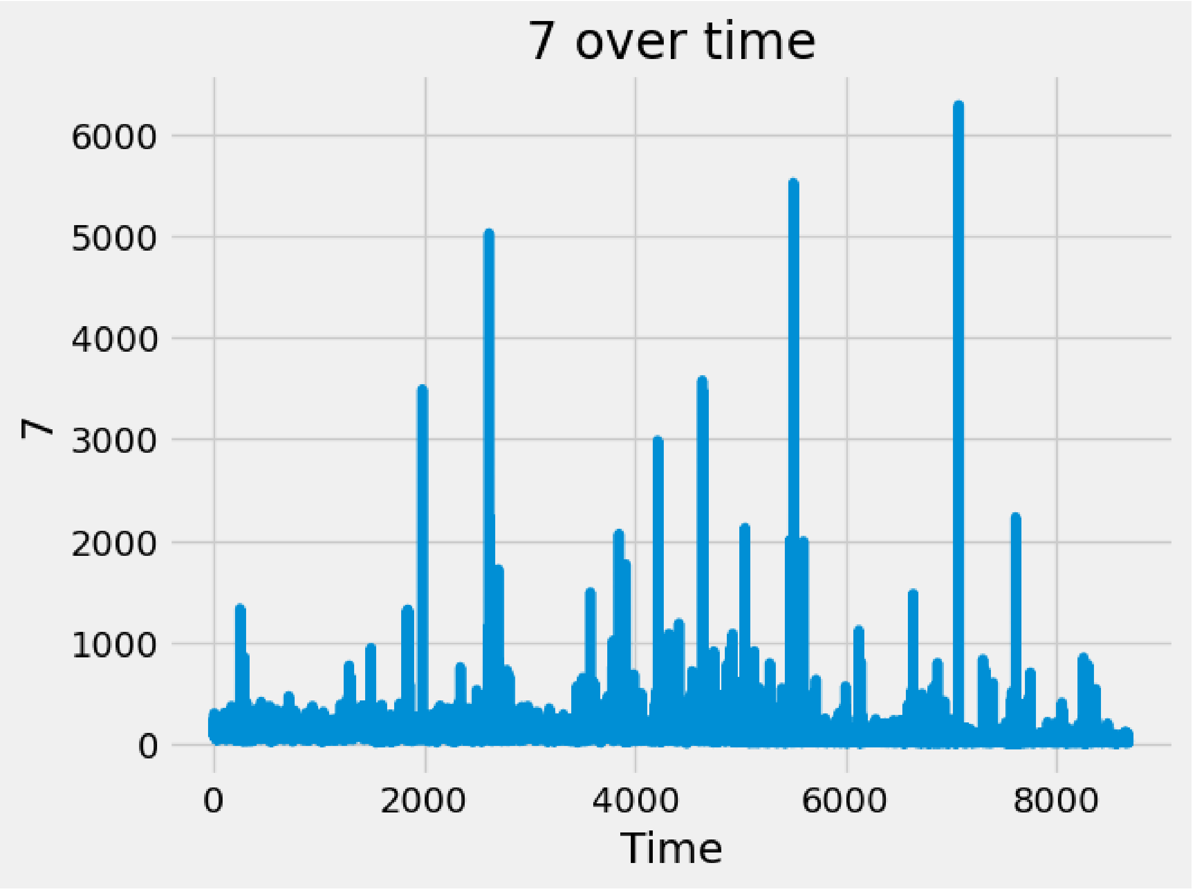

2 over time
14000
12000
10000
8000
6000
4000
2000
0
2
0
2000
4000
6000
8000
Time

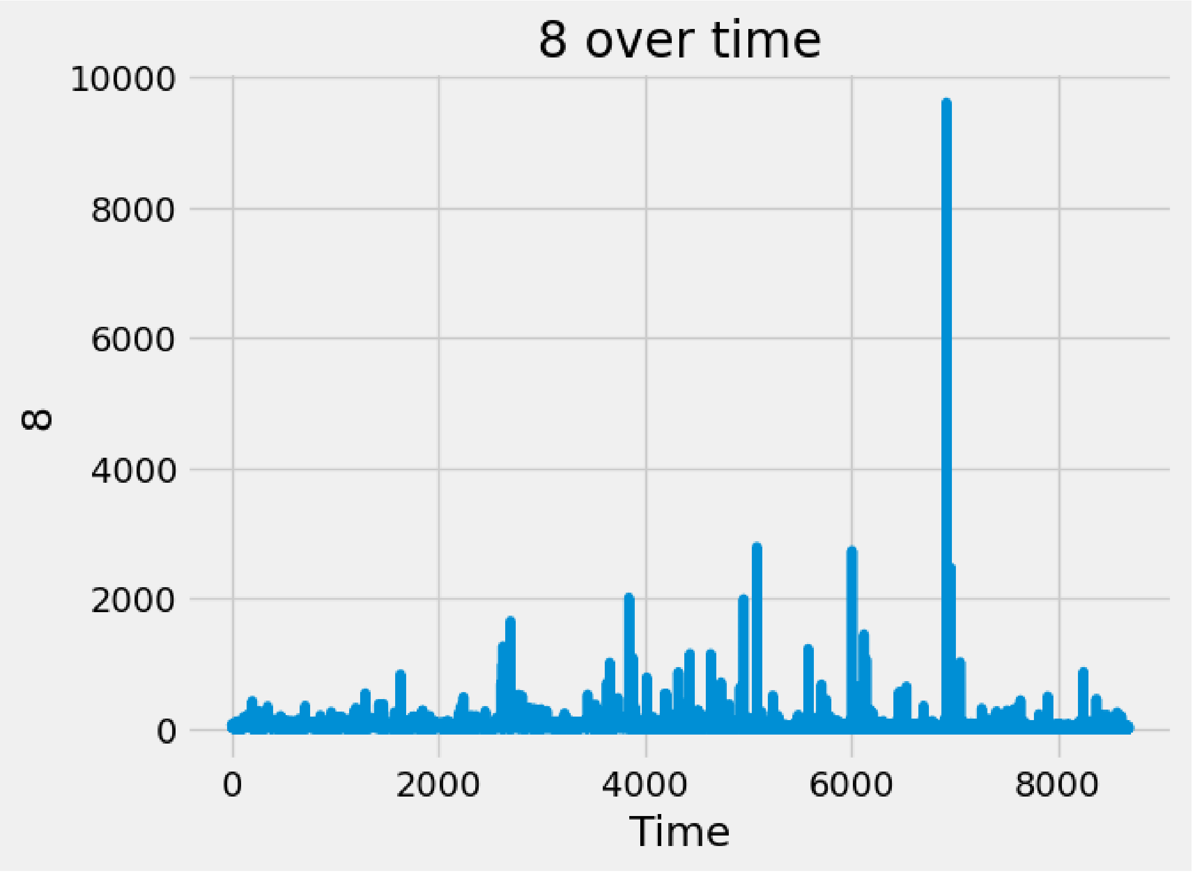

3 over time
17500
15000
12500
10000
7500
5000
2500
0
3
0
2000
4000
6000
8000
Time

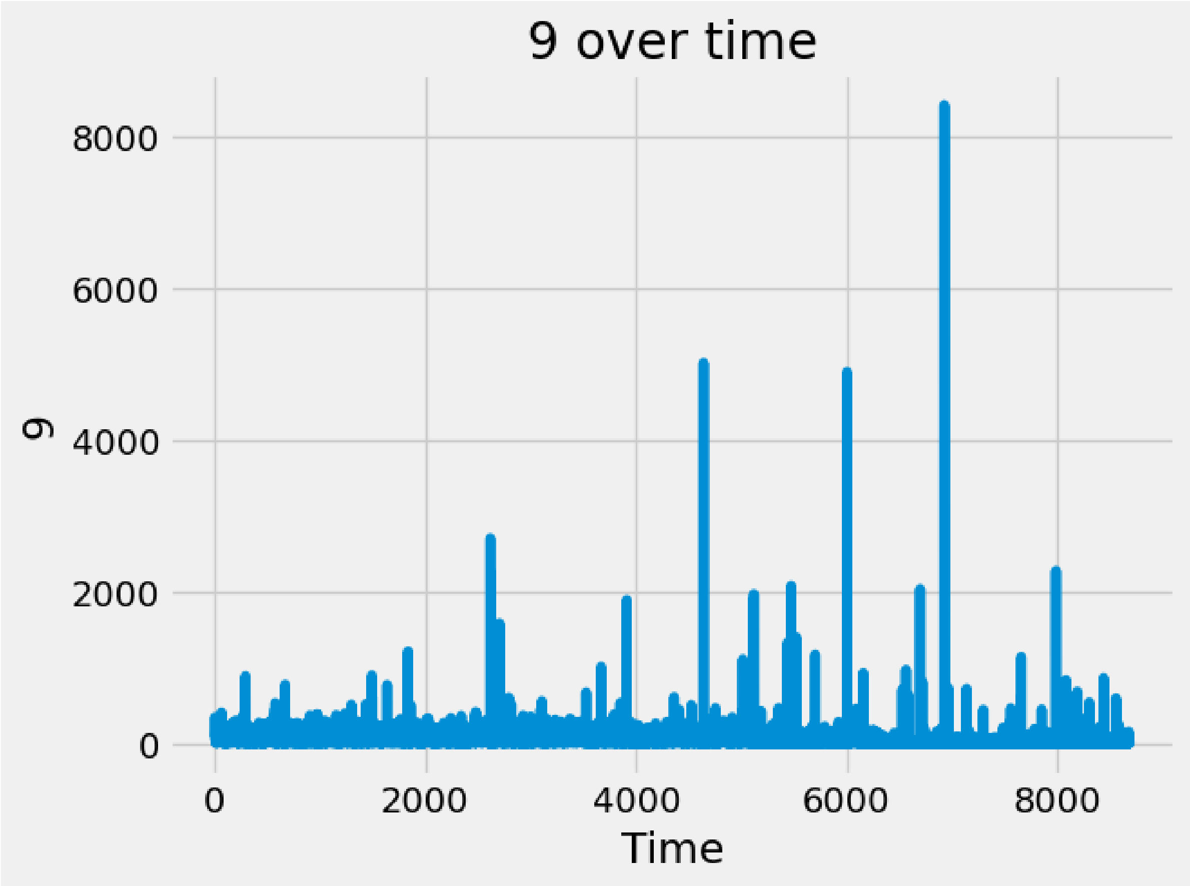

4 over time
4000
3000
2000
1000
0
4
0
2000
4000
6000
8000
Time

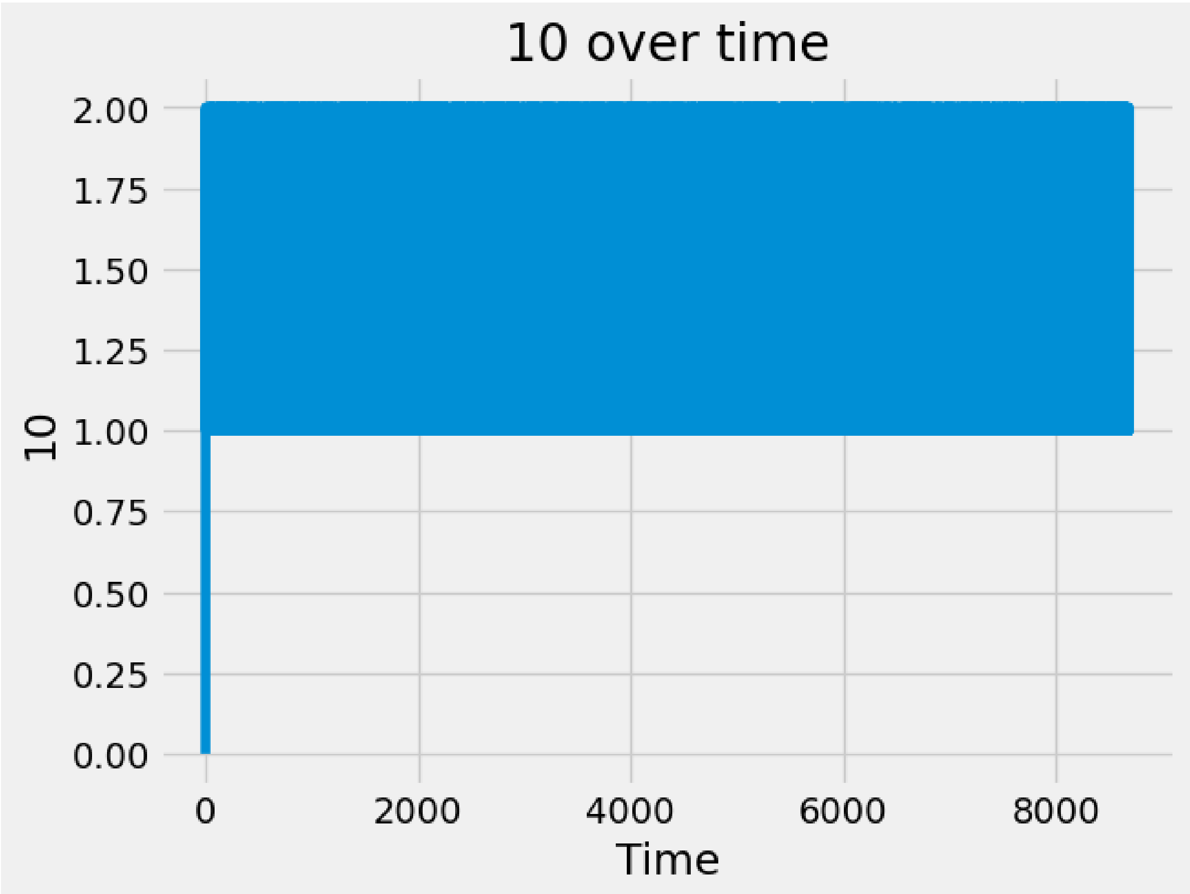

5 over time
5
Time
0
500
1000
1500
2000
2500
3000
0
2000
4000
6000
8000

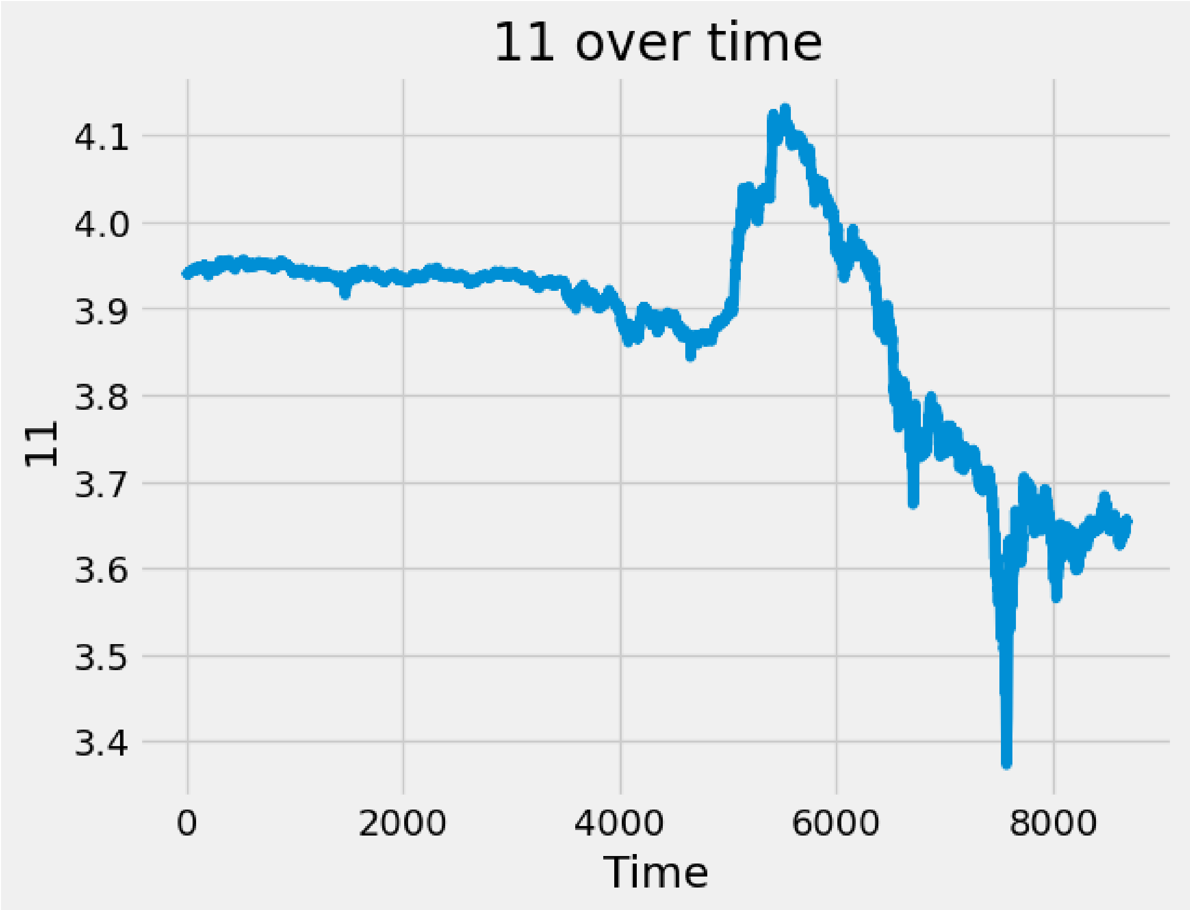

6 over time
6
Time
0
1000
2000
3000
4000
5000
6000
0
2000
4000
6000
8000

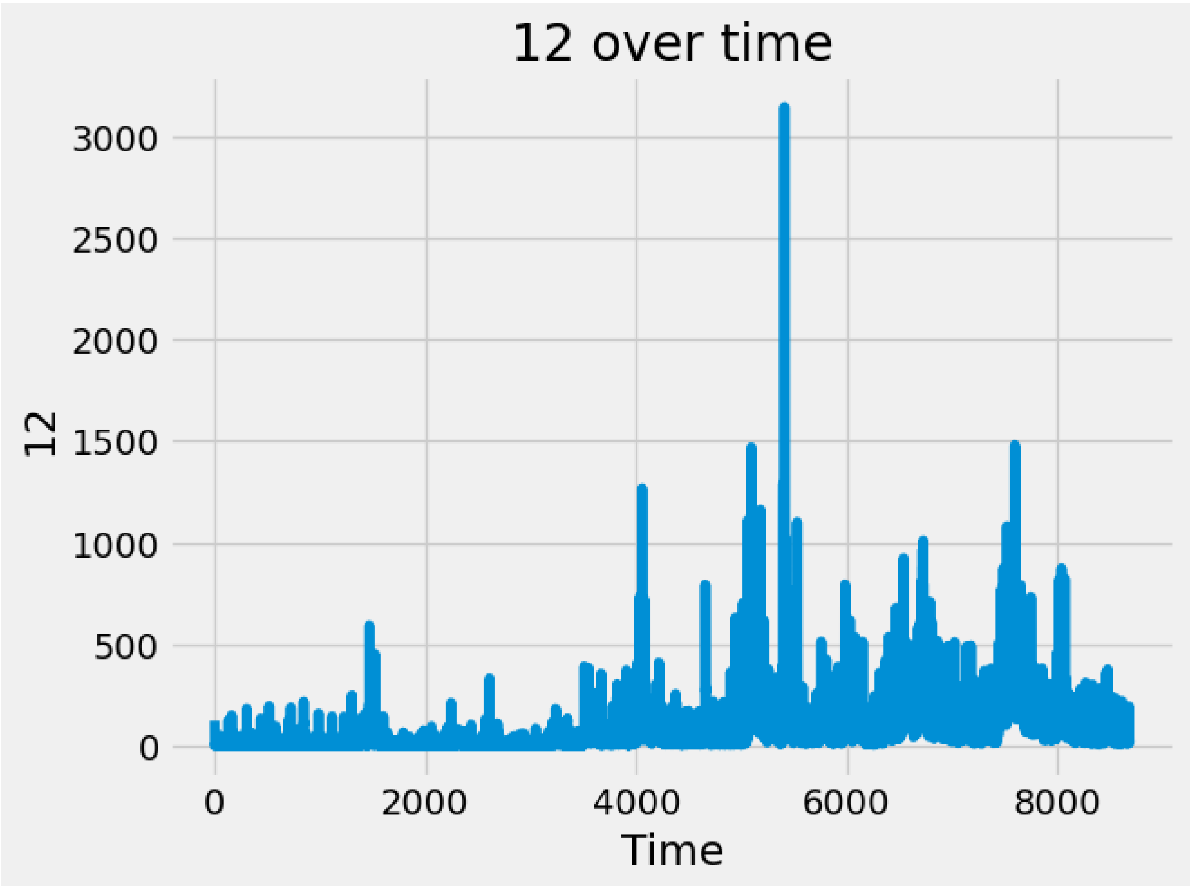

7 over time
6000
5000
4000
3000
2000
1000
0
7
0
2000
4000
6000
8000
Time

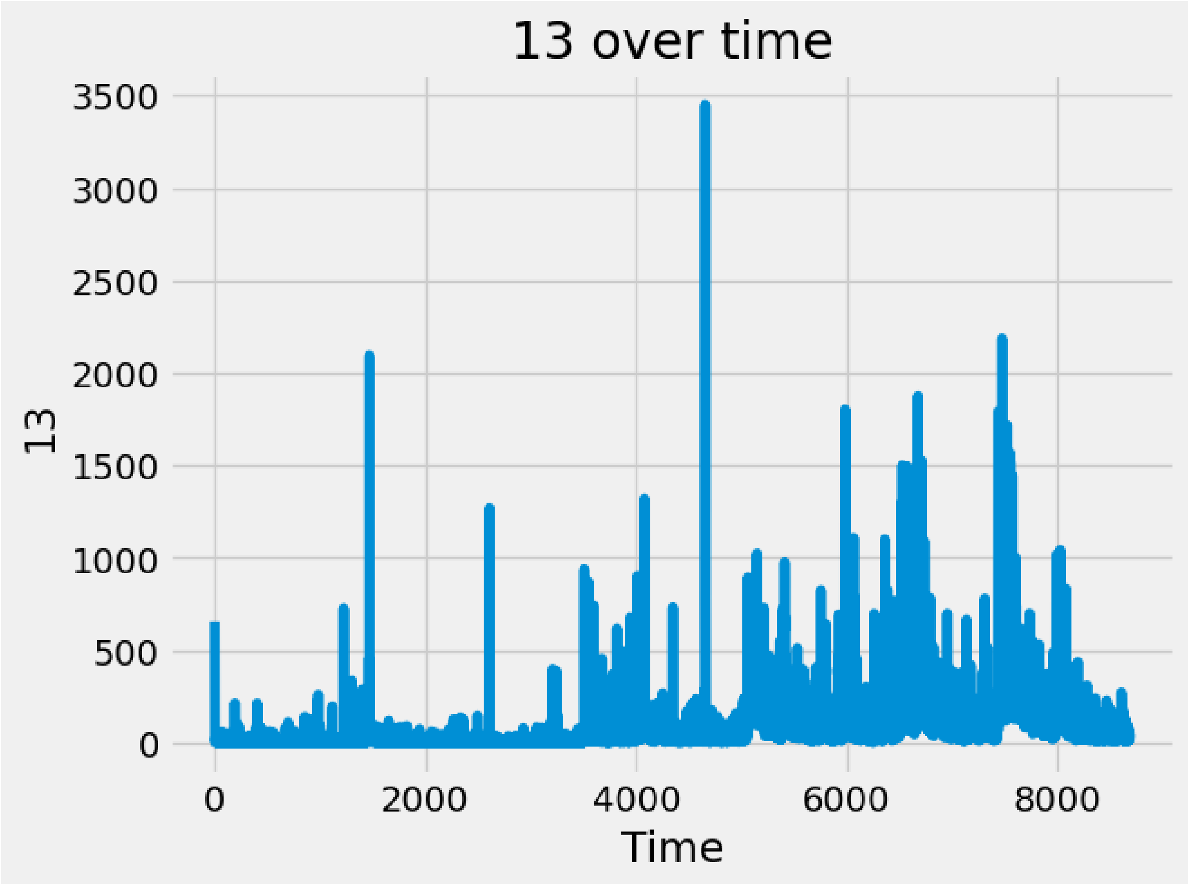

8 over time
10000
8000
6000
4000
2000
0
8
0
2000
4000
6000
8000
Time

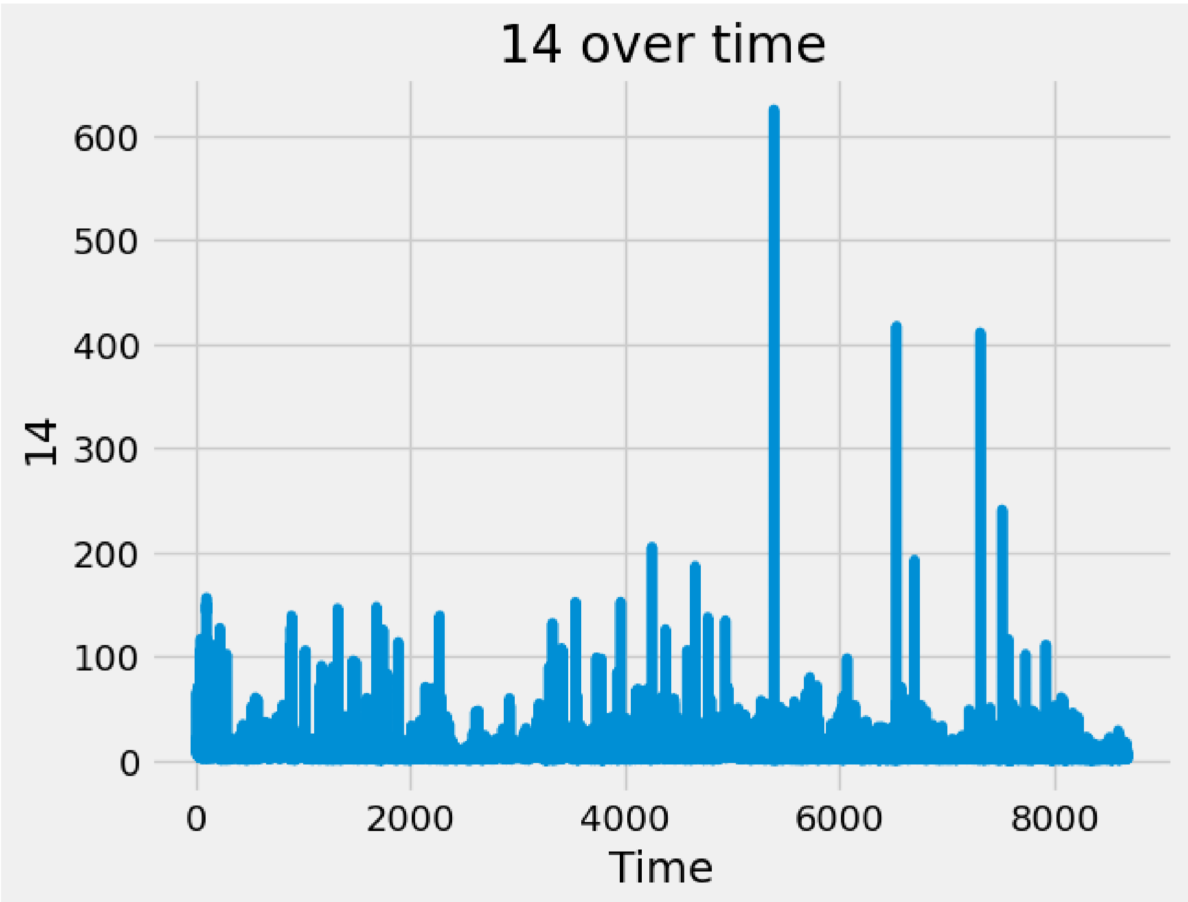

9 over time
8000
6000
4000
2000
0
9
0
2000
4000
6000
8000
Time

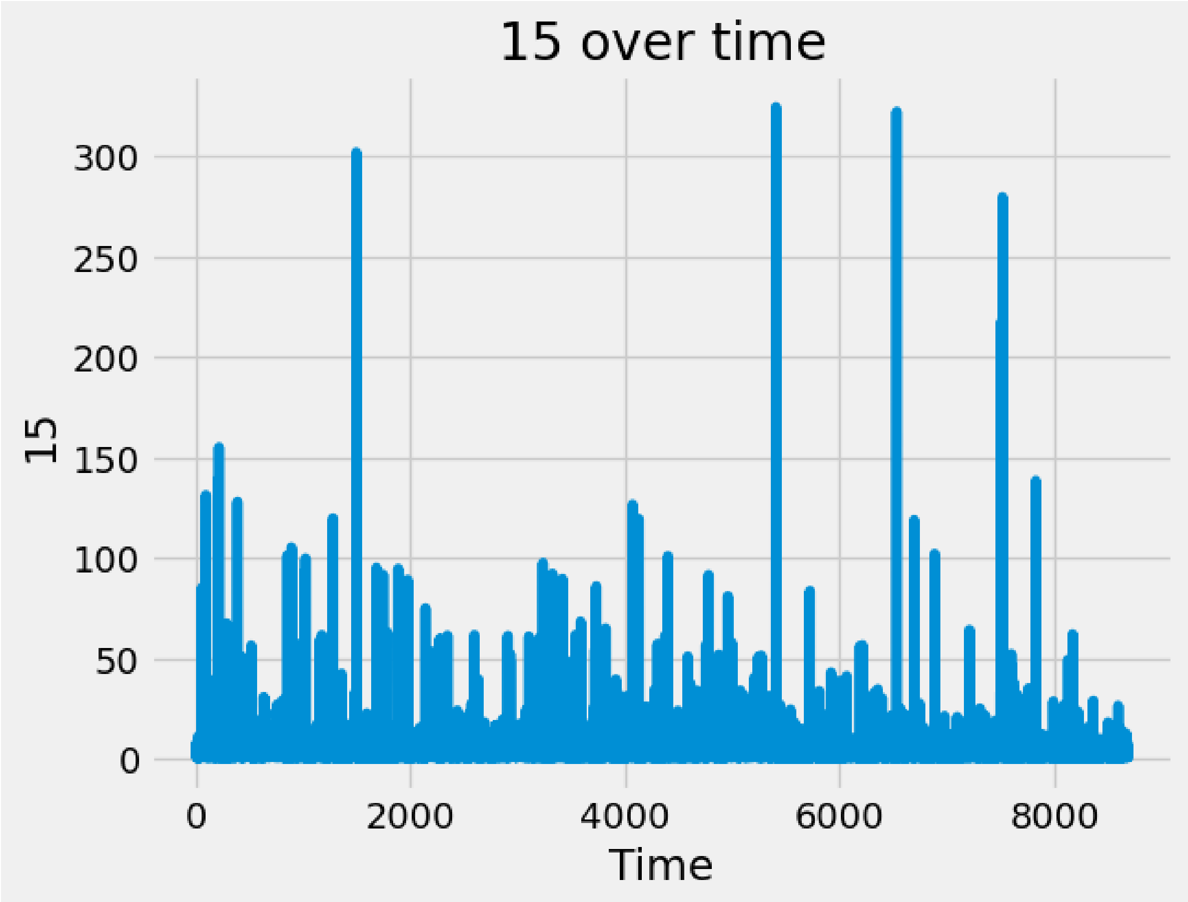

10 over time
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
10
0
2000
4000
6000
8000
Time

