CaseStudies Live 2024-11-18

November 18, 2024

1 CaseStudies Live 2024-11-18 Q6

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

1.1 Q6a

```
[52]: X = pd.read_csv("Data03.csv")
```

```
[5]: X.describe()
```

```
[5]:                 f1           f2
     count  3840.000000  3840.000000
     mean     -4.861466    -4.854412
     std       1.604436     1.564089
     min      -8.740101    -8.392300
     25%      -5.704431    -5.649969
     50%      -4.999363    -5.013370
     75%      -4.317683    -4.317863
     max       7.865602     7.463494
```

```
[6]: X.head()
```

```
[6]:              timestamp        f1        f2
     0  2023-07-20 06:00:00 -5.143580 -5.382025
     1  2023-07-20 06:01:00 -4.605052 -5.503890
     2  2023-07-20 06:02:00 -5.690262 -5.285184
     3  2023-07-20 06:03:00 -3.917527 -6.951911
     4  2023-07-20 06:04:00 -4.788862 -5.260651
```

```
[7]: X.timestamp
```

```
[7]: 0       2023-07-20 06:00:00
     1       2023-07-20 06:01:00
     2       2023-07-20 06:02:00
     3       2023-07-20 06:03:00
     4       2023-07-20 06:04:00
```

```
                 …
3835    2023-07-23 21:55:00
3836    2023-07-23 21:56:00
3837    2023-07-23 21:57:00
3838    2023-07-23 21:58:00
3839    2023-07-23 21:59:00
Name: timestamp, Length: 3840, dtype: object
```

