Credit Card Fraud Detection: Using a Generative Adversarial Network (GAN) to Generate Synthetic Data to Resolve Class Imbalance

Yu chien (Calvin) Ma

- In certain datasets, like credit card fraud detection, one class (e.g., non-fraudulent transactions) often greatly outnumbers the other (fraudulent transactions), leading to class imbalance. This imbalance can introduce biases when training machine learning models, as the model may become overly biased toward predicting the majority class.
- In this project, I first applied Principal Component Analysis (PCA) to reduce the dimensionality of the dataset, which originally contained 29 features, down to 2 components. This dimensionality reduction helps to visualize the structure of the data, revealing how fraudulent transactions compare to genuine ones in a lower-dimensional space.
- Next, I implemented a Generative Adversarial Network (GAN) to generate synthetic fraudulent transactions. The GAN consists of two components: a Generator that creates new, synthetic fraudulent data, and a Discriminator that evaluates the authenticity of the generated data against real fraudulent transactions. By training these two components in opposition, the GAN is able to generate increasingly realistic fraudulent transactions.
- Finally, I visualized the comparison between the real and synthetic fraudulent transactions across the 28 features. This allowed me to analyze how each feature behaves in both the genuine and synthetic fraud samples, providing insights into the characteristics of fraudulent transactions and how well the GAN can replicate those characteristics.

Importing the Dataset

In [80]:

```python
import numpy as np
import pandas as pd

# Importing neural network modules
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, BatchNormalization, LeakyR
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.initializers import RandomNormal
# Importing some machine learning modules
```

```python
from sklearn.utils import shuffle
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
# Import data visualization modules
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
```

Check the data shape

In [82]:

```python
data = pd.read_csv("Creditcard_dataset.csv")
data.head()
```

Out[82]:

	Time	V1	V2	V3	V4	V5	V6	V7
0	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609
1	7	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631
2	10	1.449044	-1.176339	0.913860	-1.375667	-1.971383	-0.629152	-1.423236
3	10	0.384978	0.616109	-0.874300	-0.094019	2.924584	3.317027	0.470455
4	11	1.069374	0.287722	0.828613	2.712520	-0.178398	0.337544	-0.096717

5 rows × 31 columns

Display number of genuine and fraudulent records we have (0 means genuine, and 1 means fraudulent)

In [8]:

```python
data.Class.value_counts()
```

Out[8]:

```
Class
0    50000
1      492
Name: count, dtype: int64
```

Data Preprocessing and Exploration

- Removing all the rows with `Nan` values
- Removing `Time` column
- Feature Scaling `Amount` column
- Split the data into features and labels
- Data Exploration

Removing the rows `Nan` values in the dataset

In [11]:

```python
data = data.dropna()
```

Removing Time column

In [13]:

```python
data = data.drop(axis=1, columns = "Time")
```

Feature Scaling of Amount column

In [15]:

```python
data
```

Out[15]:

	V1	V2	V3	V4	V5	V6	V7
0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609
1	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631
2	1.449044	-1.176339	0.913860	-1.375667	-1.971383	-0.629152	-1.423236
3	0.384978	0.616109	-0.874300	-0.094019	2.924584	3.317027	0.470455
4	1.069374	0.287722	0.828613	2.712520	-0.178398	0.337544	-0.096717
...	...	...	...	...	...	...	...
50487	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850
50488	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170
50489	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739
50490	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002
50491	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050

50492 rows × 30 columns

In [16]:

```python
# to scale the "Amount" column to match the other features as large, unscale

scaler = StandardScaler()
data["Amount"] = scaler.fit_transform(data[["Amount"]])
```

In [17]:

```python
data.head()
```

Out[17]:

	V1	V2	V3	V4	V5	V6	V7	
0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.37
1	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.80
2	1.449044	-1.176339	0.913860	-1.375667	-1.971383	-0.629152	-1.423236	0.04
3	0.384978	0.616109	-0.874300	-0.094019	2.924584	3.317027	0.470455	0.53
4	1.069374	0.287722	0.828613	2.712520	-0.178398	0.337544	-0.096717	0.11

5 rows × 30 columns

Splitting the genuine and fraud records into separate dataframes

In [19]:

```
data_fraudulent = data[data.Class == 1]
data_genuine = data[data.Class == 0]
```

In [20]:

```
data_fraudulent.shape
```

Out[20]:

```
(492, 30)
```

In [21]:

```
data_genuine.shape
```

Out[21]:

```
(50000, 30)
```

Split the data into features and labels

In [23]:

```
X = data.drop(axis = 1, columns = "Class")
Y = data.Class
```

Data Exploration

- Apply PCA to reduce the dimensionality of features `X` into two dimensions
- Reduce from 29 to 2 dimensions
- Use a scatter plot to visualize our data

In [25]:

```
pca = PCA(2)
df = pd.DataFrame(pca.fit_transform(X))
```

In [26]:

```
df.head()
```

Out[26]:

	0	1
0	0.447840	-1.197485
1	0.582393	-0.258062
2	0.939390	0.728299
3	0.630766	0.499103
4	0.536287	1.055403

In [27]:

```
df["label"] = Y
```

In [28]:

```
df.head()
```

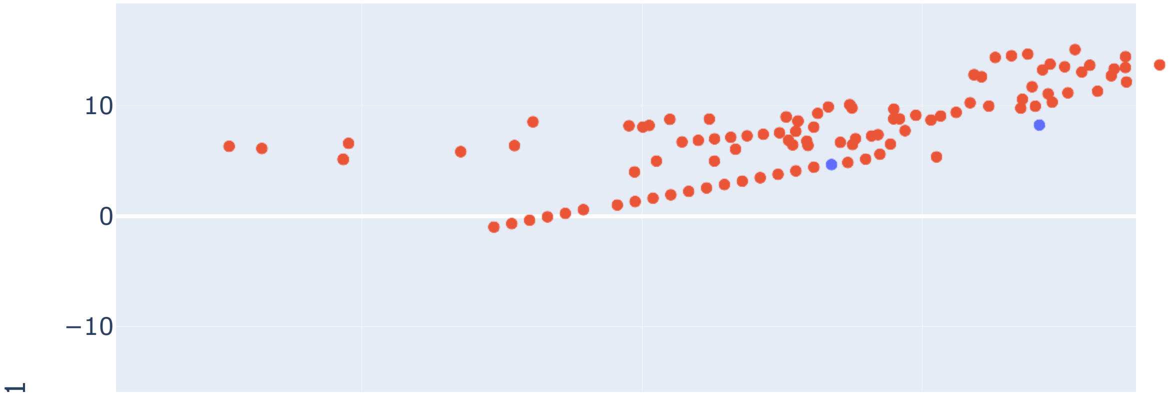
Out[28]:

	0	1	label
0	0.447840	-1.197485	0
1	0.582393	-0.258062	0
2	0.939390	0.728299	0
3	0.630766	0.499103	0
4	0.536287	1.055403	0

Plotting the two PCA features to visualize the genuine and fradulent transcations

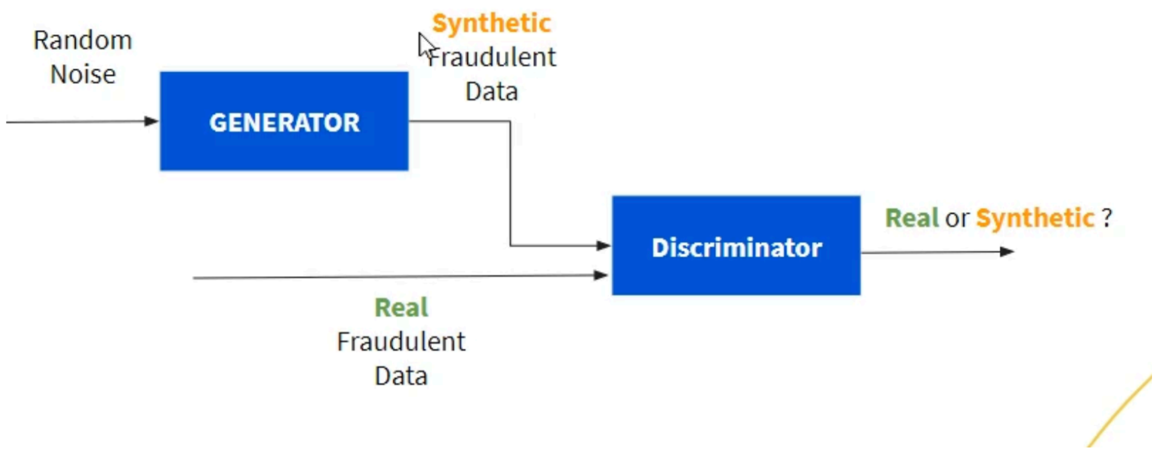
In [30]:

```
px.scatter(df, x=0, y=1, color = df.label.astype(str))
```



Building the Generator Model

Generative Adversarial Networks



Write a method to create the Generator model architecture

In [34]:
```
def build_generator():
    model = Sequential()
    model.add(Dense(32, activation = "relu", input_dim = 29, kernel_initiali
    model.add(BatchNormalization())
    model.add(Dense(64, activation = "relu"))
    model.add(BatchNormalization())
    model.add(Dense(128, activation = "relu"))
    model.add(BatchNormalization())

    #output later
    model.add(Dense(29, activation = "relu"))
    model.compile(optimizer = "adam", loss = "binary_crossentropy")
    model.summary()

    return model
```

In [35]:
```
build_generator()
```

```
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/core/dense.py:8
7: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Seq
uential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
```

Model: "sequential"

Layer (type)	Output Shape	Par
dense (Dense)	(None, 32)	
batch_normalization (BatchNormalization)	(None, 32)	
dense_1 (Dense)	(None, 64)	2
batch_normalization_1 (BatchNormalization)	(None, 64)	
dense_2 (Dense)	(None, 128)	8
batch_normalization_2 (BatchNormalization)	(None, 128)	
dense_3 (Dense)	(None, 29)	3

```
Total params: 16,029 (62.61 KB)
Trainable params: 15,581 (60.86 KB)
Non-trainable params: 448 (1.75 KB)
```

Out[35]:

```
<Sequential name=sequential, built=True>
```

Building the Discriminator Model

Write a method to create the Discriminator model architecture

In [38]:

```python
def build_discriminator():
    model = Sequential()
    model.add(Dense(128, activation = "relu", input_dim = 29, kernel_initial
    model.add(Dense(64, activation = "relu"))
    model.add(Dense(32, activation = "relu"))
    model.add(Dense(32, activation = "relu"))
    model.add(Dense(16, activation = "relu"))
    model.add(Dense(1, activation = "sigmoid"))
    model.compile(optimizer = "adam", loss = "binary_crossentropy")
    model.summary()
    return model
```