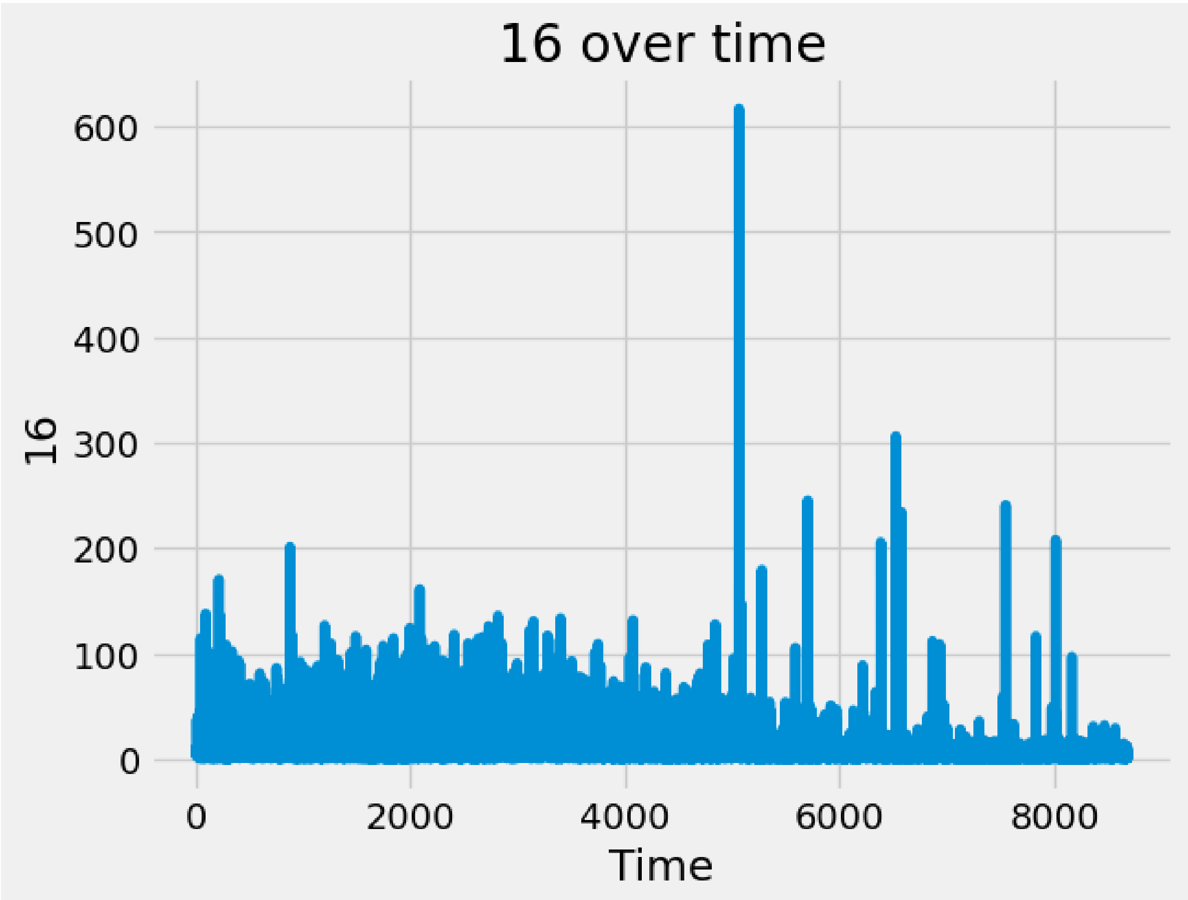

11 over time
4.1
4.0
3.9
3.8
3.7
3.6
3.5
3.4
11
0
2000
4000
6000
8000
Time

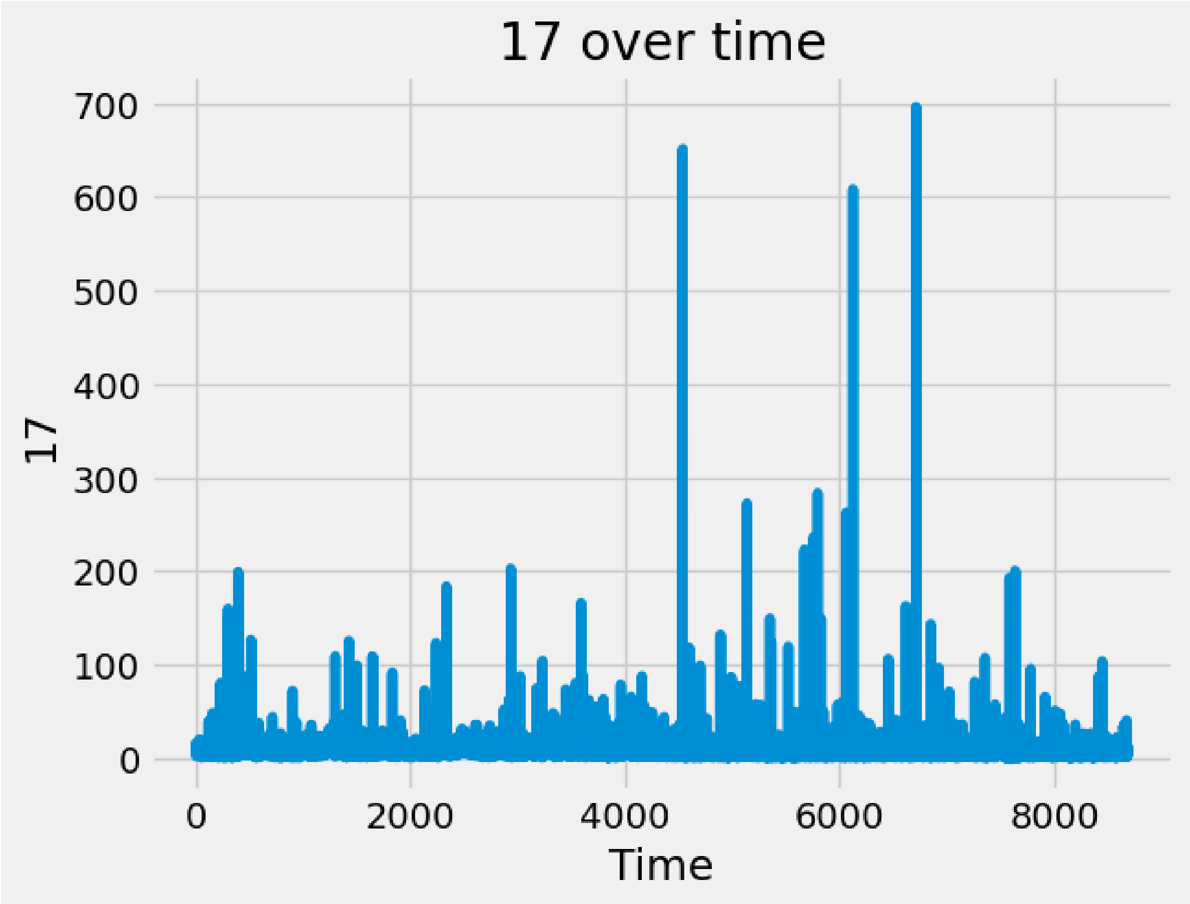

12 over time
3000
2500
2000
1500
1000
500
0
12
0
2000
4000
6000
8000
Time

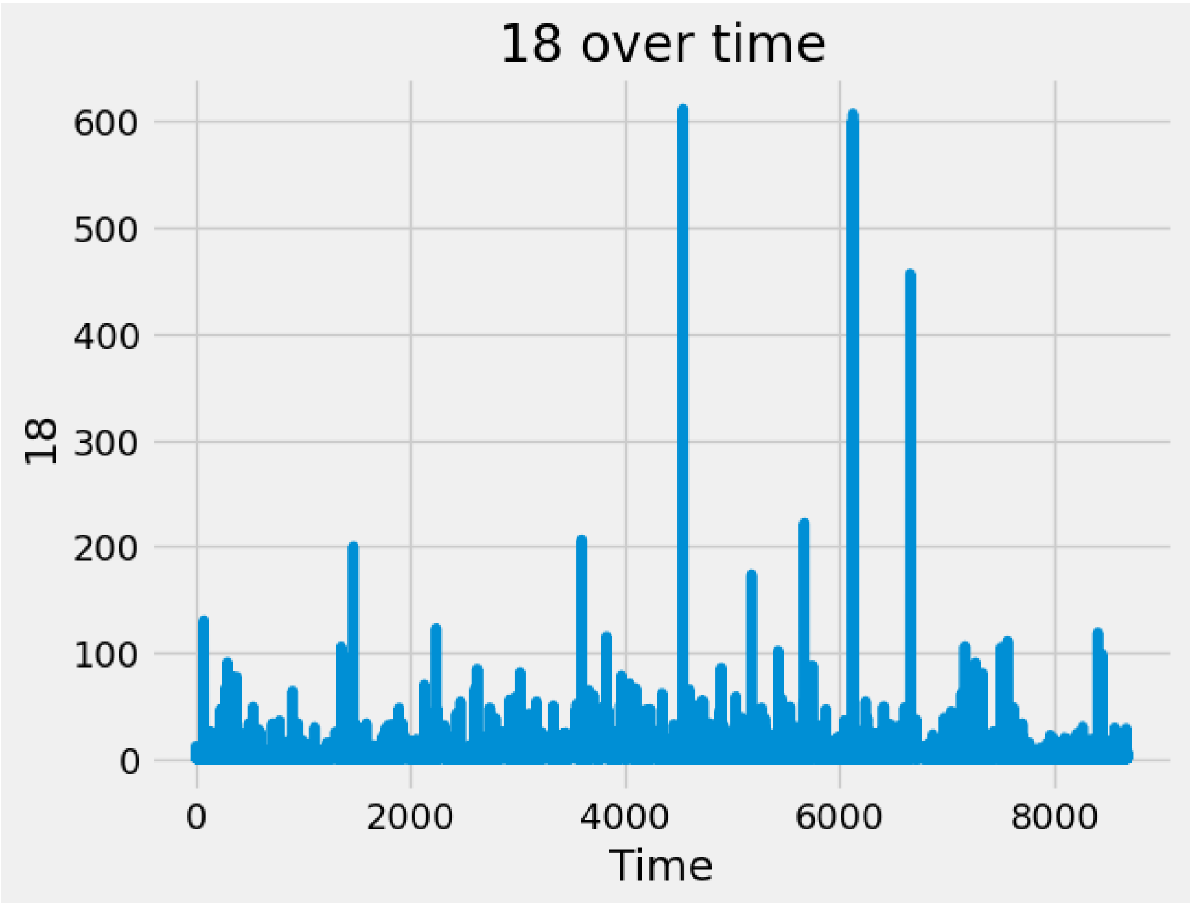

13 over time
3500
3000
2500
2000
1500
1000
500
0
13
0
2000
4000
6000
8000
Time

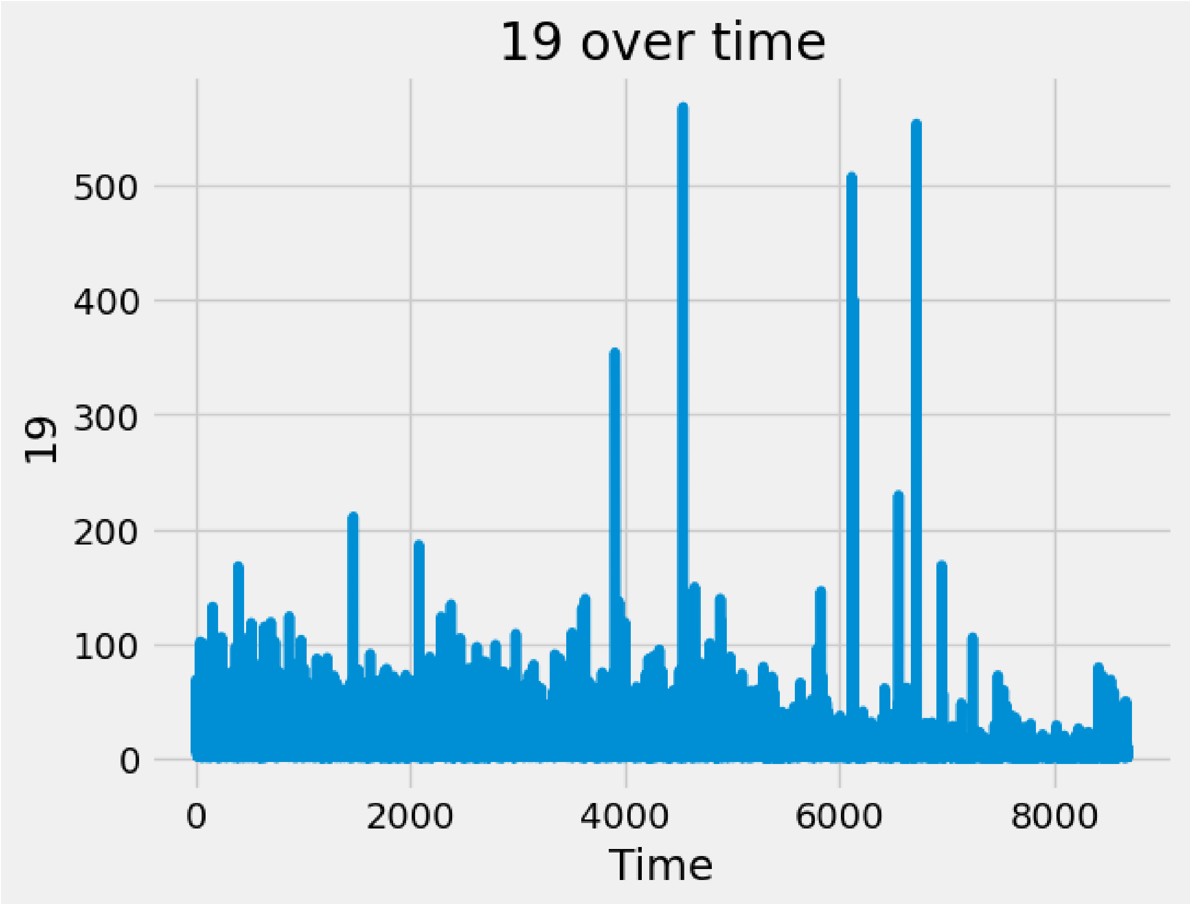

14 over time
600
500
400
300
200
100
0
14
0
2000
4000
6000
8000
Time

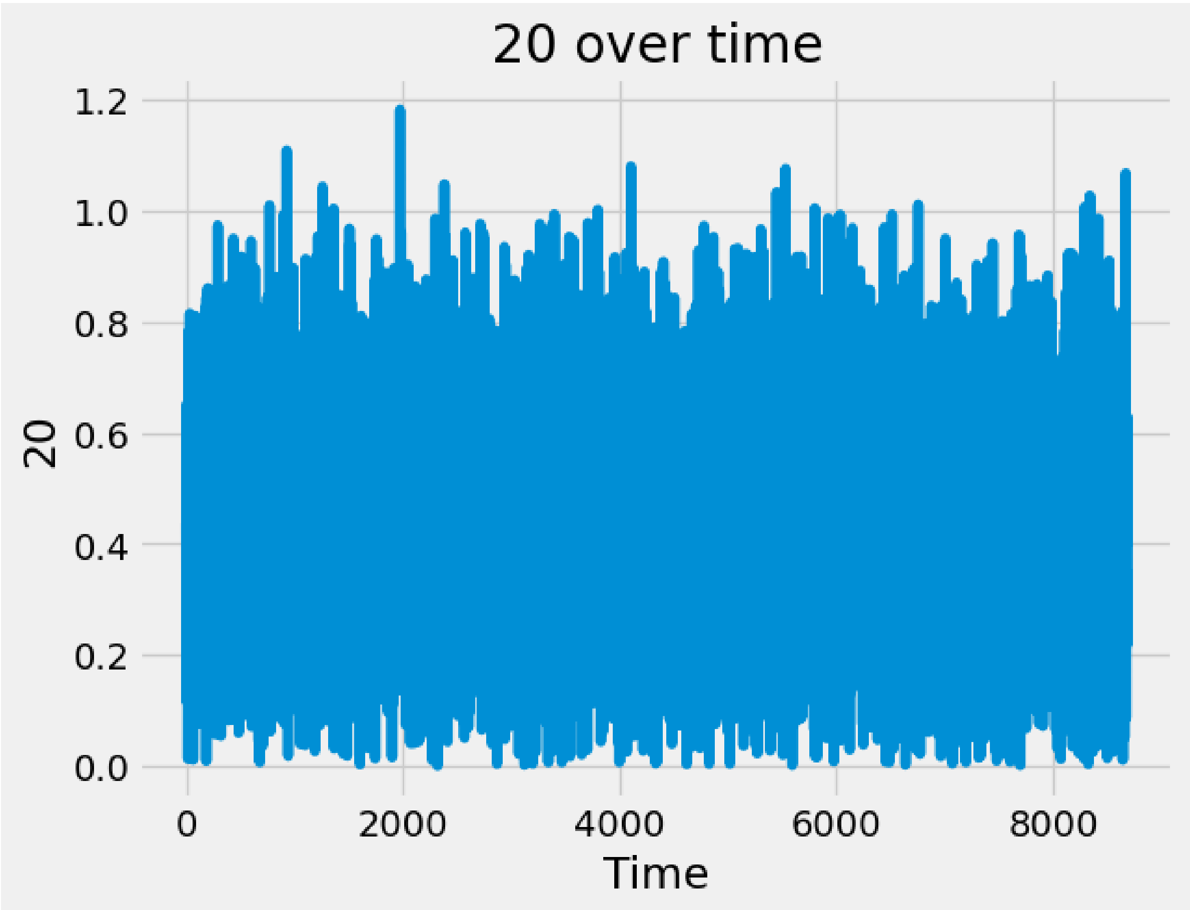

15 over time
15
Time
0
50
100
150
200
250
300
0
2000
4000
6000
8000

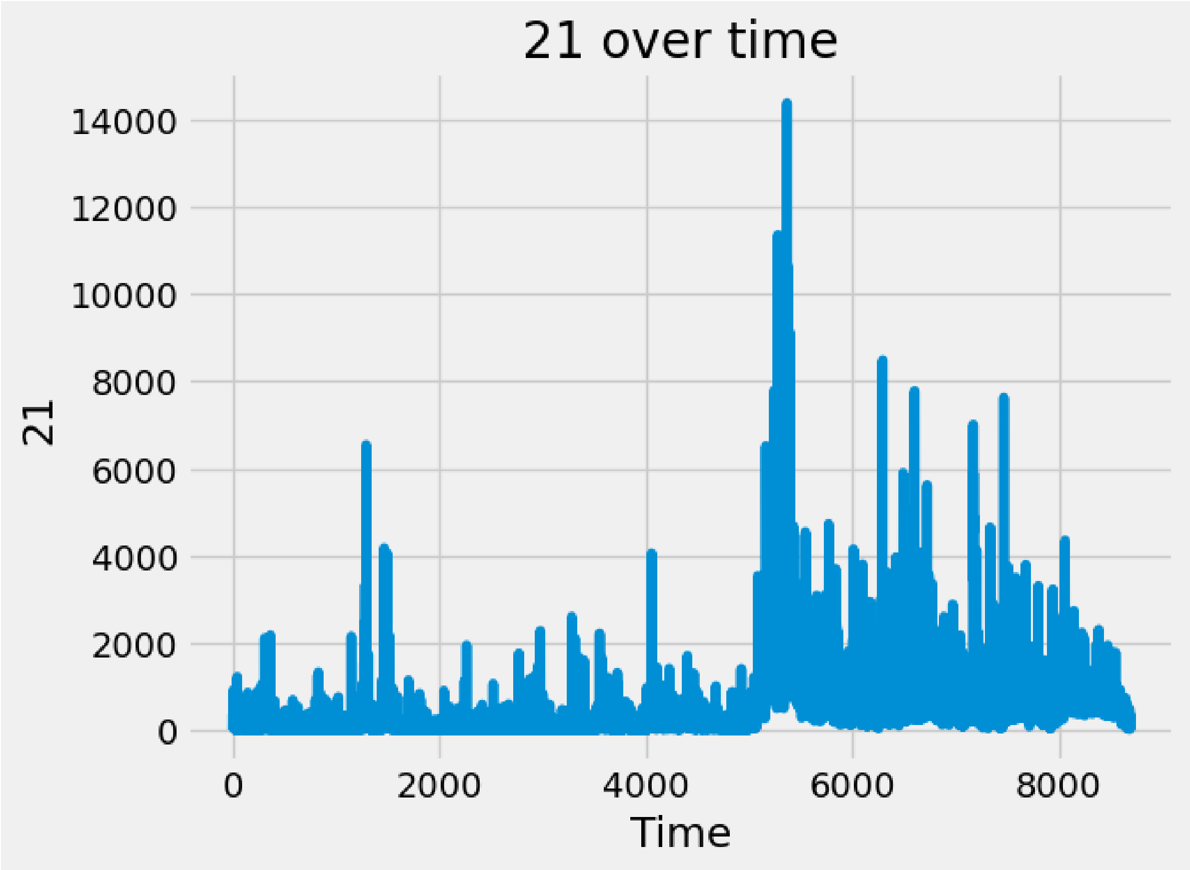

16 over time
16
Time
0
100
200
300
400
500
600
0
2000
4000
6000
8000


17 over time
17
Time
0
2000
4000
6000
8000
0
100
200
300
400
500
600
700

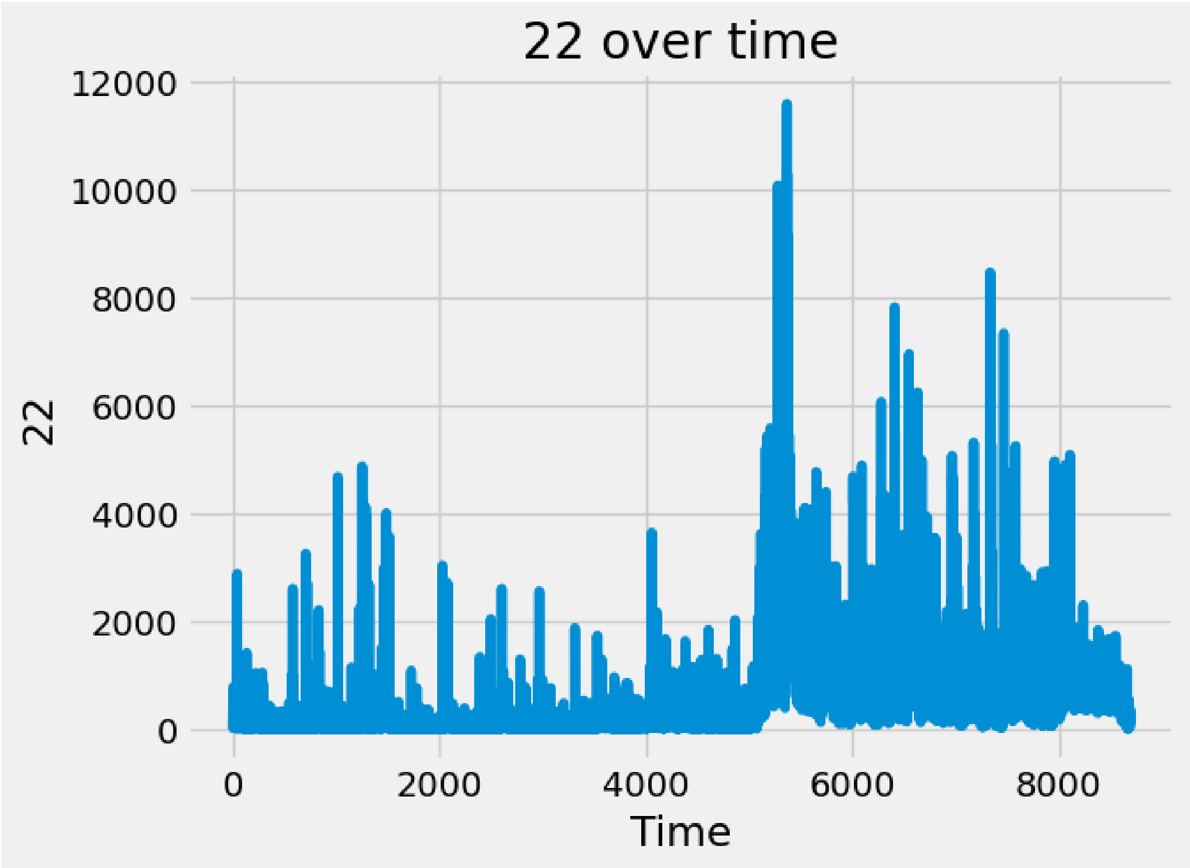

18 over time
18
Time
0
2000
4000
6000
8000
0
100
200
300
400
500
600


19 over time
19
500
400
300
200
100
0
0
2000
4000
6000
8000
Time

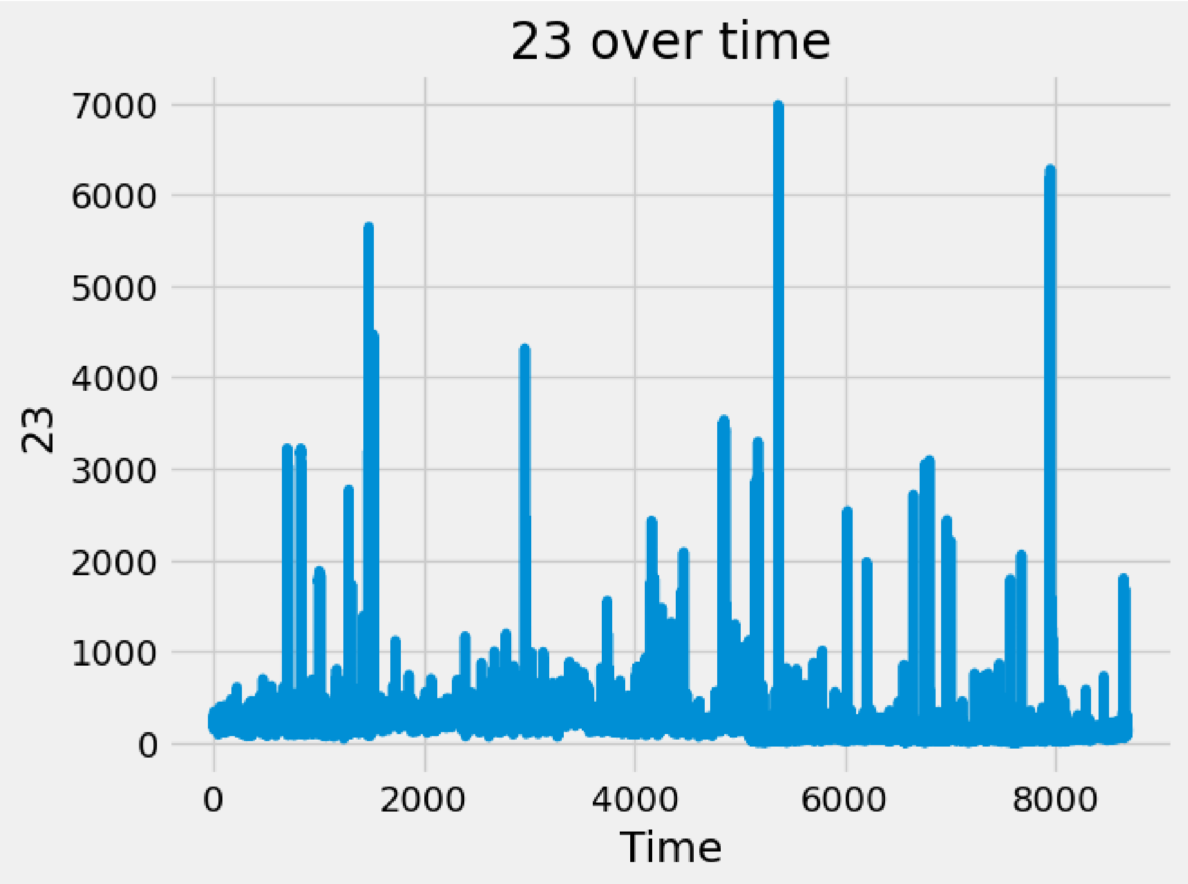

20 over time
20
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
Time

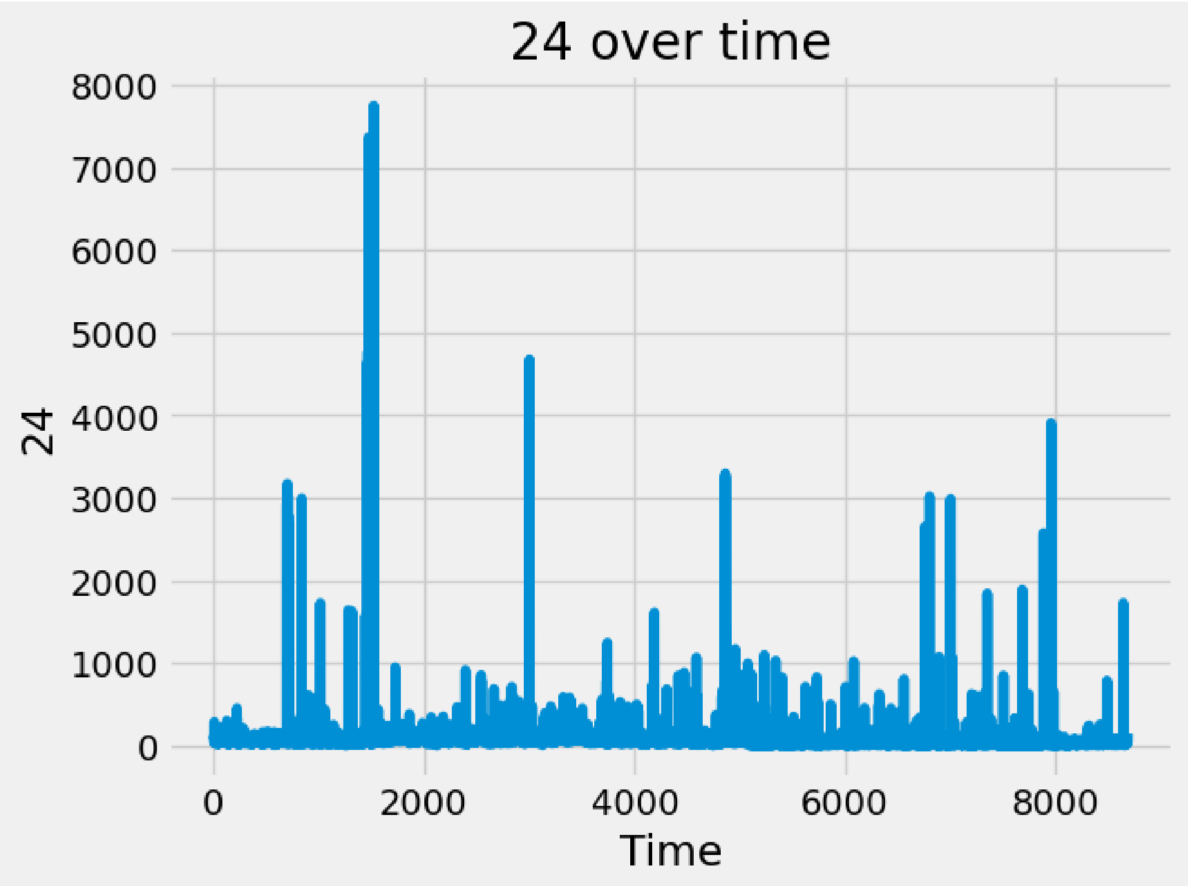


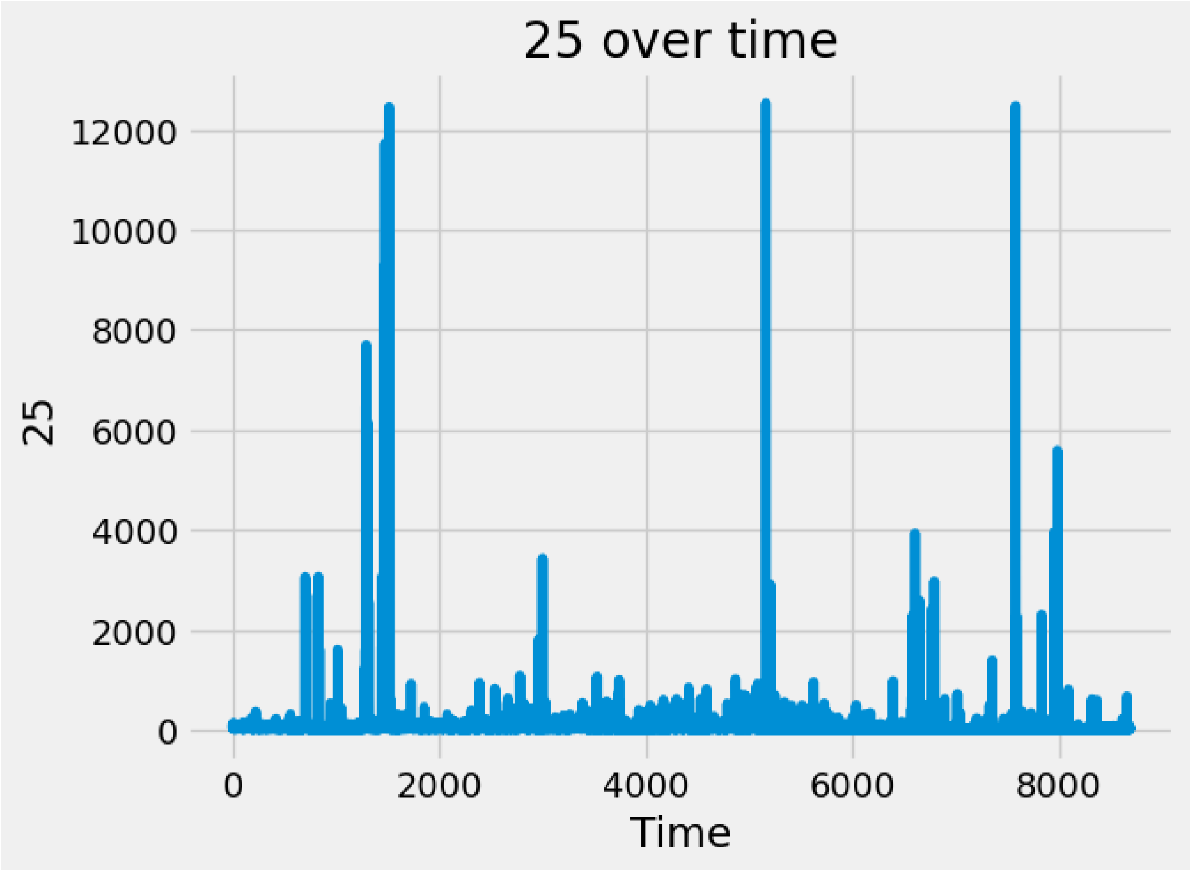
21 over time
14000
12000
10000
8000
6000
4000
2000
0
21
0
2000
4000
6000
8000
Time

