Visual Data Analysis of Fraudulent Transactions

Your CFO has also requested detailed trends data on specific card holders. Use the starter notebook to query your database and generate visualizations that supply the requested information as follows, then add your visualizations and observations to your markdown report.

In [1]:
```
# Initial imports
!pip install psycopg2-binary
!pip install connectorx
import pandas as pd
import calendar
import hvplot.pandas
from sqlalchemy import create_engine
import psycopg2
import connectorx as cx
```

```
Requirement already satisfied: psycopg2-binary in c:\users\yohan\anaconda3\envs\dev\envs
\dev\lib\site-packages (2.9.3)
Requirement already satisfied: connectorx in c:\users\yohan\anaconda3\envs\dev\envs\dev
\lib\site-packages (0.3.0)
```

In [2]:
```
# Create a connection to the database
engine = create_engine("postgresql://postgres:postgres@localhost:5432/fraud_detection")
```

Data Analysis Question 1

The two most important customers of the firm may have been hacked. Verify if there are any fraudulent transactions in their history. For privacy reasons, you only know that their cardholder IDs are 2 and 18.

- Using hvPlot, create a line plot representing the time series of transactions over the course of the year for each cardholder separately.
- Next, to better compare their patterns, create a single line plot that containins both card holders' trend data.
- What difference do you observe between the consumption patterns? Does the difference suggest a fraudulent transaction? Explain your rationale in the markdown report.

In [3]:
```
# loading data for card holder 2 and 18 from the database
# Write the query

#First, get the card numbers from customers id 2 and 18 from table credit_card which onl
#customer id. This step is first given that the transaction table does not have customer
query = "SELECT cardholder_id, card FROM credit_card WHERE cardholder_id = 2 OR cardhold

# Create a DataFrame from the query result.


two_eighteen_df=pd.read_sql(query,engine)
two_eighteen_df
```

Out[3]:

	cardholder_id	card

0	2	4866761290278198714
1	2	675911140852
2	18	4498002758300
3	18	344119623920892
4	25	4319653513507
5	25	372414832802279

In [4]:
```
#now we can match the cardnumbers with the transaction history in the transaction table:

query_c2 = "SELECT date, transaction_amount, card FROM transaction WHERE card = '4866761

c2_df = pd.read_sql(query_c2,engine)
c2_df
```

Out[4]:

	date	transaction_amount	card
0	2018-01-06 02:16:41	1.33	4866761290278198714
1	2018-01-06 05:13:20	10.82	4866761290278198714
2	2018-01-07 15:10:27	17.29	4866761290278198714
3	2018-01-10 10:07:20	10.91	675911140852
4	2018-01-16 06:29:35	17.64	675911140852
...	...	...	...
94	2018-12-13 06:21:43	19.36	4866761290278198714
95	2018-12-13 15:28:18	10.06	675911140852
96	2018-12-16 13:44:25	11.38	4866761290278198714
97	2018-12-22 23:29:09	10.20	4866761290278198714
98	2018-12-28 15:30:55	11.03	675911140852

99 rows × 3 columns

In [5]:
```
#we do the same for cardholder 18:
query_c18 = "SELECT date, transaction_amount, card FROM transaction WHERE card = '449800

c18_df = pd.read_sql(query_c18,engine)
c18_df
```

Out[5]:

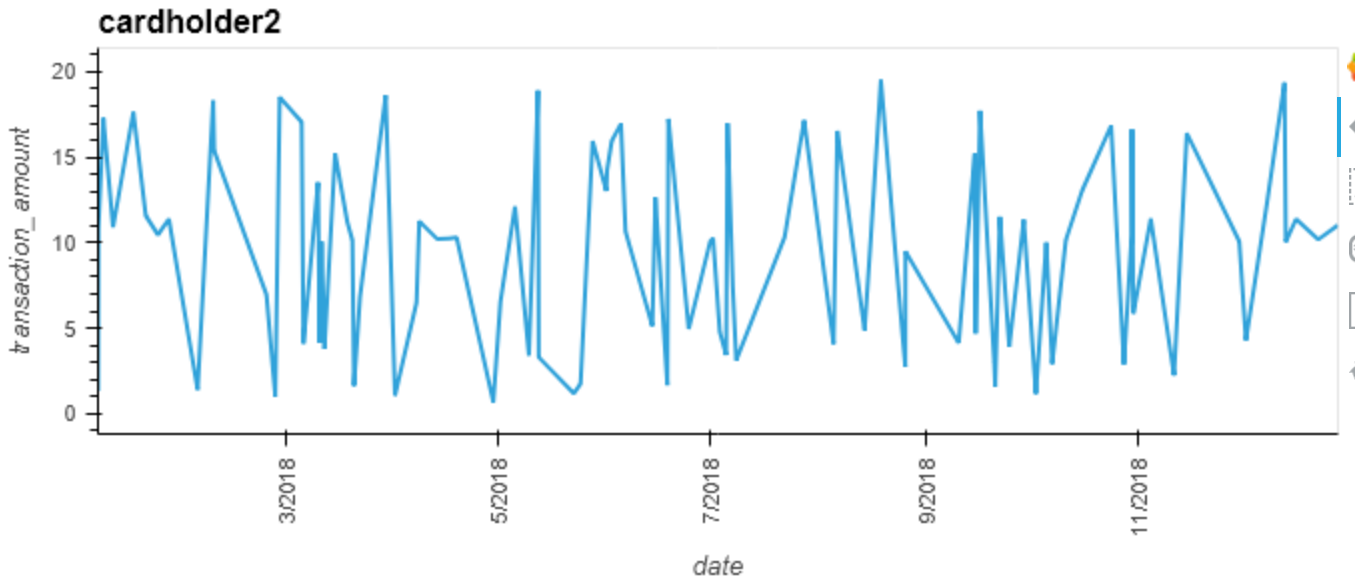
	date	transaction_amount	card
0	2018-01-01 23:15:10	2.95	4498002758300
1	2018-01-05 07:19:27	1.36	344119623920892
2	2018-01-07 01:10:54	175.00	344119623920892
3	2018-01-08 11:15:36	333.00	344119623920892
4	2018-01-08 20:10:59	11.55	344119623920892
...	...	...	...
128	2018-12-23 03:33:56	4.36	344119623920892
129	2018-12-27 18:46:57	1.70	344119623920892

130	2018-12-28 08:45:26	3.46	4498002758300
131	2018-12-28 09:00:45	12.88	344119623920892
132	2018-12-29 08:11:55	12.25	4498002758300