In [39]:

```python
build_discriminator()
```

```
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/core/dense.py:8
7: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Seq
uential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
```

Model: "sequential_1"

Layer (type)	Output Shape	Par
dense_4 (Dense)	(None, 128)	3
dense_5 (Dense)	(None, 64)	8
dense_6 (Dense)	(None, 32)	2
dense_7 (Dense)	(None, 32)	1
dense_8 (Dense)	(None, 16)	
dense_9 (Dense)	(None, 1)	

```
Total params: 15,777 (61.63 KB)
Trainable params: 15,777 (61.63 KB)
Non-trainable params: 0 (0.00 B)
```

Out[39]:
```
<Sequential name=sequential_1, built=True>
```

Combine Generator and Discriminator Models to Build the GAN

In [41]:
```python
def build_GAN(generator, discriminator):
    gan_input = Input(shape = (generator.input_shape[1],))
    x = generator(gan_input)
    gan_output = discriminator(x)
    gan = Model(gan_input, gan_output)
    gan.compile(optimizer="adam", loss="binary_crossentropy")
    gan.summary()

    # Freeze the discriminator from training
    discriminator.trainable = False
    return gan
```

Let's create a method that generates synthetic data using the Generator

In [43]:
```python
def generate_synthetic_data(generator, num_sample):
    noise = np.random.normal(0, 1, (num_sample, generator.input_shape[1]))
    fake_data = generator.predict(noise)
    return fake_data
```

Train and evaluate our GAN

- Defining some variables
- Creating our GAN
- Training the GAN
- Monitor the GAN performance using PCA

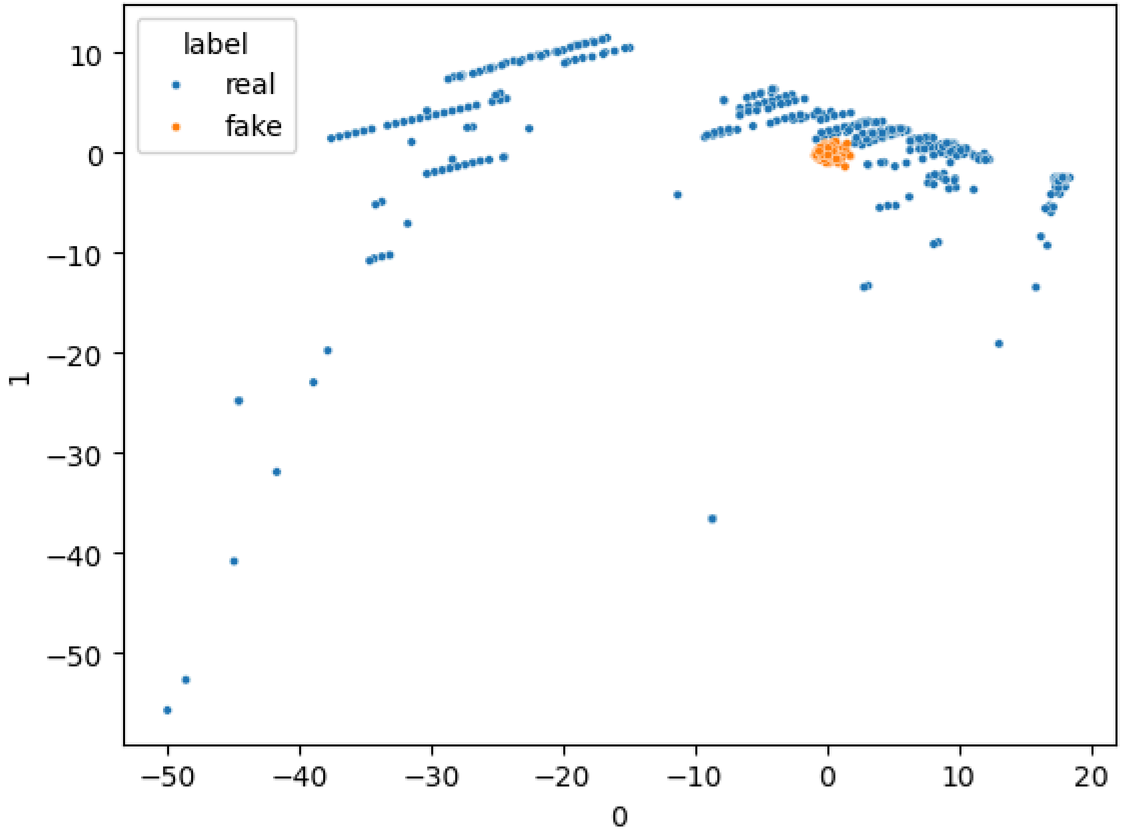
In [71]:
```python
def monitor_generator(generator):
    # Initialize a PCA (Principal Component Analysis) object with 2 componer
    pca = PCA(n_components=2)
    # Drop the 'Class' column from the fraud dataset to get real data
    real_fraud_data = data_fraudulent.drop("Class", axis=1)

    # Transform the real fraud data using PCA
    transformed_data_real = pca.fit_transform(real_fraud_data.values)

    # Create a DataFrame for the transformed real data and add a 'label' col
    df_real = pd.DataFrame(transformed_data_real)
    df_real['label'] = "real"

    # Generate synthetic fraud data using the provided generator and specify
    synthetic_fraud_data = generate_synthetic_data(generator, 492)

    # Transform the synthetic fraud data using PCA
    transformed_data_fake = pca.fit_transform(synthetic_fraud_data)

    # Create a DataFrame for the transformed fake data and add a 'label' col
    df_fake = pd.DataFrame(transformed_data_fake)
    df_fake["label"] = "fake"

    # Concatenate the real and fake data DataFrames
    df_combined = pd.concat([df_real, df_fake])

    # Create a scatterplot to visualize the data points, using the first and
    # and color points based on the 'label' column, with a size of 10
    plt.figure()
    sns.scatterplot(data=df_combined, x=0, y=1, hue='label', s=10)
    plt.show()
```

In [75]:

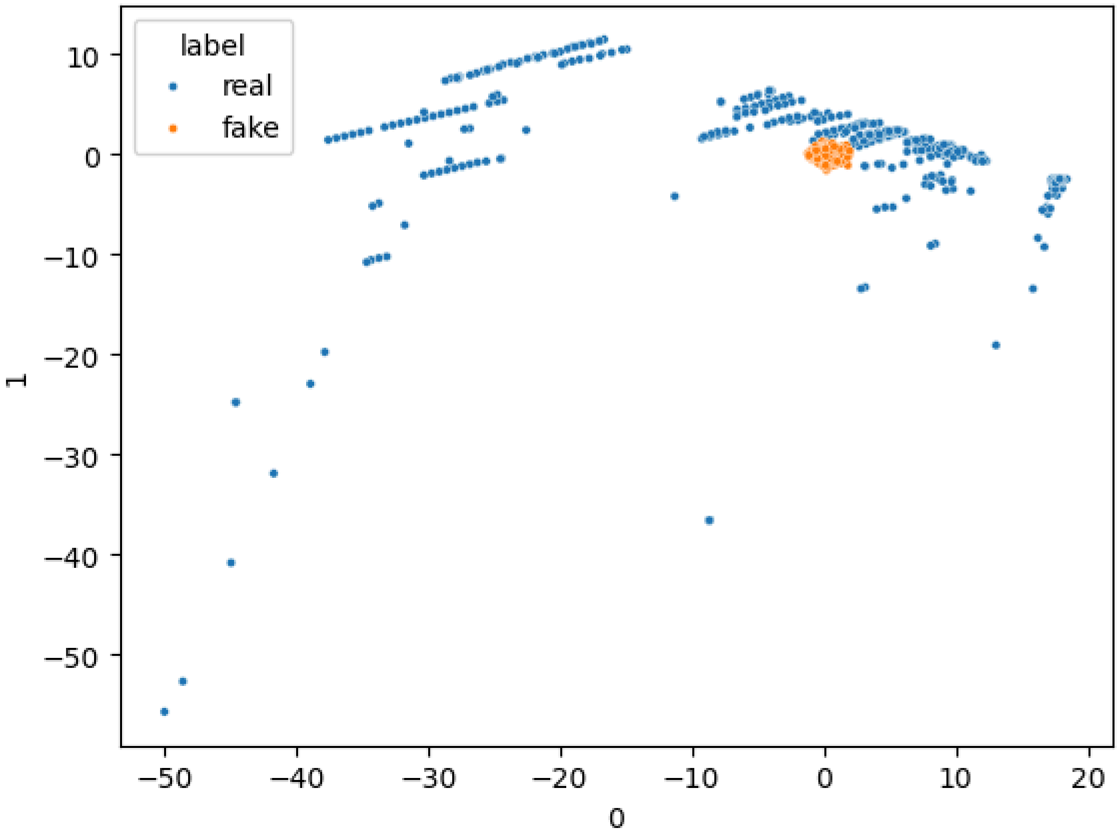
```
generator = build_generator()
discriminator = build_discriminator()
gan = build_GAN(generator, discriminator)
gan.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy

num_epochs = 100
batch_size = 64
half_batch = int(batch_size / 2)

for epoch in range(num_epochs):
    # Generate synthetic data
    X_fake = generate_synthetic_data(generator, half_batch)
    y_fake = np.zeros((half_batch, 1))

    # Sample a batch of real data
    X_real = data_fraudulent.drop("Class", axis=1).sample(n=half_batch, ranc
    y_real = np.ones((half_batch, 1))

    discriminator.compile(optimizer="adam", loss="binary_crossentropy")

    # Train the discriminator
    discriminator.trainable = True
    discriminator.train_on_batch(X_real, y_real)
    discriminator.train_on_batch(X_fake, y_fake)

    # Train the GAN (generator part)
    noise = np.random.normal(0, 1, (batch_size, 29))
    gan.train_on_batch(noise, np.ones((batch_size, 1)))

    if epoch%10 == 0:
        monitor_generator(generator)
```

```
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/core/dense.py:8
7: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Seq
uential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
```