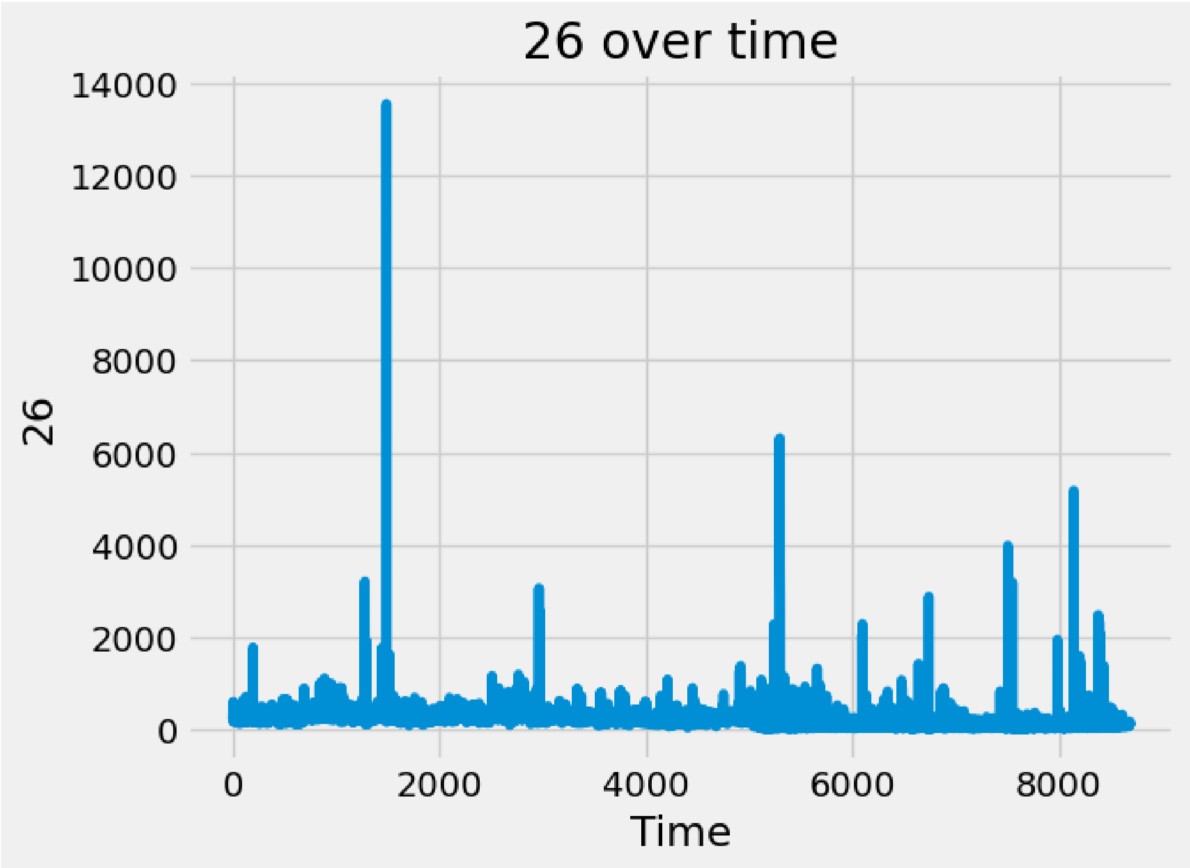


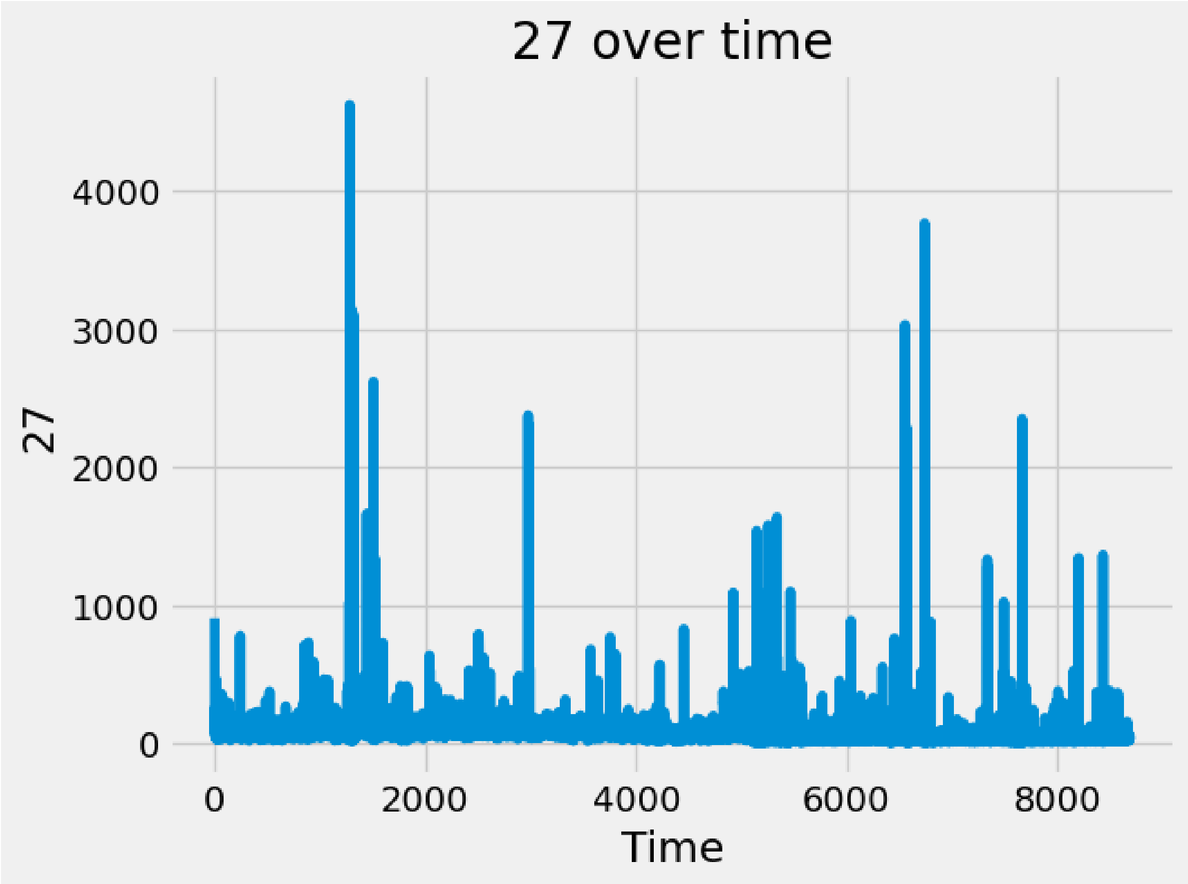
22 over time
12000
10000
8000
6000
4000
2000
0
22
0
2000
4000
6000
8000
Time

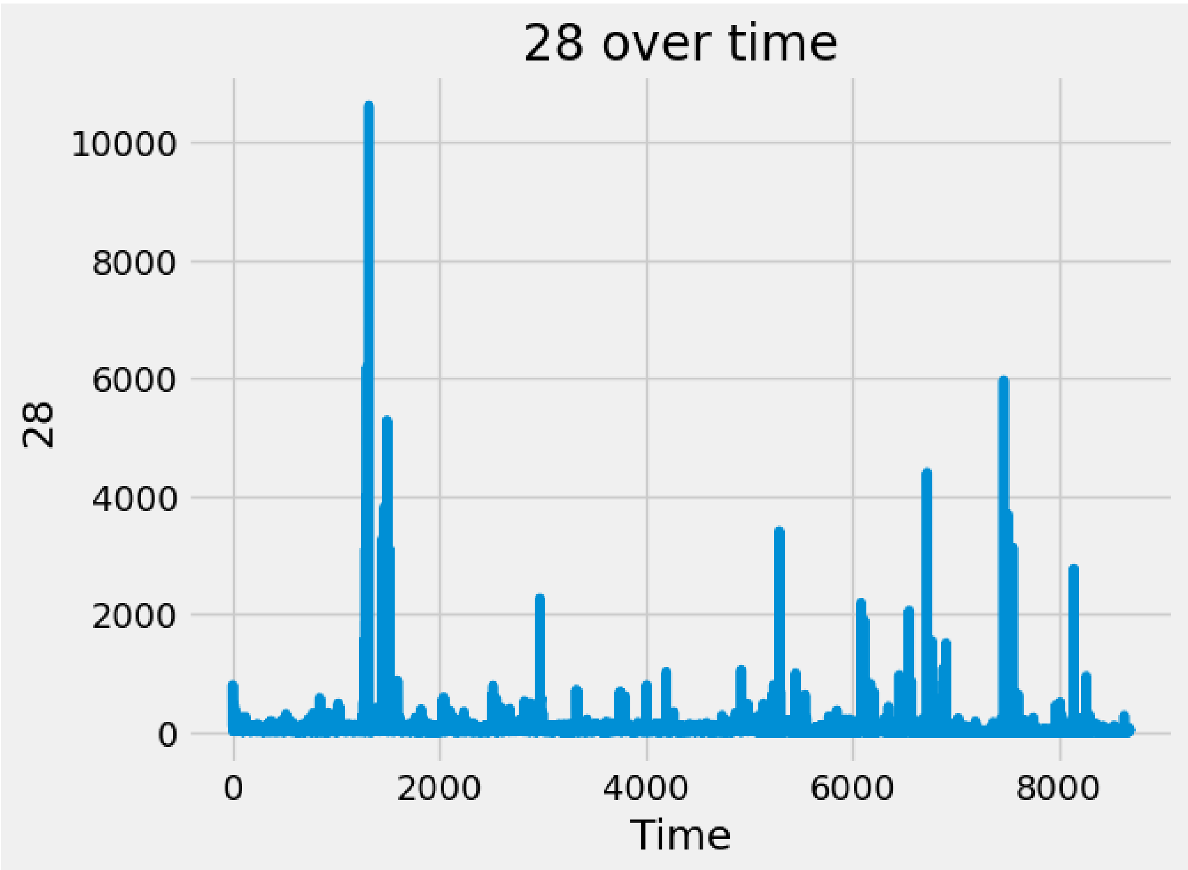

23 over time
7000
6000
5000
4000
3000
2000
1000
0
23
0
2000
4000
6000
8000
Time

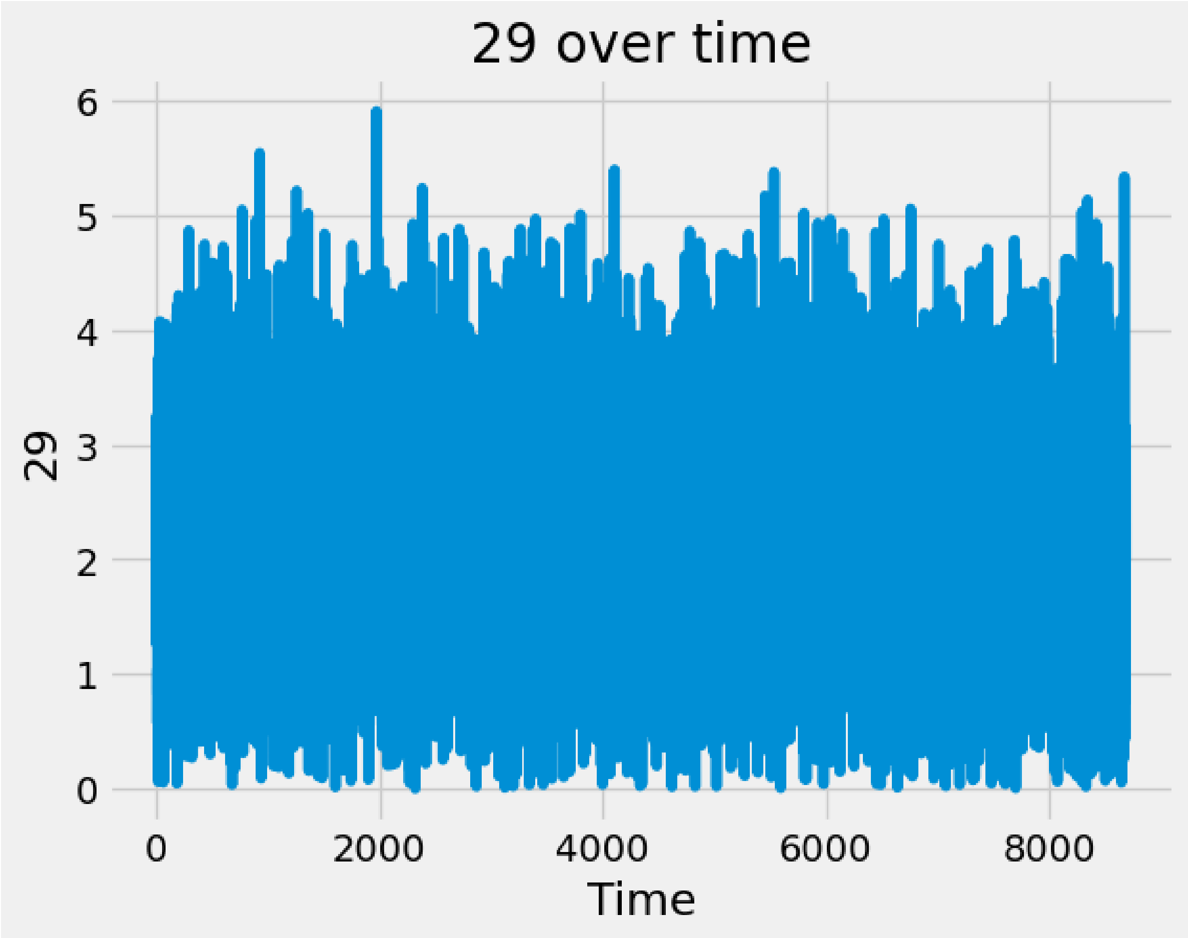

24 over time
8000
7000
6000
5000
4000
3000
2000
1000
0
24
0
2000
4000
6000
8000
Time

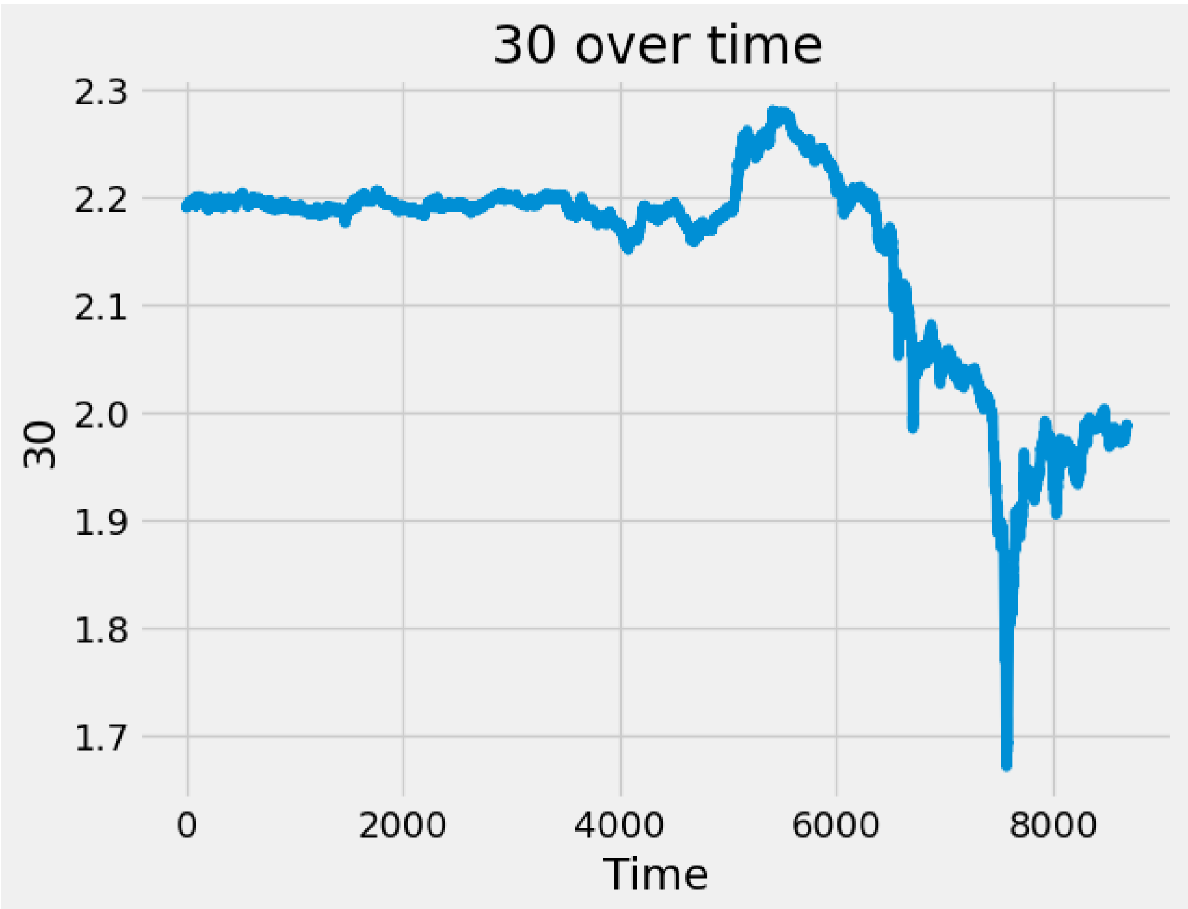

25 over time
25
Time
12000
10000
8000
6000
4000
2000
0
0
2000
4000
6000
8000

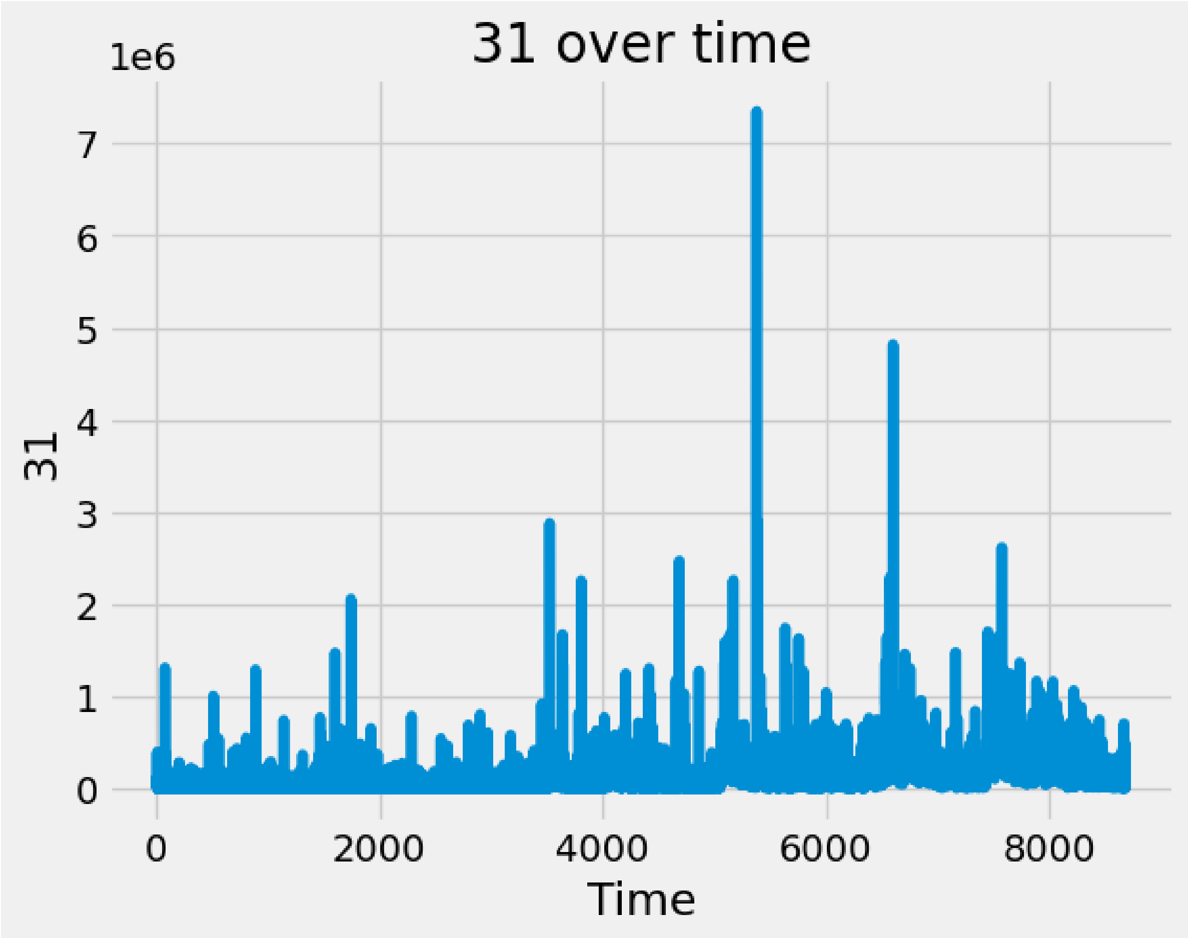

26 over time
26
Time
14000
12000
10000
8000
6000
4000
2000
0
0
2000
4000
6000
8000