133 rows × 3 columns

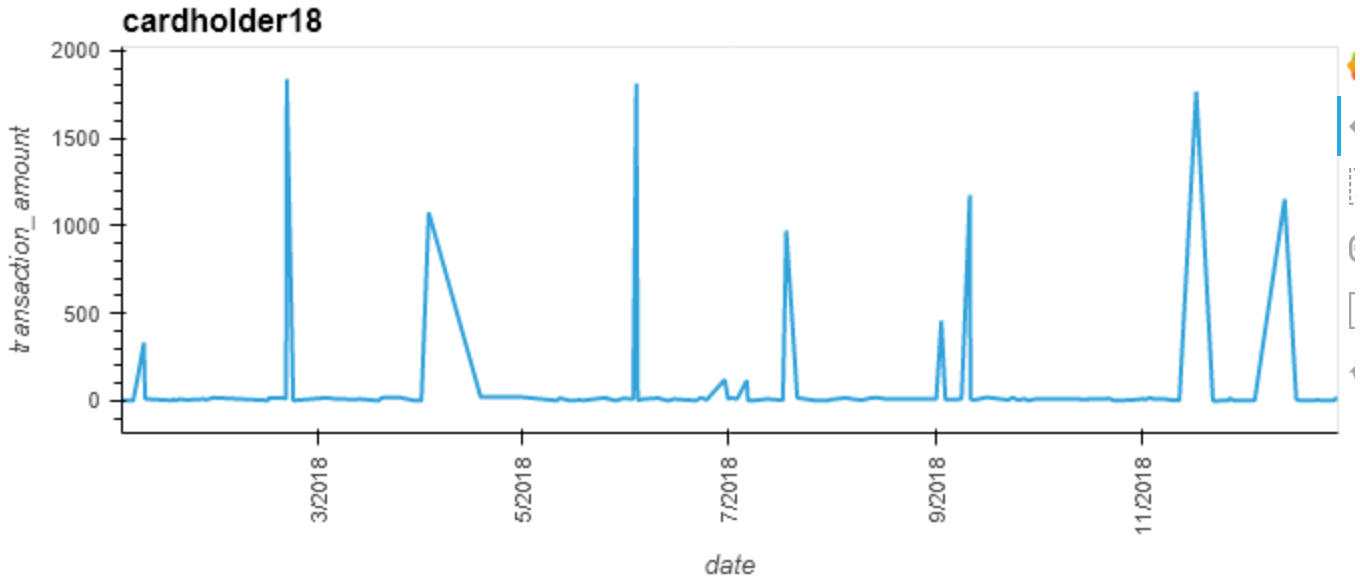
In [6]:
```
# Plot for cardholder 2
c2_plot = c2_df.hvplot.line(x="date", y="transaction_amount", rot=90, label="cardholder2
c2_plot
```

Out[6]:



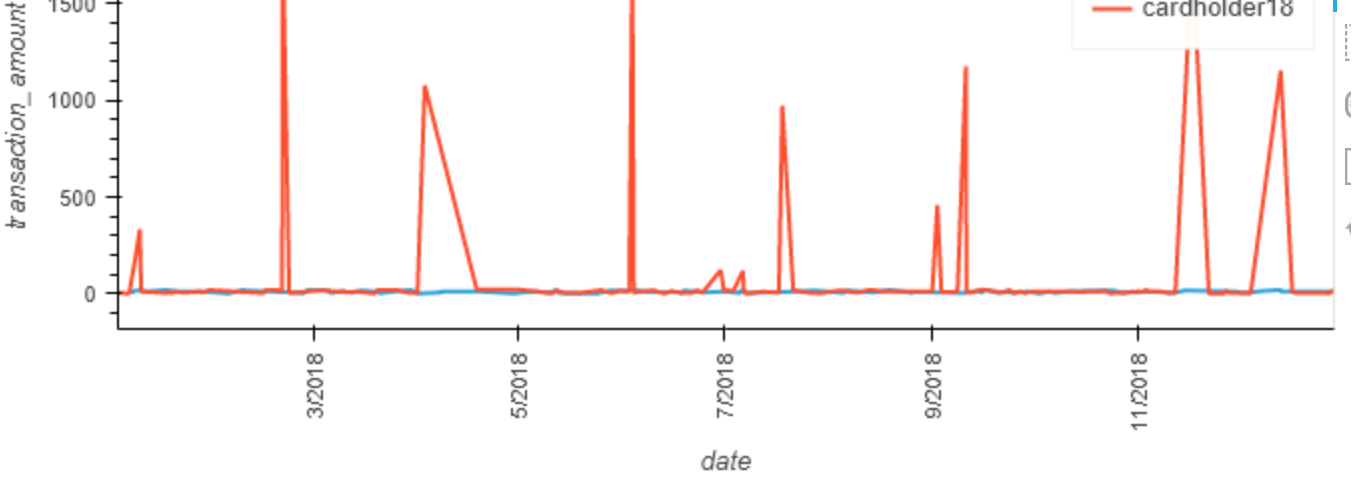
In [7]:
```
# Plot for cardholder 18
c18_plot=c18_df.hvplot.line(x="date", y="transaction_amount", rot=90, label="cardholder1
c18_plot
```