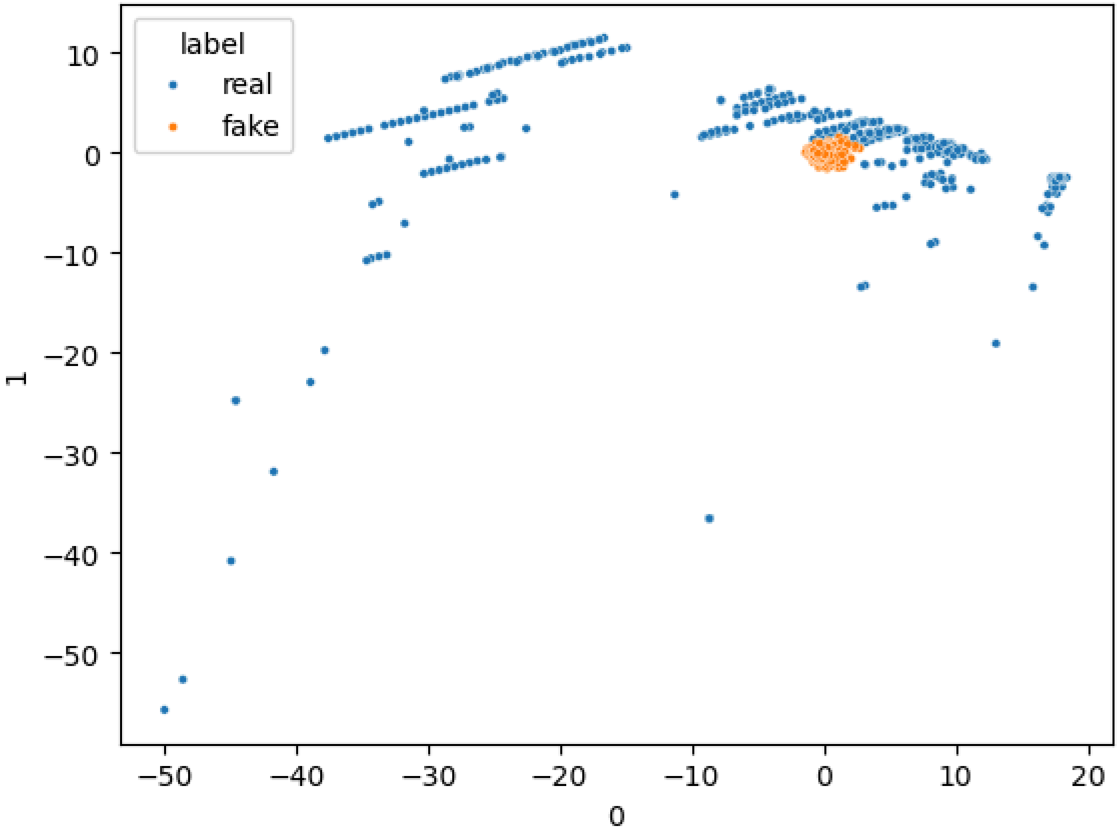
```
Model: "sequential_12"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
┃ Layer (type)                    ┃ Output Shape           ┃       Par
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
│ dense_60 (Dense)                │ (None, 32)             │
├─────────────────────────────────┼────────────────────────┼──────────
│ batch_normalization_18          │ (None, 32)             │
│ (BatchNormalization)            │                        │
├─────────────────────────────────┼────────────────────────┼──────────
│ dense_61 (Dense)                │ (None, 64)             │         2
├─────────────────────────────────┼────────────────────────┼──────────
│ batch_normalization_19          │ (None, 64)             │
│ (BatchNormalization)            │                        │
├─────────────────────────────────┼────────────────────────┼──────────
│ dense_62 (Dense)                │ (None, 128)            │         8
├─────────────────────────────────┼────────────────────────┼──────────
│ batch_normalization_20          │ (None, 128)            │
│ (BatchNormalization)            │                        │
├─────────────────────────────────┼────────────────────────┼──────────
│ dense_63 (Dense)                │ (None, 29)             │         3
└─────────────────────────────────┴────────────────────────┴──────────

 Total params: 16,029 (62.61 KB)

 Trainable params: 15,581 (60.86 KB)

 Non-trainable params: 448 (1.75 KB)

Model: "sequential_13"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
┃ Layer (type)                    ┃ Output Shape           ┃       Par
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
│ dense_64 (Dense)                │ (None, 128)            │         3
├─────────────────────────────────┼────────────────────────┼──────────
│ dense_65 (Dense)                │ (None, 64)             │         8
├─────────────────────────────────┼────────────────────────┼──────────
│ dense_66 (Dense)                │ (None, 32)             │         2
├─────────────────────────────────┼────────────────────────┼──────────
│ dense_67 (Dense)                │ (None, 32)             │         1
├─────────────────────────────────┼────────────────────────┼──────────
│ dense_68 (Dense)                │ (None, 16)             │
├─────────────────────────────────┼────────────────────────┼──────────
│ dense_69 (Dense)                │ (None, 1)              │
└─────────────────────────────────┴────────────────────────┴──────────

 Total params: 15,777 (61.63 KB)

 Trainable params: 15,777 (61.63 KB)
 Non-trainable params: 0 (0.00 B)
Model: "functional_96"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━
┃ Layer (type)                       ┃ Output Shape            ┃     Par
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━
│ input_layer_19 (InputLayer)        │ (None, 29)              │
├────────────────────────────────────┼─────────────────────────┼────────
│ sequential_12 (Sequential)         │ (None, 29)              │      16
├────────────────────────────────────┼─────────────────────────┼────────
│ sequential_13 (Sequential)         │ (None, 1)               │      15
└────────────────────────────────────┴─────────────────────────┴────────
 Total params: 31,806 (124.24 KB)
 Trainable params: 31,358 (122.49 KB)
 Non-trainable params: 448 (1.75 KB)
 1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
 16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```



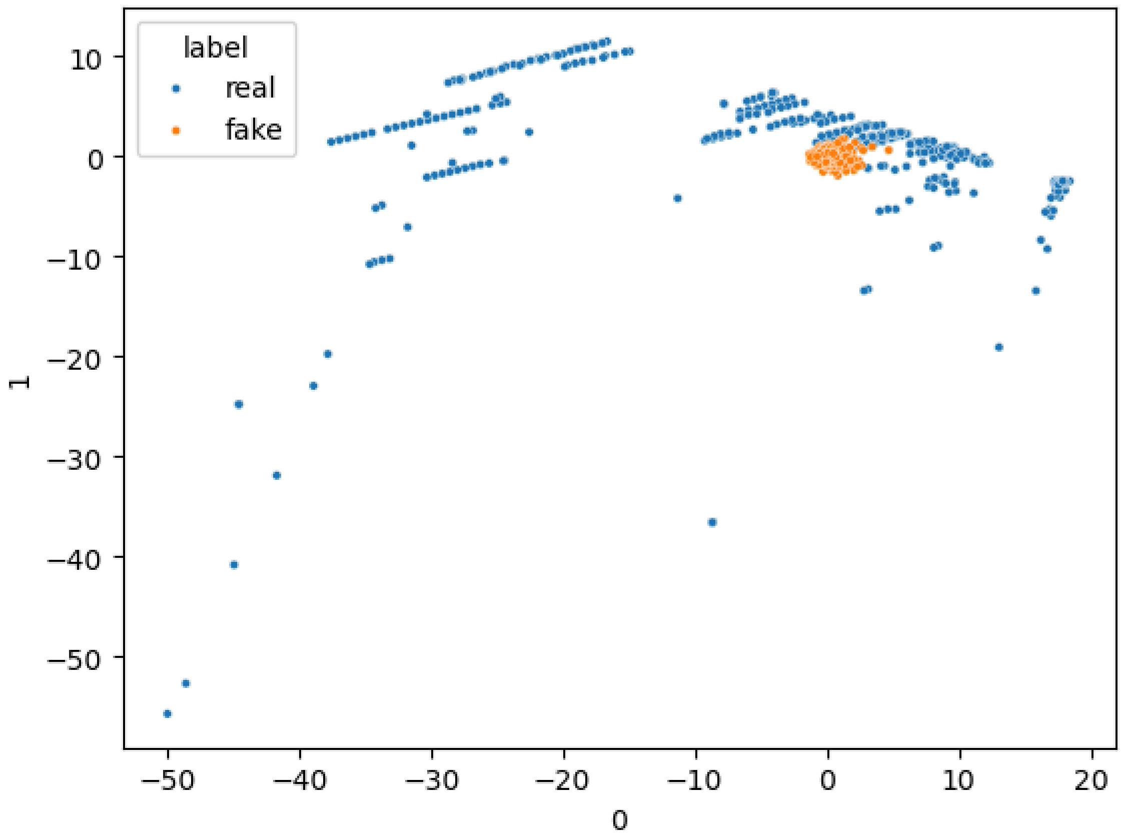
```
 1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
 1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
 1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
 1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
 1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
 1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
 1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
 1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
 1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
 1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
 16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 666us/step
```



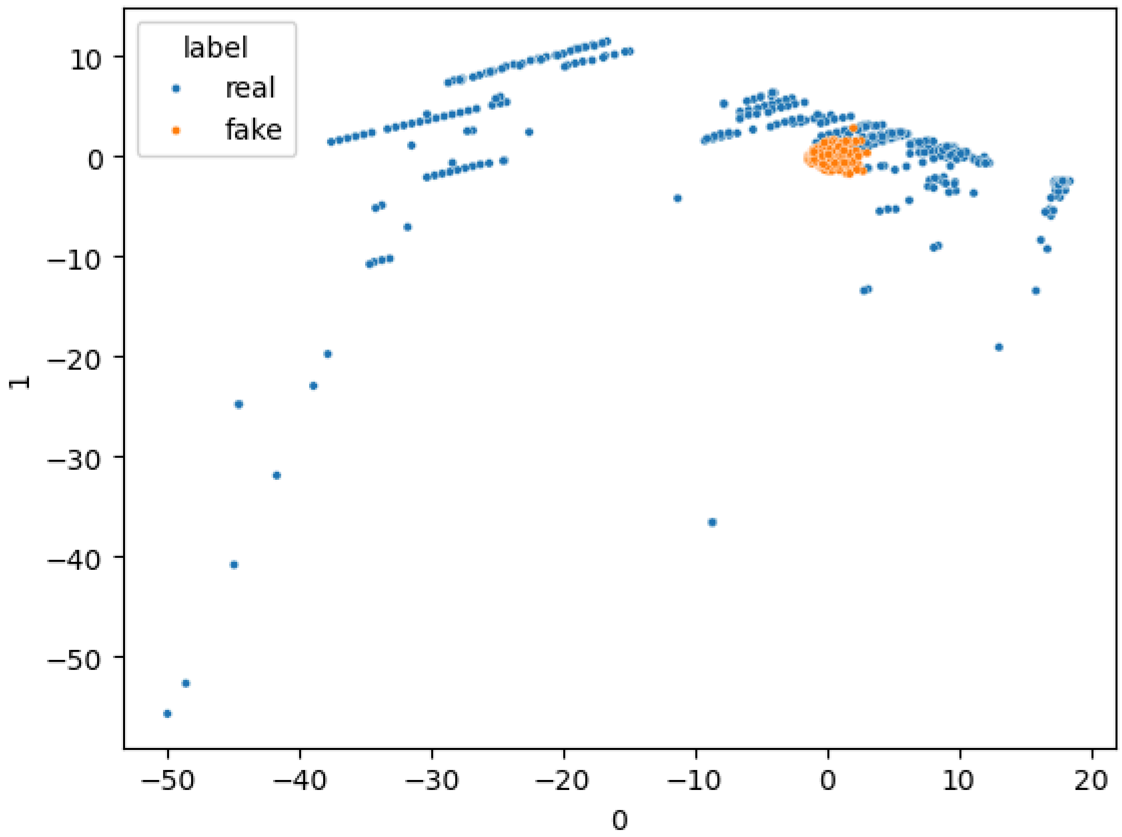
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 869us/step
```