[8]:

```
X.timestamp.mean()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[8], line 1
----> 1 X.timestamp.mean()

File /opt/conda/lib/python3.10/site-packages/pandas/core/series.py:6226, in␣
 ↪Series.mean(self, axis, skipna, numeric_only, **kwargs)
   6218 @doc(make_doc("mean", ndim=1))
   6219 def mean(
   6220     self,
   (…)
   6224     **kwargs,
   6225 ):
-> 6226     return NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)

File /opt/conda/lib/python3.10/site-packages/pandas/core/generic.py:11969, in␣
 ↪NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)
  11962 def mean(
  11963     self,
  11964     axis: Axis | None = 0,
   (…)
  11967     **kwargs,
  11968 ) -> Series | float:
> 11969     return self._stat_function(
  11970         "mean", nanops.nanmean, axis, skipna, numeric_only, **kwargs
  11971     )

File /opt/conda/lib/python3.10/site-packages/pandas/core/generic.py:11926, in␣
 ↪NDFrame._stat_function(self, name, func, axis, skipna, numeric_only, **kwargs
  11922 nv.validate_func(name, (), kwargs)
  11924 validate_bool_kwarg(skipna, "skipna", none_allowed=False)
> 11926 return self._reduce(
  11927     func, name=name, axis=axis, skipna=skipna, numeric_only=numeric_onl
  11928 )

File /opt/conda/lib/python3.10/site-packages/pandas/core/series.py:6134, in␣
 ↪Series._reduce(self, op, name, axis, skipna, numeric_only, filter_type, **kwds)
```

```
   6129     # GH#47500 - change to TypeError to match other methods
   6130     raise TypeError(
   6131         f"Series.{name} does not allow {kwd_name}={numeric_only} "
   6132         "with non-numeric dtypes."
   6133     )
-> 6134 return op(delegate, skipna=skipna, **kwds)

File /opt/conda/lib/python3.10/site-packages/pandas/core/nanops.py:147, in bottleneck_switch.__call__.<locals>.f(values, axis, skipna, **kwds)
    145         result = alt(values, axis=axis, skipna=skipna, **kwds)
    146 else:
--> 147     result = alt(values, axis=axis, skipna=skipna, **kwds)
    149 return result

File /opt/conda/lib/python3.10/site-packages/pandas/core/nanops.py:404, in _datetimelike_compat.<locals>.new_func(values, axis, skipna, mask, **kwargs)
    401 if datetimelike and mask is None:
    402     mask = isna(values)
--> 404 result = func(values, axis=axis, skipna=skipna, mask=mask, **kwargs)
    406 if datetimelike:
    407     result = _wrap_results(result, orig_values.dtype, fill_value=iNaT)

File /opt/conda/lib/python3.10/site-packages/pandas/core/nanops.py:720, in nanmean(values, axis, skipna, mask)
    718 count = _get_counts(values.shape, mask, axis, dtype=dtype_count)
    719 the_sum = values.sum(axis, dtype=dtype_sum)
--> 720 the_sum = _ensure_numeric(the_sum)
    722 if axis is not None and getattr(the_sum, "ndim", False):
    723     count = cast(np.ndarray, count)

File /opt/conda/lib/python3.10/site-packages/pandas/core/nanops.py:1693, in _ensure_numeric(x)
   1690 elif not (is_float(x) or is_integer(x) or is_complex(x)):
   1691     if isinstance(x, str):
   1692         # GH#44008, GH#36703 avoid casting e.g. strings to numeric
-> 1693         raise TypeError(f"Could not convert string '{x}' to numeric")
   1694     try:
   1695         x = float(x)
```

```
TypeError: Could not convert string '2023-07-20 06:00:002023-07-20 06:01:
  ↪002023-07-20 06:02:002023-07-20 06:03:002023-07-20 06:04:002023-07-20 06:05:
  ↪002023-07-20 06:06:002023-07-20 06:07:002023-07-20 06:08:002023-07-20 06:09:
  ↪002023-07-20 06:10:002023-07-20 06:11:002023-07-20 06:12:002023-07-20 06:13:
  ↪002023-07-20 06:14:002023-07-20 06:15:002023-07-20 06:16:002023-07-20 06:17:
  ↪002023-07-20 06:18:002023-07-20 06:19:002023-07-20 06:20:002023-07-20 06:21:
  ↪002023-07-20 06:22:002023-07-20 06:23:002023-07-20 06:24:002023-07-20 06:25:
  ↪002023-07-20 06:26:002023-07-20 06:27:002023-07-20 06:28:002023-07-20 06:29:
  ↪002023-07-20 06:30:002023-07-20 06:31:002023-07-20 06:32:002023-07-20 06:33:
  ↪002023-07-20 06:34:002023-07-20 06:35:002023-07-20 06:36:002023-07-20 06:37:
  ↪002023-07-20 06:38:002023-07-20 06:39:002023-07-20 06:40:002023-07-20 06:41:
  ↪002023-07-20 06:42:002023-07-20 06:43:002023-07-20 06:44:002023-07-20 06:45:
  ↪002023-07-20 06:46:002023-07-20 06:47:002023-07-20 06:48:002023-07-20 06:49:
  ↪002023-07-20 06:50:002023-07-20 06:51:002023-07-20 06:52:002023-07-20 06:53:
  ↪002023-07-20 06:54:002023-07-20 06:55:002023-07-20 06:56:002023-07-20 06:57:
  ↪002023-07-20 06:58:002023-07-20 06:59:002023-07-20 07:00:002023-07-20 07:01:
  ↪002023-07-20 07:02:002023-07-20 07:03:002023-07-20 07:04:002023-07-20 07:05:
  ↪002023-07-20 07:06:002023-07-20 07:07:002023-07-20 07:08:002023-07-20 07:09:
  ↪002023-07-20 07:10:002023-07-20 07:11:002023-07-20 07:12:002023-07-20 07:13:
  ↪002023-07-20 07:14:002023-07-20 07:15:002023-07-20 07:16:002023-07-20 07:17:
  ↪002023-07-20 07:18:002023-07-20 07:19:002023-07-20 07:20:002023-07-20 07:21:
  ↪002023-07-20 07:22:002023-07-20 07:23:002023-07-20 07:24:002023-07-20 07:25:
  ↪002023-07-20 07:26:002023-07-20 07:27:002023-07-20 07:28:002023-07-20 07:29:
  ↪002023-07-20 07:30:002023-07-20 07:31:002023-07-20 07:32:002023-07-20 07:33:
  ↪002023-07-20 07:34:002023-07-20 07:35:002023-07-20 07:36:002023-07-20 07:37:
  ↪002023-07-20 07:38:002023-07-20 07:39:002023-07-20 07:40:002023-07-20 07:41:
  ↪002023-07-20 07:42:002023-07-20 07:43:002023-07-20 07:44:002023-07-20 07:45:
  ↪002023-07-20 07:46:002023-07-20 07:47:002023-07-20 07:48:002023-07-20 07:49:
  ↪002023-07-20 07:50:002023-07-20 07:51:002023-07-20 07:52:002023-07-20 07:53:
  ↪002023-07-20 07:54:002023-07-20 07:55:002023-07-20 07:56:002023-07-20 07:57:
  ↪002023-07-20 07:58:002023-07-20 07:59:002023-07-20 08:00:002023-07-20 08:01:
  ↪002023-07-20 08:02:002023-07-20 08:03:002023-07-20 08:04:002023-07-20 08:05:
  ↪002023-07-20 08:06:002023-07-20 08:07:002023-07-20 08:08:002023-07-20 08:09:
  ↪002023-07-20 08:10:002023-07-20 08:11:002023-07-20 08:12:002023-07-20 08:13:
  ↪002023-07-20 08:14:002023-07-20 08:15:002023-07-20 08:16:002023-07-20 08:17:
  ↪002023-07-20 08:18:002023-07-20 08:19:002023-07-20 08:20:002023-07-20 08:21:
  ↪002023-07-20 08:22:002023-07-20 08:23:002023-07-20 08:24:002023-07-20 08:25:
  ↪002023-07-20 08:26:002023-07-20 08:27:002023-07-20 08:28:002023-07-20 08:29:
  ↪002023-07-20 08:30:002023-07-20 08:31:002023-07-20 08:32:002023-07-20 08:33:
  ↪002023-07-20 08:34:002023-07-20 08:35:002023-07-20 08:36:002023-07-20 08:37:
  ↪002023-07-20 08:38:002023-07-20 08:39:002023-07-20 08:40:002023-07-20 08:41:
  ↪002023-07-20 08:42:002023-07-20 08:43:002023-07-20 08:44:002023-07-20 08:45:
  ↪002023-07-20 08:46:002023-07-20 08:47:002023-07-20 08:48:002023-07-20 08:49:
  ↪002023-07-20 08:50:002023-07-20 08:51:002023-07-20 08:52:002023-07-20 08:53:
  ↪002023-07-20 08:54:002023-07-20 08:55:002023-07-20 08:56:002023-07-20 08:57:
  ↪002023-07-20 08:58:002023-07-20 08:59:002023-07-20 09:00:002023-07-20 09:01:
  ↪002023-07-20 09:02:002023-07-20 09:03:002023-07-20 09:04:002023-07-20 09:05:
  ↪002023-07-20 09:06:002023-07-20 09:07:002023-07-20 09:08:002023-07-20 09:09:
  ↪002023-07-20 09:10:002023-07-20 09:11:002023-07-20 09:12:002023-07-20 09:13:
  ↪002023-07-20 09:14:002023-07-20 09:15:002023-07-20 09:16:002023-07-20 09:17:
  ↪002023-07-20 09:18:002023-07-20 09:19:002023-07-20 09:20:002023-07-20 09:21:
  ↪002023-07-20 09:22:002023-07-20 09:23:002023-07-20 09:24:002023-07-20 09:25:
  ↪002023-07-20 09:26:002023-07-20 09:27:002023-07-20 09:28:002023-07-20 09:29:
  ↪002023-07-20 09:30:002023-07-20 09:31:002023-07-20 09:32:002023-07-20 09:33:
  ↪002023-07-20 09:34:002023-07-20 09:35:002023-07-20 09:36:002023-07-20 09:37:
  ↪002023-07-20 09:38:002023-07-20 09:39:002023-07-20 09:40:002023-07-20 09:41:
  ↪002023-07-20 09:42:002023-07-20 09:43:002023-07-20 09:44:002023-07-20 09:45:
  ↪002023-07-20 09:46:002023-07-20 09:47:002023-07-20 09:48:002023-07-20 09:49:
  ↪002023-07-20 09:50:002023-07-20 09:51:002023-07-20 09:52:002023-07-20 09:53:
  ↪002023-07-20 09:54:002023-07-20 09:55:002023-07-20 09:56:002023-07-20 09:57:
  ↪002023-07-20 09:58:002023-07-20 09:59:002023-07-20 10:00:002023-07-20 10:01:
  ↪002023-07-20 10:02:002023-07-20 10:03:002023-07-20 10:04:002023-07-20 10:05:
  ↪002023-07-20 10:06:002023-07-20 10:07:002023-07-20 10:08:002023-07-20 10:09:
  ↪002023-07-20 10:10:002023-07-20 10:11:002023-07-20 10:12:002023-07-20 10:13:
  ↪002023-07-20 10:14:002023-07-20 10:15:002023-07-20 10:16:002023-07-20 10:17:
  ↪002023-07-20 10:18:002023-07-20 10:19:002023-07-20 10:20:002023-07-20 10:21:
  ↪002023-07-20 10:22:002023-07-20 10:23:002023-07-20 10:24:002023-07-20 10:25:
  ↪002023-07-20 10:26:002023-07-20 10:27:002023-07-20 10:28:002023-07-20 10:29:
  ↪002023-07-20 10:30:002023-07-20 10:31:002023-07-20 10:32:002023-07-20 10:33:
  ↪002023-07-20 10:34:002023-07-20 10:35:002023-07-20 10:36:002023-07-20 10:37:
  ↪002023-07-20 10:38:002023-07-20 10:39:002023-07-20 10:40:002023-07-20 10:41:
  ↪002023-07-20 10:42:002023-07-20 10:43:002023-07-20 10:44:002023-07-20 10:45:
  ↪002023-07-20 10:46:002023-07-20 10:47:002023-07-20 10:48:002023-07-20 10:49:
  ↪002023-07-20 10:50:002023-07-20 10:51:002023-07-20 10:52:002023-07-20 10:53:
  ↪002023-07-20 10:54:002023-07-20 10:55:002023-07-20 10:56:002023-07-20 10:57:
  ↪002023-07-20 10:58:002023-07-20 10:59:002023-07-20 11:00:002023-07-20 11:01:
  ↪002023-07-20 11:02:002023-07-20 11:03:002023-07-20 11:04:002023-07-20 11:05:
  ↪002023-07-20 11:06:002023-07-20 11:07:002023-07-20 11:08:002023-07-20 11:09:
  ↪002023-07-20 11:10:002023-07-20 11:11:002023-07-20 11:12:002023-07-20 11:13:
  ↪002023-07-20 11:14:002023-07-20 11:15:002023-07-20 11:16:002023-07-20 11:17:
  ↪002023-07-20 11:18:002023-07-20 11:19:002023-07-20 11:20:002023-07-20 11:21:
  ↪002023-07-20 11:22:002023-07-20 11:23:002023-07-20 11:24:002023-07-20 11:25:
  ↪002023-07-20 11:26:002023-07-20 11:27:002023-07-20 11:28:002023-07-20 11:29:
  ↪002023-07-20 11:30:002023-07-20 11:31:002023-07-20 11:32:002023-07-20 11:33:
  ↪002023-07-20 11:34:002023-07-20 11:35:002023-07-20 11:36:002023-07-20 11:37:
  ↪002023-07-20 11:38:002023-07-20 11:39:002023-07-20 11:40:002023-07-20 11:41:
  ↪002023-07-20 11:42:002023-07-20 11:43:002023-07-20 11:44:002023-07-20 11:45:
  ↪002023-07-20 11:46:002023-07-20 11:47:002023-07-20 11:48:002023-07-20 11:49:
  ↪002023-07-20 11:50:002023-07-20 11:51:002023-07-20 11:52:002023-07-20 11:53:
  ↪002023-07-20 11:54:002023-07-20 11:55:002023-07-20 11:56:002023-07-20 11:57:
  ↪002023-07-20 11:58:002023-07-20 11:59:002023-07-20 12:00:002023-07-20 12:01:
  ↪002023-07-20 12:02:002023-07-20 12:03:002023-07-20 12:04:002023-07-20 12:05:
  ↪002023-07-20 12:06:002023-07-20 12:07:002023-07-20 12:08:002023-07-20 12:09:
  ↪002023-07-20 12:10:002023-07-20 12:11:002023-07-20 12:12:002023-07-20 12:13:
  ↪002023-07-20 12:14:002023-07-20 12:15:002023-07-20 12:16:002023-07-20 12:17:
  ↪002023-07-20 12:18:002023-07-20 12:19:002023-07-20 12:20:002023-07-20 12:21:
  ↪002023-07-20 12:22:002023-07-20 12:23:002023-07-20 12:24:002023-07-20 12:25:
  ↪002023-07-20 12:26:002023-07-20 12:27:002023-07-20 12:28:002023-07-20 12:29:
  ↪002023-07-20 12:30:002023-07-20 12:31:002023-07-20 12:32:002023-07-20 12:33:
  ↪002023-07-20 12:34:002023-07-20 12:35:002023-07-20 12:36:002023-07-20 12:37:
  ↪002023-07-20 12:38:002023-07-20 12:39:002023-07-20 12:40:002023-07-20 12:41:
  ↪002023-07-20 12:42:002023-07-20 12:43:002023-07-20 12:44:002023-07-20 12:45:
  ↪002023-07-20 12:46:002023-07-20 12:47:002023-07-20 12:48:002023-07-20 12:49:
```