Out[7]:



In [8]:
```
# Combined plot for card holders 2 and 18
c2_plot * c18_plot
```

Out[8]:



In [9]:
```
#It seems that cardholder 18 has been hacked. Cardholder 2 has a lot more stable and sma
#But cardholder 18 has these sudden bursts of expenditures sometimes close to $2000.
```

Data Analysis Question 2

The CEO of the biggest customer of the firm suspects that someone has used her corporate credit card without authorization in the first quarter of 2018 to pay quite expensive restaurant bills. Again, for privacy reasons, you know only that the cardholder ID in question is 25.

- Using hvPlot, create a box plot, representing the expenditure data from January 2018 to June 2018 for cardholder ID 25.
- Are there any outliers for cardholder ID 25? How many outliers are there per month?
- Do you notice any anomalies? Describe your observations and conclusions in your markdown report.

In [9]:
```
# loading data of daily transactions from jan to jun 2018 for card holder 25
# Write the query
query_c25 = """
            SELECT date, transaction_amount, card FROM transaction
            WHERE (card = '4319653513507' OR card = '372414832802279') AND date BETWEEN
            """
#where event_date between '2020-01-01 12:00:00' and '2020-01-01 23:30:00';

# Create a DataFrame from the query result.
c25_df = pd.read_sql(query_c25,engine)
c25_df
```

Out[9]:

	date	transaction_amount	card
0	2018-01-02 02:06:21	1.46	4319653513507
1	2018-01-05 06:26:45	10.74	372414832802279
2	2018-01-07 14:57:23	2.93	4319653513507
3	2018-01-10 00:25:40	1.39	372414832802279
4	2018-01-14 05:02:22	17.84	372414832802279
5	2018-01-16 02:26:16	1.65	372414832802279
6	2018-01-18 12:41:06	15.86	4319653513507
7	2018-01-21 23:04:02	2.22	372414832802279

8	2018-01-30 18:31:00	1177.00	4319653513507
9	2018-01-31 05:46:43	2.75	4319653513507
10	2018-02-02 11:31:33	10.75	372414832802279
11	2018-02-05 21:59:07	10.81	372414832802279
12	2018-02-07 00:20:11	5.97	372414832802279
13	2018-02-12 03:44:20	3.69	4319653513507
14	2018-02-18 20:43:22	16.70	4319653513507
15	2018-02-23 10:13:27	1.26	4319653513507
16	2018-02-23 12:26:19	2.63	4319653513507
17	2018-02-23 18:49:22	11.01	372414832802279
18	2018-02-28 02:06:08	0.91	372414832802279
19	2018-02-28 13:56:12	1.18	372414832802279
20	2018-03-02 23:23:52	12.42	4319653513507
21	2018-03-05 07:34:15	16.58	4319653513507
22	2018-03-06 07:18:09	1334.00	4319653513507
23	2018-03-07 16:45:37	2.88	4319653513507
24	2018-03-09 03:59:06	2.04	372414832802279
25	2018-03-11 19:37:02	13.57	4319653513507
26	2018-03-12 01:00:24	10.10	372414832802279
27	2018-03-12 09:08:18	1.65	372414832802279
28	2018-03-12 17:16:34	3.08	372414832802279
29	2018-03-16 02:04:54	4.20	372414832802279
30	2018-03-17 18:22:07	2.56	4319653513507
31	2018-03-18 12:29:39	18.28	372414832802279
32	2018-03-31 20:12:10	21.04	372414832802279
33	2018-04-01 07:17:21	100.00	4319653513507
34	2018-04-01 21:08:23	2.62	4319653513507
35	2018-04-02 01:50:15	7.08	372414832802279
36	2018-04-02 18:34:50	17.15	372414832802279
37	2018-04-08 06:03:50	1063.00	4319653513507
38	2018-04-08 18:03:55	10.15	372414832802279
39	2018-04-08 18:14:22	10.06	4319653513507
40	2018-04-09 18:28:25	269.00	4319653513507
41	2018-04-10 23:03:20	10.24	4319653513507
42	2018-04-18 10:12:40	7.39	372414832802279
43	2018-04-19 18:30:14	6.01	4319653513507
44	2018-04-20 17:02:27	20.03	372414832802279
45	2018-04-26 02:16:45	2.79	372414832802279

