Live CaseStudies 2025-11-12

November 12, 2025

1 Live Case Study 1 2025-11-12

Matriculation: 0815

[2]:
```
import pandas as pd

X = pd.read_csv("../PMCM_WS2425/Data02.csv")
```

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

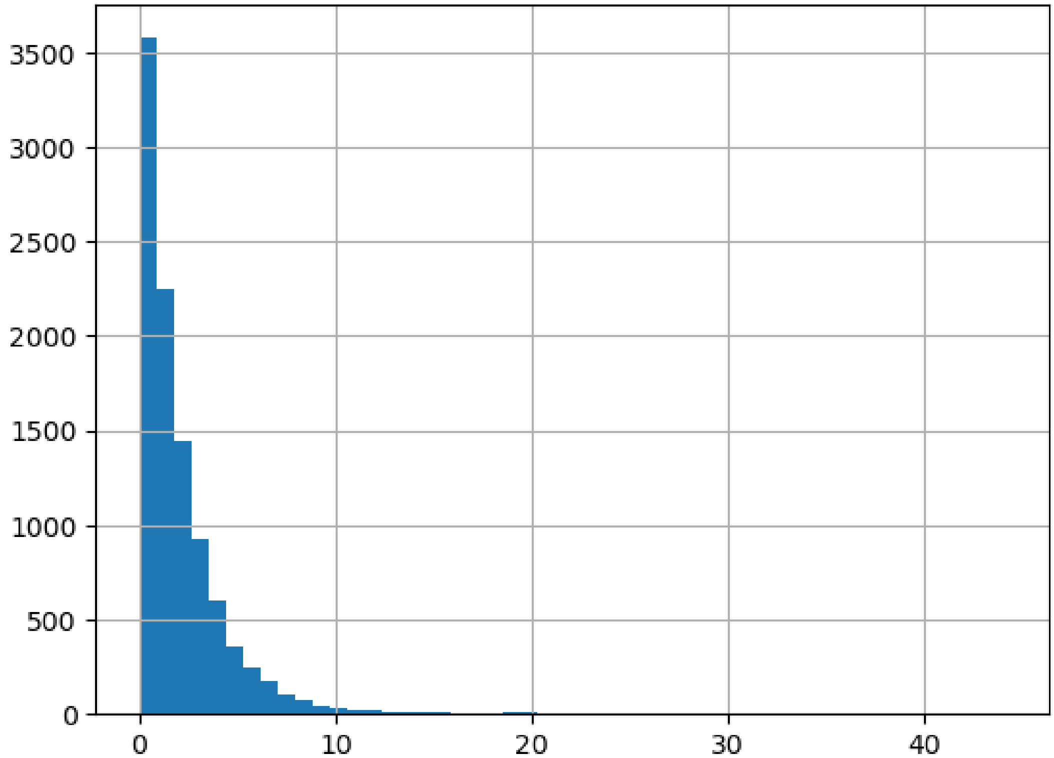
[6]:

	f1	f2	error
count	10000.000000	10000.000000	10000.000000
mean	-1.477134	2.914468	2.205596
std	1.079116	1.019233	2.832223
min	-5.973536	-1.874929	0.000158
25%	-2.199008	2.260771	0.587542
50%	-1.516600	2.912011	1.399142
75%	-0.808268	3.580329	2.859412
max	4.359184	6.409433	44.133879

1.1 Q5a

[4]:
```
import matplotlib.pyplot as plt
```

[8]:
```
plt.hist(X.error,bins=50)
plt.grid()
```



1.2 Q5b

[9]:
```
X[X['error']>6].count()
```

[9]:
```
f1       640
f2       640
error    640
dtype: int64
```

640 bad parts

[10]:
```
X[X['error']<=6].count()
```

[10]:
```
f1       9360
f2       9360
error    9360
dtype: int64
```

9360 good parts

[11]:
```
#Production yield = good / (good + bad)
9360 / (9360 + 640)
```

[11]: 0.936

1.3 Q5c

[14]:
```
# we have a classification problem
# most stupid predictor would predict all as good (majority class)
# accuracy = correct / total = tp + tn / (tp + tn + fp + fn)

acc_stupid = 9360 / (9360 + 640)
print(f"The baseline is an accuracy of {acc_stupid}")
```

```
The baseline is an accuracy of 0.936
```

[20]:
```
## alternative way
from sklearn.dummy import DummyClassifier
from sklearn.metrics import accuracy_score

clf = DummyClassifier(strategy="most_frequent")

y_true = X['error']<=6
xx = X[['f1', 'f2']]

clf.fit(xx,y_true)

y_pred = clf.predict(xx)

acc = accuracy_score(y_true,y_pred)
print(f"Baseline accuracy {acc}")
```

```
Baseline accuracy 0.936
```

1.4 Q5d

The dataset is unbalanced, because the good class is represented in 93.6 % of the instances and the bad class only in 6.4 %.

Firstly we have to split in train and test data. Then train on the train and test on the test :-)

Because of imbalance we have to make sure that we get bad instances in train and in test!