:-((timestamp was not loaded as a timestamp!!!

Try to load it properly

```
[11]: X = pd.read_csv("Data03.csv",parse_dates=[0])
```

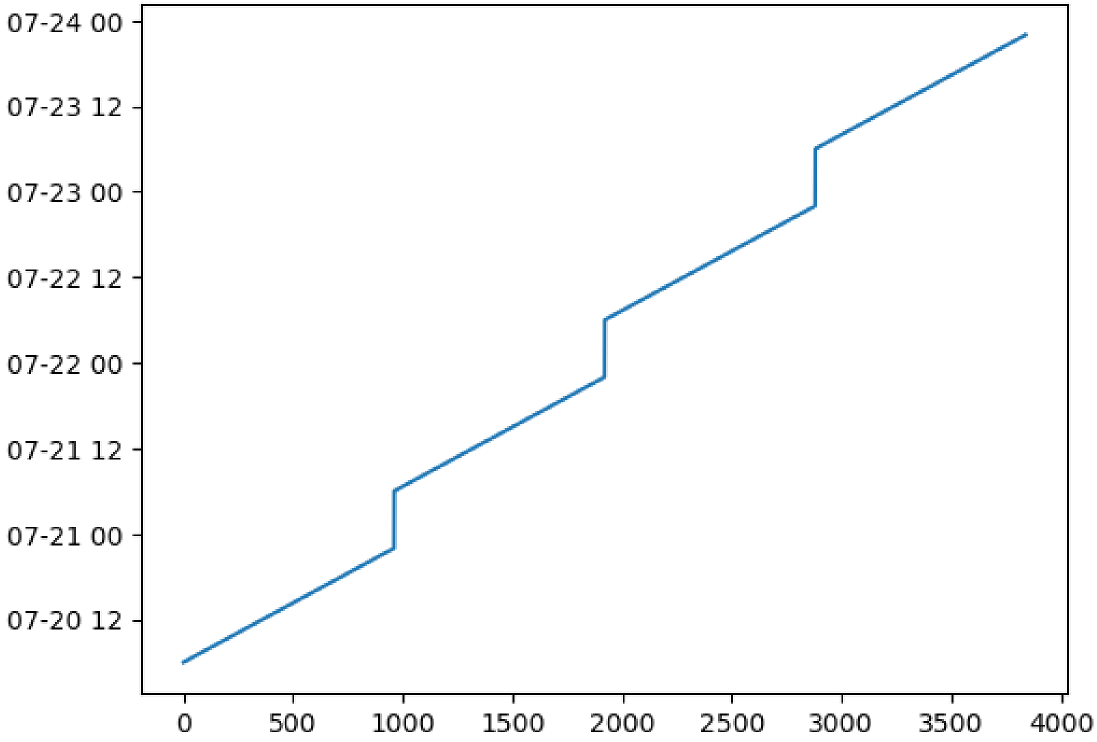
```
[12]: X.describe()
```

```
[12]:                  timestamp           f1           f2
      count                 3840  3840.000000  3840.000000
      mean   2023-07-22 01:59:30    -4.861466    -4.854412
      min    2023-07-20 06:00:00    -8.740101    -8.392300
      25%    2023-07-21 03:59:45    -5.704431    -5.649969
      50%    2023-07-22 01:59:30    -4.999363    -5.013370
      75%    2023-07-22 23:59:15    -4.317683    -4.317863
      max    2023-07-23 21:59:00     7.865602     7.463494
      std                    NaN     1.604436     1.564089
```

The timestamps are in the range 2023-07-20 06:00:00 to 2023-07-23 21:59:00. This is the timespan covered by the data.