46	2018-04-26 19:49:31	10.02	4319653513507
47	2018-04-26 23:09:51	15.66	372414832802279
48	2018-04-29 02:41:44	16.50	372414832802279
49	2018-05-06 04:38:27	1.10	372414832802279
50	2018-05-13 06:31:20	1046.00	4319653513507
51	2018-05-17 21:32:51	12.15	4319653513507
52	2018-05-19 09:12:20	2.27	4319653513507
53	2018-05-29 14:34:36	5.97	372414832802279

In [10]:
```
c25_df["month"] = ""
c25_df.head()
```

Out[10]:

	date	transaction_amount	card	month
0	2018-01-02 02:06:21	1.46	4319653513507	
1	2018-01-05 06:26:45	10.74	372414832802279	
2	2018-01-07 14:57:23	2.93	4319653513507	
3	2018-01-10 00:25:40	1.39	372414832802279	
4	2018-01-14 05:02:22	17.84	372414832802279	

In [11]:
```
# loop to change the numeric month to month names

for i in range(0, len(c25_df["date"])):
    if c25_df["date"][i] < pd.Timestamp('2018-02-01'):
        c25_df["month"][i] = "January"
    elif c25_df["date"][i] < pd.Timestamp('2018-03-01'):
        c25_df["month"][i] = "February"
    elif c25_df["date"][i] < pd.Timestamp('2018-04-01'):
        c25_df["month"][i] = "March"
    elif c25_df["date"][i] < pd.Timestamp('2018-05-01'):
        c25_df["month"][i] = "April"
    elif c25_df["date"][i] < pd.Timestamp('2018-06-01'):
        c25_df["month"][i] = "May"

c25_df
```

```
C:\Users\yohan\anaconda3\envs\dev\envs\dev\lib\site-packages\ipykernel_launcher.py:5: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  """
C:\Users\yohan\anaconda3\envs\dev\envs\dev\lib\site-packages\ipykernel_launcher.py:7: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  import sys
C:\Users\yohan\anaconda3\envs\dev\envs\dev\lib\site-packages\ipykernel_launcher.py:9: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
```

```
guide/indexing.html#returning-a-view-versus-a-copy
  if __name__ == '__main__':
C:\Users\yohan\anaconda3\envs\dev\envs\dev\lib\site-packages\ipykernel_launcher.py:11: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  # This is added back by InteractiveShellApp.init_path()
C:\Users\yohan\anaconda3\envs\dev\envs\dev\lib\site-packages\ipykernel_launcher.py:13: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  del sys.path[0]
```