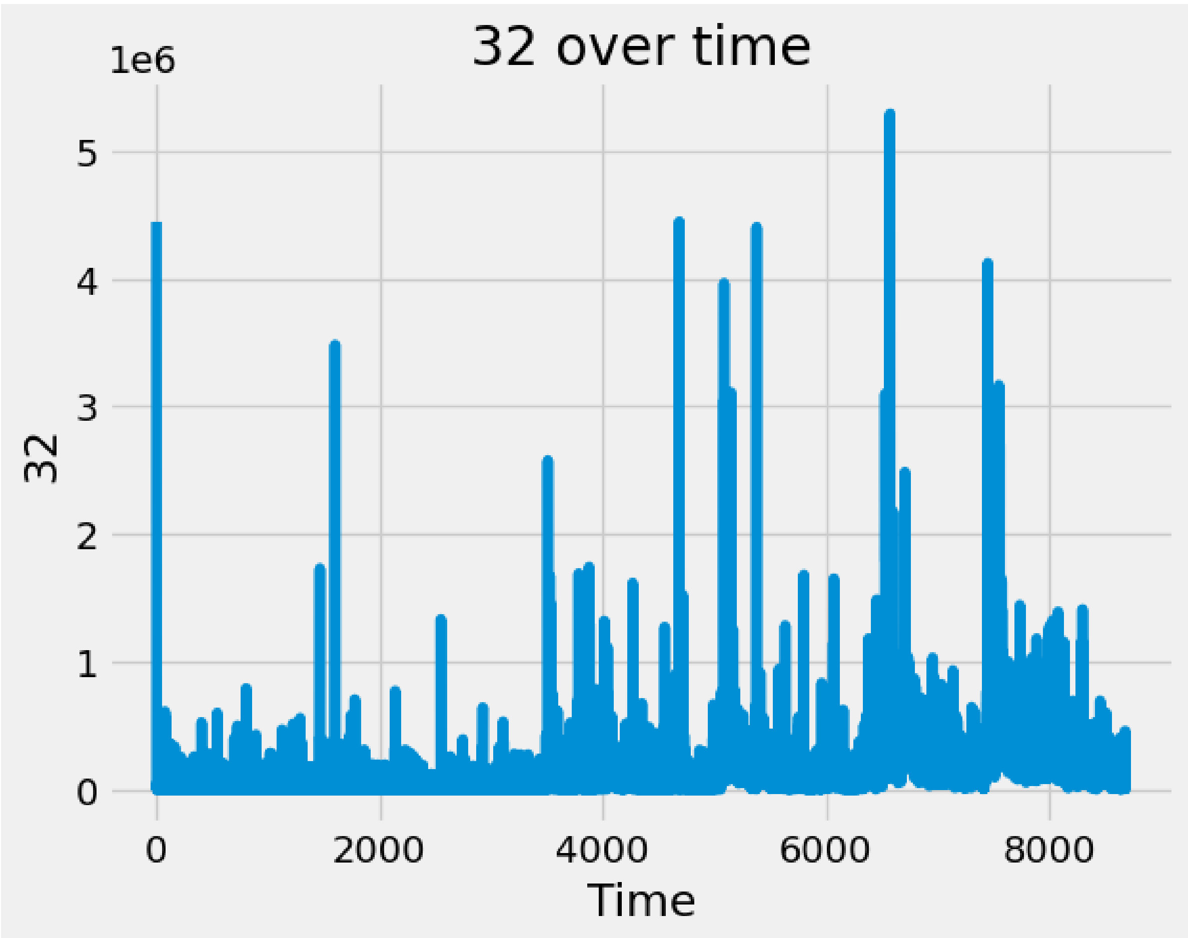

27 over time
27
4000
3000
2000
1000
0
0
2000
4000
6000
8000
Time

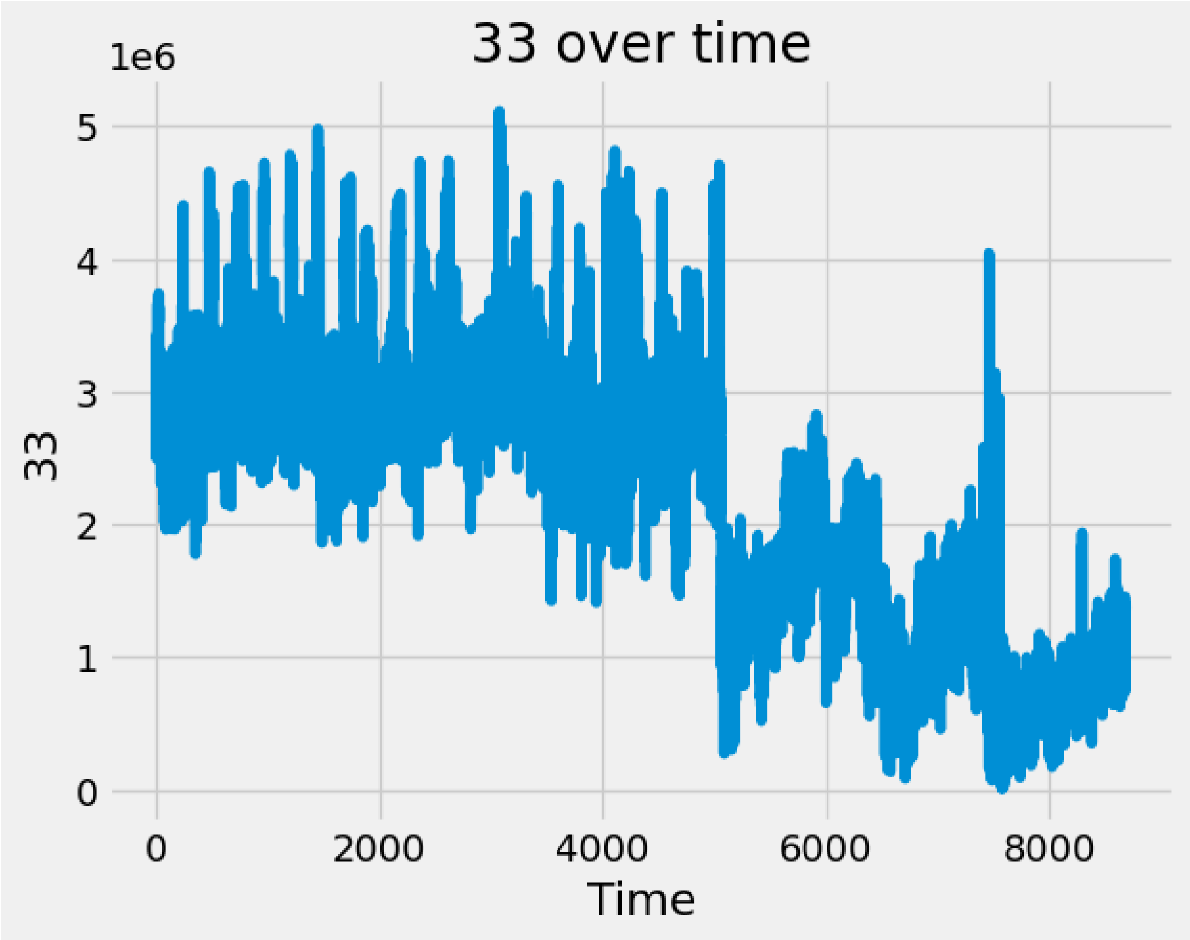

28 over time
28
10000
8000
6000
4000
2000
0
0
2000
4000
6000
8000
Time


29 over time
29
Time
0
1
2
3
4
5
6
0
2000
4000
6000
8000

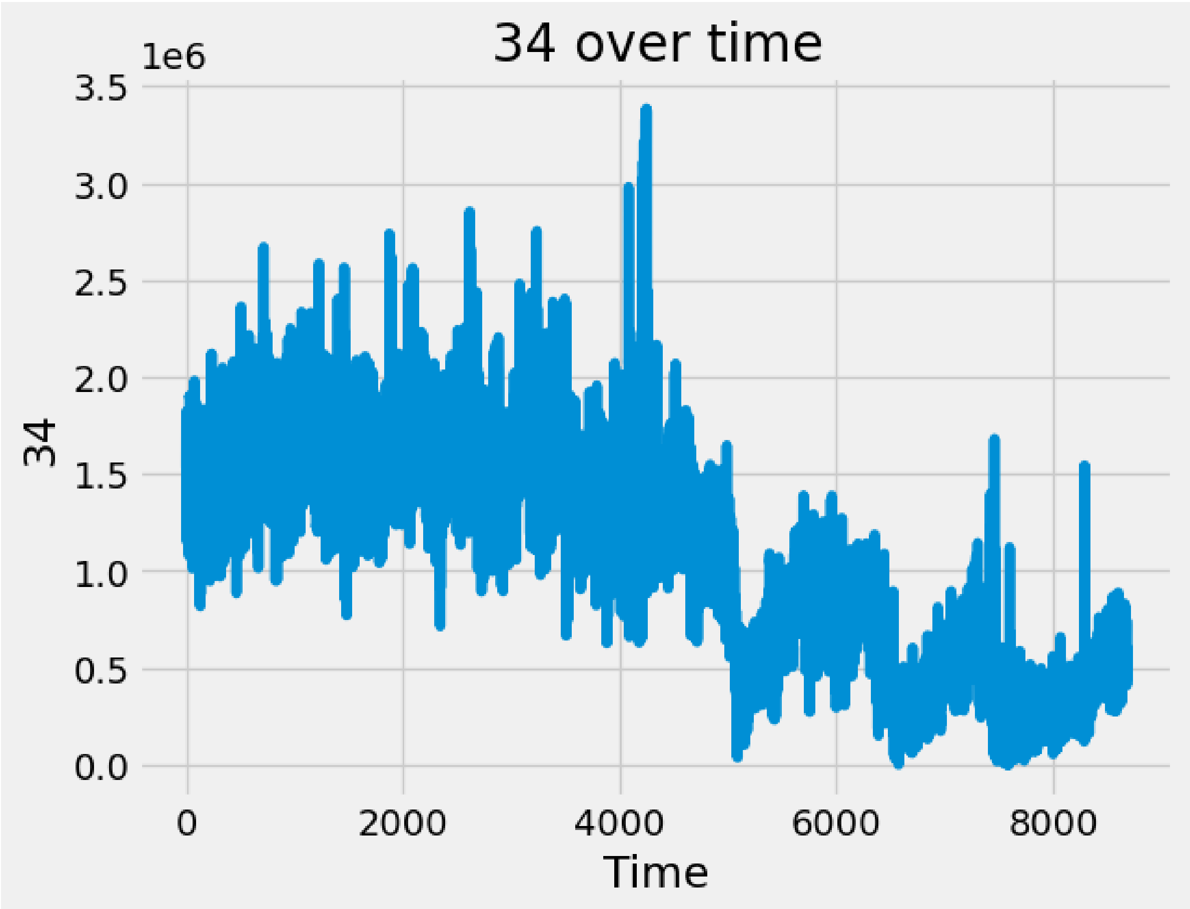

30 over time
30
Time
1.7
1.8
1.9
2.0
2.1
2.2
2.3
0
2000
4000
6000
8000


31 over time
1e6
7
6
5
4
3
2
1
0
31
0
2000
4000
6000
8000
Time

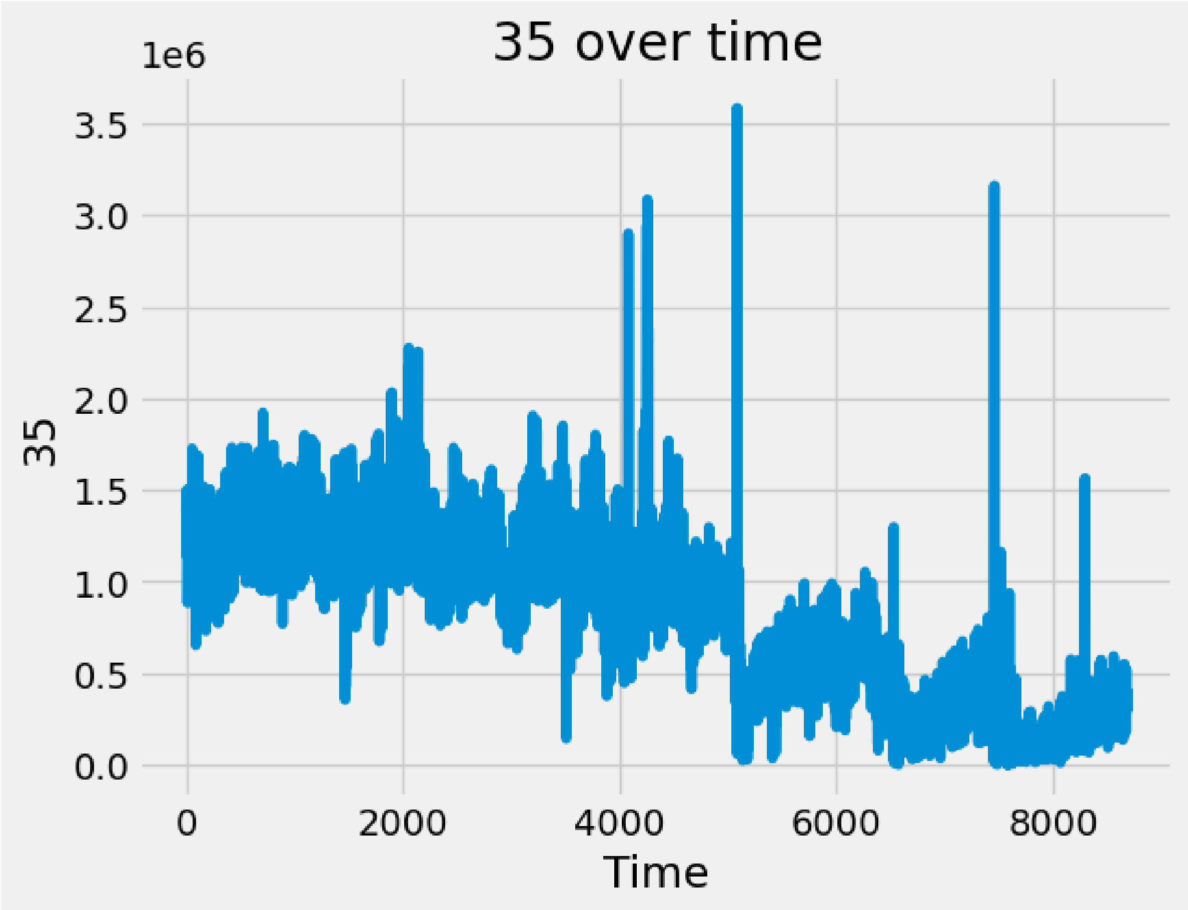

32 over time
1e6
5
4
3
2
1
0
32
0
2000
4000
6000
8000
Time

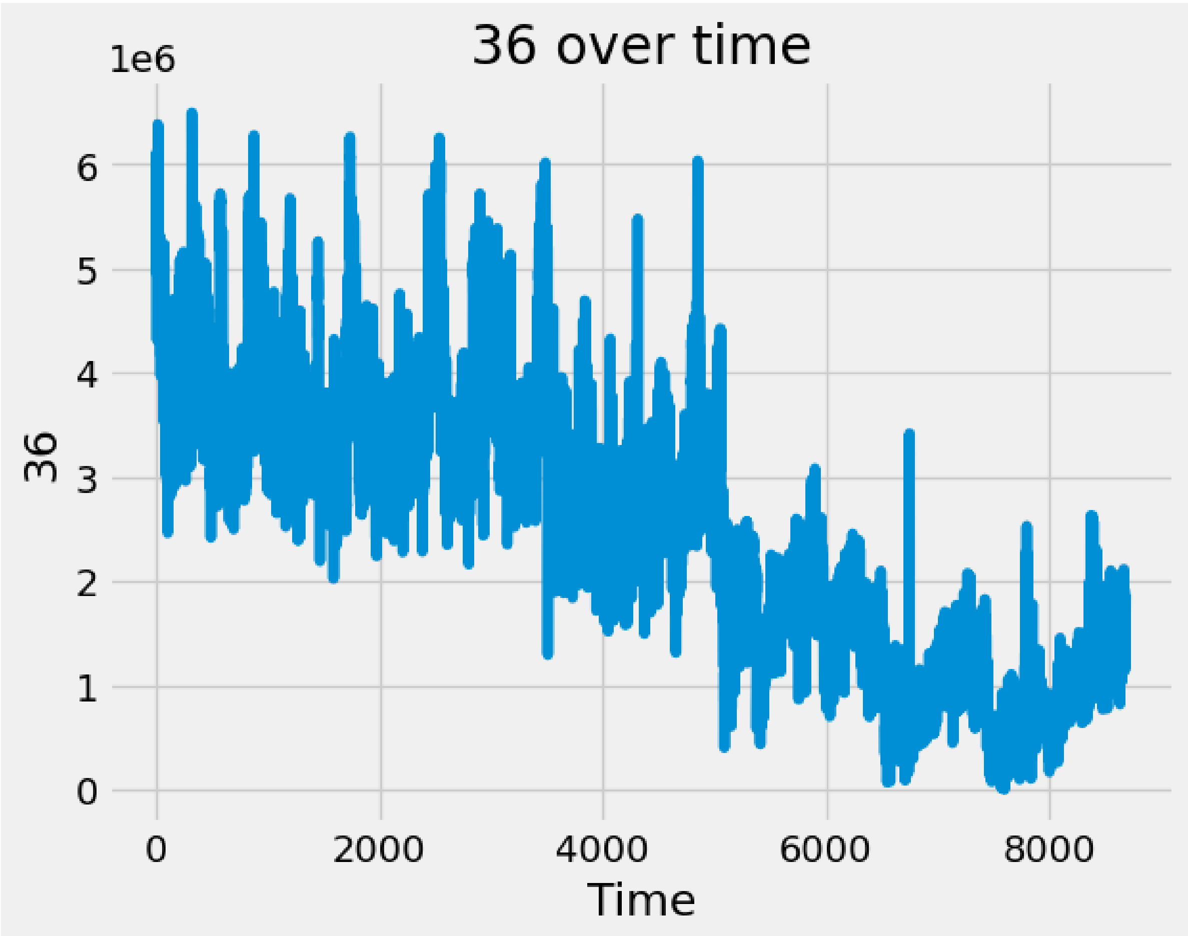


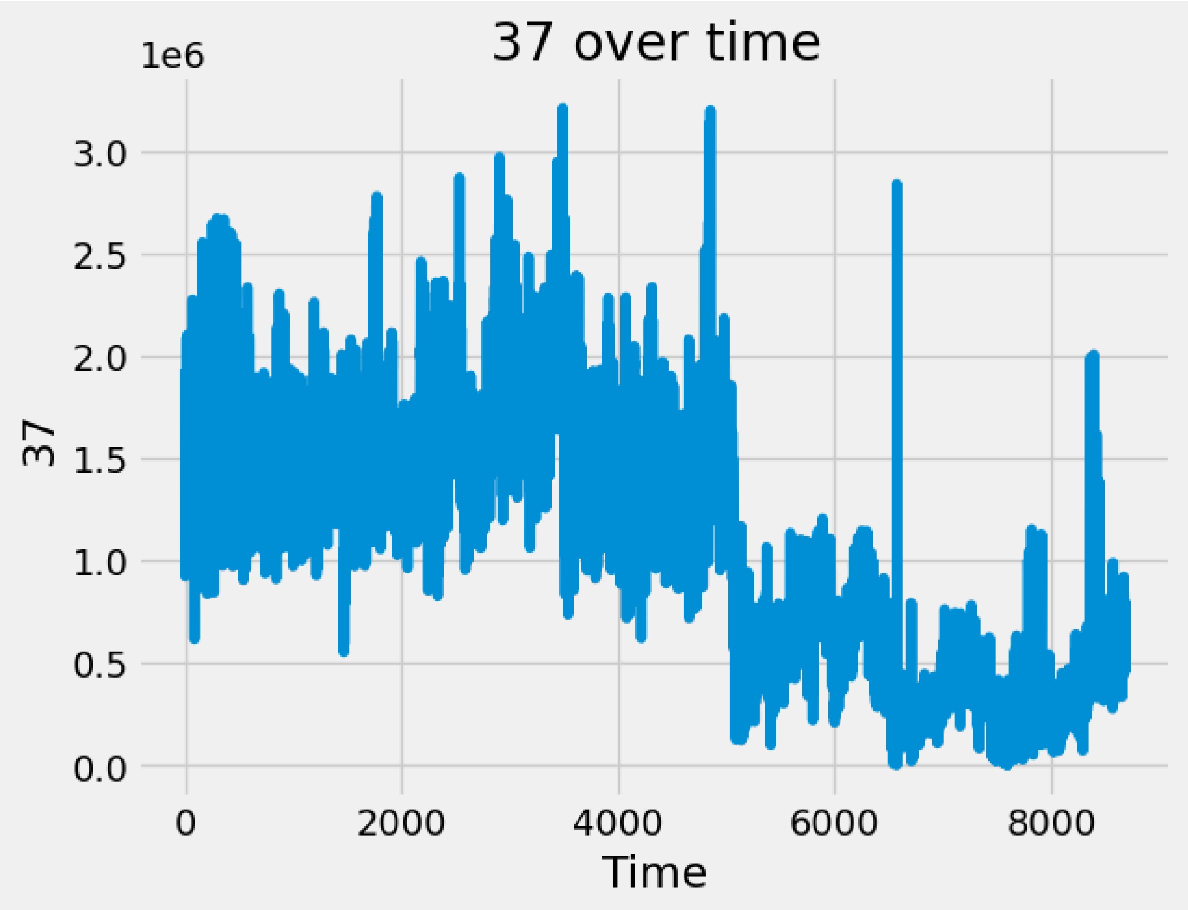
33 over time
1e6
5
4
3
2
1
0
33
0
2000
4000
6000
8000
Time

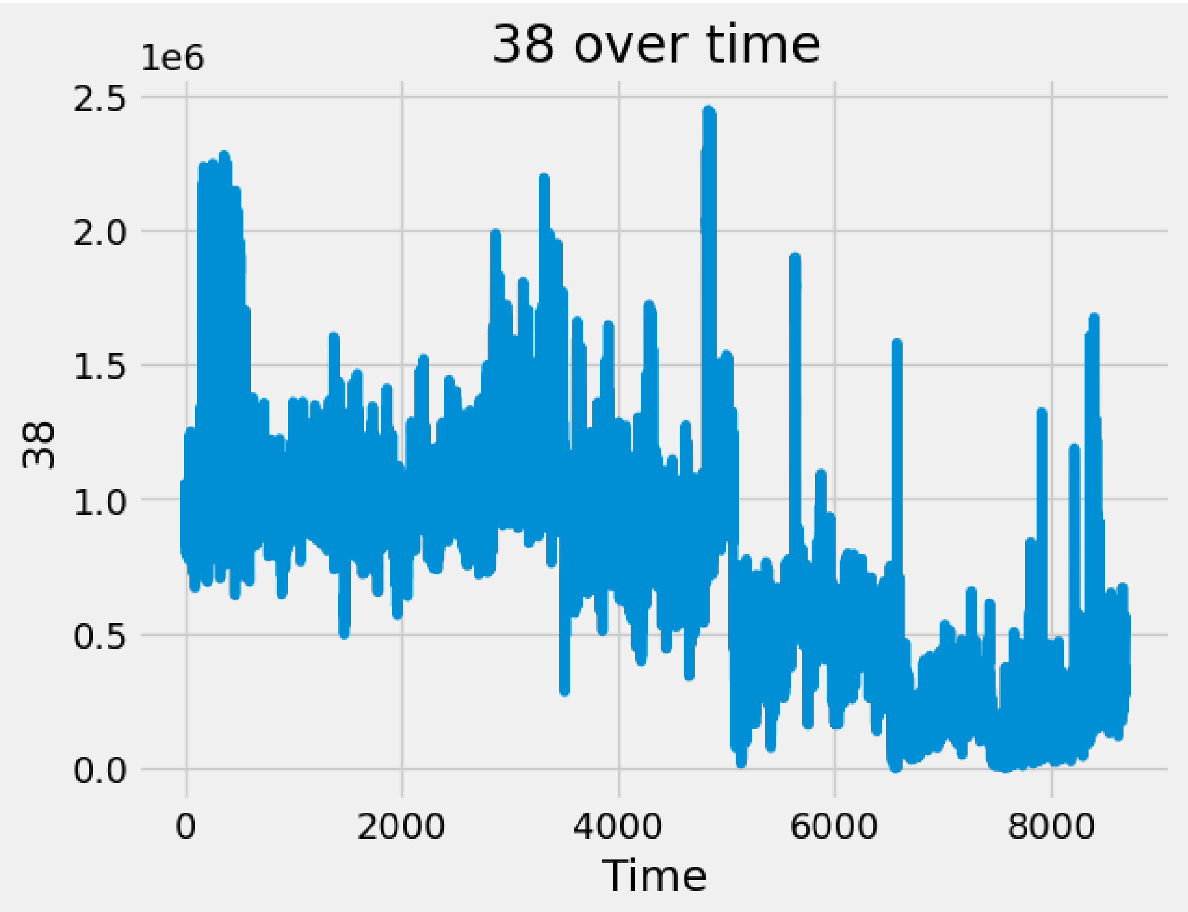


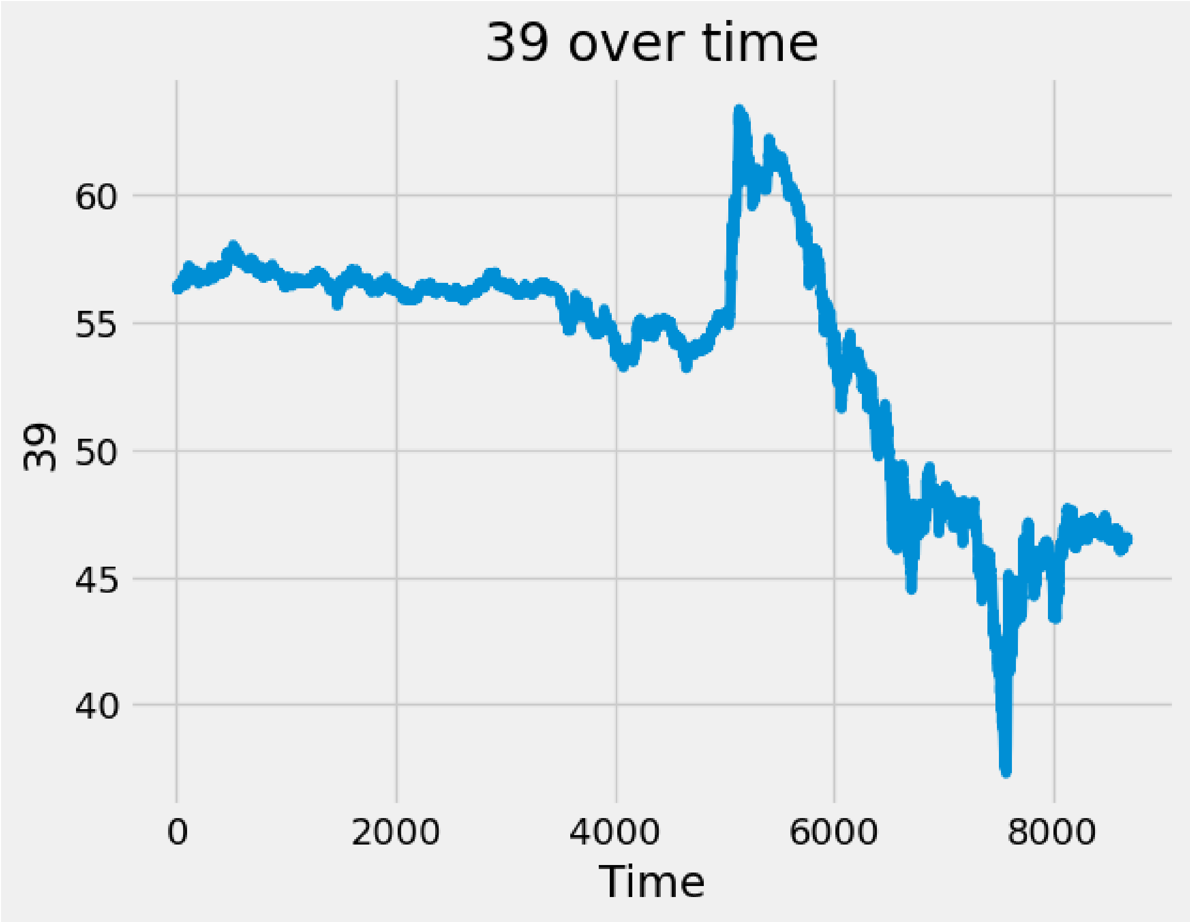
34 over time
1e6
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
34
0
2000
4000
6000
8000
Time

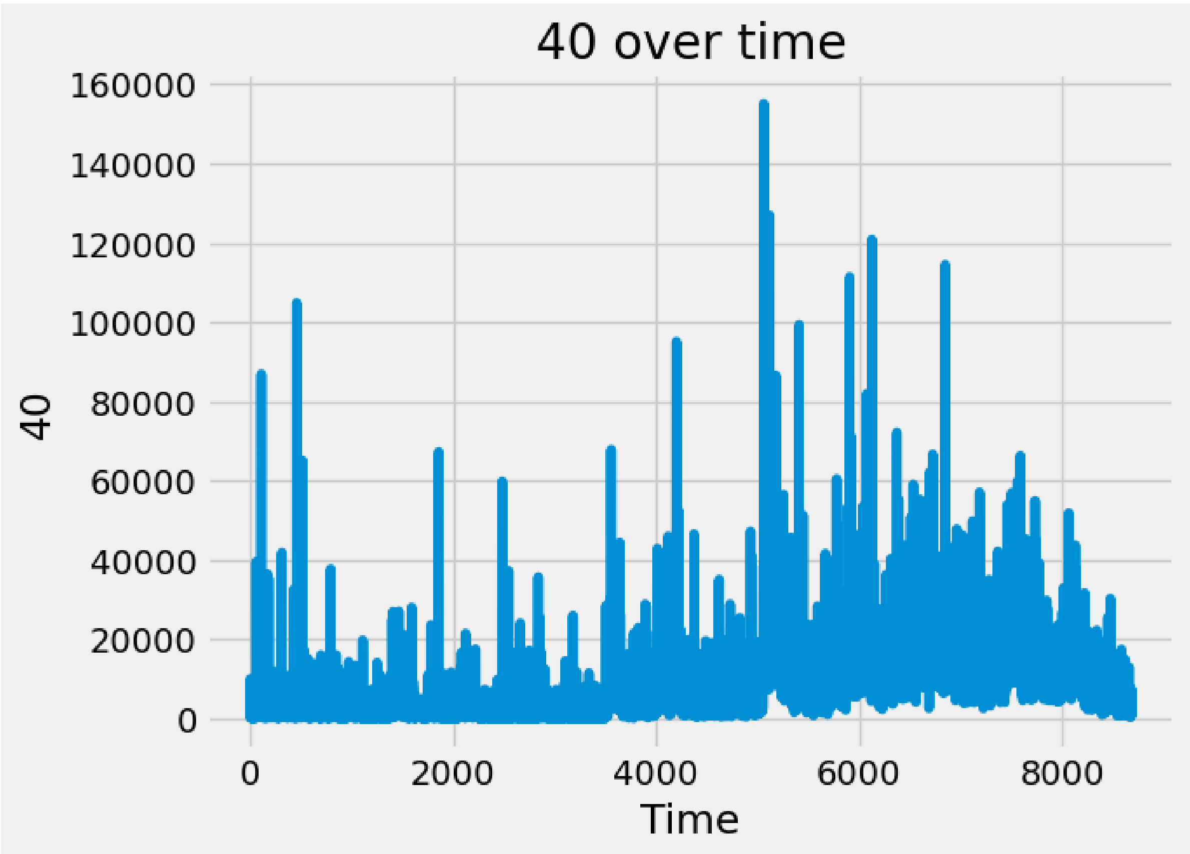

35 over time
1e6
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
35
0
2000
4000
6000
8000
Time

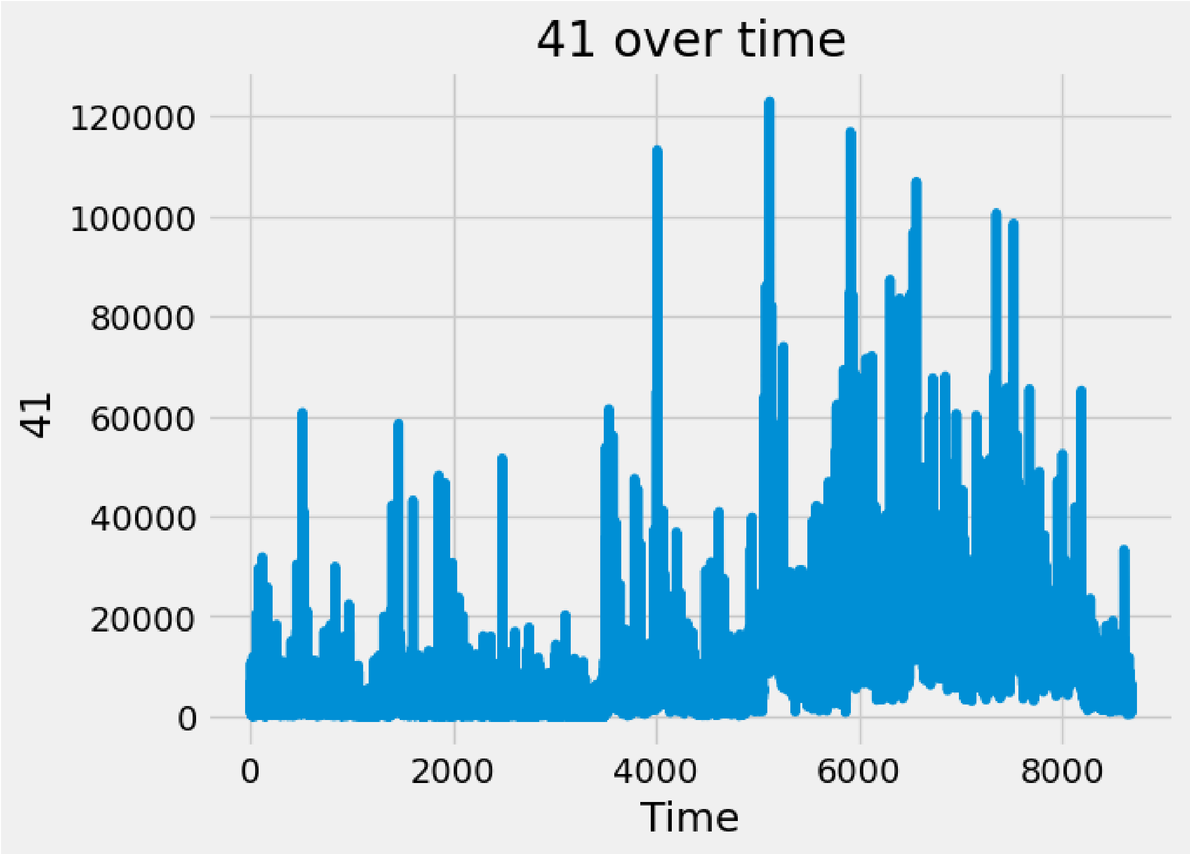

36 over time
1e6
6
5
4
3
2
1
0
36
0
2000
4000
6000
8000
Time

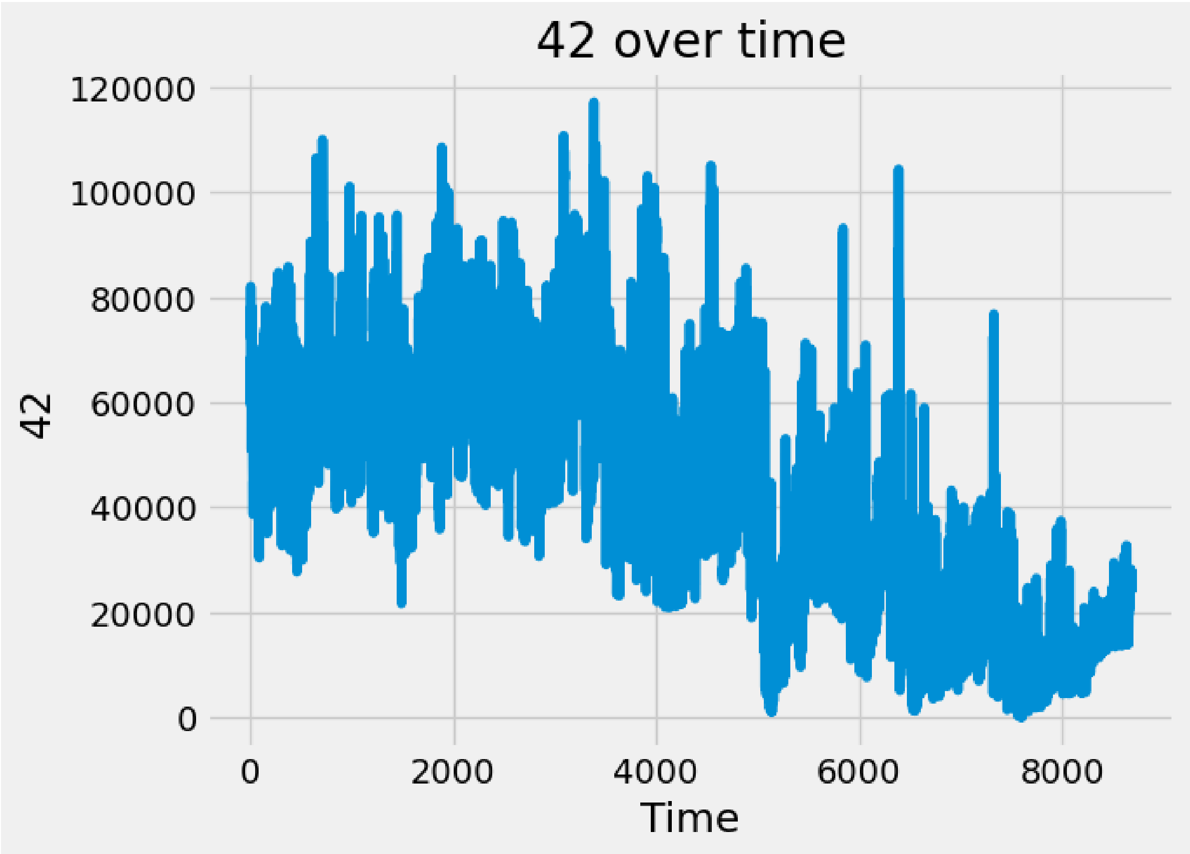

37 over time
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
37
0
2000
4000
6000
8000
Time