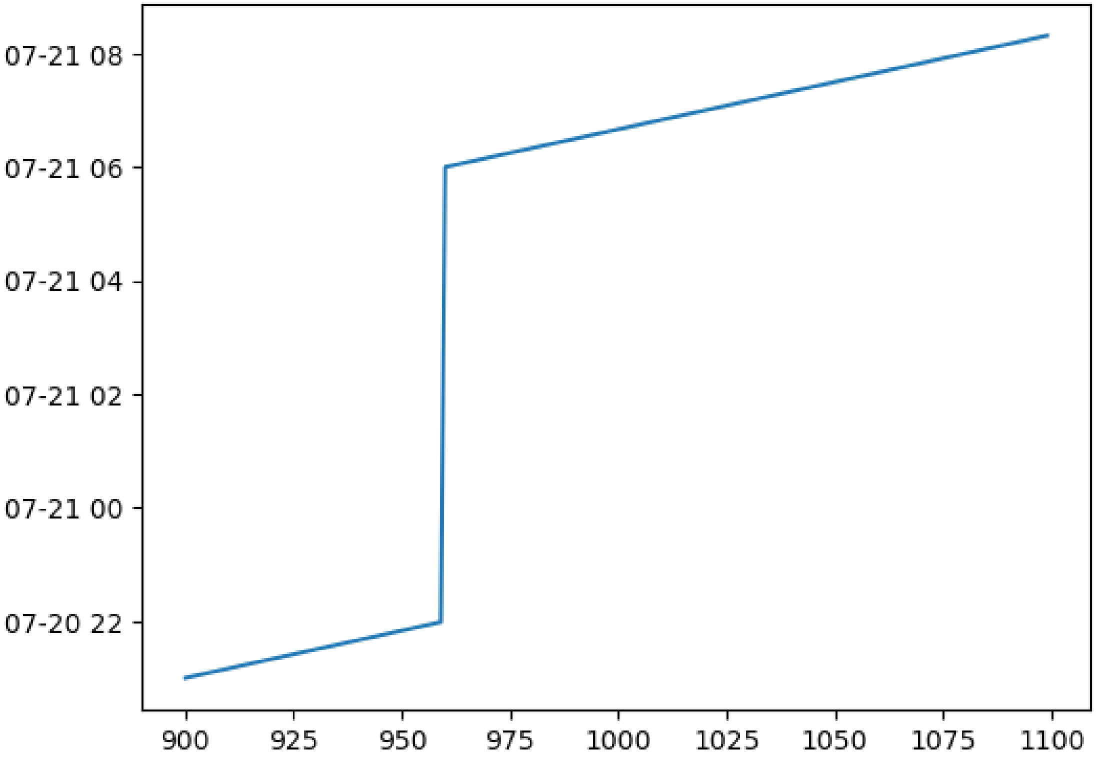
```
[13]: plt.plot(X.timestamp)
```

```
[13]: [<matplotlib.lines.Line2D at 0x7f81139916f0>]
```



[14]:
```
plt.plot(X.timestamp[900:1100])
```

[14]:
```
[<matplotlib.lines.Line2D at 0x7f8113996560>]
```



[16]:
```
X.timestamp[950:970]
```

[16]:
```
950   2023-07-20 21:50:00
951   2023-07-20 21:51:00
952   2023-07-20 21:52:00
953   2023-07-20 21:53:00
954   2023-07-20 21:54:00
955   2023-07-20 21:55:00
956   2023-07-20 21:56:00
957   2023-07-20 21:57:00
958   2023-07-20 21:58:00
959   2023-07-20 21:59:00
960   2023-07-21 06:00:00
961   2023-07-21 06:01:00
962   2023-07-21 06:02:00
963   2023-07-21 06:03:00
964   2023-07-21 06:04:00
965   2023-07-21 06:05:00
```

```
966    2023-07-21 06:06:00
967    2023-07-21 06:07:00
968    2023-07-21 06:08:00
969    2023-07-21 06:09:00
Name: timestamp, dtype: datetime64[ns]
```

```
[18]: X.timestamp[950:970].diff().max()
```

```
[18]: Timedelta('0 days 08:01:00')
```

```
[26]: X.timestamp.diff()[X.timestamp.diff()/pd.Timedelta(days=1) > 0.5/24]/pd.
       ↪Timedelta(days=1)
```

```
[26]: 960     0.334028
      1920    0.334028
      2880    0.334028
      Name: timestamp, dtype: float64
```