One way to do that would be a stratified split (not just a normal split). Or we could * augment data with more bad cases (but how?? :-(() * reduce number of good cases

1.5 Q6

[22]:
```
X = pd.read_csv("../PMCM_WS2425/Data03.csv")
```

[25]:
```
X.head()
```

[25]:

	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

[26]:
```
X['timestamp_parsed'] = pd.to_datetime(X.timestamp)
```

[27]:
```
X.head()
```

[27]:

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

[28]:
```
X.describe()
```

[28]:

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

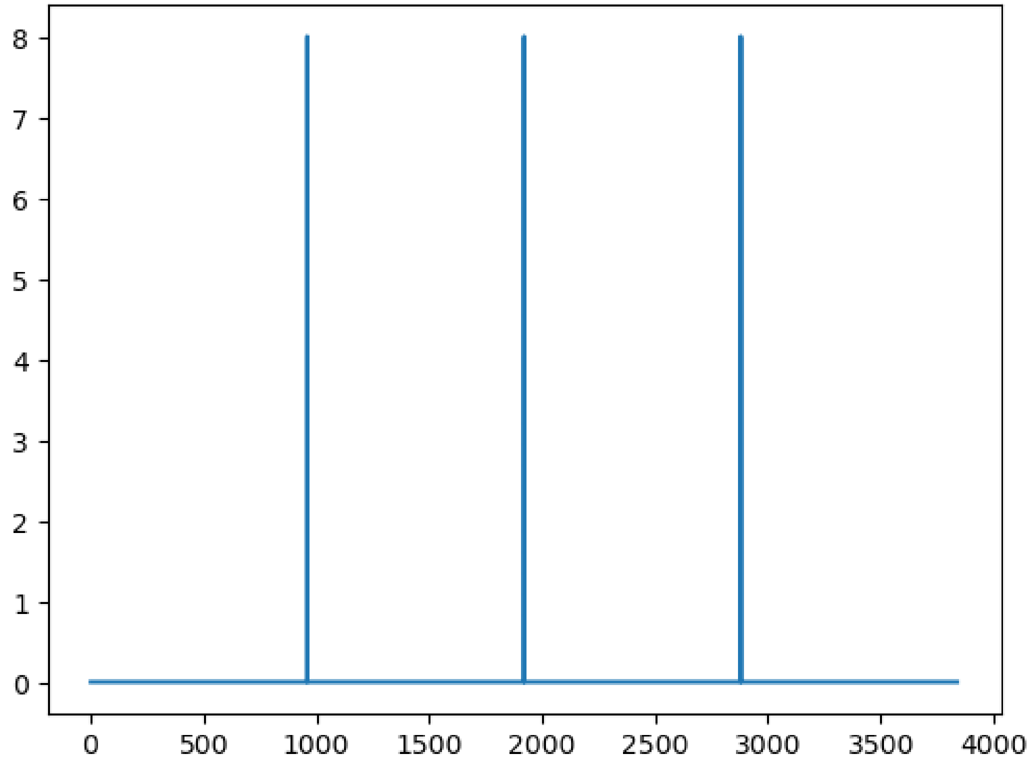
[29]:
```
print(f"Timestamp ranges from {X.timestamp_parsed.min()} to {X.timestamp_parsed.
↪max()}")
```

```
Timestamp ranges from 2023-07-20 06:00:00 to 2023-07-23 21:59:00
```

[30]:
```
deltas = X.timestamp_parsed.diff()/pd.Timedelta(hours=1)
```

[31]:
```
plt.plot(deltas)
```

[31]:
```
[<matplotlib.lines.Line2D at 0x7f4193e5b6d0>]
```



[37]:
```
X[deltas > 0.1].timestamp_parsed
```

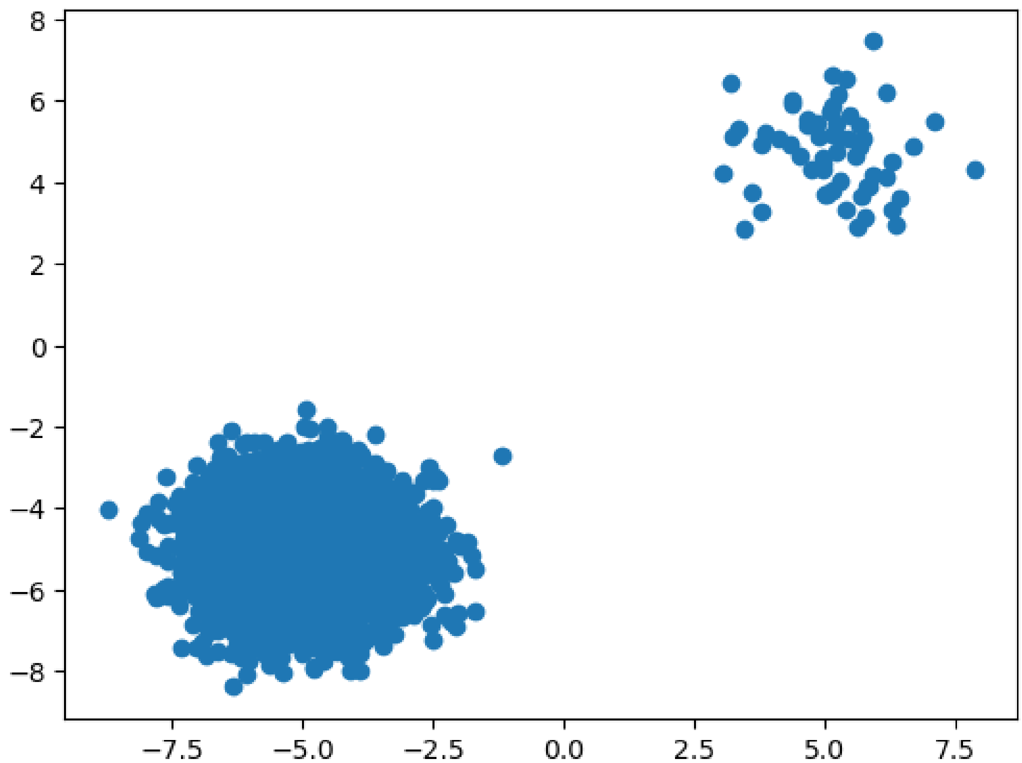
[37]:
```
960    2023-07-21 06:00:00
1920   2023-07-22 06:00:00
2880   2023-07-23 06:00:00
Name: timestamp_parsed, dtype: datetime64[ns]
```