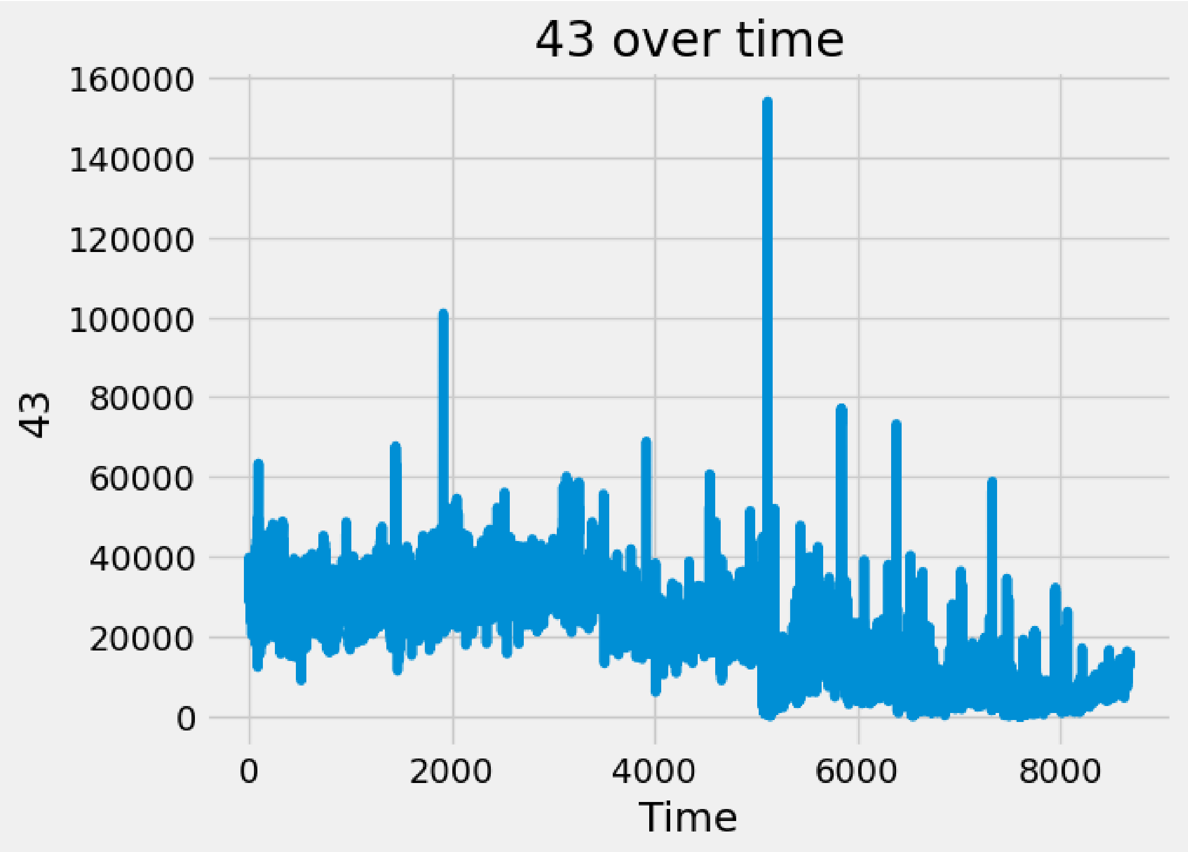

38 over time
1e6
2.5
2.0
1.5
1.0
0.5
0.0
38
0
2000
4000
6000
8000
Time

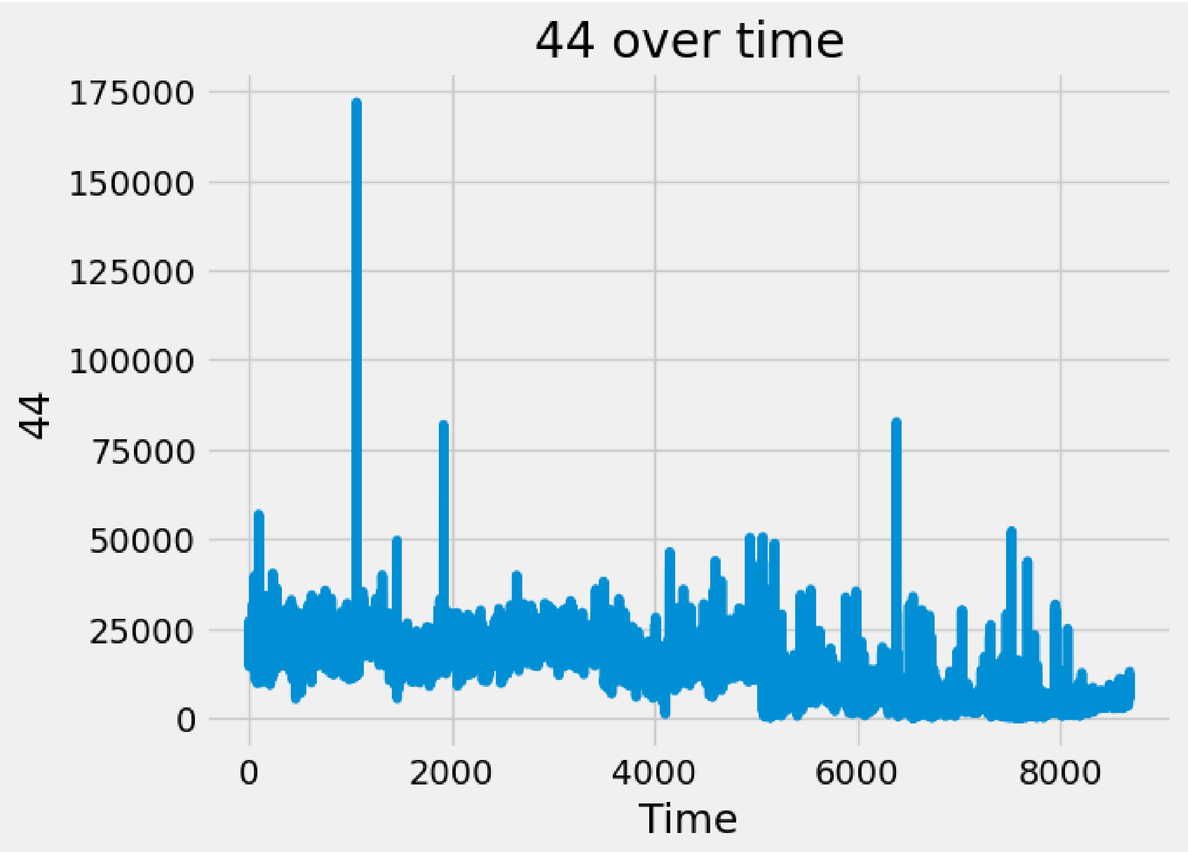

39 over time
60
55
50
45
40
39
0
2000
4000
6000
8000
Time

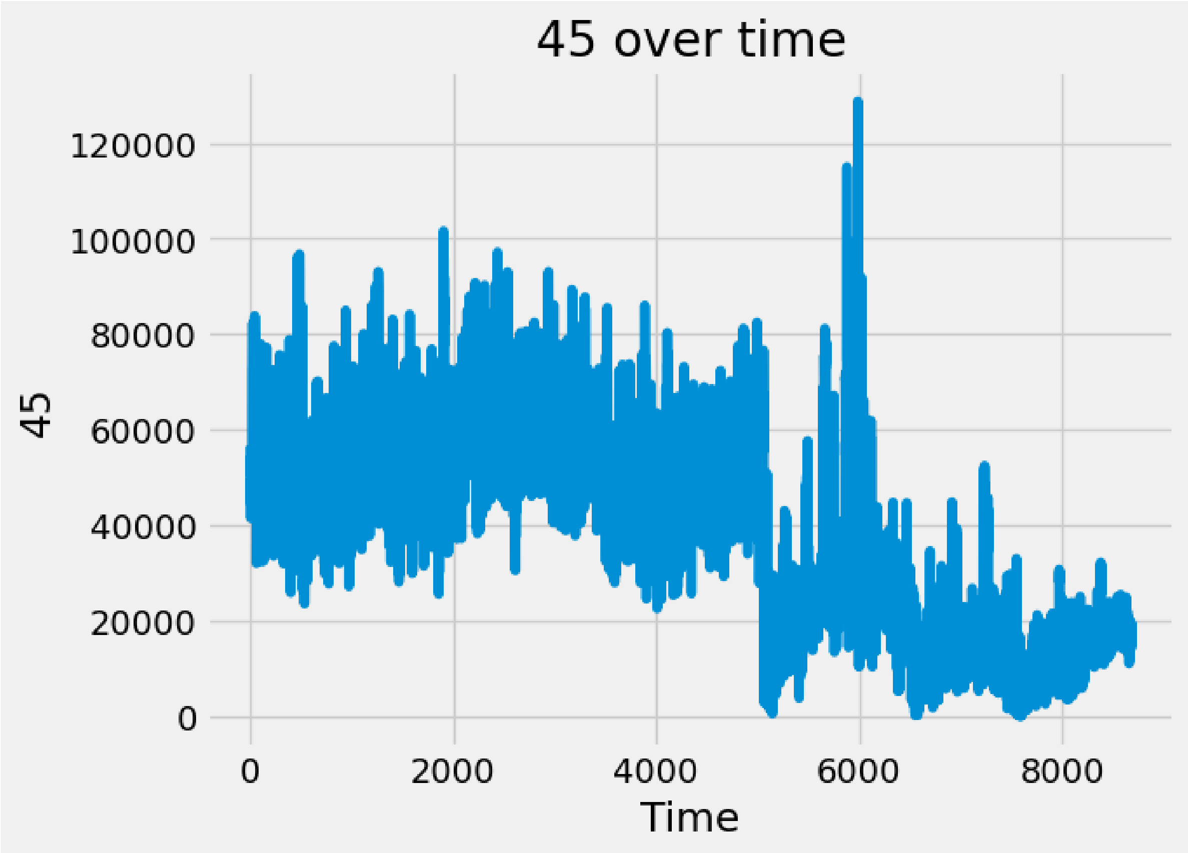

40 over time
160000
140000
120000
100000
80000
60000
40000
20000
0
40
0
2000
4000
6000
8000
Time

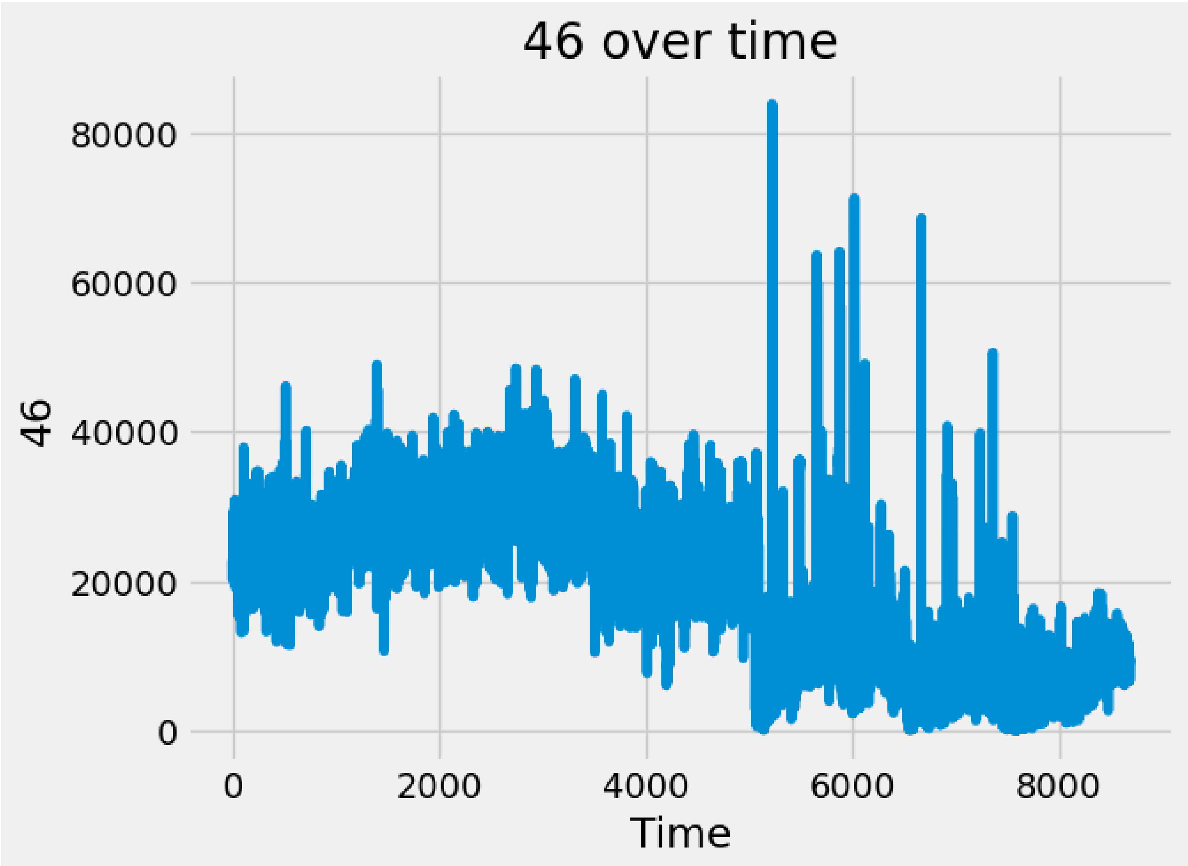

41 over time
41
Time
120000
100000
80000
60000
40000
20000
0
0
2000
4000
6000
8000

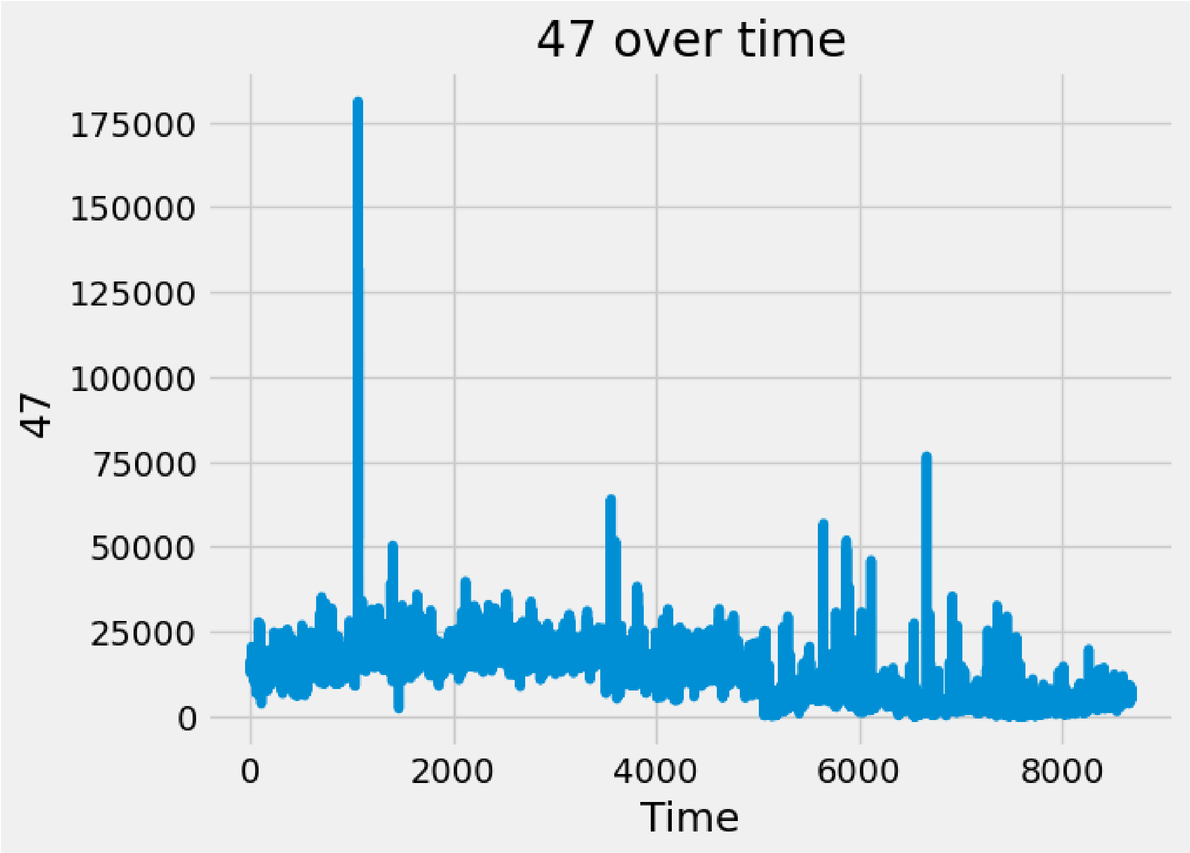

42 over time
42
Time
120000
100000
80000
60000
40000
20000
0
0
2000
4000
6000
8000

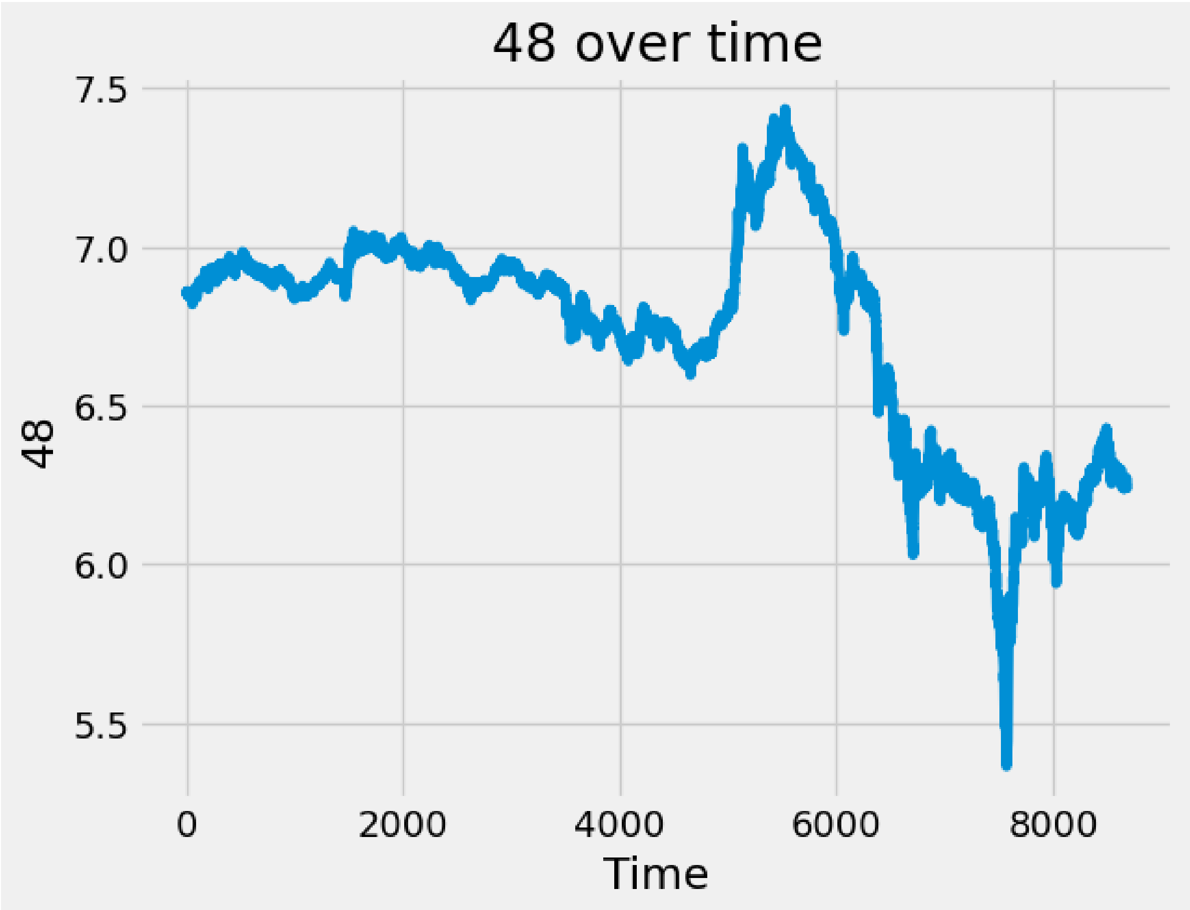

43 over time
160000
140000
120000
100000
80000
60000
40000
20000
0
43
0
2000
4000
6000
8000
Time

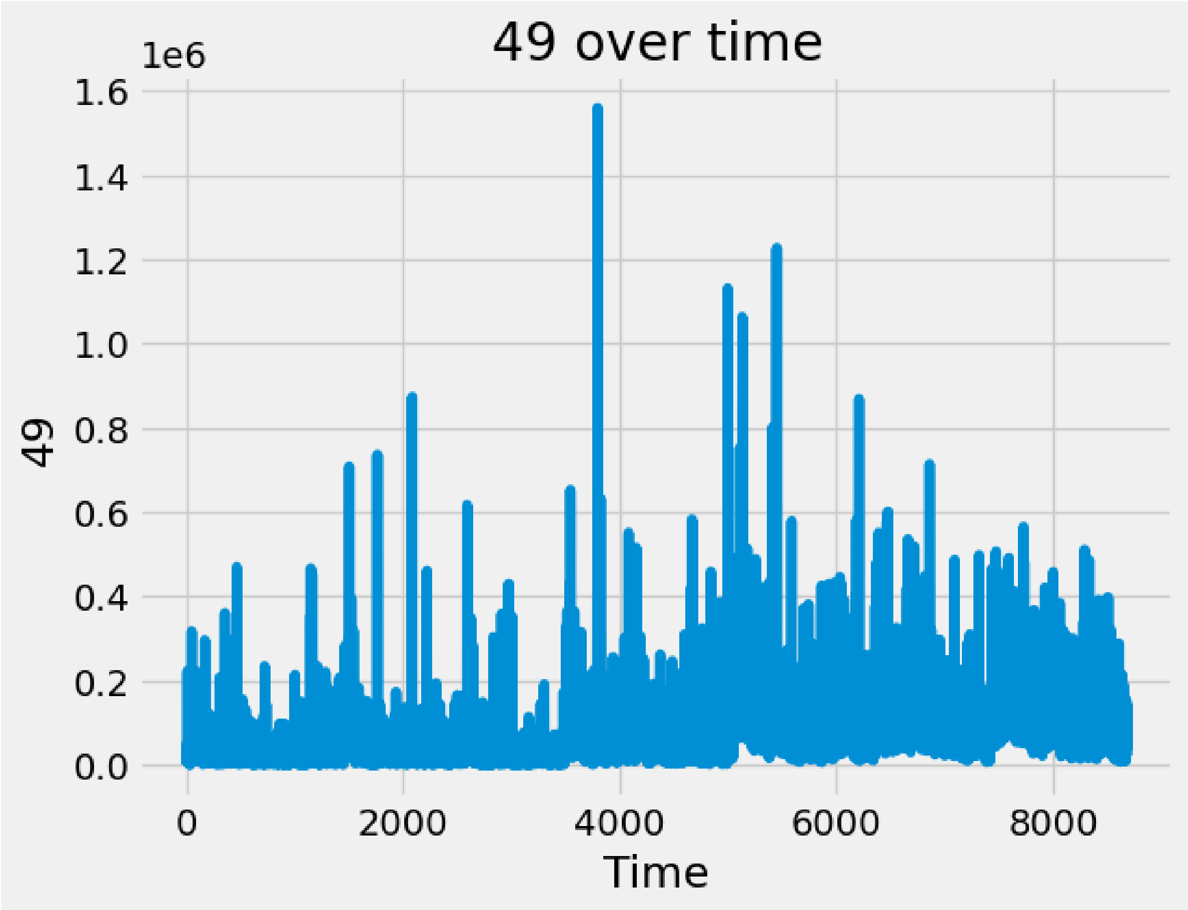

44 over time
175000
150000
125000
100000
75000
50000
25000
0
44
0
2000
4000
6000
8000
Time


45 over time
45
120000
100000
80000
60000
40000
20000
0
0
2000
4000
6000
8000
Time

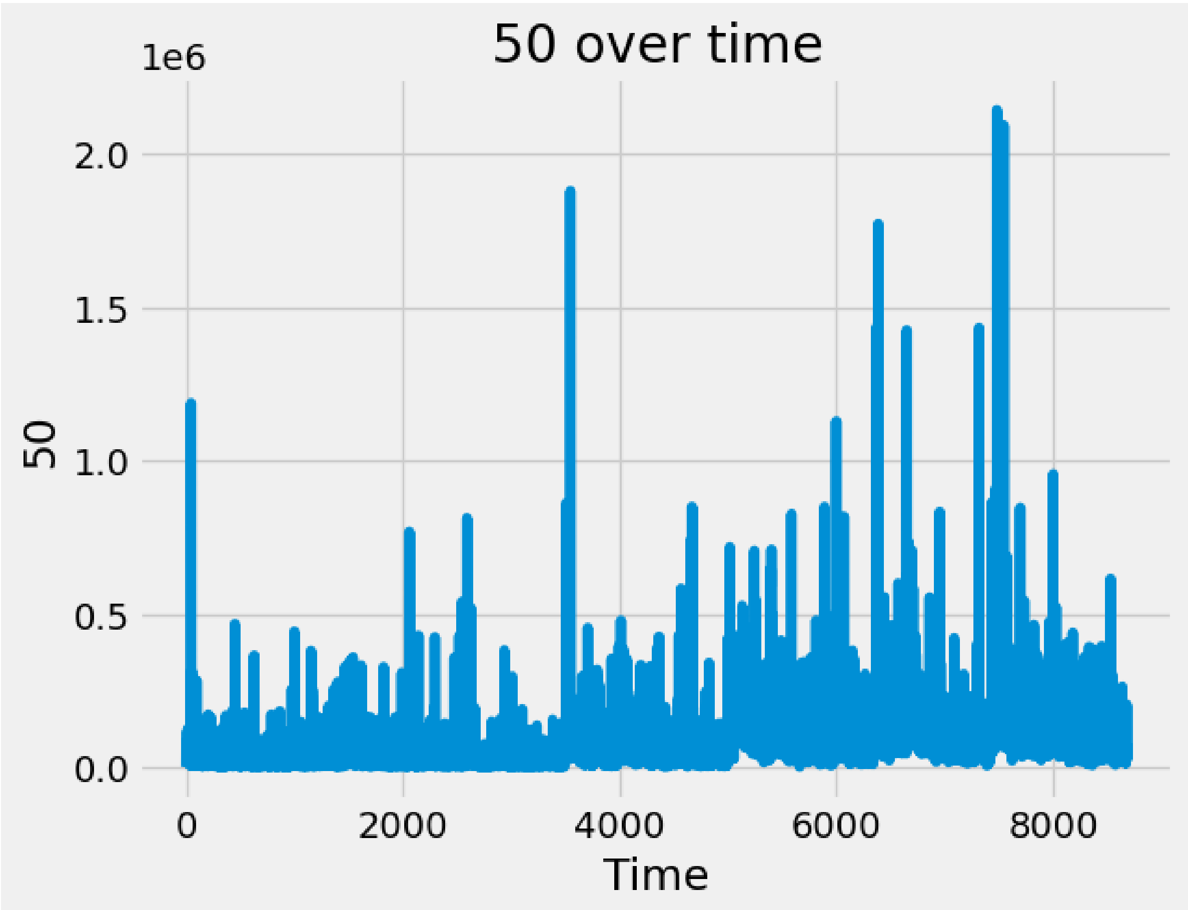

46 over time
46
80000
60000
40000
20000
0
0
2000
4000
6000
8000
Time


47 over time
175000
150000
125000
100000
75000
50000
25000
0
47
0
2000
4000
6000
8000
Time

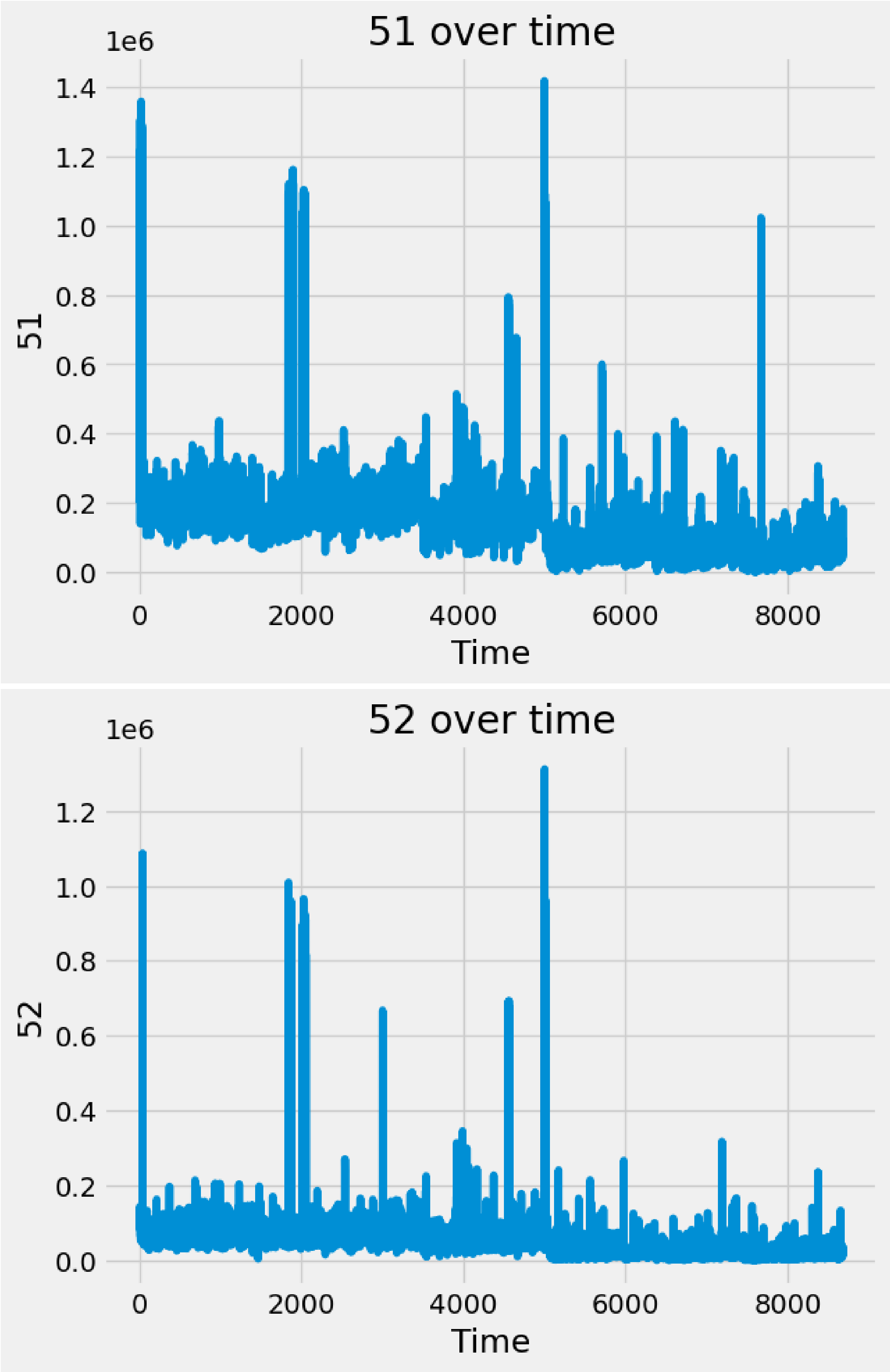

48 over time
7.5
7.0
6.5
6.0
5.5
48
0
2000
4000
6000
8000
Time

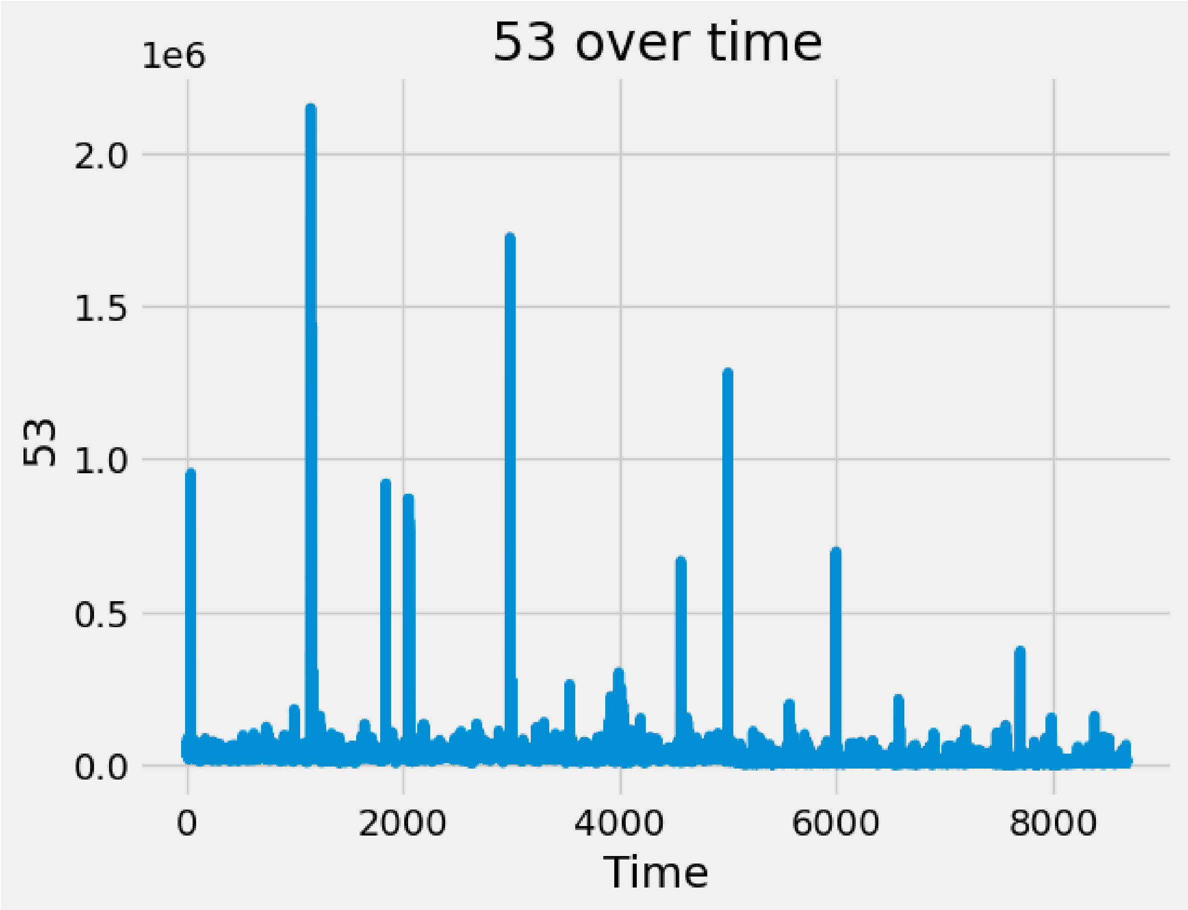


49 over time
1e6
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
49
0
2000
4000
6000
8000
Time