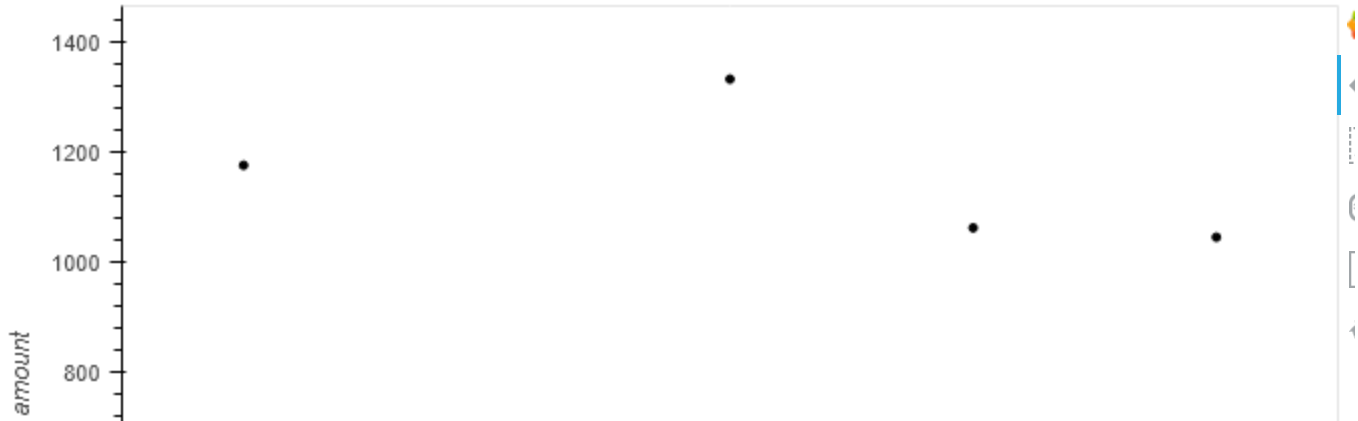
Out[11]:

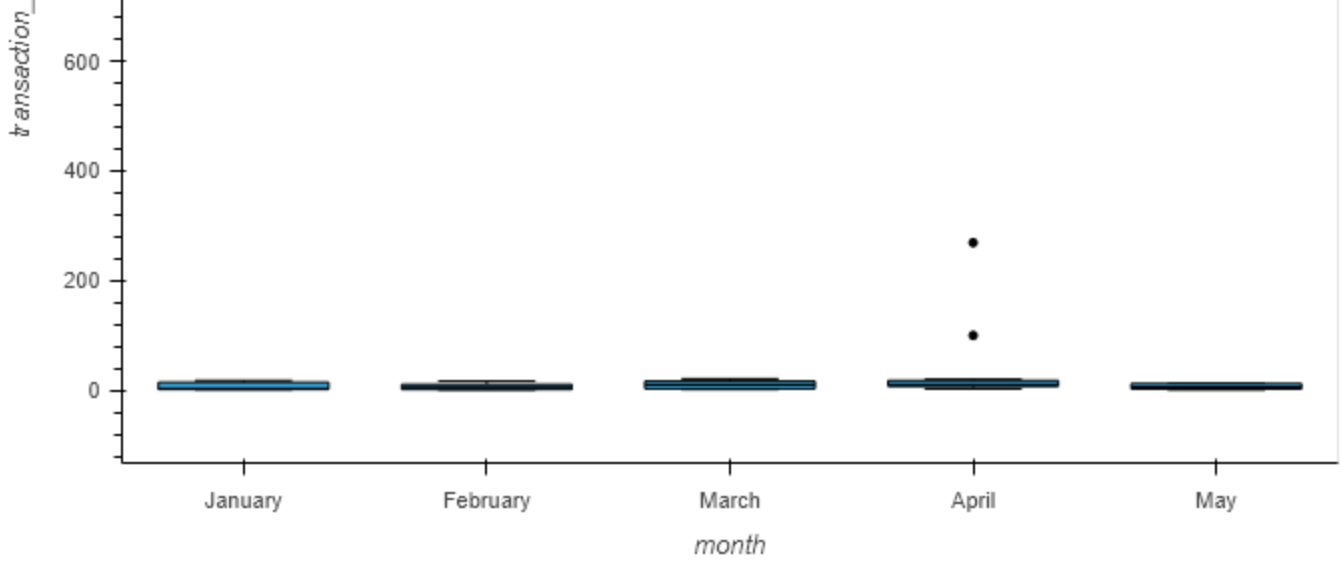
	date	transaction_amount	card	month
0	2018-01-02 02:06:21	1.46	4319653513507	January
1	2018-01-05 06:26:45	10.74	372414832802279	January
2	2018-01-07 14:57:23	2.93	4319653513507	January
3	2018-01-10 00:25:40	1.39	372414832802279	January
4	2018-01-14 05:02:22	17.84	372414832802279	January
5	2018-01-16 02:26:16	1.65	372414832802279	January
6	2018-01-18 12:41:06	15.86	4319653513507	January
7	2018-01-21 23:04:02	2.22	372414832802279	January
8	2018-01-30 18:31:00	1177.00	4319653513507	January
9	2018-01-31 05:46:43	2.75	4319653513507	January
10	2018-02-02 11:31:33	10.75	372414832802279	February
11	2018-02-05 21:59:07	10.81	372414832802279	February
12	2018-02-07 00:20:11	5.97	372414832802279	February
13	2018-02-12 03:44:20	3.69	4319653513507	February
14	2018-02-18 20:43:22	16.70	4319653513507	February
15	2018-02-23 10:13:27	1.26	4319653513507	February
16	2018-02-23 12:26:19	2.63	4319653513507	February
17	2018-02-23 18:49:22	11.01	372414832802279	February
18	2018-02-28 02:06:08	0.91	372414832802279	February
19	2018-02-28 13:56:12	1.18	372414832802279	February
20	2018-03-02 23:23:52	12.42	4319653513507	March
21	2018-03-05 07:34:15	16.58	4319653513507	March
22	2018-03-06 07:18:09	1334.00	4319653513507	March
23	2018-03-07 16:45:37	2.88	4319653513507	March
24	2018-03-09 03:59:06	2.04	372414832802279	March
25	2018-03-11 19:37:02	13.57	4319653513507	March
26	2018-03-12 01:00:24	10.10	372414832802279	March