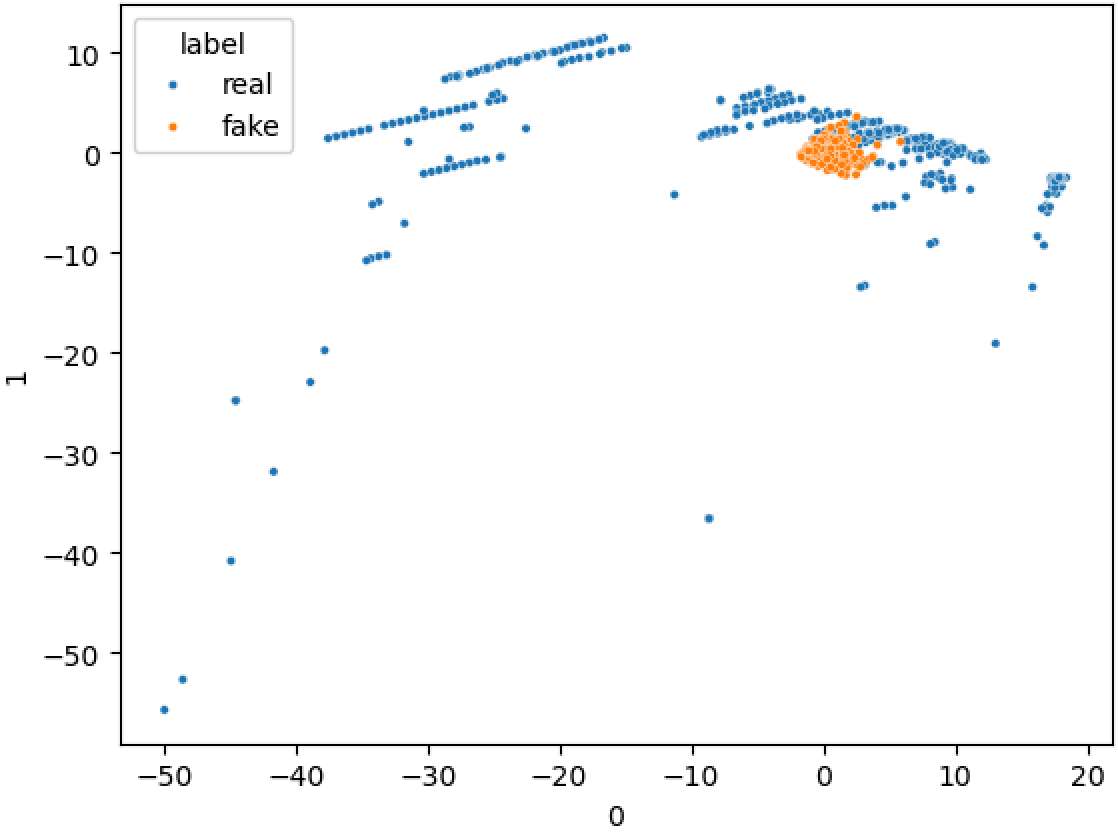
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 596us/step
```



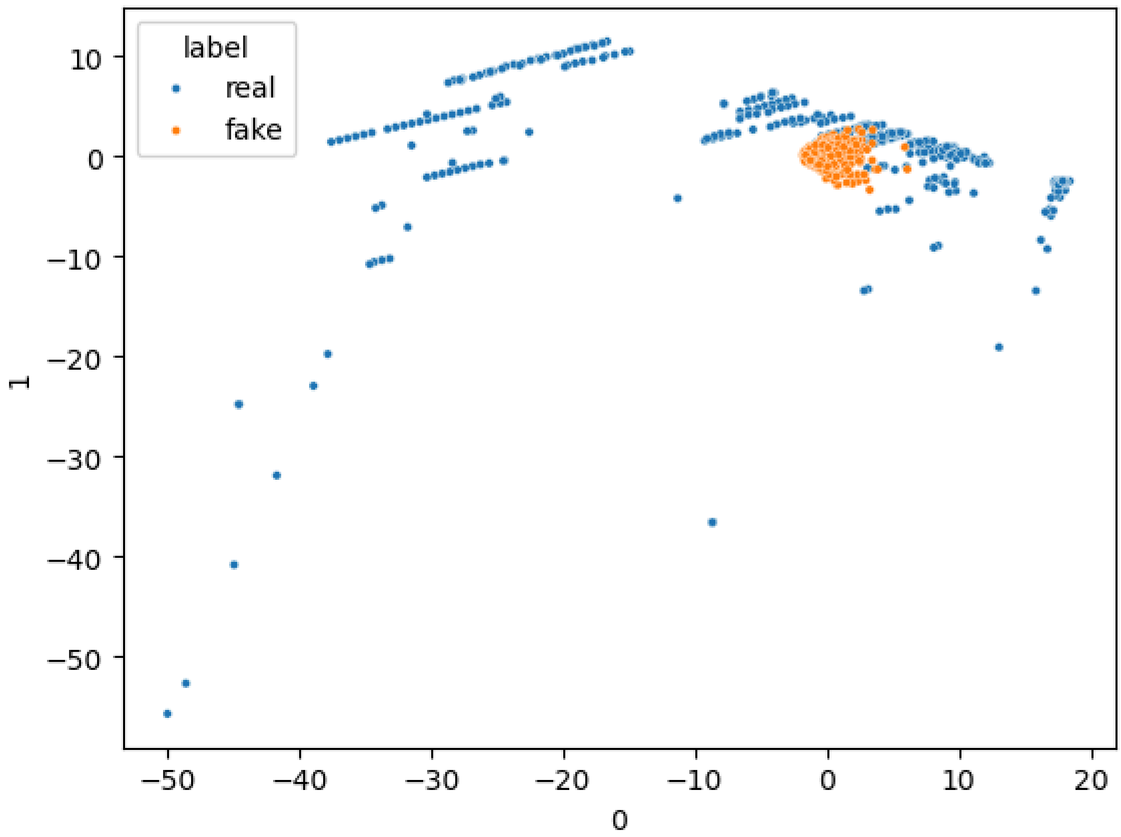
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 585us/step
```