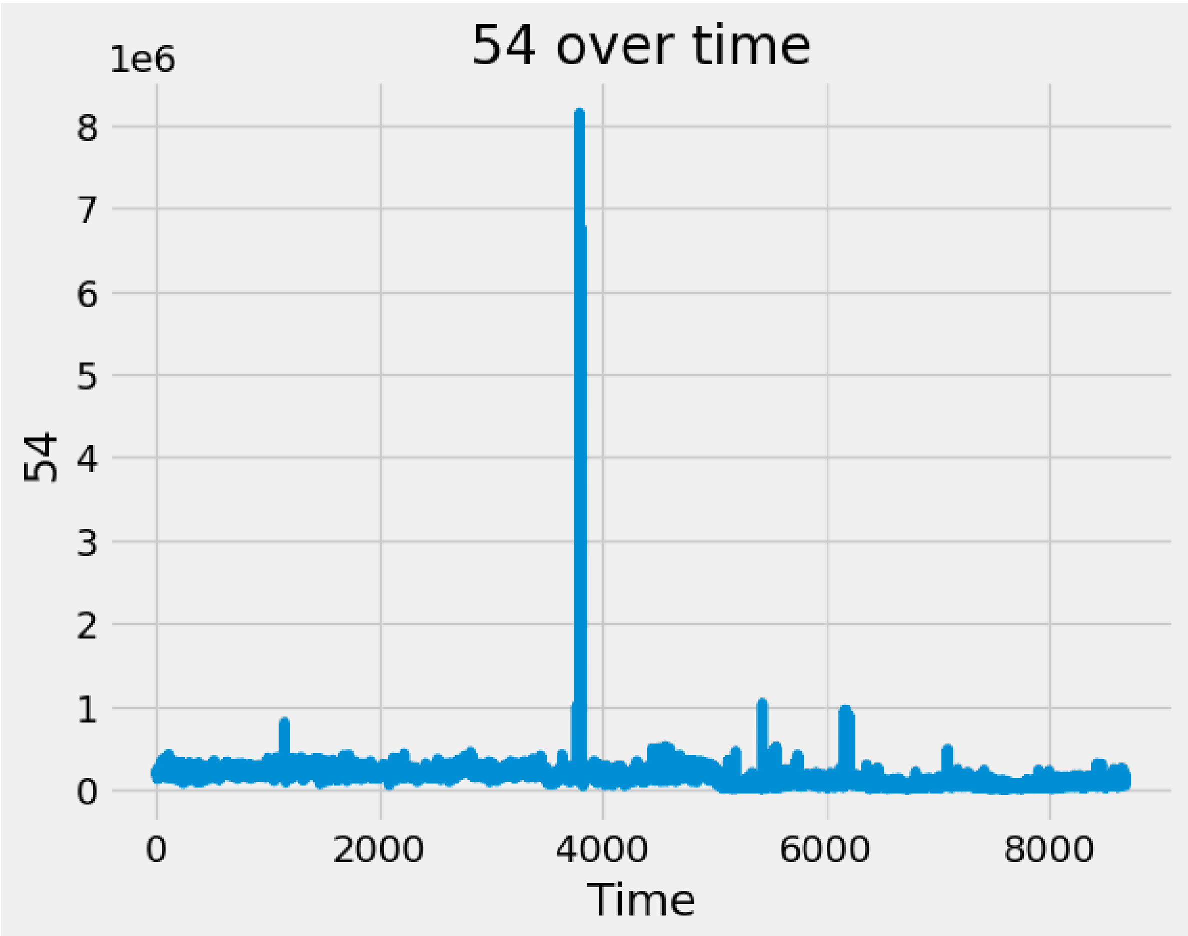


50 over time
1e6
2.0
1.5
1.0
0.5
0.0
50
0
2000
4000
6000
8000
Time

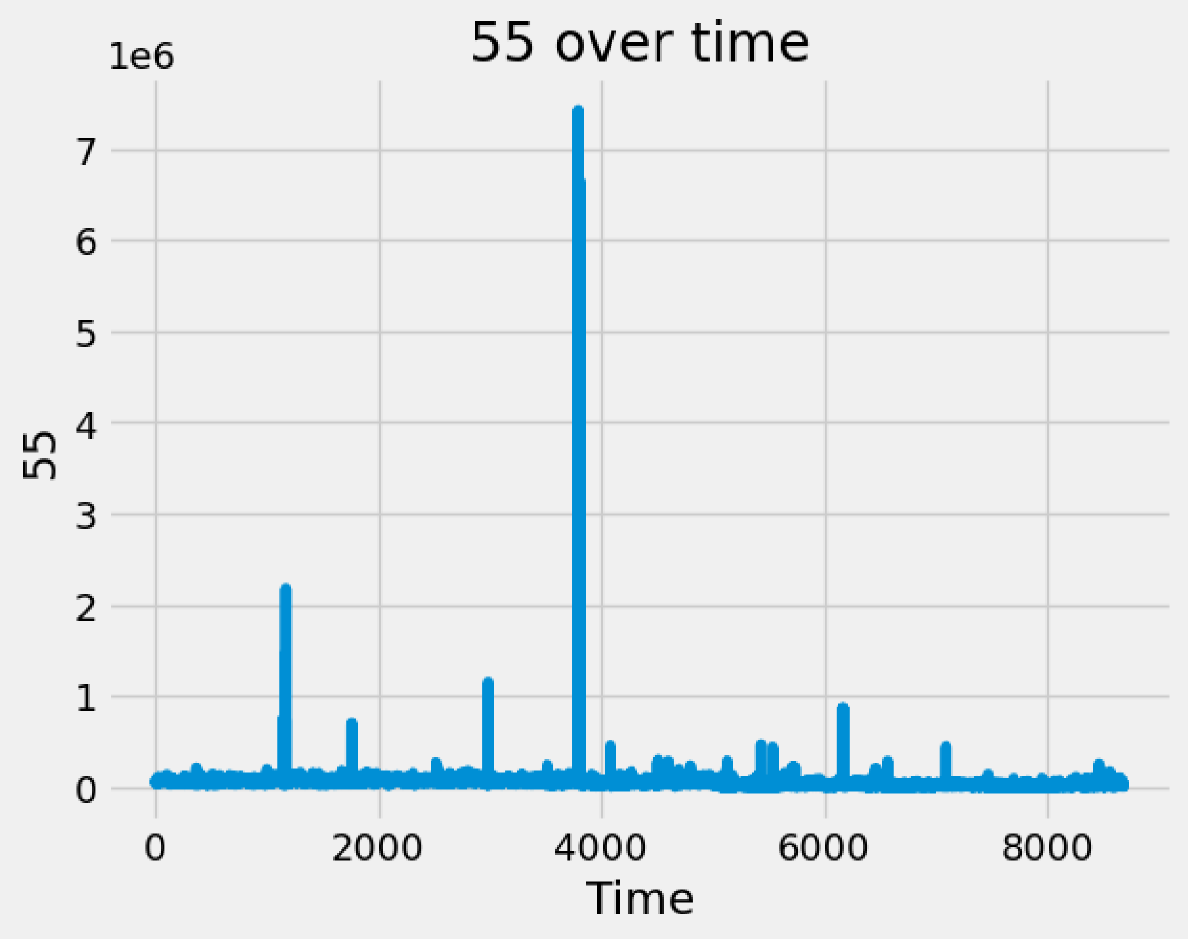

51 over time
1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
51
0
2000
4000
6000
8000
Time
52 over time
1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
52
0
2000
4000
6000
8000
Time

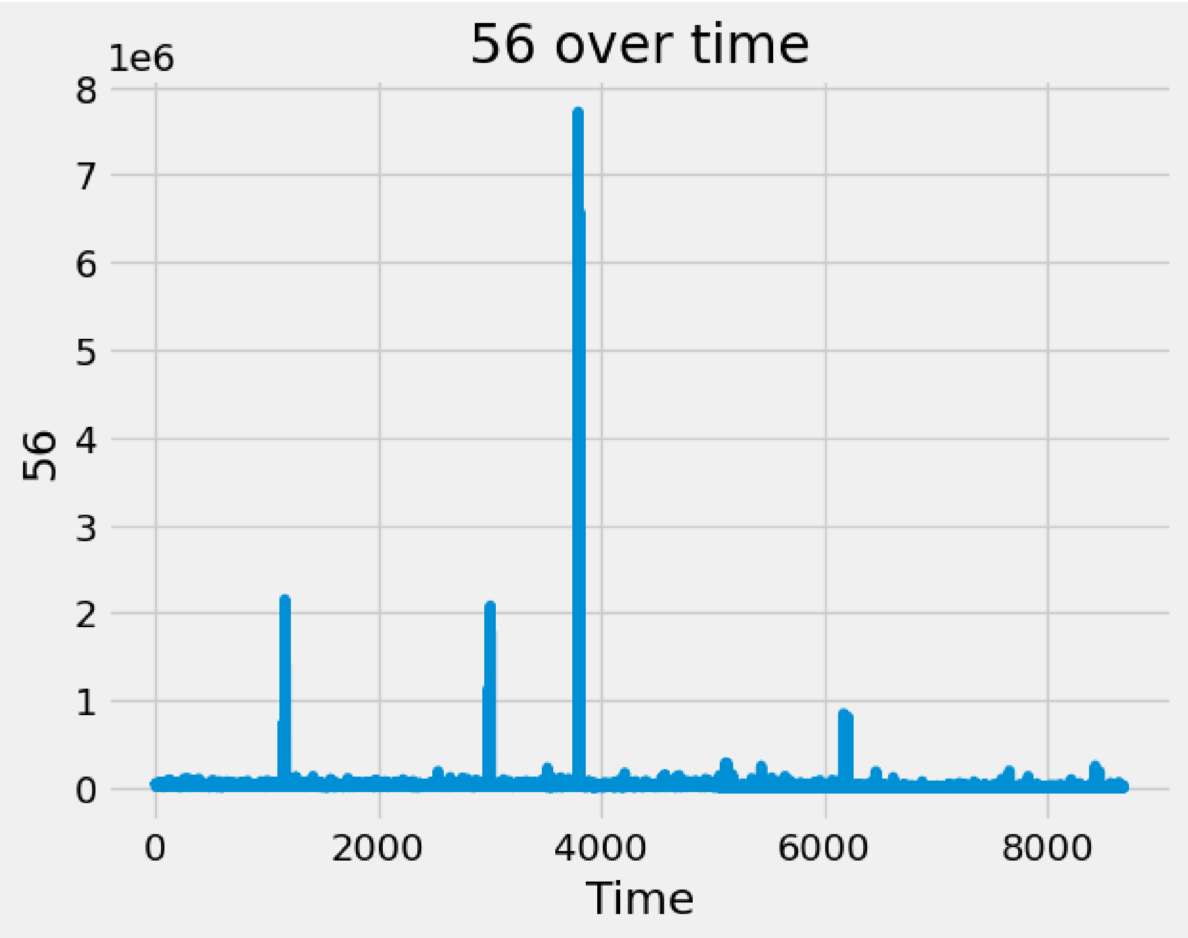


53 over time
1e6
2.0
1.5
1.0
0.5
0.0
53
0
2000
4000
6000
8000
Time



54 over time
1e6
8
7
6
5
4
3
2
1
0
54
0
2000
4000
6000
8000
Time



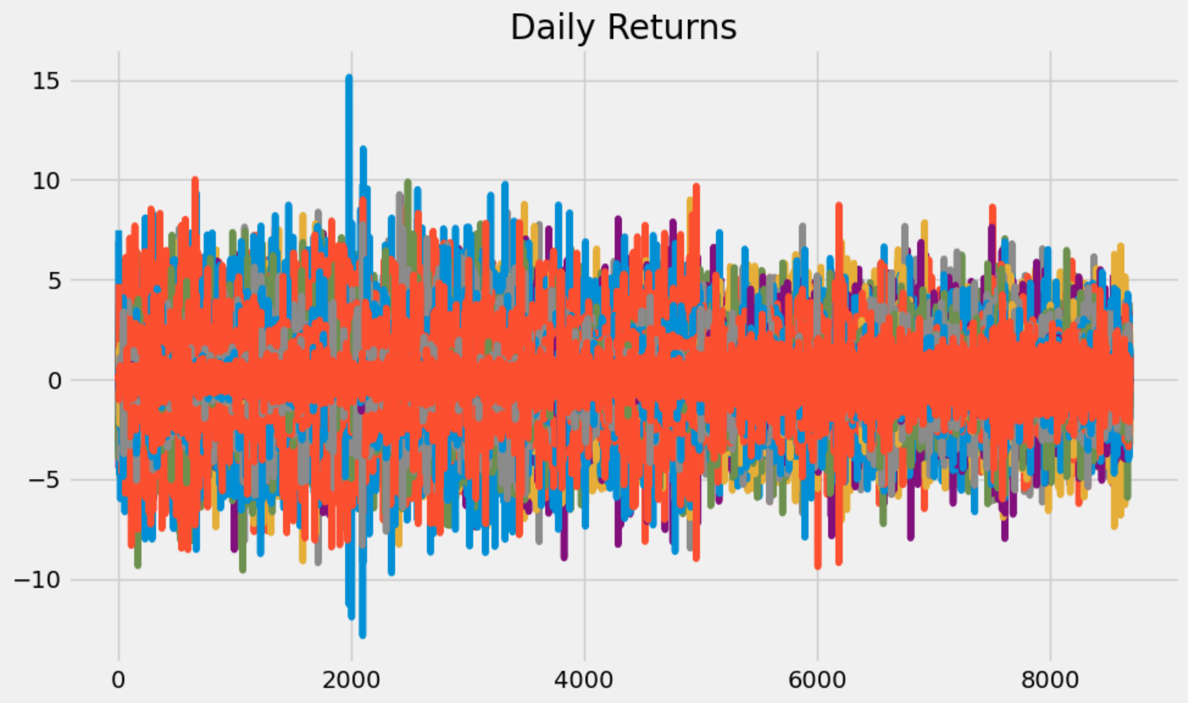


In [9]:
```
#Convert from price to return for stationary time series
daily_returns = df.apply(np.log).diff(1)
print(daily_returns)
```

```
             1         2         3         4         5         6         7  \
0          NaN       NaN       NaN       NaN       NaN       NaN       NaN
1    -0.000809 -1.320597  2.500295 -0.109236 -2.353321  1.380466 -1.079312
2     0.000108  2.427471  0.143855  0.240051  1.729944 -0.206041  0.657566
3    -0.000094 -2.039937 -2.527217 -0.593498  2.299630 -0.588164 -0.842351
4    -0.000067 -0.000021  2.457623 -0.035655 -0.284377  0.386773  1.018091
...        ...       ...       ...       ...       ...       ...       ...
8683 -0.000669 -0.112570  0.295691  0.455691  3.770461 -1.991021  2.842751
8684 -0.000167 -0.257338 -0.811593  0.295251 -2.725541  2.071490 -1.173476
8685  0.000540  0.341572 -0.080244 -0.559142 -0.364254 -1.362467  0.182436
8686  0.000456 -0.558141 -0.264928  0.159726  0.981803 -1.154374 -0.282337
8687 -0.000524 -0.071444  0.392641 -1.962891 -1.421185  0.994087 -0.578050

             8         9        10  ...        47        48        49  \
0          NaN       NaN       NaN  ...       NaN       NaN       NaN
1    -0.417719  0.410406       inf  ... -0.151011 -0.000874  6.930495
2     0.356071  0.726417  0.000000  ... -0.047459  0.000087 -0.485136
3    -0.104361  0.264002  0.000000  ...  0.126364  0.000000  0.155253
4    -0.996353 -1.292897  0.000000  ...  0.014109 -0.000175 -0.605348
...        ...       ...       ...  ...       ...       ...       ...
8683  2.424335  2.074412  0.000000  ... -0.131352 -0.001535 -0.940842
8684 -1.719953 -0.929053  0.000000  ...  0.102064 -0.001249  0.134029
8685 -0.326246 -1.281060 -0.693147  ... -0.110634  0.000192  0.071244
8686  1.259446  0.596595  0.000000  ...  0.106063 -0.000144 -0.329537
8687 -0.063737 -0.773150  0.693147  ...  0.071328  0.000432 -0.831546

            50        51        52        53        54        55        56
0          NaN       NaN       NaN       NaN       NaN       NaN       NaN
1     4.777988  0.135562 -0.011419  0.078858 -0.428625 -0.431409  0.285850
2    -0.139767  0.069952  0.151293 -0.050287  0.289095  0.500303 -0.038758
3    -0.326150 -0.042259 -0.035262 -0.206438  0.050230  0.012914  0.222378
4     0.272093 -0.002165  0.073815  0.217587 -0.100392 -0.015781 -1.027443
...        ...       ...       ...       ...       ...       ...       ...
8683  1.795921  0.849530 -1.190369  0.750469 -0.760805 -0.509473  0.162802
8684 -1.226597 -0.566090  1.109213  0.616866  1.161600  1.701020  0.555011
8685 -0.795259 -0.624621 -0.010354 -1.196883 -0.232636  0.000317 -0.734031
8686  0.610491 -0.062638  0.038649 -0.443751  0.006112  0.178336 -0.406945
8687 -0.476913  0.284154 -0.234183  0.175666 -0.101750 -0.015864  1.120515

[8688 rows x 56 columns]
```

In [10]:

```python
#Plot returns clusters of all assets
plt.style.use('fivethirtyeight')
daily_returns.plot(legend=0, figsize=(10,6), grid=True, title='Daily Returns')
plt.tight_layout()
```



In [11]:
```
#Forward fill Nan values
daily_returns = daily_returns.fillna(method='ffill')
daily_returns
```

Out[11]:

	1	2	3	4	5	6	7	8	
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
1	-0.000809	-1.320597	2.500295	-0.109236	-2.353321	1.380466	-1.079312	-0.417719	0.410
2	0.000108	2.427471	0.143855	0.240051	1.729944	-0.206041	0.657566	0.356071	0.726
3	-0.000094	-2.039937	-2.527217	-0.593498	2.299630	-0.588164	-0.842351	-0.104361	0.264
4	-0.000067	-0.000021	2.457623	-0.035655	-0.284377	0.386773	1.018091	-0.996353	-1.292
...	...	...	...	...	...	...	...	...	
8683	-0.000669	-0.112570	0.295691	0.455691	3.770461	-1.991021	2.842751	2.424335	2.074
8684	-0.000167	-0.257338	-0.811593	0.295251	-2.725541	2.071490	-1.173476	-1.719953	-0.929
8685	0.000540	0.341572	-0.080244	-0.559142	-0.364254	-1.362467	0.182436	-0.326246	-1.281
8686	0.000456	-0.558141	-0.264928	0.159726	0.981803	-1.154374	-0.282337	1.259446	0.596
8687	-0.000524	-0.071444	0.392641	-1.962891	-1.421185	0.994087	-0.578050	-0.063737	-0.773

8688 rows × 56 columns

In [12]:
```
#Replace infinity with Nan
daily_returns=np.array(daily_returns)
daily_returns[~np.isfinite(daily_returns)] = np.nan
```

In [13]:
```
#Remove Nan values
daily_returns= daily_returns[~np.isnan(daily_returns)]
```

In [14]:
```
#Normality test
from scipy.stats import shapiro
# Perform Shapiro-Wilk normality test
stat, p = shapiro(daily_returns)
```

```
# Interpret results
alpha = 0.05
if p > alpha:
    print('Sample looks Gaussian (fail to reject H0)')
else:
    print('Sample does not look Gaussian (reject H0)')
```

```
Sample does not look Gaussian (reject H0)
C:\Users\sigma\anaconda3\lib\site-packages\scipy\stats\_morestats.py:1800: UserWar
ning: p-value may not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

In [16]:
```
#Reshape 1D array into 2D array
daily_returns= daily_returns.reshape(-1, 1)
```

In [17]:
```
#standarized data
standard_returns=(daily_returns-daily_returns.mean())/daily_returns.std()
```

In [18]:
```
#Principal Components Analsyis (PCA) for dimension reduction
#Use first differenced data as the scaling factor for PCA due
#to its simplicity computationally.

diff_ = df.diff(-1)
diff_.dropna(inplace=True)
diff_.tail()
```

Out[18]:

	1	2	3	4	5	6	7	8	9	10	...	
8682	0.0440	59.219	-148.172	-24.581	-148.697	48.997	-93.260	-43.279	-73.879	0.0	...	8
8683	0.0110	112.768	321.746	-23.068	142.233	-53.752	68.401	38.980	51.125	0.0	...	-6
8684	-0.0355	-156.447	19.824	38.646	3.044	45.755	-6.130	2.367	24.101	1.0	...	6
8685	-0.0300	231.268	55.221	-8.934	-11.563	10.782	9.042	-15.484	-7.562	0.0	...	-6
8686	0.0345	21.335	-87.543	52.019	14.026	-8.450	12.168	1.335	9.062	-1.0	...	-4

5 rows × 56 columns

In [19]:
```
#Covariance
cov_= pd.DataFrame(np.cov(diff_, rowvar=False)*252/10000, columns=diff_.columns, i
cov_.style.format("{:.4%}")
```

Out[19]:

	1	2	3	4	5	
1	0.0155%	34.1823%	-67.9606%	-0.3191%	-0.4149%	-1.54
2	34.1823%	2069823.4441%	783751.1190%	3695.4401%	2379.4432%	6263.55
3	-67.9606%	783751.1190%	2900190.5881%	-1876.3356%	-430.8547%	9788.19
4	-0.3191%	3695.4401%	-1876.3356%	115143.2334%	-4400.1349%	-5730.45
5	-0.4149%	2379.4432%	-430.8547%	-4400.1349%	31105.8823%	-573.93
6	-1.5492%	6263.5507%	9788.1959%	-5730.4570%	-573.9358%	96483.16
7	0.0690%	-9790.1282%	8313.1914%	-124.3066%	567.9897%	-297.56
8	0.6957%	-12058.4327%	3583.5424%	238.5467%	470.1012%	-733.86
9	0.9268%	-3389.7929%	-4401.7551%	-310.1542%	-1095.0435%	-7268.20
10	0.0014%	-3.5448%	17.4908%	4.9345%	-0.2064%	-4.44
11	0.0005%	0.8517%	-1.8192%	-0.0324%	-0.0180%	-0.05
12	3.3288%	75459.3077%	33467.8788%	-801.7043%	529.1395%	-178.90
13	-6.1872%	44742.5847%	133772.6116%	-854.9656%	1763.4925%	2821.50
14	-0.1437%	-331.9945%	1844.9537%	587.3368%	99.4608%	196.89
15	-0.0542%	2335.6426%	1410.8487%	-35.7599%	-14.3482%	337.65
16	0.0359%	-421.0942%	-1846.6037%	35.5357%	78.4967%	1236.10
17	0.1711%	723.6582%	-977.4490%	86.3764%	12.4115%	-129.08
18	0.0173%	-217.5724%	718.3888%	-48.2741%	-31.0347%	-162.40
19	-0.0566%	-838.8068%	61.8013%	-7.9433%	-49.8361%	-679.29
20	-0.0001%	8.4702%	8.6838%	1.1061%	-1.8748%	-0.41
21	-1.2125%	27857.3909%	18763.5839%	6689.9825%	-2753.0420%	-2946.36
22	-5.0468%	-12180.6191%	47344.7647%	-7267.4457%	1051.3618%	-9229.12
23	-0.2837%	-3040.2716%	8035.5374%	1098.4079%	-601.0969%	-2698.45
24	-0.9953%	-20552.8459%	9037.4472%	-912.1068%	600.2845%	-95.57
25	0.0377%	6571.3426%	10620.5945%	-1849.3096%	-1505.0058%	1863.18
26	-0.6647%	-11067.1568%	-12353.2429%	-1665.4237%	218.6657%	1270.69
27	0.3827%	2244.9543%	-3560.3280%	-269.0303%	308.1480%	623.51
28	-0.1028%	-11490.8318%	3209.8442%	-1327.3516%	-859.9411%	-9313.16
29	-0.0005%	42.3510%	43.4192%	5.5304%	-9.3742%	-2.05
30	0.0003%	0.3480%	-1.1175%	-0.0162%	-0.0032%	-0.05
31	4520.2763%	93156529.0312%	26590900.5748%	93935.0393%	326732.1873%	2099536.72
32	-7703.1591%	35405144.8942%	134849283.6966%	157104.2040%	232783.3247%	6501363.96
33	-580.7750%	-35871306.6017%	-42078389.3799%	-3720605.3066%	-139317.3818%	3182038.41
34	-2874.5581%	-47525273.4070%	-34588088.5659%	1684880.3123%	-1115409.3558%	1176336.76
35	-621.0689%	671669.8531%	-11318388.1589%	853017.0086%	777624.7784%	-195722.34
36	1023.5294%	-59573692.6273%	-51496885.9938%	-3076307.4828%	-4458330.4488%	-8869466.22

	1	2	3	4	5	
37	5000.7367%	-1104610.4890%	-49176097.5273%	-817020.5755%	621551.6583%	56296.59
38	830.2313%	-21282341.8714%	-18504494.3367%	470599.6241%	-324795.5257%	-211283.84
39	0.0126%	16.3743%	-47.8097%	-0.7529%	0.0307%	-0.00
40	185.9299%	2378779.5442%	312472.8395%	-50075.9878%	14552.0964%	4.44
41	-193.2543%	690454.3395%	3248995.5549%	-95603.4079%	-29024.1536%	-74584.92
42	-242.5475%	-1328925.5592%	1265331.3700%	7609.7201%	5930.0541%	-67811.58
43	-68.0929%	-724960.2759%	205108.0656%	32819.0490%	25072.6144%	22591.63
44	-40.0715%	-350039.7071%	-175530.1291%	68029.9970%	-16324.0685%	-19409.51
45	126.0730%	-21590.7008%	-1265649.5420%	-80947.9648%	-23059.4762%	-19334.03
46	49.6026%	-335810.1060%	-586594.7656%	-14638.7146%	-19543.9828%	12011.74
47	19.9570%	-56925.2996%	-252200.6906%	28265.9460%	15375.0295%	56876.01
48	0.0012%	2.3562%	-4.4983%	-0.0939%	-0.0370%	0.00
49	2387.8280%	37934867.7950%	4970719.0996%	-632898.8748%	-63756.1760%	-136807.82
50	-3285.9059%	4607733.2527%	55870783.1352%	-163744.4513%	268009.5259%	200361.75
51	-696.1741%	-8126421.0327%	-2993134.6097%	-518327.7403%	88921.3183%	744236.94
52	-169.7282%	-3020808.4318%	-1886705.3478%	995266.1051%	-194490.5721%	-1127944.31
53	-264.4201%	-2620979.1214%	2303615.7824%	-312364.1802%	-15731.2888%	71352.06
54	-241.5703%	-5071814.2951%	658475.3391%	511283.7462%	-1622809.6409%	957267.14
55	727.4850%	144792.3629%	-1320861.3240%	438236.1874%	306816.1554%	-630374.31
56	409.0389%	10009409.6942%	-5378699.3002%	-36523.4121%	-298638.6733%	243443.88

In [20]:
```
#Eigen
# Perform eigen decomposition
eigenvalues, eigenvectors = np.linalg.eig(cov_)
```

In [21]:
```
# Sort values
idx = eigenvalues.argsort()[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:,idx]
```

In [22]:
```
# Format into a DataFrame
df_eigval = pd.DataFrame({"Eigenvalues": eigenvalues})

eigenvalues
```

Out[22]:
```
array([ 3.56567562e+09,  3.07437801e+09,  2.72391583e+09,  2.05584444e+09,
        1.30114594e+09,  1.10745017e+09,  1.03191146e+09,  8.69946587e+08,
        7.36333267e+08,  4.87944845e+08,  3.25293464e+08,  2.65216862e+08,
        2.16432654e+08,  1.53037505e+08,  1.14420427e+08,  6.04586634e+07,
        2.66469073e+06,  2.33428326e+06,  1.67885600e+06,  1.17405597e+06,
        1.00594322e+06,  7.99816453e+05,  6.71604108e+05,  5.82359370e+05,
        3.11733646e+04,  1.43982022e+04,  1.35456959e+04,  7.35201169e+03,
        6.64531610e+03,  3.97600747e+03,  2.54851774e+03,  2.20084596e+03,
        1.95423236e+03,  1.41693860e+03,  1.21998798e+03,  1.16061245e+03,
        9.33412165e+02,  9.14230023e+02,  9.08112401e+02,  3.73032475e+02,
        3.02383357e+02,  1.77645728e+02,  2.16936638e+01,  1.60386635e+01,
        1.34884265e+01,  1.04600506e+01,  1.01342176e+01,  5.63738086e+00,
        4.68842167e-02,  8.17508593e-03,  2.90387050e-04,  7.32873787e-05,
        9.02902092e-07,  7.41426687e-08,  3.81111535e-08, -2.76721811e-11])
```

In [23]:
```python
#Explained variance
# Work out explained proportion
df_eigval["Explained proportion"] = df_eigval["Eigenvalues"] / np.sum(df_eigval["E
df_eigval = df_eigval[:10]
df_eigval
```

Out[23]:

	Eigenvalues	Explained proportion
0	3.565676e+09	0.196994
1	3.074378e+09	0.169851
2	2.723916e+09	0.150489
3	2.055844e+09	0.113580
4	1.301146e+09	0.071885
5	1.107450e+09	0.061184
6	1.031911e+09	0.057010
7	8.699466e+08	0.048062
8	7.363333e+08	0.040680
9	4.879448e+08	0.026958

In [24]:
```python
#Format as percentage
df_eigval.style.format({"Explained proportion": "{:.2%}"})
```

Out[24]:

	Eigenvalues	Explained proportion
0	3565675617.799420	19.70%
1	3074378012.781180	16.99%
2	2723915829.827961	15.05%
3	2055844443.934842	11.36%
4	1301145940.448869	7.19%
5	1107450173.655597	6.12%
6	1031911457.763237	5.70%
7	869946586.862567	4.81%
8	736333266.594387	4.07%
9	487944845.288714	2.70%

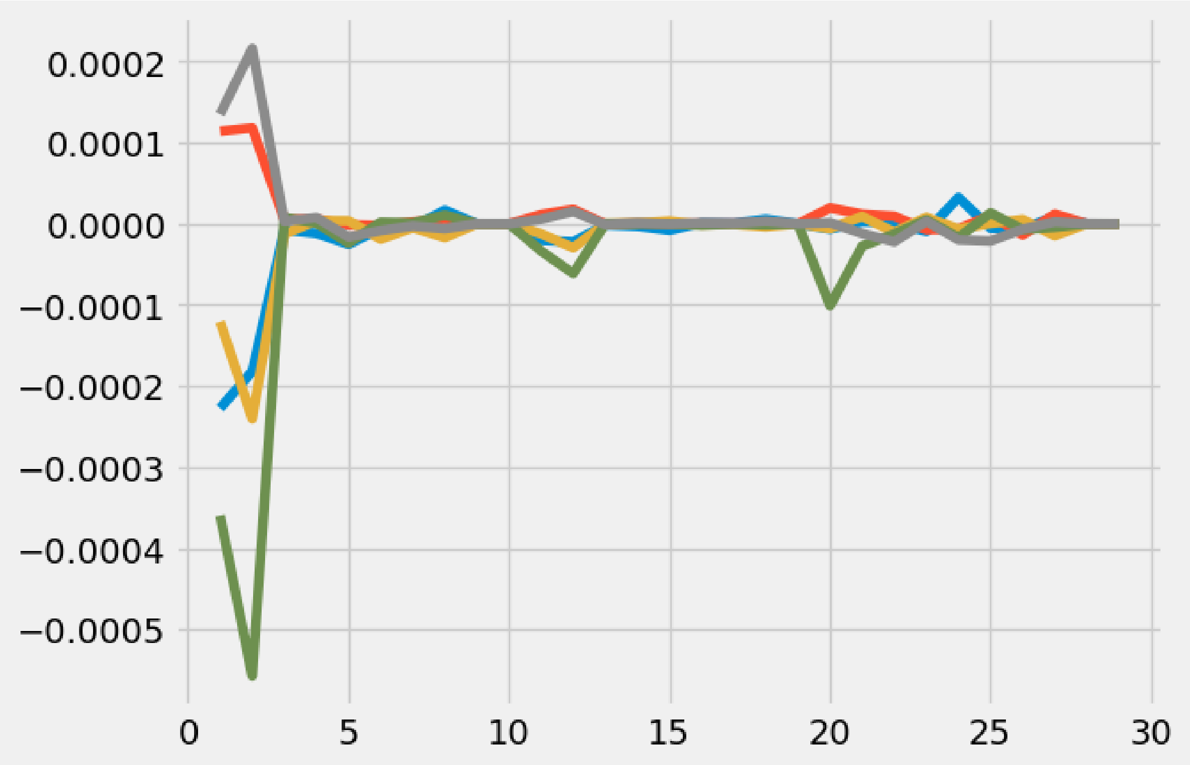
In [25]:
```
#Based on the ranking of eigenvalues and explained proportion,
#I subsume the top 5 components into a dataframe.
pcadf = pd.DataFrame(eigenvectors[:,0:5], columns=['PC1','PC2','PC3', 'PC4', 'PC5'
pcadf[:10]
```

Out[25]:

	PC1	PC2	PC3	PC4	PC5
0	-1.228251e-09	5.011755e-10	9.404022e-09	1.573288e-08	-4.547522e-09
1	-2.272843e-04	1.141434e-04	-1.206761e-04	-3.596104e-04	1.350565e-04
2	-1.817419e-04	1.182757e-04	-2.401763e-04	-5.573426e-04	2.161227e-04
3	-7.551782e-06	5.325700e-06	-1.267283e-05	7.738908e-06	2.996736e-06
4	-1.181960e-05	5.983324e-06	3.896880e-06	8.959288e-07	7.648094e-06
5	-2.518283e-05	-1.306735e-06	3.731348e-06	-2.384105e-05	-1.632771e-05
6	-6.077556e-06	-2.140034e-06	-1.907157e-05	1.864047e-06	-8.372842e-06
7	-3.911336e-06	2.310614e-06	-4.788029e-06	1.142855e-06	-2.633991e-06
8	1.667558e-05	5.979013e-06	-1.704649e-05	1.070744e-05	-5.935757e-06
9	3.001291e-08	-3.142292e-10	-2.942342e-08	1.179973e-08	4.619046e-08

In [26]:
```
#Convergence of eivenvectors
plt.plot(pcadf[1:30])
```

Out[26]:
```
[<matplotlib.lines.Line2D at 0x23d5414eca0>,
 <matplotlib.lines.Line2D at 0x23d5414e220>,
 <matplotlib.lines.Line2D at 0x23d5414ef10>,
 <matplotlib.lines.Line2D at 0x23d5414e670>,
 <matplotlib.lines.Line2D at 0x23d5414e820>]
```



In [27]:
```
#Conclusion from PCA
#We can attribute the first five principal components to the following:
#1. monetary policy shock 2. inflation shock
#3. Changes in the curvature of yield curve
#4. Geopolitical event 5. Shift in demand for technology
```

In [28]:

```
#Mean reversion with half-life estimation for pair trading between two assets

def estimate_half_life(spread):
    x=spread.shift().iloc[1:].to_frame().assign(const=1)
    y=spread.diff().iloc[1:]
    beta=(np.linalg.inv(x.T@x)@x.T@y).iloc[0]
    halflife=int(round(-np.log(2)/beta,0))
    return max(halflife,1)
```

In [29]:

```
#Best pair for convergence pair trading
from scipy.stats import pearsonr
# Find the best pair simply by looking at correlations
corr_matrix = diff_.corr()

# Find the pair with the highest correlation coefficient
pairs = [(corr_matrix.iloc[i, j], corr_matrix.columns[i], corr_matrix.columns[j]) f
best_pair = max(pairs)

# Output the best pair
print('Best pair for pair trading:', best_pair[1], 'and', best_pair[2])
```

```
Best pair for pair trading: 20 and 29
```

In [30]:

```
#Pick the suggested best two (asset 20 & 29) for a pair
#Then calculate the half life (mean reversion speed)
a=df.iloc[7:1007,19]
a
b=df.iloc[7:1007,28]
b
spread=a-b
estimate_half_life(spread)
```

Out[30]:

```
1
```

In [28]:

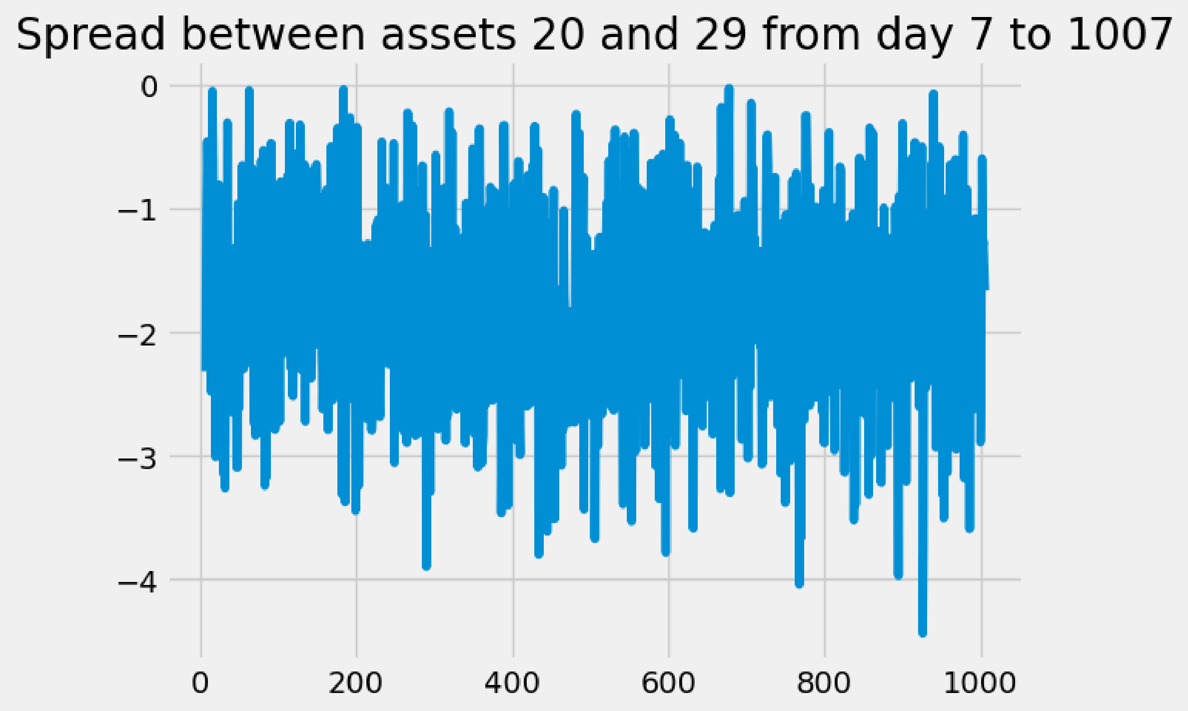
```
#Therefore, it takes 1*2 = 2 days for the pair spread to converge to the long run r
```

In [31]:

```
plt.plot(spread)
plt.title("Spread between assets 20 and 29 from day 7 to 1007")
```

Out[31]:

```
Text(0.5, 1.0, 'Spread between assets 20 and 29 from day 7 to 1007')
```



In [45]:

```
#Univariate forecast
#Let's say we want to forecast the spread that
#is more suitable for divergence
#trading
# Output the best pair for divergence trading
print('Best pair for divergence trading:', best_pair_dir[1], 'and', best_pair_dir[
#Best pair for divergence trading: 54 and 55

c=df.iloc[7:1007,53]
d=df.iloc[7:1007,54]
spread_divergence=c-d
```

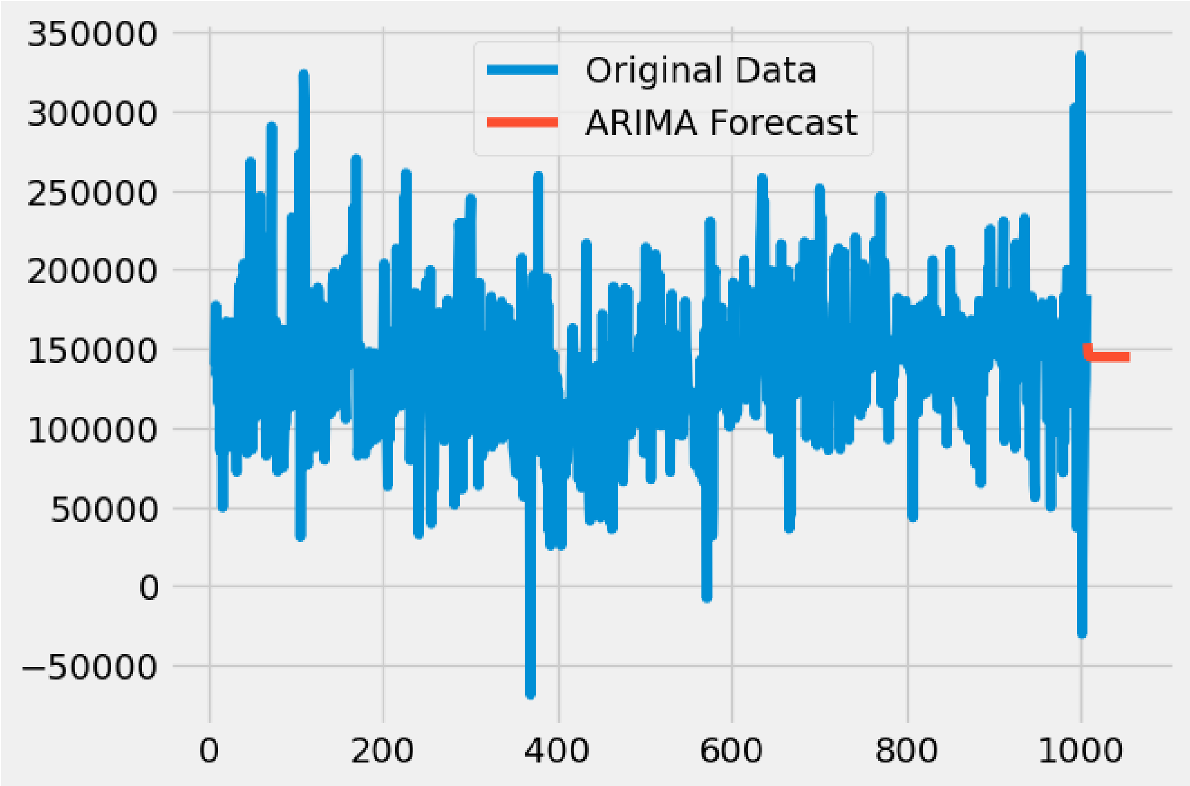
```
Best pair for divergence trading: 54 and 55
```

In [46]:

```
# Fit an ARIMA model to the data
from statsmodels.tsa.arima.model import ARIMA
ARIMAmodel = ARIMA(spread_divergence, order=(1, 1, 1))
ARIMAmodel_fit = ARIMAmodel.fit()
```

In [47]:

```
# Make a forecast for the next 50 period
ARIMAforecast = ARIMAmodel_fit.forecast(50)
ARIMAforecast
plt.plot(spread_divergence, label='Original Data')
plt.plot(ARIMAforecast, label='ARIMA Forecast')
plt.legend()
plt.show()
```

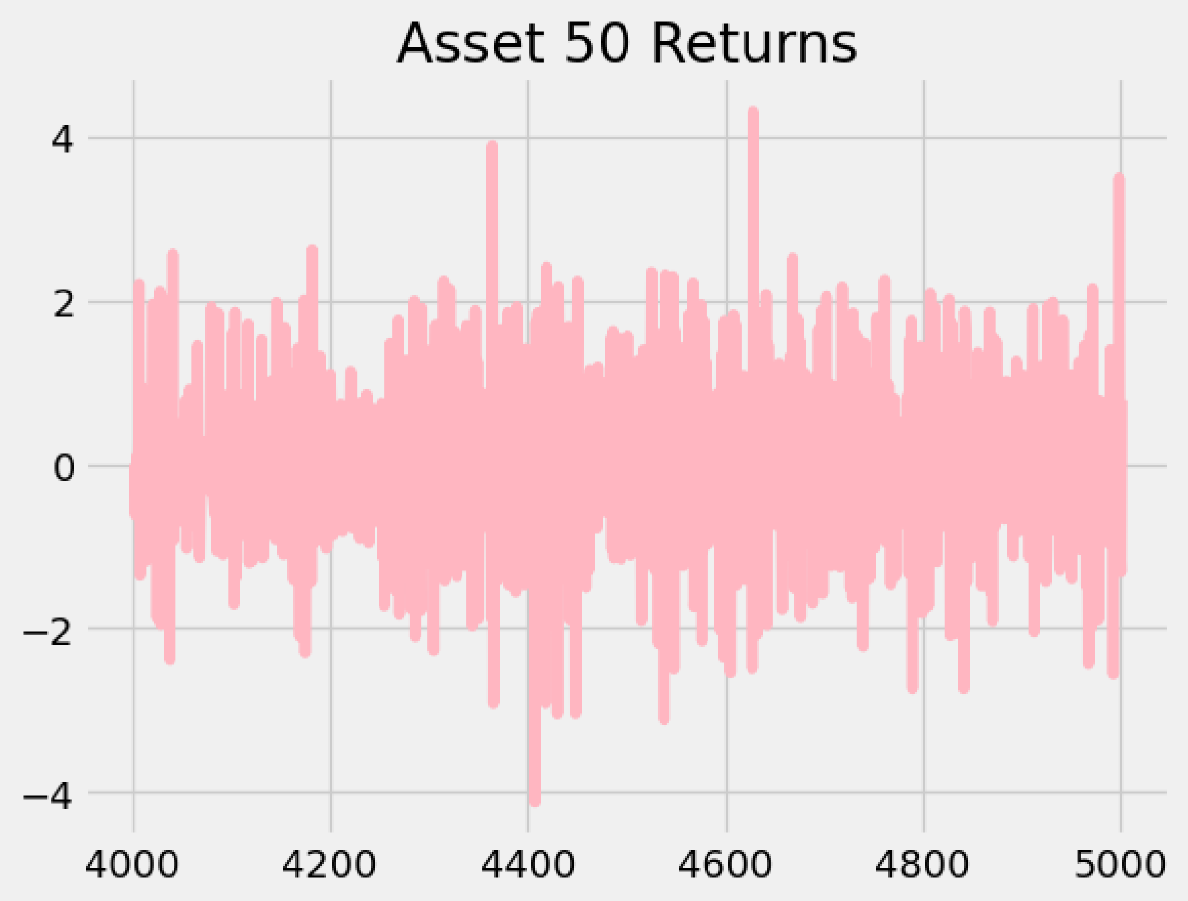


In [39]:
```
#Volatility analysis
# Randomly select 1 asset from the DataFrame for volatility analysis
import random
random_asset = df.columns[random.randint(0, len(df.columns)-1)]
print(random_asset)
from arch import arch_model
```

1

In [40]:
```
#Assume it randomly selects asset 50
A50=df.iloc[4002:5002,49]

A50_returns = np.log(A50).diff().fillna(0)
```

In [41]:
```
# Visualize the selected asset's daily returns
plt.plot(A50_returns, color='lightpink')
plt.title('Asset 50 Returns')
plt.grid(True)
```



In [35]:
```
#Skew and kurtosis
from scipy.stats import skew, kurtosis
skewness = skew(A50_returns)
kurt = kurtosis(A50_returns)
print(skewness)
print(kurt)
#Interestingly, kurtosis of the selected sample period and asset
#does not appear to be leptokurtic.
```

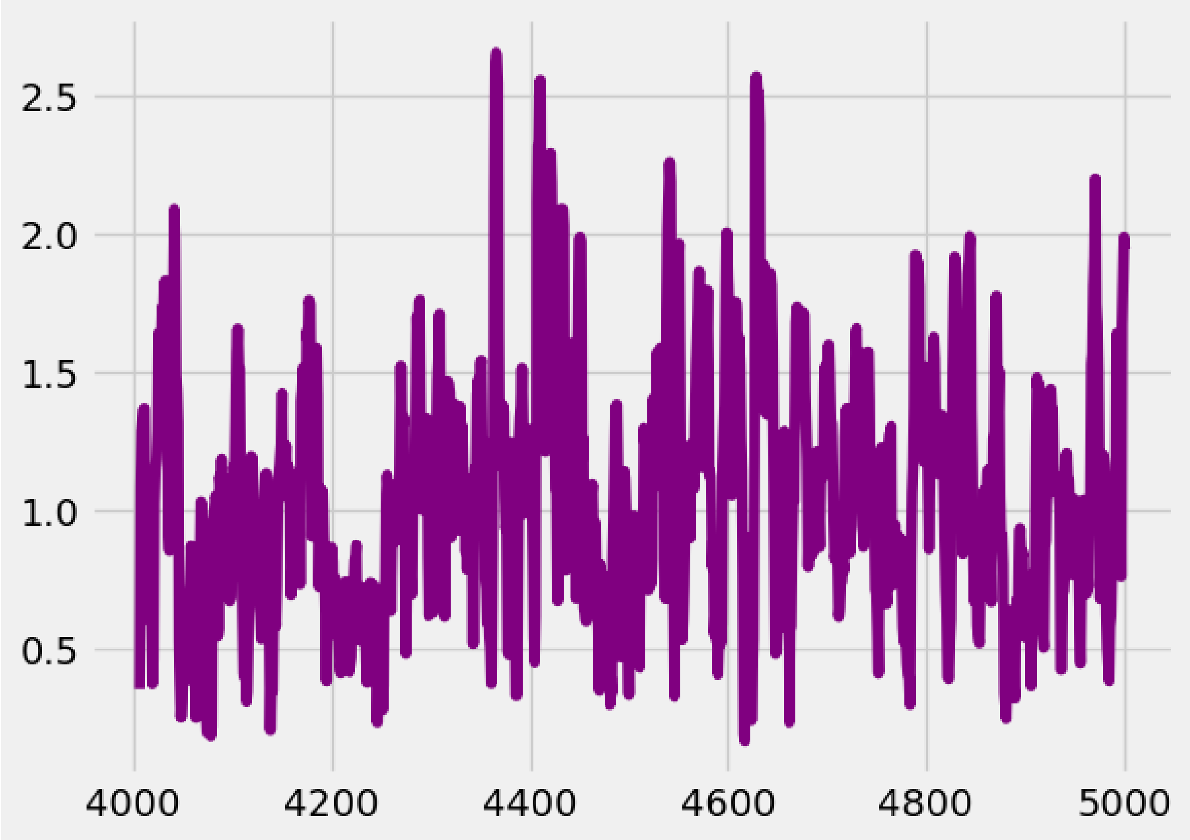
```
0.06771824868900031
0.39029090447849324
```

In [36]:
```
# Perform Shapiro-Wilk normality test for A50 returns
stat, p = shapiro(A50_returns)

# Interpret results
alpha = 0.05
if p > alpha:
    print('Sample looks Gaussian (fail to reject H0)')
else:
    print('Sample does not look Gaussian (reject H0)')
```

```
Sample does not look Gaussian (reject H0)
```

In [37]:
```
#5-Day Historical volatility of the selected asset
HV_5D=A50_returns.rolling(5).std()
HV_5D
plt.plot(HV_5D, color='purple')
```

Out[37]:
```
[<matplotlib.lines.Line2D at 0x18fd0ccfee0>]
```



In [38]:
```
HV_5D_stat=HV_5D.describe()
HV_5D_stat
```

Out[38]:
```
count    996.000000
mean       1.057749
std        0.455351
min        0.166460
25%        0.720385
50%        1.029976
75%        1.343695
max        2.651230
Name: 50, dtype: float64
```

In [39]:
```
##Exponential GARCH (1,1,1) that captures asymmetric news shock
am = arch_model(A50_returns, vol="Garch", p=1, o=1, q=1, dist="Normal")
res = am.fit(update_freq=5)
vforecasts = res.forecast(reindex=False)
print(vforecasts.mean.iloc[-3:])
print(vforecasts.residual_variance.iloc[-3:])
print(vforecasts.variance.iloc[-3:])
```

```
Iteration:      5,   Func. Count:     39,   Neg. LLF: 1450.6991498494976
Iteration:     10,   Func. Count:     69,   Neg. LLF: 1450.1534425347095
Optimization terminated successfully    (Exit mode 0)
            Current function value: 1450.1534425345408
            Iterations: 10
            Function evaluations: 69
            Gradient evaluations: 10
          h.1
5001  0.00616
           h.1
5001  1.024106
           h.1
5001  1.024106
```

In [40]:
```
#Volatility forecast in the next five days
vforecasts = res.forecast(horizon=5, reindex=False)
```

```
print(vforecasts.residual_variance.iloc[-3:])
```

```
           h.1       h.2      h.3      h.4       h.5
5001  1.024106  1.084071  1.10646  1.11482  1.117941
```

In [41]:
```
#Derivatives pricing
from scipy import sparse
#pip install quantecon
import quantecon as qe
from quantecon.markov import DiscreteDP, backward_induction, sa_indices
```

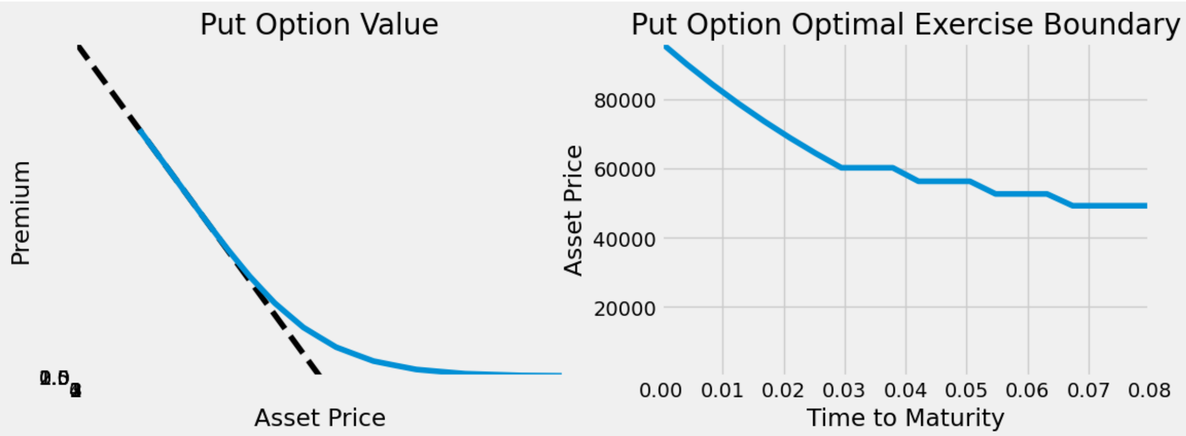
In [42]:
```
A50_price=df.iloc[6432,49]
#Options pricing model inputs
T = 0.08       # Time expiration (years)
vol = 1.06    # Annual volatility
r = 0.039     # Annual interest rate
strike = A50_price+200  # Strike price
p0 =A50_price         # Current price
N = 20       # Number of periods to expiration
```

In [43]:
```
# Time length of a period
tau = T/N
# Discount factor
beta = np.exp(-r*tau)
# Up-jump factor
u = np.exp(vol*np.sqrt(tau))
# Up-jump probability
q = 1/2 + np.sqrt(tau)*(r - (vol**2)/2)/(2*vol)
# Possible price values
ps = u**np.arange(-N, N+1) * p0
# Number of states
n = len(ps) + 1  # State n-1: "the option has been exercised"
# Number of actions
m = 2  # 0: hold, 1: exercise
# Number of feasible state-action pairs
L = n*m - 1  # At state n-1, there is only one action "do nothing"
# Arrays of state and action indices
s_indices, a_indices = sa_indices(n, m)
s_indices, a_indices = s_indices[:-1], a_indices[:-1]
# Reward vector
R = np.empty((n, m))
R[:, 0] = 0
R[:-1, 1] = strike - ps
R = R.ravel()[:-1]
```

In [44]:
```
# Transition probability array
Q = sparse.lil_matrix((L, n))
for i in range(L-1):
    if a_indices[i] == 0:
        Q[i, min(s_indices[i]+1, len(ps)-1)] = q
        Q[i, max(s_indices[i]-1, 0)] = 1 - q
    else:
        Q[i, n-1] = 1
Q[L-1, n-1] = 1
```

In [45]:
```
# Put options optimal exercise boundary
ddp = DiscreteDP(R, Q, beta, s_indices, a_indices)

vs, sigmas = backward_induction(ddp, N)

v = vs[0]
max_exercise_price = ps[sigmas[::-1].sum(-1)-1]
```

```
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

axes[0].plot([0, strike], [strike, 0], 'k--')
axes[0].plot(ps, v[:-1])
axes[0].set_xlim(0, strike*2)
axes[0].set_xticks(np.linspace(0, 4, 5, endpoint=True))
axes[0].set_ylim(0, strike)
axes[0].set_yticks(np.linspace(0, 2, 5, endpoint=True))
axes[0].set_xlabel('Asset Price')
axes[0].set_ylabel('Premium')
axes[0].set_title('Put Option Value')

axes[1].plot(np.linspace(0, T, N), max_exercise_price)
axes[1].set_xlim(0, T)
axes[1].set_ylim(1.6, strike)
axes[1].set_xlabel('Time to Maturity')
axes[1].set_ylabel('Asset Price')
axes[1].set_title('Put Option Optimal Exercise Boundary')
axes[1].tick_params(right='on')

plt.show()
```



In [46]:
```
#Implied Volatility
# Data Manipulation
import pandas as pd
from numpy import *
from datetime import timedelta
#import yfinance as yf
from tabulate import tabulate

# Math & Optimization
from scipy.stats import norm
from scipy.optimize import fsolve

# Plotting
import matplotlib.pyplot as plt
import cufflinks as cf
cf.set_config_file(offline=True)
```

In [47]:
```
class BS:

    """
    This is a class for Options contract for pricing European options on stocks/inc

    Attributes:
        spot          : int or float
        strike        : int or float
```

```
        rate          : float
        dte           : int or float [days to expiration in number of years]
        volatility    : float
        callprice     : int or float [default None]
        putprice      : int or float [default None]
    """

    def __init__(self, spot, strike, rate, dte, volatility, callprice=None, putpri

        # Spot Price
        self.spot = spot

        # Option Strike
        self.strike = strike

        # Interest Rate
        self.rate = rate

        # Days To Expiration
        self.dte = dte

        # Volatlity
        self.volatility = volatility

        # Callprice # mkt price
        self.callprice = callprice

        # Putprice # mkt price
        self.putprice = putprice

        # Utility
        self._a_ = self.volatility * self.dte**0.5

        if self.strike == 0:
            raise ZeroDivisionError('The strike price cannot be zero')
        else:
            self._d1_ = (log(self.spot / self.strike) + \
                     (self.rate + (self.volatility**2) / 2) * self.dte) / self._a_

        self._d2_ = self._d1_ - self._a_

        self._b_ = e**-(self.rate * self.dte)


        # The __dict__ attribute
        '''
        Contains all the attributes defined for the object itself. It maps the attr
        '''
        for i in ['callPrice', 'putPrice', 'callDelta', 'putDelta', 'callTheta', '
                  'callRho', 'putRho', 'vega', 'gamma', 'impvol']:
            self.__dict__[i] = None

        [self.callPrice, self.putPrice] = self._price()
        [self.callDelta, self.putDelta] = self._delta()
        [self.callTheta, self.putTheta] = self._theta()
        [self.callRho, self.putRho] = self._rho()
        self.vega = self._vega()
        self.gamma = self._gamma()
        self.impvol = self._impvol()

    # Option Price
    def _price(self):
        '''Returns the option price: [Call price, Put price]'''
```

```
        if self.volatility == 0 or self.dte == 0:
            call = maximum(0.0, self.spot - self.strike)
            put = maximum(0.0, self.strike - self.spot)
        else:
            call = self.spot * norm.cdf(self._d1_) - self.strike * e**(-self.rate 
                                                                    self.dte) *

            put = self.strike * e**(-self.rate * self.dte) * norm.cdf(-self._d2_)
                                                                     self.spot 
        return [call, put]

    # Option Delta
    def _delta(self):
        '''Returns the option delta: [Call delta, Put delta]'''

        if self.volatility == 0 or self.dte == 0:
            call = 1.0 if self.spot > self.strike else 0.0
            put = -1.0 if self.spot < self.strike else 0.0
        else:
            call = norm.cdf(self._d1_)
            put = -norm.cdf(-self._d1_)
        return [call, put]

    # Option Gamma
    def _gamma(self):
        '''Returns the option gamma'''
        return norm.pdf(self._d1_) / (self.spot * self._a_)

    # Option Vega
    def _vega(self):
        '''Returns the option vega'''
        if self.volatility == 0 or self.dte == 0:
            return 0.0
        else:
            return self.spot * norm.pdf(self._d1_) * self.dte**0.5 / 100

    # Option Theta
    def _theta(self):
        '''Returns the option theta: [Call theta, Put theta]'''
        call = -self.spot * norm.pdf(self._d1_) * self.volatility / (2 * self.dte*

        put = -self.spot * norm.pdf(self._d1_) * self.volatility / (2 * self.dte**
        return [call / 365, put / 365]

    # Option Rho
    def _rho(self):
        '''Returns the option rho: [Call rho, Put rho]'''
        call = self.strike * self.dte * self._b_ * norm.cdf(self._d2_) / 100
        put = -self.strike * self.dte * self._b_ * norm.cdf(-self._d2_) / 100

        return [call, put]

    # Option Implied Volatility
    def _impvol(self):
        '''Returns the option implied volatility'''
        if (self.callprice or self.putprice) is None:
            return self.volatility
        else:
            def f(sigma):
                option = BS(self.spot,self.strike,self.rate,self.dte,sigma)
                if self.callprice:
                    return option.callPrice - self.callprice # f(x) = BS_Call - Ma
                if self.putprice and not self.callprice:
                    return option.putPrice - self.putprice
```