Yes, we have three breaks of 8 h each. We found them by subtracting timestamp_i+1 - timestamp_i

1.6 Q6b

[38]:
```
plt.scatter(X.f1,X.f2)
```

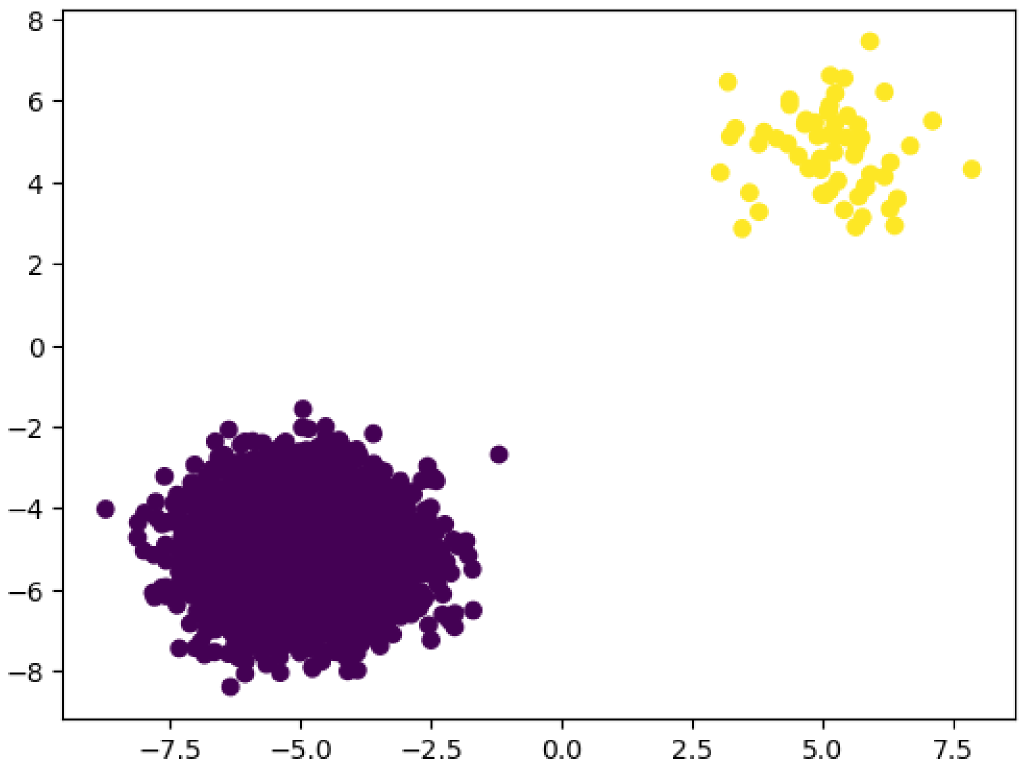
[38]:
```
<matplotlib.collections.PathCollection at 0x7f419370bcd0>
```



[41]:
```
from sklearn.cluster import KMeans

xx = X[['f1','f2']]

clusterer = KMeans(n_clusters=2)

clusterer.fit(xx)

y_cluster = pd.Series(clusterer.predict(xx))
```

[45]:
```
plt.scatter(xx.f1,xx.f2,c=y_cluster)
```

[45]: <matplotlib.collections.PathCollection at 0x7f4193e48880>



[46]:
```
y_cluster[y_cluster == 0].count()
```

[46]: np.int64(3780)

[47]:
```
y_cluster[y_cluster == 1].count()
```

[47]: np.int64(60)

[48]:
```
X[y_cluster == 1].timestamp_parsed
```

[48]:
```
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_parsed, dtype: datetime64[ns]
```

[ ]:

```

```