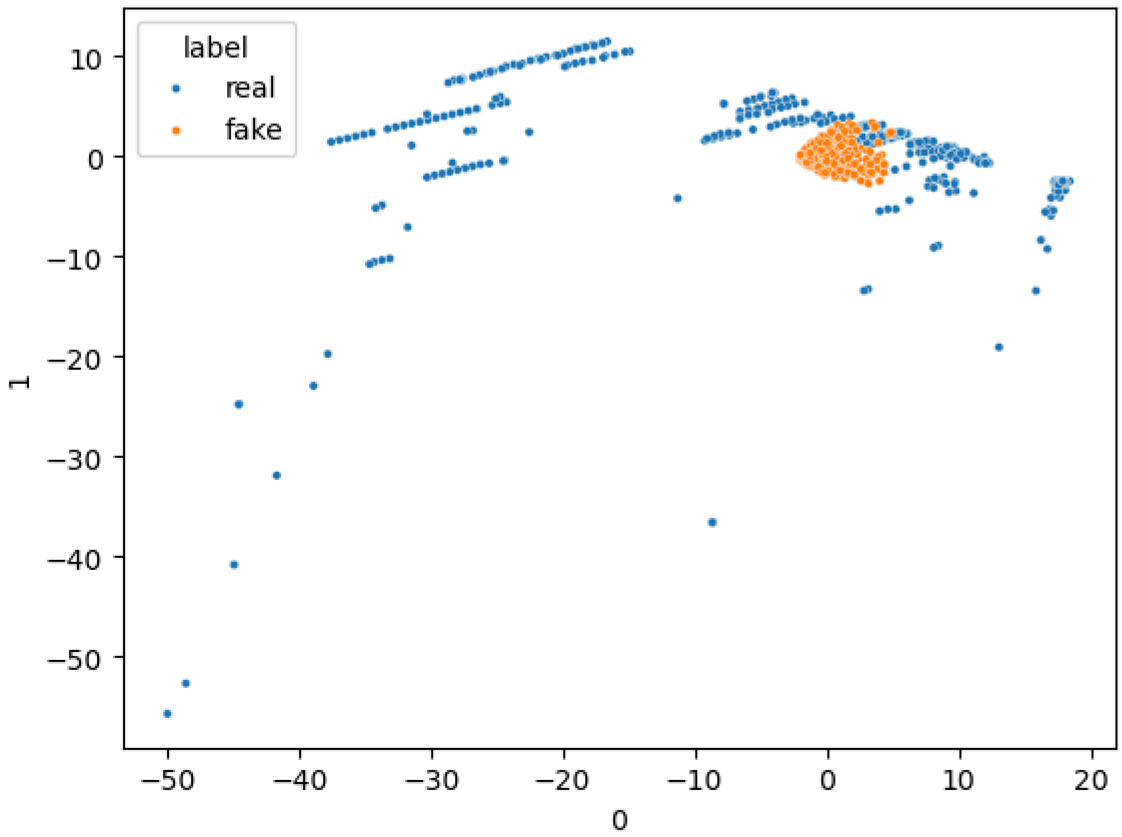
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 616us/step
```



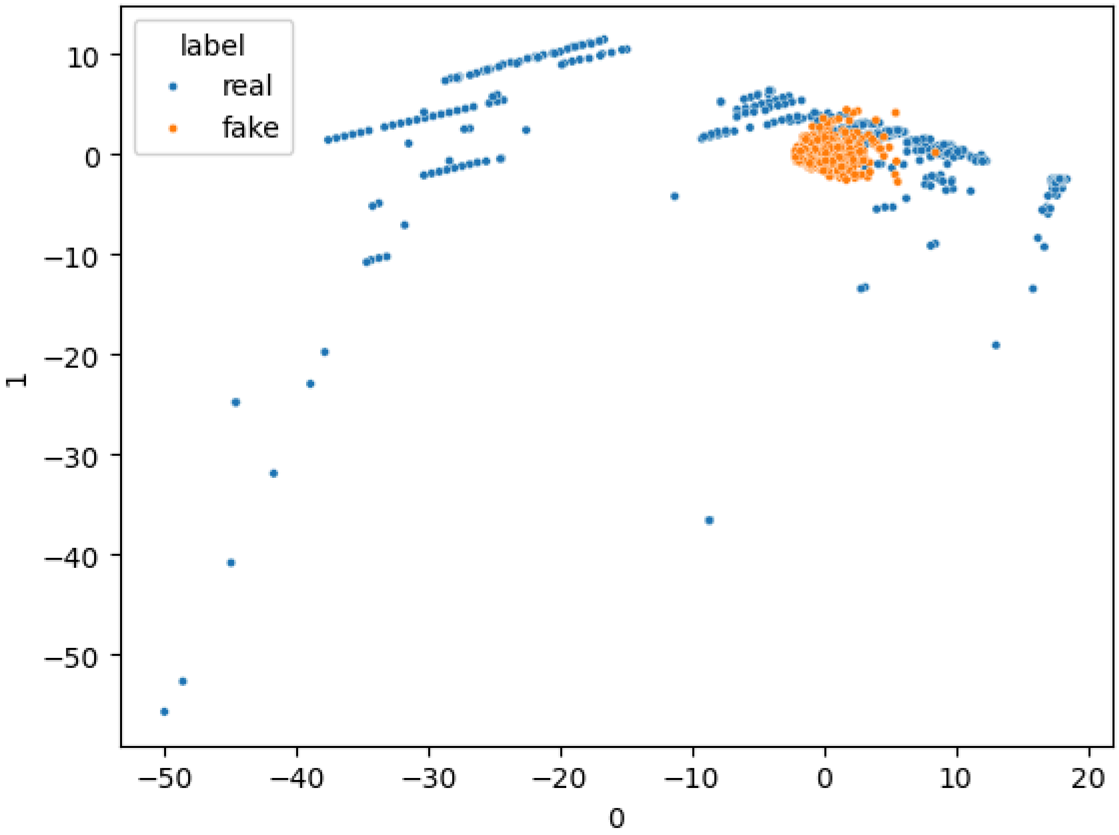
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```



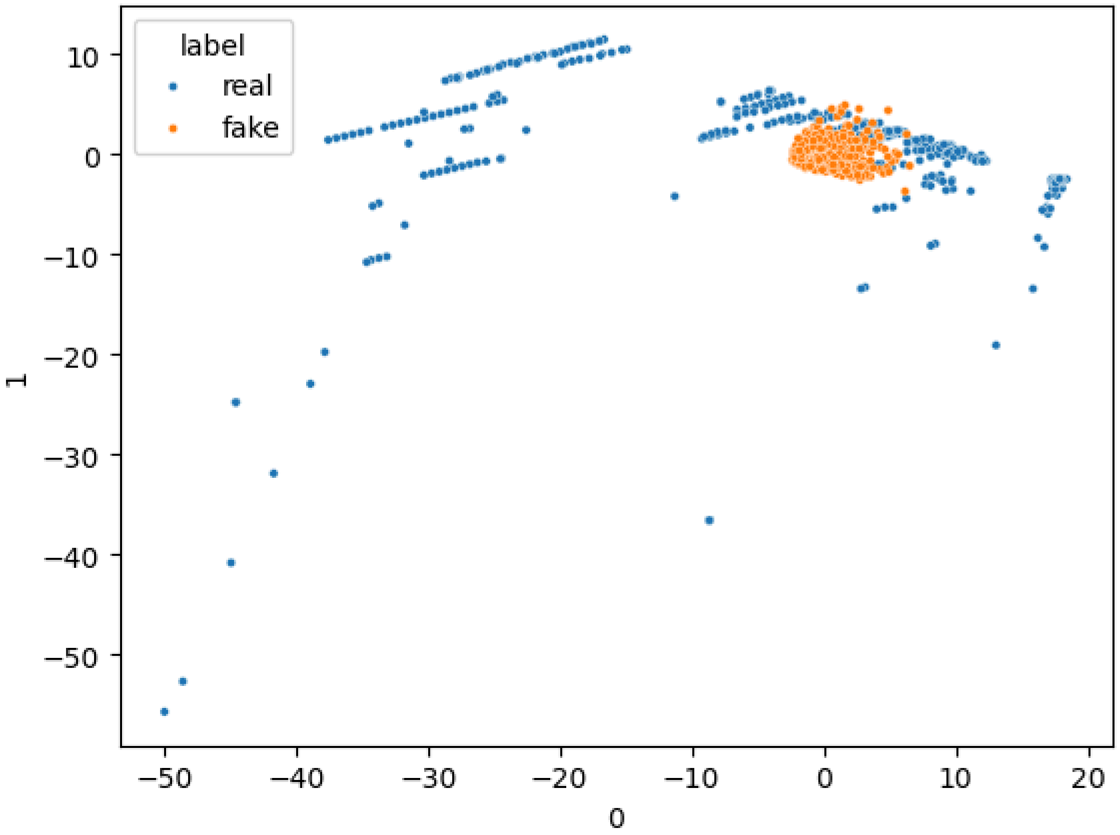
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 606us/step
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 645us/step
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 619us/step
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```

Generate synthetic data using the trained Generator

- Generate 1000 fradulent data points using the trained generator
- Compare the distribution of real and synthetic fradulent data points.

In [63]:

```
synthetic_data = generate_synthetic_data(generator, 1000)
df = pd.DataFrame(synthetic_data)
df["label"] = "fake"

df2 = data_fraudulent.drop("Class", axis = 1)
df2["label"] = "real"
df2.columns = df.columns

combined_df = pd.concat([df, df2])
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 263us/step
```

In [65]:

```
combined_df
```

Out[65]:

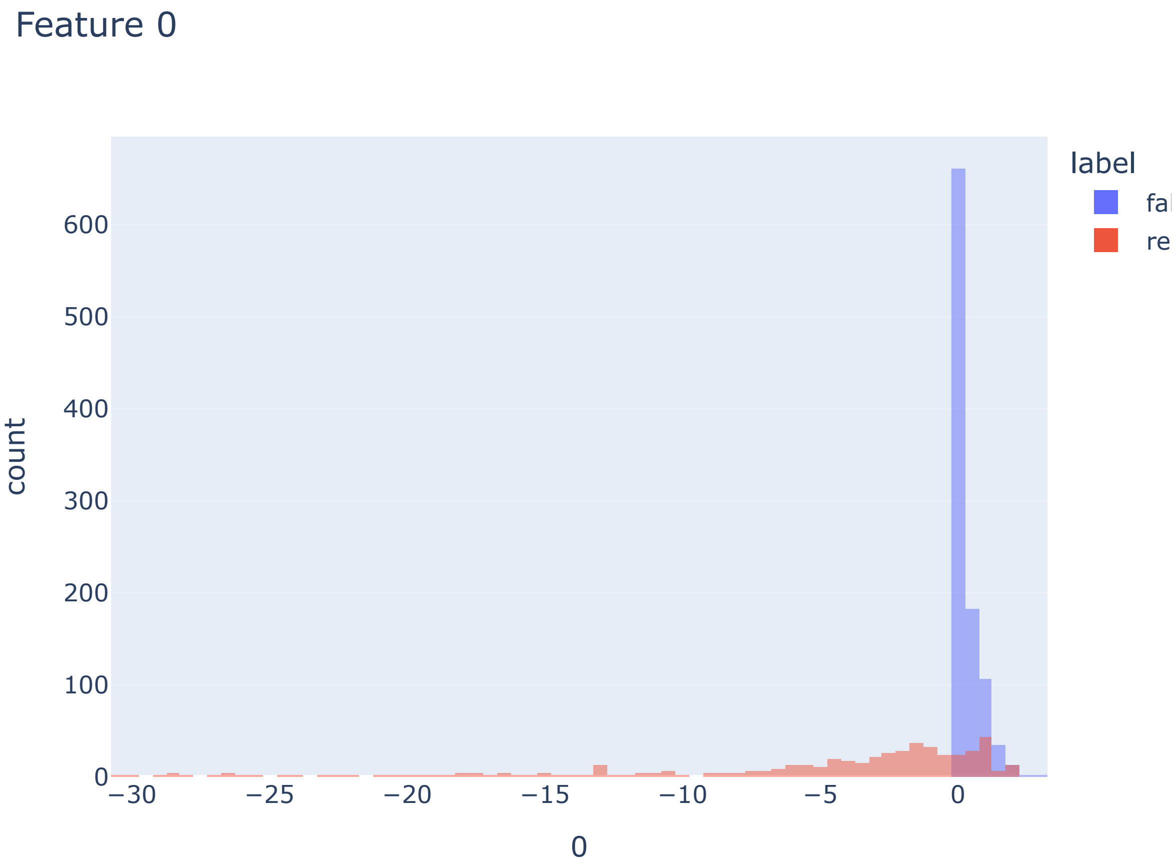
	0	1	2	3	4	5	6
0	0.488779	0.000000	0.000000	0.000000	0.192821	0.000000	0.683964
1	0.088744	0.291647	0.000000	0.000000	0.000000	0.000000	0.003341
2	0.000000	0.000000	1.074494	0.325012	0.920057	1.140243	0.000000
3	0.000000	0.000000	0.414872	0.000000	0.224516	0.000000	0.000000
4	1.341631	0.000000	0.000000	0.000000	0.058641	2.130062	0.000000
...	...	...	...	...	...	...	...
50487	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850
50488	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170
50489	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739