27	2018-03-12 09:08:18	1.65	372414832802279	March
28	2018-03-12 17:16:34	3.08	372414832802279	March
29	2018-03-16 02:04:54	4.20	372414832802279	March
30	2018-03-17 18:22:07	2.56	4319653513507	March
31	2018-03-18 12:29:39	18.28	372414832802279	March
32	2018-03-31 20:12:10	21.04	372414832802279	March
33	2018-04-01 07:17:21	100.00	4319653513507	April
34	2018-04-01 21:08:23	2.62	4319653513507	April
35	2018-04-02 01:50:15	7.08	372414832802279	April
36	2018-04-02 18:34:50	17.15	372414832802279	April
37	2018-04-08 06:03:50	1063.00	4319653513507	April
38	2018-04-08 18:03:55	10.15	372414832802279	April
39	2018-04-08 18:14:22	10.06	4319653513507	April
40	2018-04-09 18:28:25	269.00	4319653513507	April
41	2018-04-10 23:03:20	10.24	4319653513507	April
42	2018-04-18 10:12:40	7.39	372414832802279	April
43	2018-04-19 18:30:14	6.01	4319653513507	April
44	2018-04-20 17:02:27	20.03	372414832802279	April
45	2018-04-26 02:16:45	2.79	372414832802279	April
46	2018-04-26 19:49:31	10.02	4319653513507	April
47	2018-04-26 23:09:51	15.66	372414832802279	April
48	2018-04-29 02:41:44	16.50	372414832802279	April
49	2018-05-06 04:38:27	1.10	372414832802279	May
50	2018-05-13 06:31:20	1046.00	4319653513507	May
51	2018-05-17 21:32:51	12.15	4319653513507	May
52	2018-05-19 09:12:20	2.27	4319653513507	May
53	2018-05-29 14:34:36	5.97	372414832802279	May

In [12]:
```
# Creating the six box plots using hvPlot
c25_df.hvplot.box(y="transaction_amount", by="month", height=500, legend=False)
```

Out[12]:





In [15]:
```
#it seems as though there are a few outliers in the expenditures of cardholder 25. There
#sometimes in the amount of over $1000. February did not have any but April had three ou
```