Yes, we have three breaks of 8 hours each –> See plot.

```
[27]: X.timestamp[X.timestamp.diff()/pd.Timedelta(days=1) > 0.5/24]
```

```
[27]: 960    2023-07-21 06:00:00
      1920   2023-07-22 06:00:00
      2880   2023-07-23 06:00:00
      Name: timestamp, dtype: datetime64[ns]
```

```
[31]: X.timestamp[X.timestamp.index[X.timestamp.diff()/pd.Timedelta(days=1) > 0.5/24]␣
       ↪- 1]
```

```
[31]: 959    2023-07-20 21:59:00
      1919   2023-07-21 21:59:00
      2879   2023-07-22 21:59:00
      Name: timestamp, dtype: datetime64[ns]
```

1.2 Q6b

```
[32]: from sklearn.cluster import KMeans
```

```
[41]: clusterer = KMeans(n_clusters = 2)
```

```
[42]: X = X.drop(['timestamp'],axis = 1)
      X.head()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[42], line 1
```

```
----> 1 X = X.drop(['timestamp'],axis = 1)
      2 X.head()

File /opt/conda/lib/python3.10/site-packages/pandas/core/frame.py:5347, in␣
 ↪DataFrame.drop(self, labels, axis, index, columns, level, inplace, errors)
   5199 def drop(
   5200     self,
   5201     labels: IndexLabel | None = None,
   (…)
   5208     errors: IgnoreRaise = "raise",
   5209 ) -> DataFrame | None:
   5210     """
   5211     Drop specified labels from rows or columns.
   5212
   (…)
   5345             weight  1.0     0.8
   5346     """
-> 5347     return super().drop(
   5348         labels=labels,
   5349         axis=axis,
   5350         index=index,
   5351         columns=columns,
   5352         level=level,
   5353         inplace=inplace,
   5354         errors=errors,
   5355     )

File /opt/conda/lib/python3.10/site-packages/pandas/core/generic.py:4711, in␣
 ↪NDFrame.drop(self, labels, axis, index, columns, level, inplace, errors)
   4709 for axis, labels in axes.items():
   4710     if labels is not None:
-> 4711         obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   4713 if inplace:
   4714     self._update_inplace(obj)

File /opt/conda/lib/python3.10/site-packages/pandas/core/generic.py:4753, in␣
 ↪NDFrame._drop_axis(self, labels, axis, level, errors, only_slice)
   4751         new_axis = axis.drop(labels, level=level, errors=errors)
   4752     else:
-> 4753         new_axis = axis.drop(labels, errors=errors)
   4754     indexer = axis.get_indexer(new_axis)
   4756 # Case for non-unique axis
   4757 else:

File /opt/conda/lib/python3.10/site-packages/pandas/core/indexes/base.py:6992,␣
 ↪in Index.drop(self, labels, errors)
   6990 if mask.any():
   6991     if errors != "ignore":
```

```
-> 6992             raise KeyError(f"{labels[mask].tolist()} not found in axis")
   6993         indexer = indexer[~mask]
   6994 return self.delete(indexer)

KeyError: "['timestamp'] not found in axis"
```