50490	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002
50491	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050

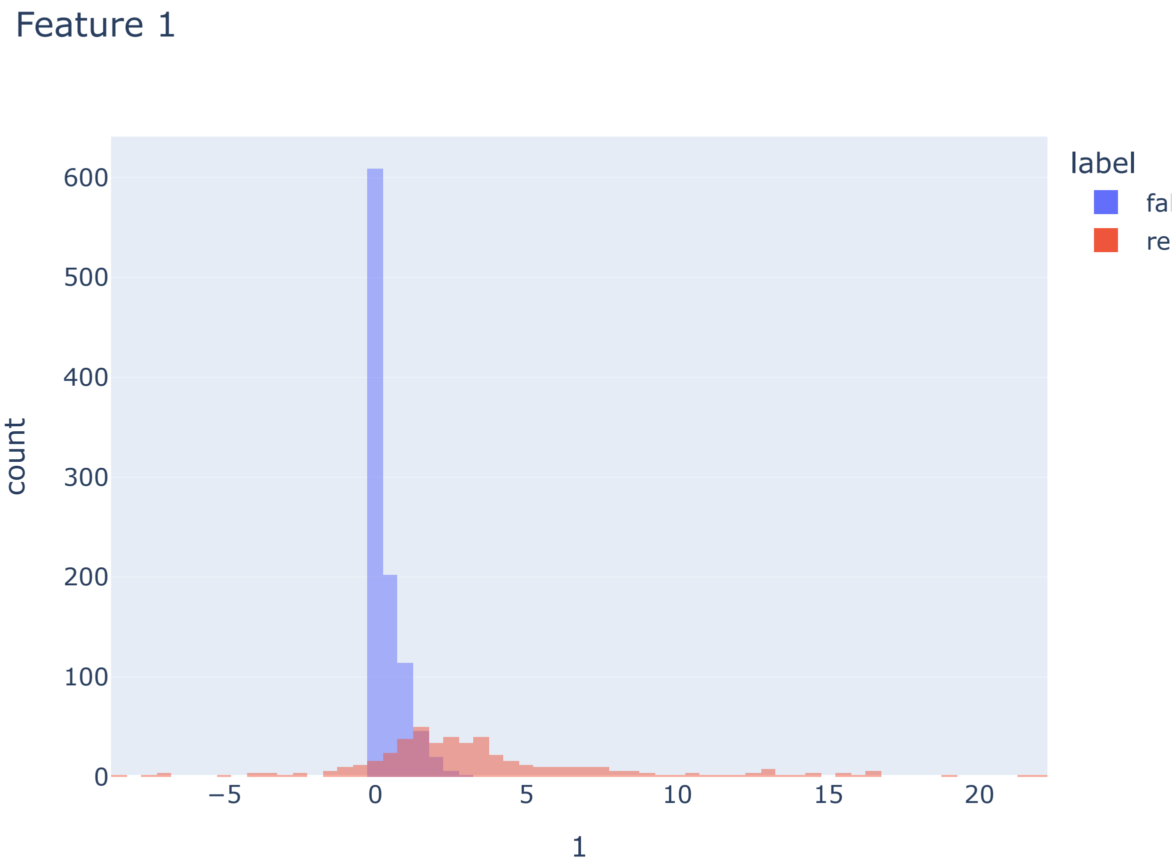
1492 rows × 30 columns

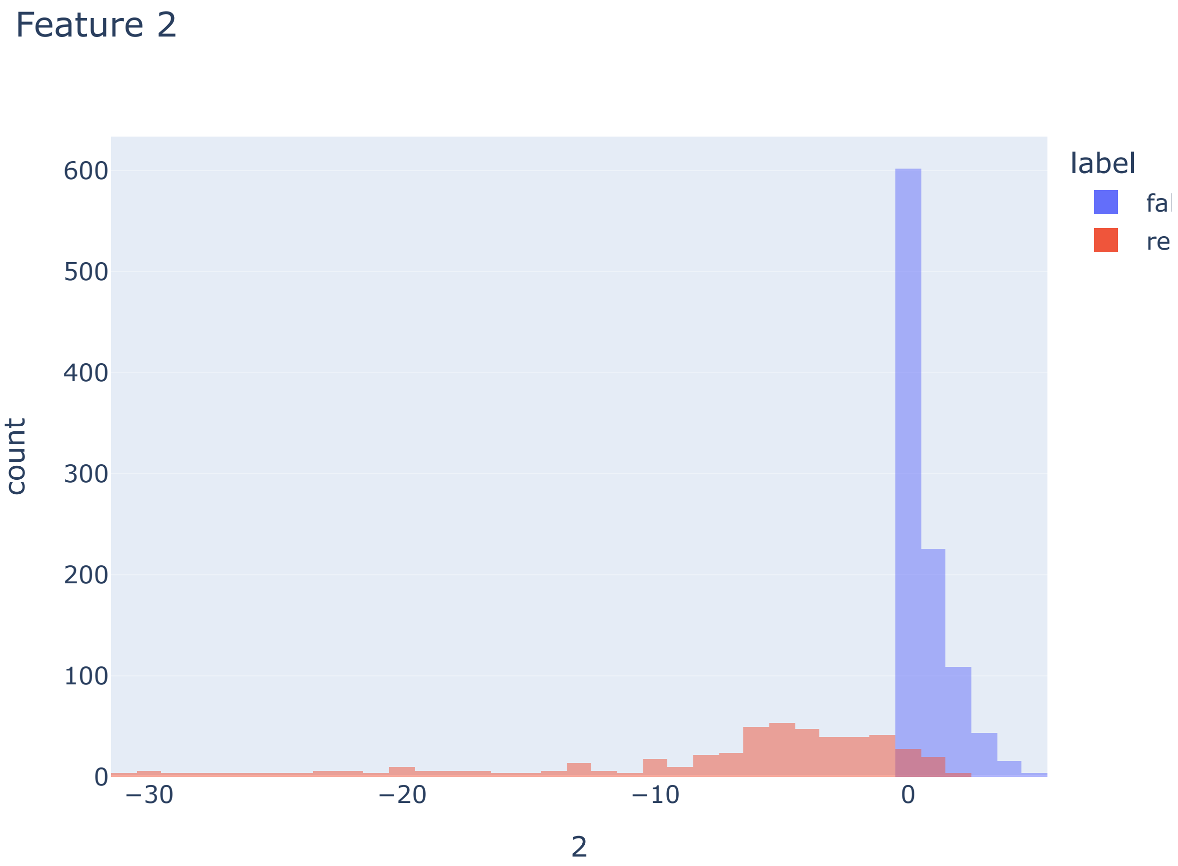
Checking the individual feature distribution of synthetic and real fraudulent data.

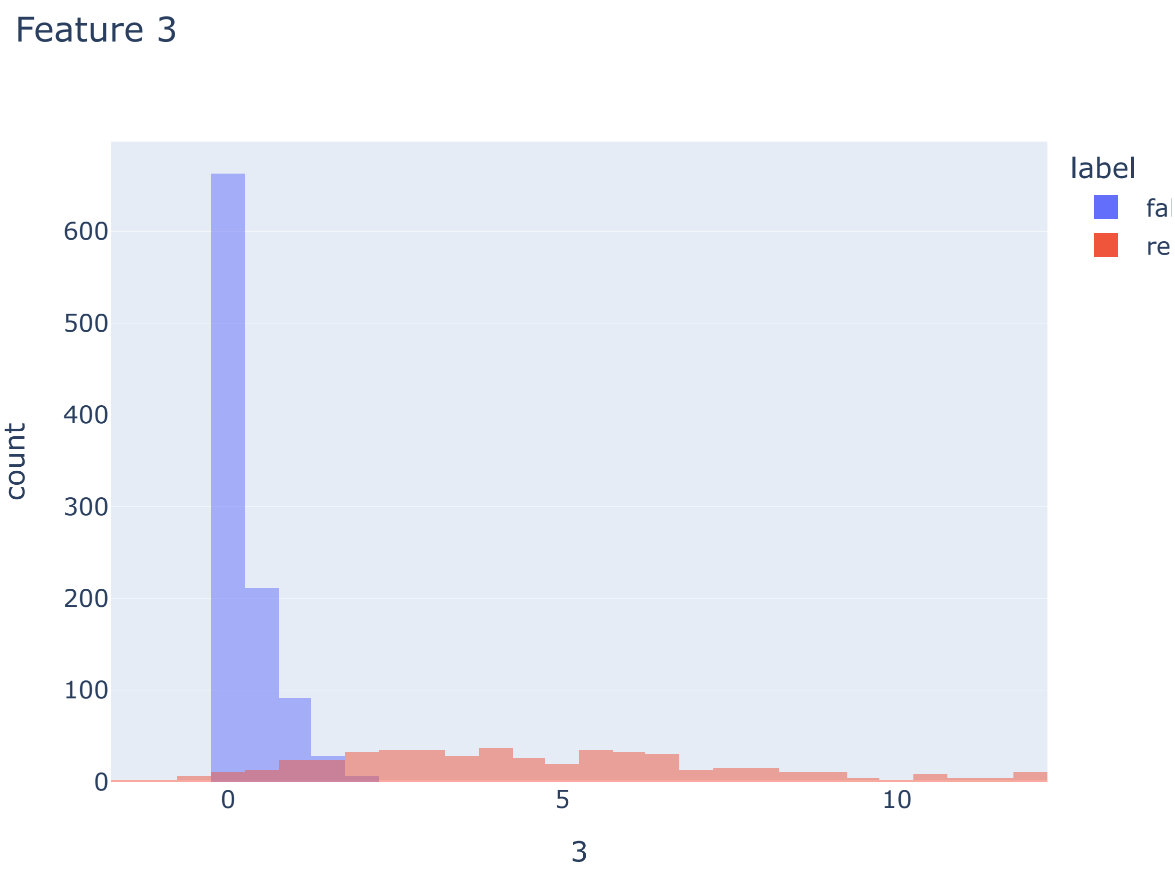
In [67]:

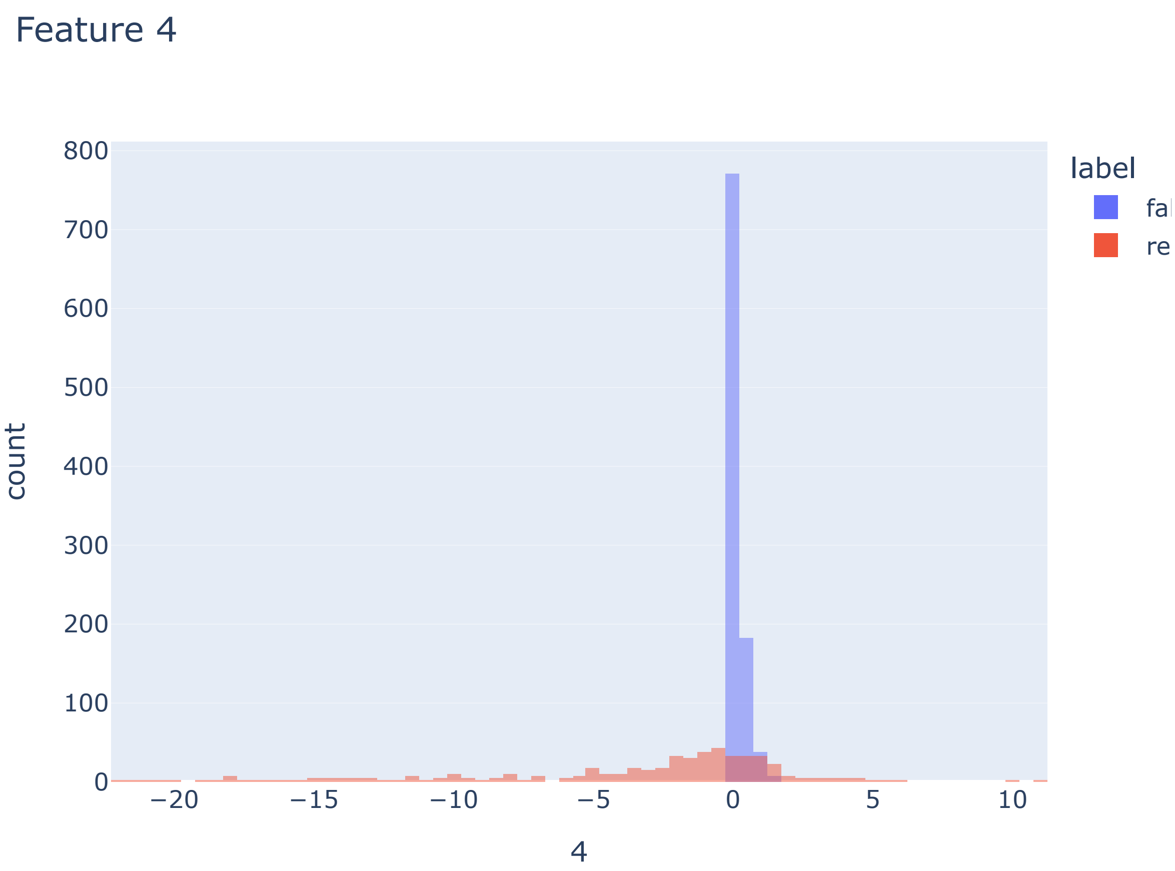
```
for col in combined_df.columns:
  plt.figure()
  fig = px.histogram(combined_df, color = 'label', x=col,barmode="overlay",
  fig.show()
```

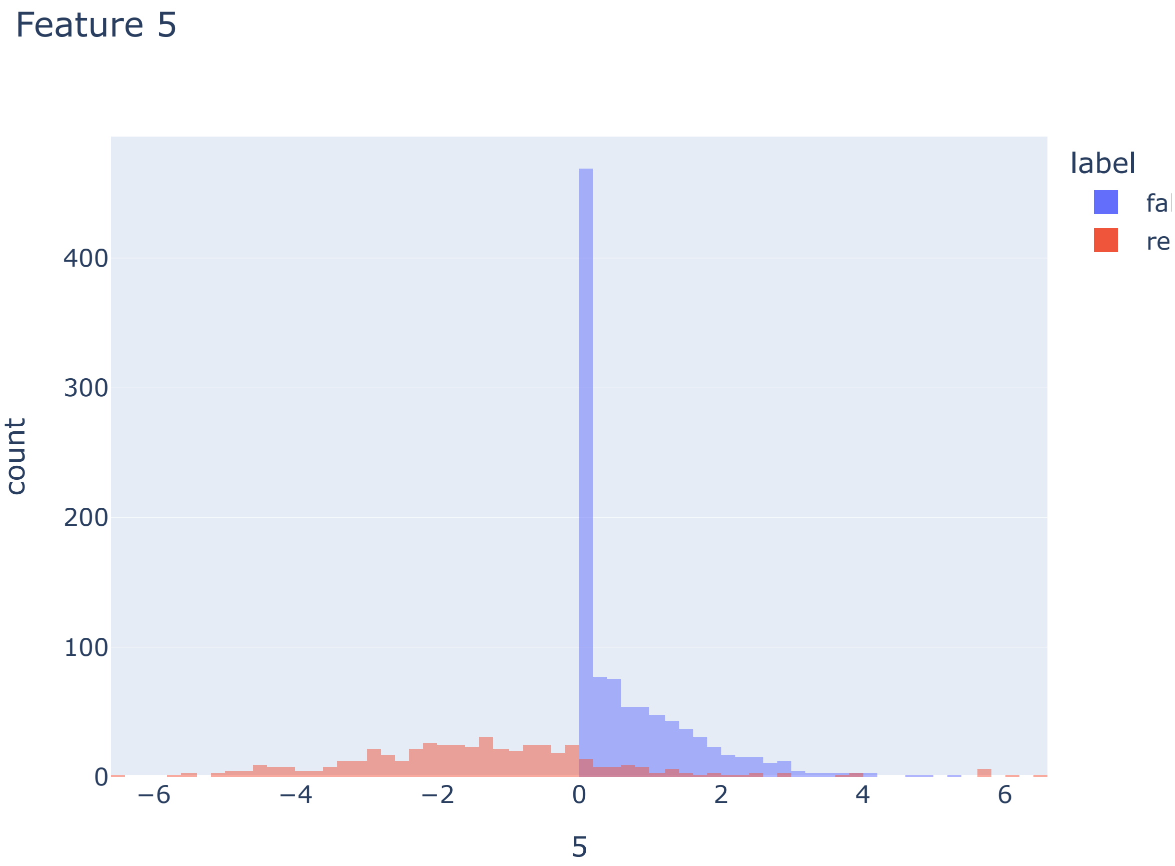

Feature 0
count
600
500
400