```
        return maximum(1e-5, fsolve(f, 0.2)[0])
```

In [48]:
```
# Initialize option output for options pricing, greeks, and implied volatility
#from BS import BS
option = BS(p0 ,strike,r,20/250,1.405,500)

header = ['Option Price', 'Delta', 'Gamma', 'Theta', 'Vega', 'Rho', 'IV']
table = [[option.callPrice, option.callDelta, option.gamma, option.callTheta, opti

print(tabulate(table,header))
```

```
  Option Price     Delta       Gamma     Theta     Vega      Rho          IV
--------------  --------  ----------  --------  -------  -------  ---------
       15210.5  0.579779  1.0215e-05  -260.533  106.488  32.5001  0.0412378
```

In [49]:
```
# Bisection Method for estimating implied volatility

def bisection_iv(className, spot, strike, rate, dte, volatility, callprice=None, pu

    if callprice:
        price = callprice
    if putprice and not callprice:
        price = putprice

    tolerance = 1e-7

    for i in range(10000):
        mid = (high + low) / 2 # c= (a+b)/2
        if mid < tolerance:
            mid = tolerance

        if callprice:
            estimate = eval(className)(spot, strike, rate, dte, mid).callPrice # B
        if putprice:
            estimate = eval(className)(spot, strike, rate, dte, mid).putPrice

        if round(estimate,6) == price:
            break
        elif estimate > price:
            high = mid # b = c
        elif estimate < price:
            low = mid # a = c

    return mid
```

In [50]:
```
#Call price
bisection_iv('BS',p0 ,strike,r,20/250,1.405,callprice=300)
```

Out[50]:
```
0.022688569288220606
```

In [51]:
```
#Put price
bisection_iv('BS',p0 ,strike,r,20/250,1.405,putprice=550)
```

Out[51]:
```
0.05514821043561824
```

In [52]:
```
#Multivariate Time Series using the Vector Autoregression (VAR) Model
#Pick a subset of first five variables (Asset 40 to Asset 44) from
#3000th to 5999th day.
subset=df.iloc[3000:6000,39:44]
print(subset)
```

```
         40     41     42     43     44
3000    298     10  48721  34190  24696
3001    349     10  47897  33760  25636
3002   2361   5885  57314  34050  21118
3003   2763   2376  78676  34619  19442
3004   2337  12272  75821  44595  17740
...     ...    ...    ...    ...    ...
5995  31458  19273  19418   4712   1940
5996  33193  22295  14360   3940  11113
5997  18103  18687  34555  10317   2146
5998  11908  14835  19267   3777   2582
5999   9166  14318  26087  22472   9187

[3000 rows x 5 columns]
```

In [53]:

```
diffsubdata = subset.diff(-1)
diffsubdata.dropna(inplace=True)
diffsubdata.tail()
```

Out[53]:

	40	41	42	43	44
5994	-12531.0	32133.0	6997.0	3675.0	5532.0
5995	-1735.0	-3022.0	5058.0	772.0	-9173.0
5996	15090.0	3608.0	-20195.0	-6377.0	8967.0
5997	6195.0	3852.0	15288.0	6540.0	-436.0
5998	2742.0	517.0	-6820.0	-18695.0	-6605.0

In [54]:

```
#Run cointegraton test first
from statsmodels.tsa.vector_ar.vecm import coint_johansen
cointresult = coint_johansen(subset, det_order=0, k_ar_diff=1)
cv = cointresult.cvm
p_values = 1 - cv[:, 1]
print(p_values)
```

```
[-32.8777 -26.5858 -20.1314 -13.2639  -2.8415]
```

In [55]:

```
#Conclusion from cointegration
#Since p-value > five percent significance level, we can
#conlcude that this subset of five variables is not cointegrated.
#If they are not cointergrated, then we can safely use the
#vector autoregression model for multivariate time series analysis.
```

In [56]:

```
#Stationary check with unit root test
from statsmodels.tsa.stattools import adfuller

def stationarity(data, cutoff=0.05):
   if adfuller(data)[1] < cutoff:
        print('The series is stationary')
        print('p-value = ', adfuller(data)[1])
   else:
        print('The series is NOT stationary')
        print('p-value = ', adfuller(data)[1])
```

In [57]:

```
d39=df.iloc[3000:6000,39]
d40=df.iloc[3000:6000,40]
d41=df.iloc[3000:6000,41]
d42=df.iloc[3000:6000,42]
d43=df.iloc[3000:6000,43]
d44=df.iloc[3000:6000,44]
stationarity(d39)
```

```python
stationarity(d40)
stationarity(d41)
stationarity(d42)
stationarity(d43)
stationarity(d44)
#Each of them is stationary; therfore, the entire system
#is stationary
#which is required for running VAR model.
#Note: I use first differenced method to make
#each selected variable stationary
```

```
The series is stationary
p-value =  4.8908513092332815e-05
The series is stationary
p-value =  0.00012369491911959032
The series is stationary
p-value =  0.006861808790367721
The series is stationary
p-value =  0.0034678415210305583
The series is stationary
p-value =  0.011222805234577833
The series is stationary
p-value =  0.003912305672655036
```

In [58]:
```python
model= VAR(diffsubdata)
results = model.fit()
results.summary()
```

```
C:\Users\sigma\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471:
ValueWarning:

An unsupported index was provided and will be ignored when e.g. forecasting.
```

Out[58]:

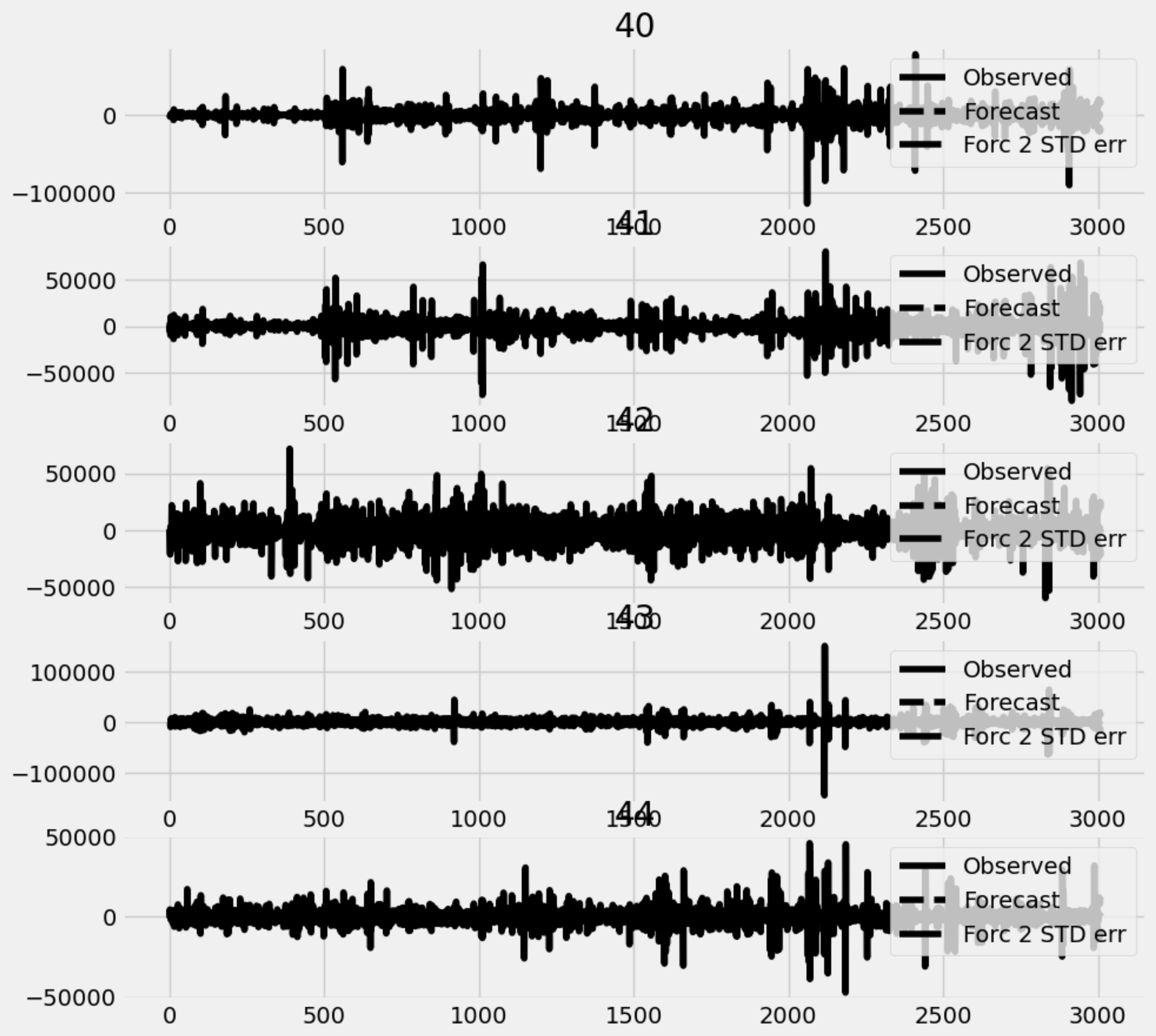
```
  Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Mon, 06, Mar, 2023
Time:                     11:06:11
--------------------------------------------------------------------
No. of Equations:         5.00000    BIC:                    89.2424
Nobs:                     2998.00    HQIC:                   89.2039
Log likelihood:          -154924.    FPE:                5.38720e+38
AIC:                      89.1823    Det(Omega_mle):     5.33361e+38
--------------------------------------------------------------------
Results for equation 40
========================================================================
           coefficient       std. error           t-stat            prob
------------------------------------------------------------------------
const        -3.925040       157.354755           -0.025           0.980
L1.40        -0.324487         0.019121          -16.970           0.000
L1.41         0.006450         0.017869            0.361           0.718
L1.42         0.023437         0.015149            1.547           0.122
L1.43        -0.055343         0.020822           -2.658           0.008
L1.44         0.012274         0.030713            0.400           0.689
========================================================================

Results for equation 41
========================================================================
           coefficient       std. error           t-stat            prob
------------------------------------------------------------------------
const        -7.598705       160.801183           -0.047           0.962
L1.40         0.127311         0.019540            6.515           0.000
L1.41        -0.428962         0.018260          -23.492           0.000
L1.42         0.042469         0.015481            2.743           0.006
L1.43        -0.002326         0.021278           -0.109           0.913
L1.44         0.109049         0.031386            3.474           0.001
========================================================================

Results for equation 42
========================================================================
           coefficient       std. error           t-stat            prob
------------------------------------------------------------------------
const         8.855591       187.178250            0.047           0.962
L1.40        -0.081186         0.022746           -3.569           0.000
L1.41        -0.001312         0.021255           -0.062           0.951
L1.42        -0.250109         0.018020          -13.879           0.000
L1.43         0.076494         0.024768            3.088           0.002
L1.44        -0.029880         0.036534           -0.818           0.413
========================================================================

Results for equation 43
========================================================================
           coefficient       std. error           t-stat            prob
------------------------------------------------------------------------
const         6.958074       129.916142            0.054           0.957
L1.40         0.021742         0.015787            1.377           0.168
L1.41        -0.038731         0.014753           -2.625           0.009
L1.42         0.025408         0.012507            2.031           0.042
L1.43        -0.398593         0.017191          -23.186           0.000
L1.44         0.066926         0.025358            2.639           0.008
========================================================================

Results for equation 44
========================================================================
           coefficient       std. error           t-stat            prob
------------------------------------------------------------------------
```

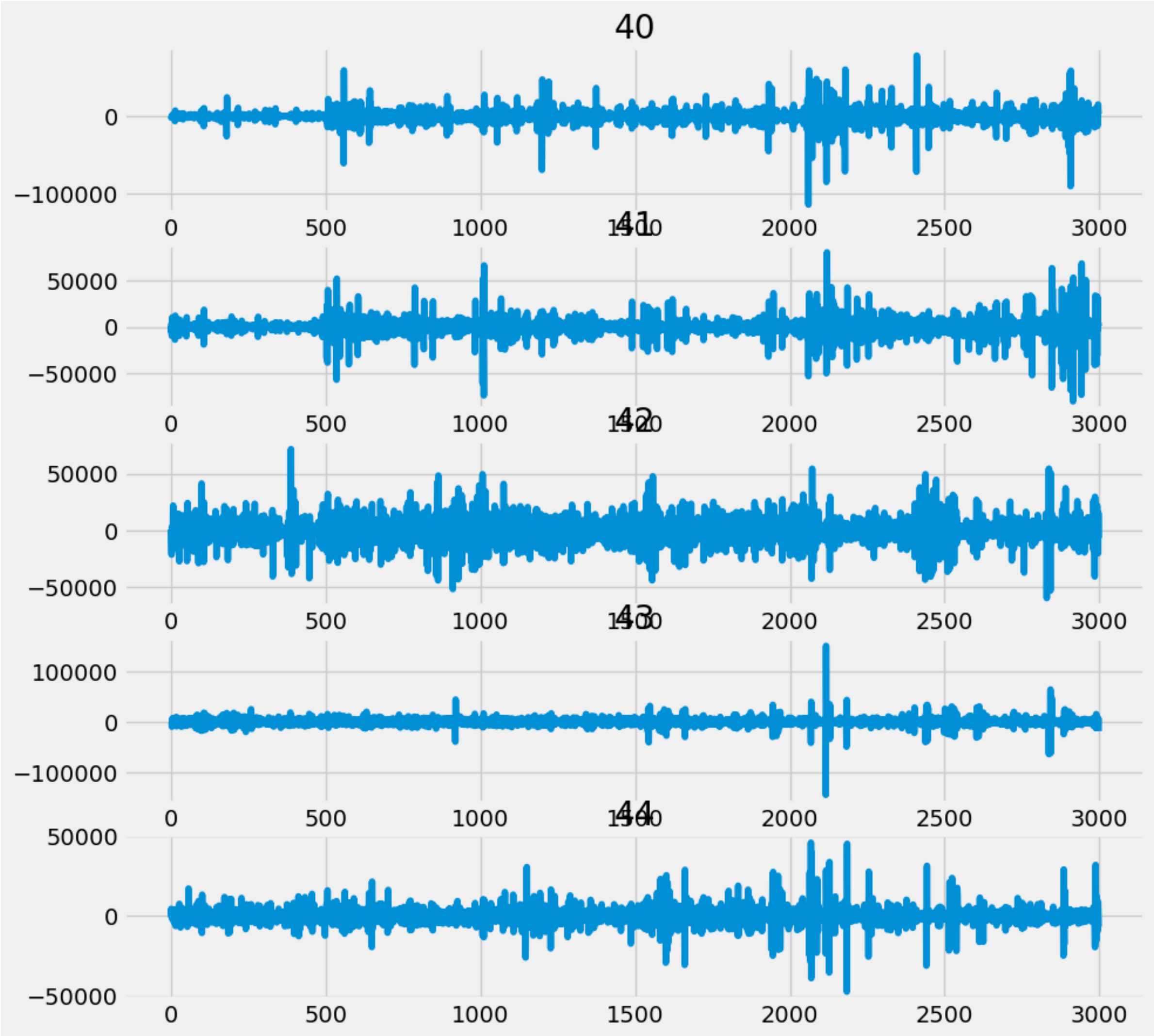
```
const          7.892486       88.730228          0.089           0.929
L1.40         -0.032926        0.010782         -3.054           0.002
L1.41          0.019359        0.010076          1.921           0.055
L1.42         -0.005243        0.008542         -0.614           0.539
L1.43          0.029301        0.011741          2.496           0.013
L1.44         -0.363780        0.017319        -21.005           0.000
========================================================================

Correlation matrix of residuals
             40        41        42        43        44
40    1.000000  0.429600 -0.139500 -0.148957 -0.116267
41    0.429600  1.000000  0.020419 -0.132764 -0.100554
42   -0.139500  0.020419  1.000000 -0.104419  0.006767
43   -0.148957 -0.132764 -0.104419  1.000000 -0.078095
44   -0.116267 -0.100554  0.006767 -0.078095  1.000000
```

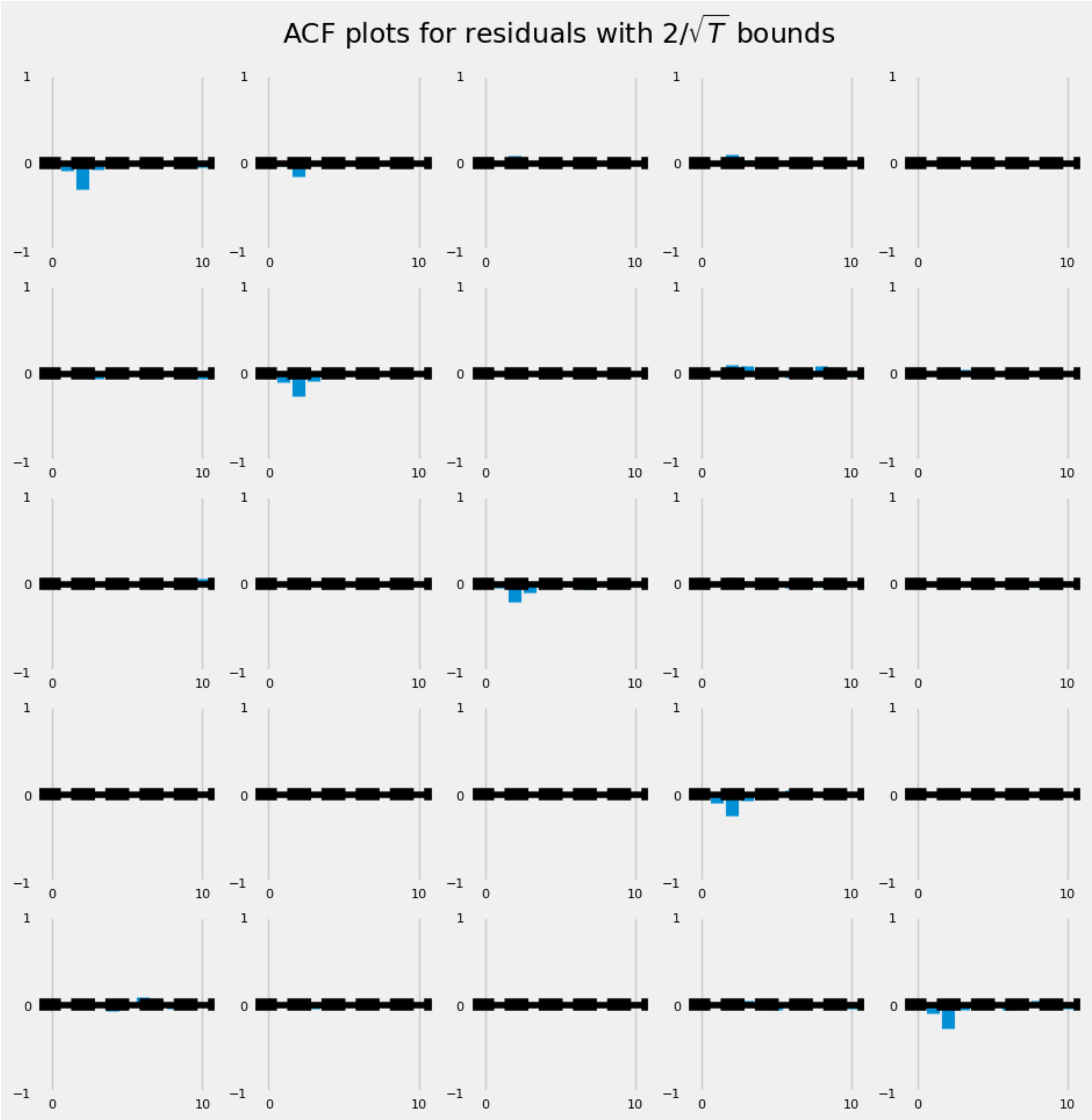
In [59]:

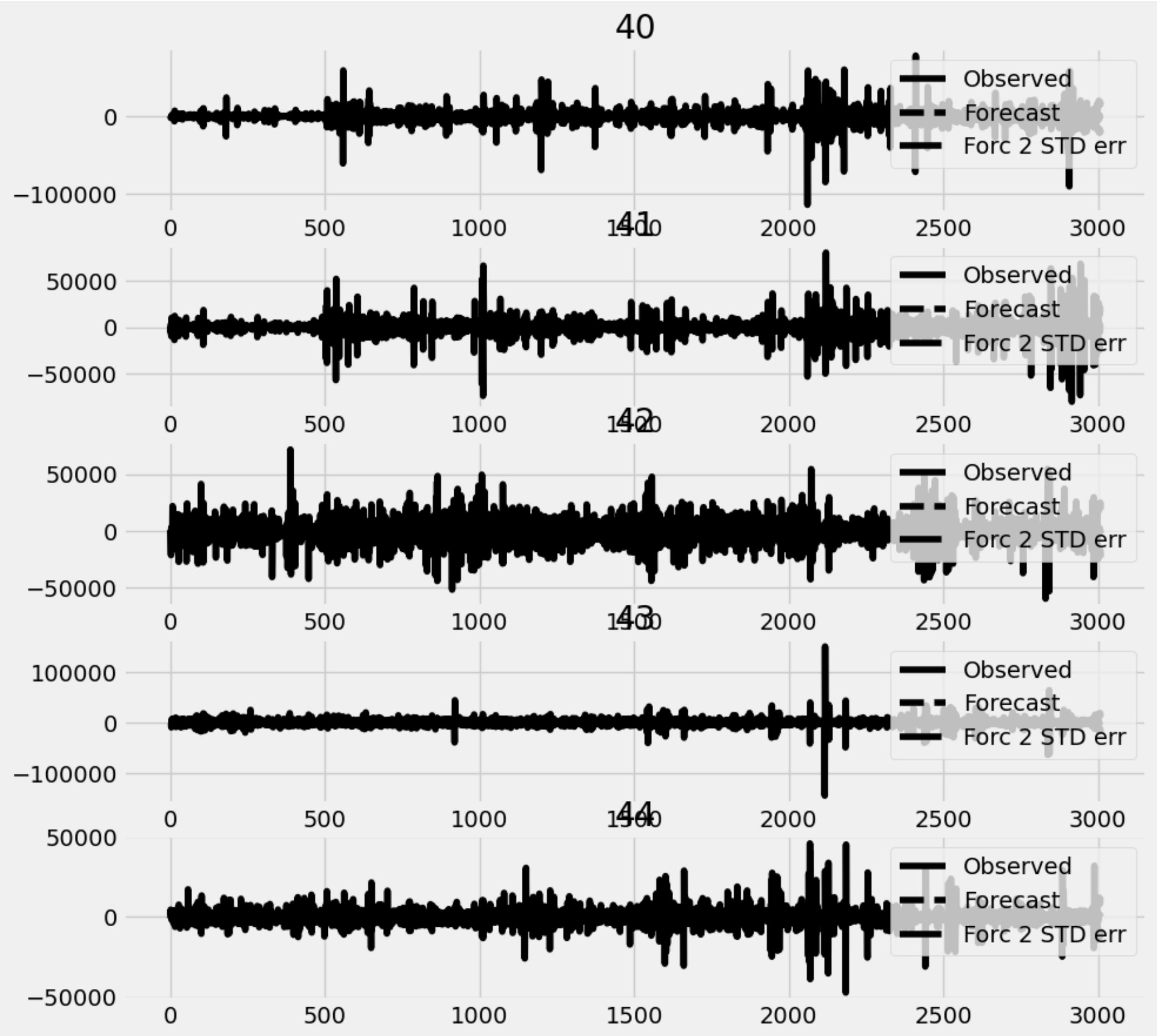
```
results.plot()
results.plot_acorr()
model.select_order(15)
results = model.fit(maxlags=15, ic='aic')
lag_order = results.k_ar
results.plot_forecast(10)
```

Out[59]:



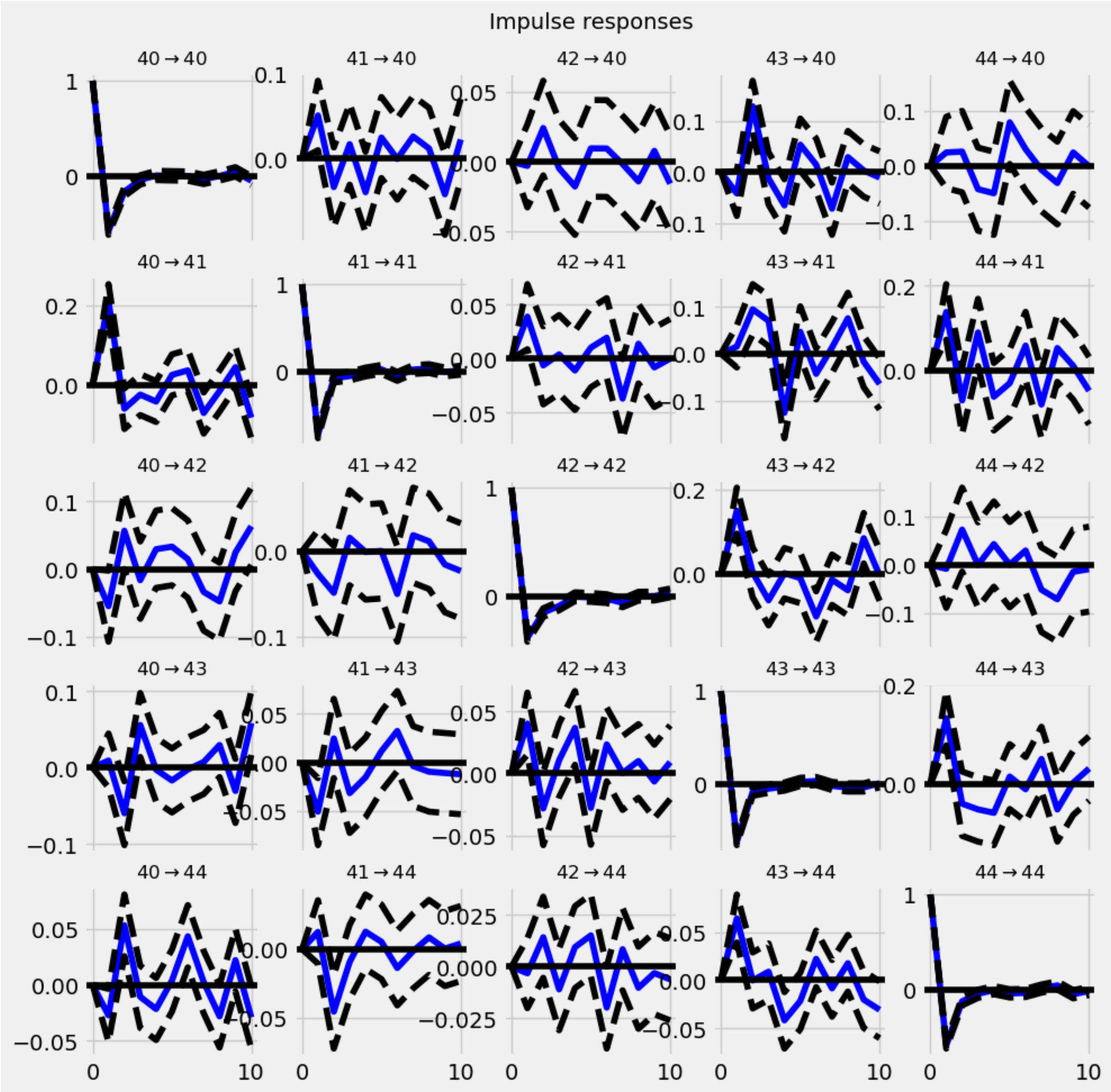

40
0
−100000
41
50000
0
−50000
42
50000
0
−50000
43
100000
0
−100000
44
50000
0
−50000
0
500
1000
1500
2000
2500
3000

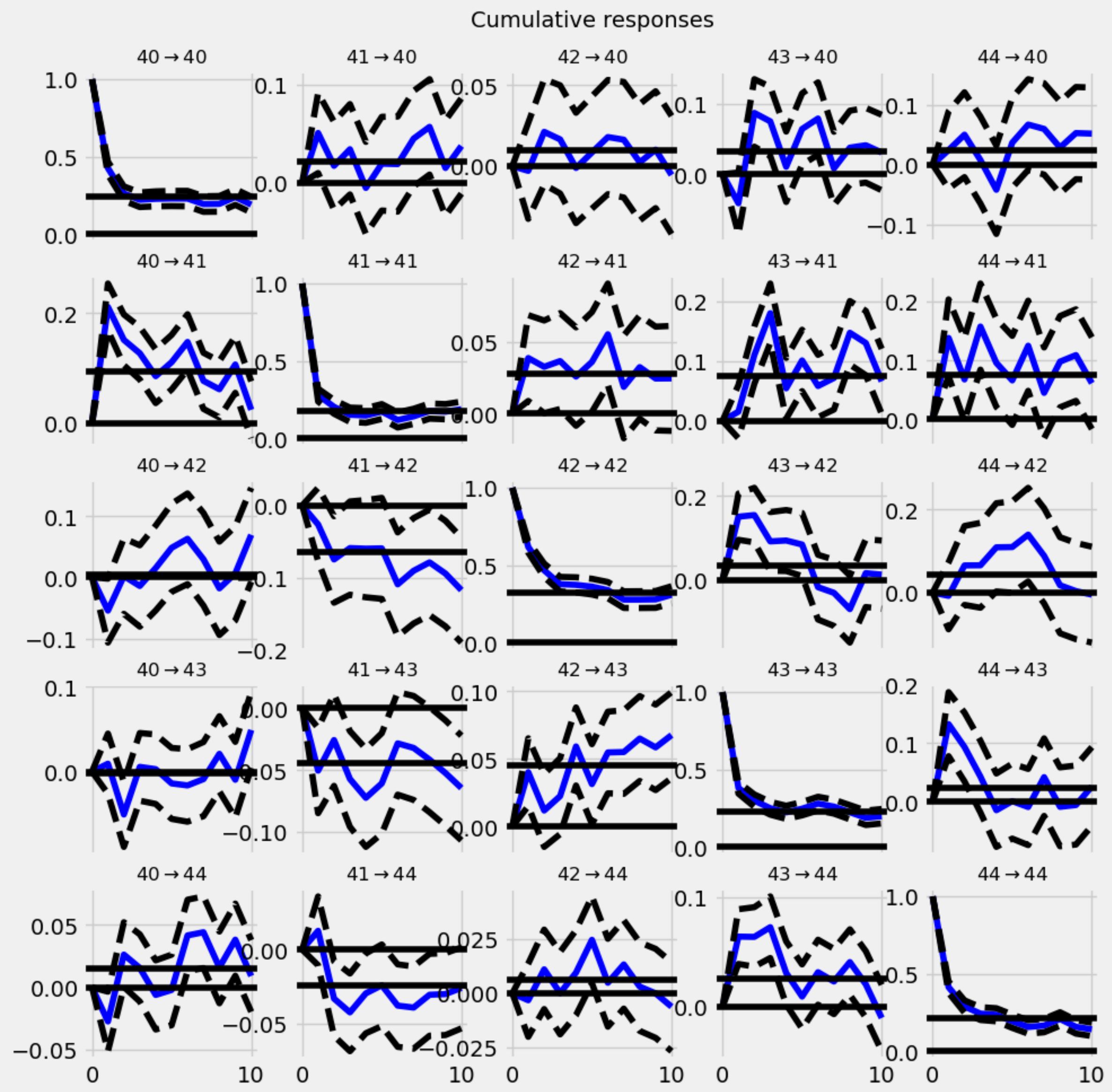

ACF plots for residuals with $2/\sqrt{T}$ bounds



In [60]:

```
#Impulse response
irf = results.irf(10)
irf.plot(orth=False)
irf.plot_cum_effects(orth=False)
fevd = results.fevd(5)
```


Impulse responses
40 → 40
41 → 40
42 → 40
43 → 40
44 → 40
40 → 41
41 → 41
42 → 41
43 → 41
44 → 41
40 → 42
41 → 42
42 → 42
43 → 42
44 → 42
40 → 43
41 → 43
42 → 43
43 → 43
44 → 43
40 → 44
41 → 44
42 → 44
43 → 44
44 → 44



In [62]:

```
# Remark from the VAR model's impulse response and regression result:
#An unexpected shock in asset 43 will cause a significant boost in asset 40
#within a few months.
#An unexpected shock in asset 40 will cause a significant downside in asset 41
#within a few months.
#An unexpected shock in asset 40 will cause a significant and long run incrsase
#in asset 42.
#An unexpected shock in asset 42 will cause a significant and long run increase
#in asset 43.
#An unexpected shock in asset 43 will cause a significant and long run decrease
#in asset 44.
```