[43]:
```
clusterer.fit(X)
y = clusterer.predict(X)
```

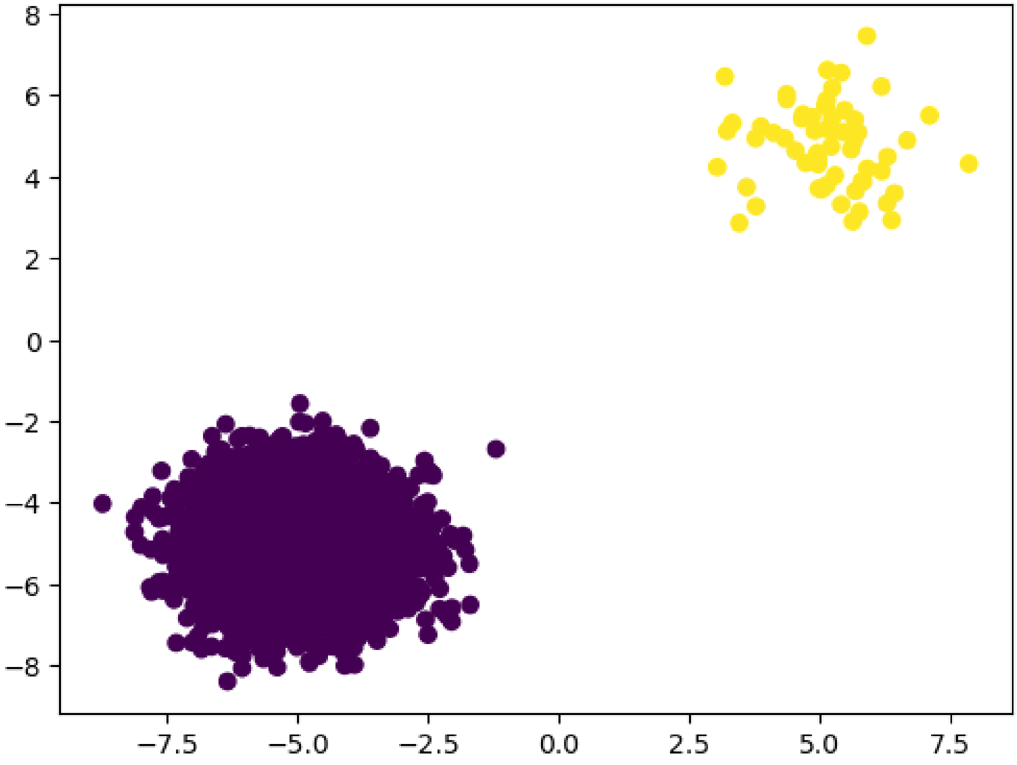
```
/opt/conda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

[44]:
```
y
```

[44]:
```
array([0, 0, 0, …, 0, 0, 0], dtype=int32)
```

[45]:
```
plt.scatter(X.f1,X.f2,c=y)
```

[45]:
```
<matplotlib.collections.PathCollection at 0x7f8111658a90>
```



A scatter plot indicates two distinct clusters: one at the bottom left, one at the top right ==> we look for two clusters

```
[49]: y = pd.Series(y)
      y.value_counts()
```

```
[49]: 0    3780
      1      60
      Name: count, dtype: int64
```

```
[53]: X.timestamp[y == 1]
```

```
[53]: 193     2023-07-20 09:13:00
      198     2023-07-20 09:18:00
      246     2023-07-20 10:06:00
      327     2023-07-20 11:27:00
      332     2023-07-20 11:32:00
      415     2023-07-20 12:55:00
      646     2023-07-20 16:46:00
      794     2023-07-20 19:14:00
      807     2023-07-20 19:27:00
      821     2023-07-20 19:41:00
      928     2023-07-20 21:28:00
      962     2023-07-21 06:02:00
      968     2023-07-21 06:08:00
      981     2023-07-21 06:21:00
      1100    2023-07-21 08:20:00
      1144    2023-07-21 09:04:00
      1239    2023-07-21 10:39:00
      1338    2023-07-21 12:18:00
      1379    2023-07-21 12:59:00
      1385    2023-07-21 13:05:00
      1470    2023-07-21 14:30:00
      1518    2023-07-21 15:18:00
      1572    2023-07-21 16:12:00
      1716    2023-07-21 18:36:00
      1741    2023-07-21 19:01:00
      1872    2023-07-21 21:12:00
      1876    2023-07-21 21:16:00
      1913    2023-07-21 21:53:00
      2015    2023-07-22 07:35:00
      2019    2023-07-22 07:39:00
      2087    2023-07-22 08:47:00
      2320    2023-07-22 12:40:00
      2379    2023-07-22 13:39:00
      2382    2023-07-22 13:42:00
      2432    2023-07-22 14:32:00
      2523    2023-07-22 16:03:00
```

```
2644    2023-07-22 18:04:00
2715    2023-07-22 19:15:00
2716    2023-07-22 19:16:00
2863    2023-07-22 21:43:00
2911    2023-07-23 06:31:00
2975    2023-07-23 07:35:00
3002    2023-07-23 08:02:00
3045    2023-07-23 08:45:00
3060    2023-07-23 09:00:00
3207    2023-07-23 11:27:00
3224    2023-07-23 11:44:00
3250    2023-07-23 12:10:00
3287    2023-07-23 12:47:00
3431    2023-07-23 15:11:00
3509    2023-07-23 16:29:00
3539    2023-07-23 16:59:00
3553    2023-07-23 17:13:00
3559    2023-07-23 17:19:00
3622    2023-07-23 18:22:00
3625    2023-07-23 18:25:00
3728    2023-07-23 20:08:00
3759    2023-07-23 20:39:00
3761    2023-07-23 20:41:00
3794    2023-07-23 21:14:00
Name: timestamp, dtype: object
```

```
[ ]:
```