300
200
100
0
−30
−25
−20
−15
−10
−5
0
0
label
fal
re

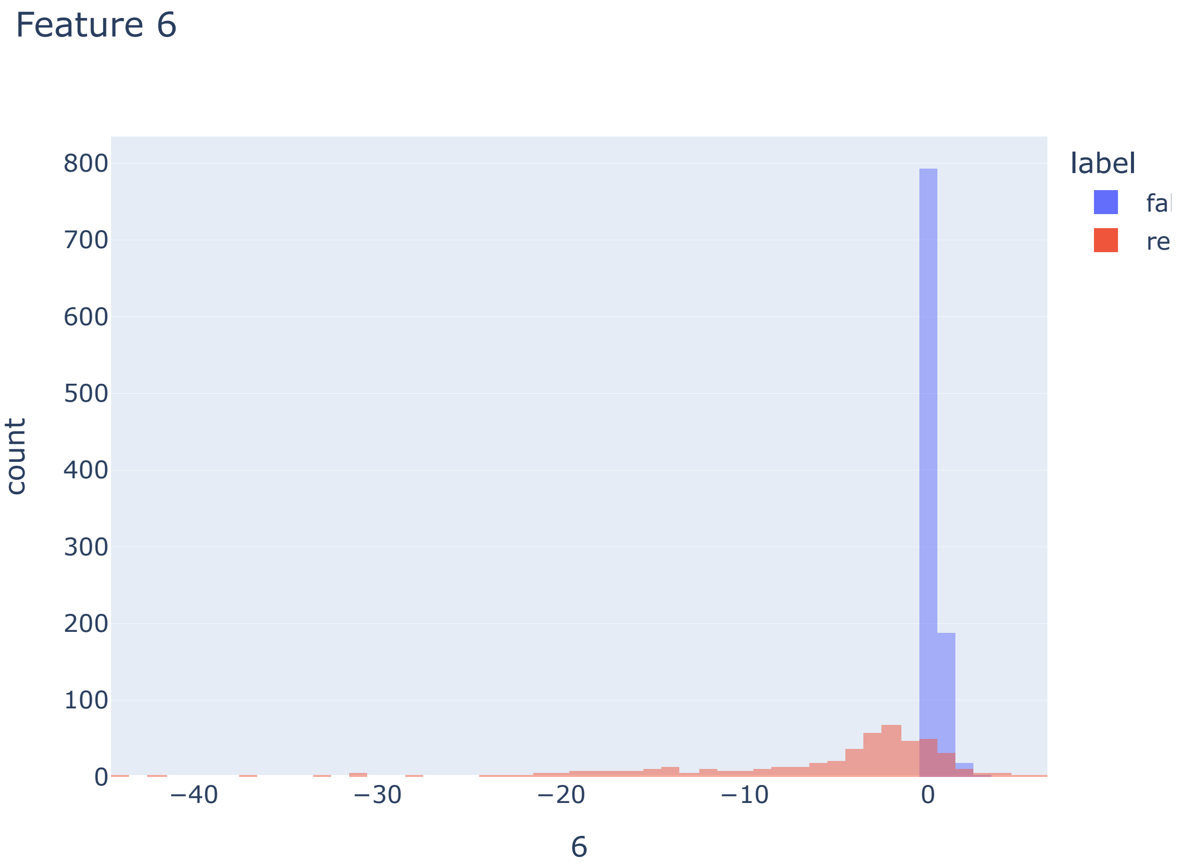

Feature 1
label
fal
re
count
600
500
400
300
200
100
0
−5
0
5
10
15
20
1

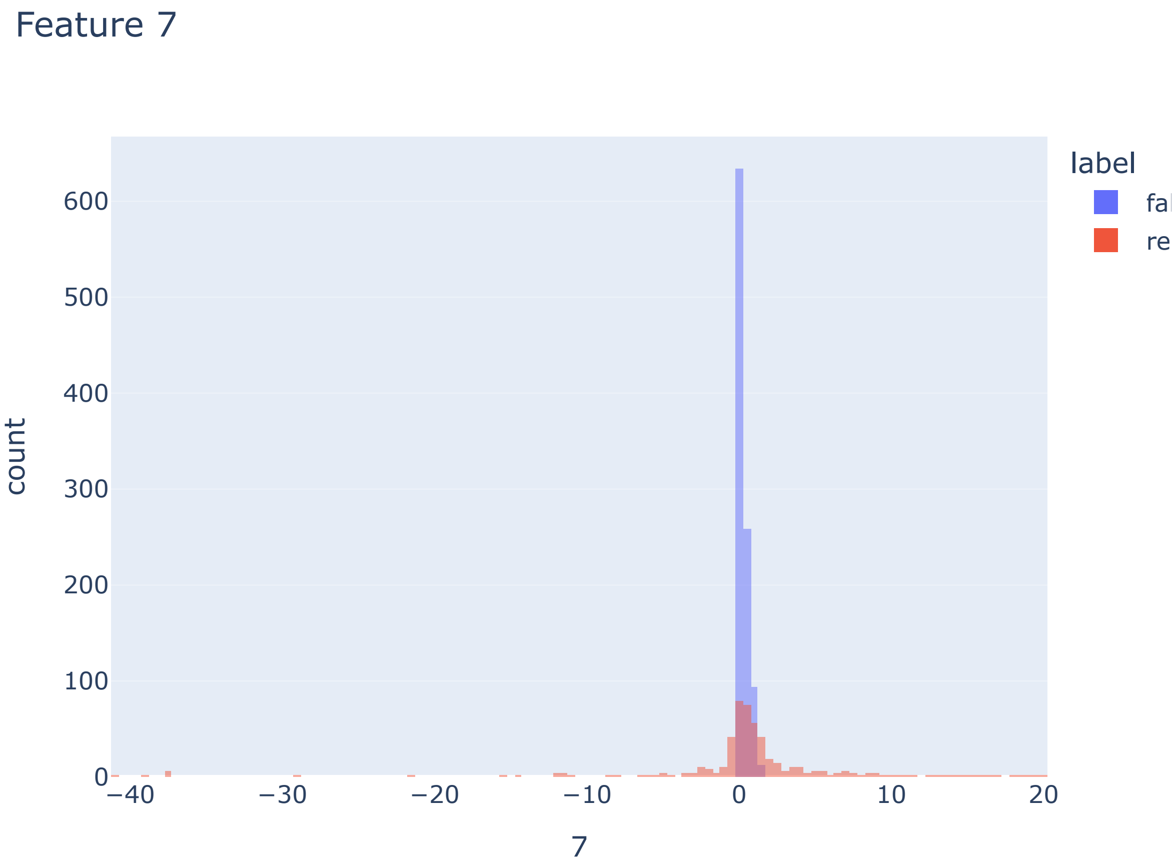

Feature 2
label
fal
re
count
600
500
400
300
200
100
0
−30
−20
−10
0
2

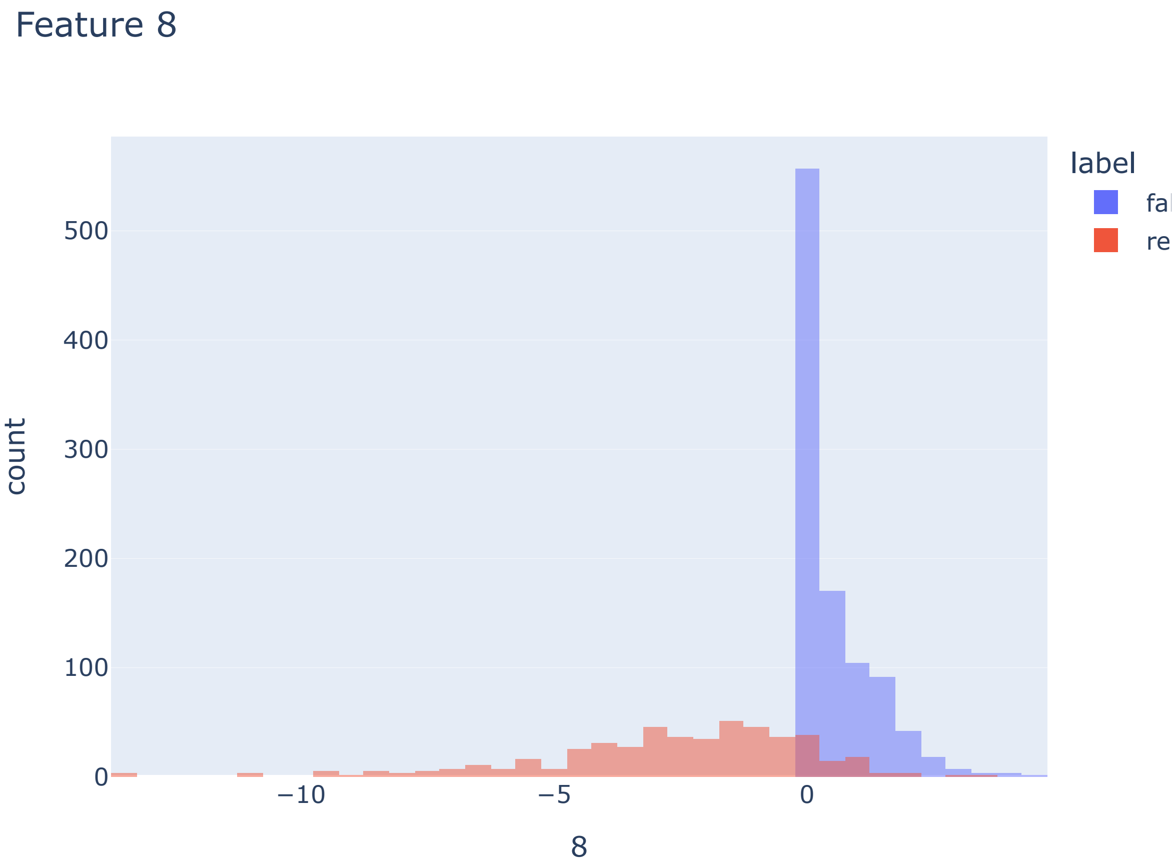

Feature 3
label
fal
re
count
600
500
400
300
200
100
0
0
5
10
3

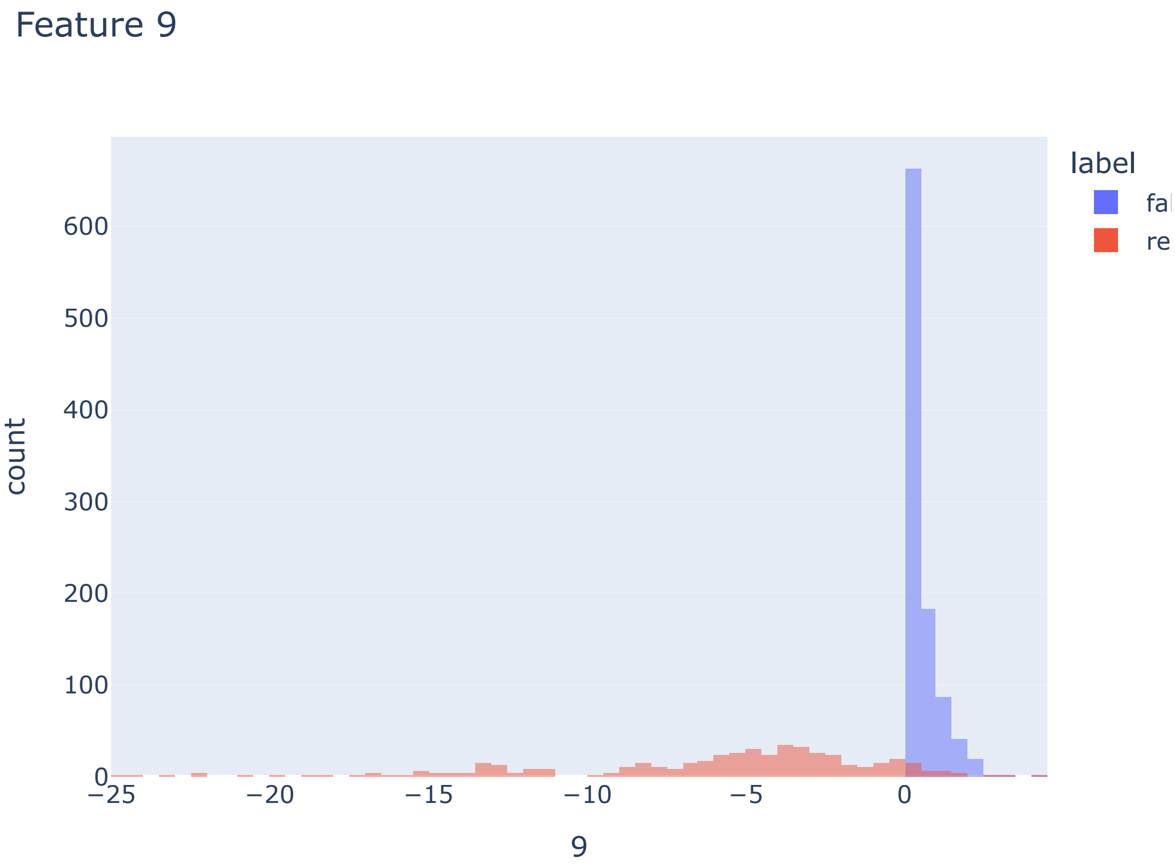

Feature 4
count
800
700
600
500
400
300
200
100
0
−20
−15
−10
−5
0
5
10
4
label
fal
re

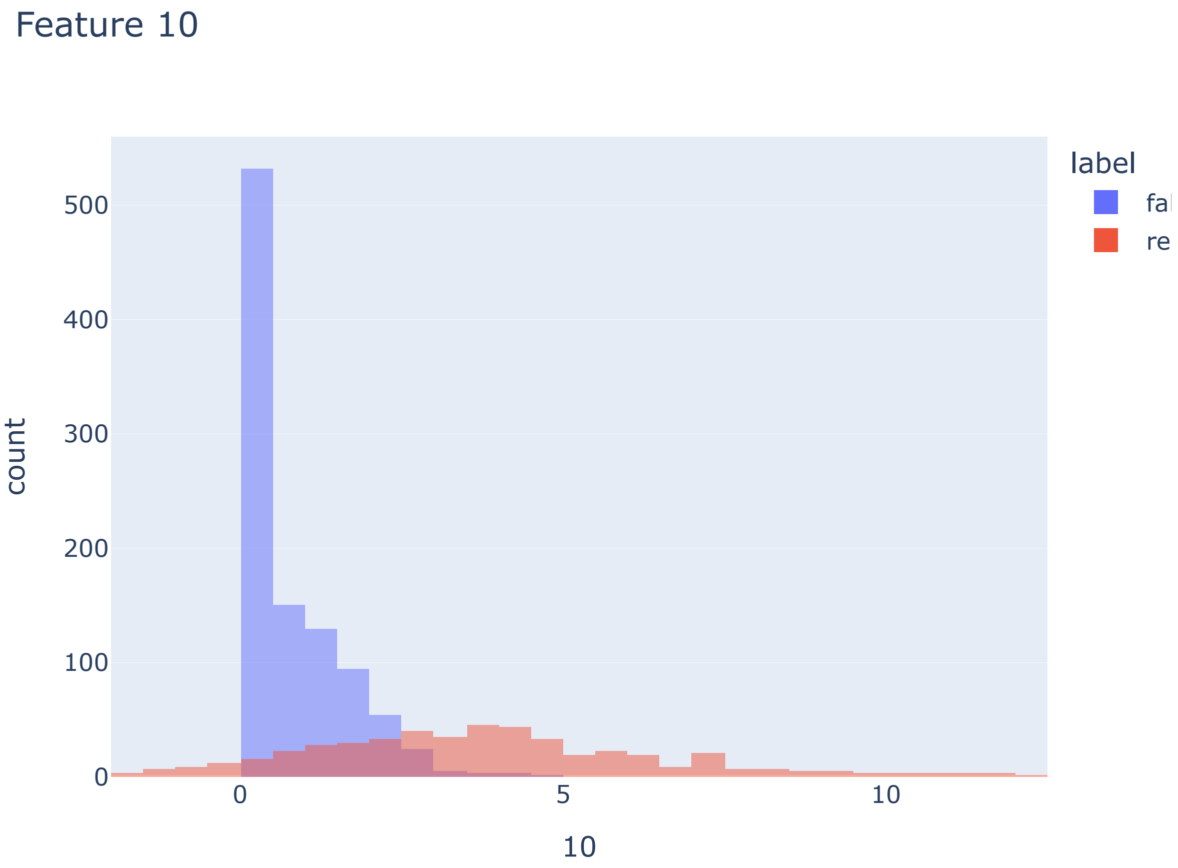

Feature 5
label
fal
re
count
400
300
200
100
0
−6
−4
−2
0
2
4
6
5

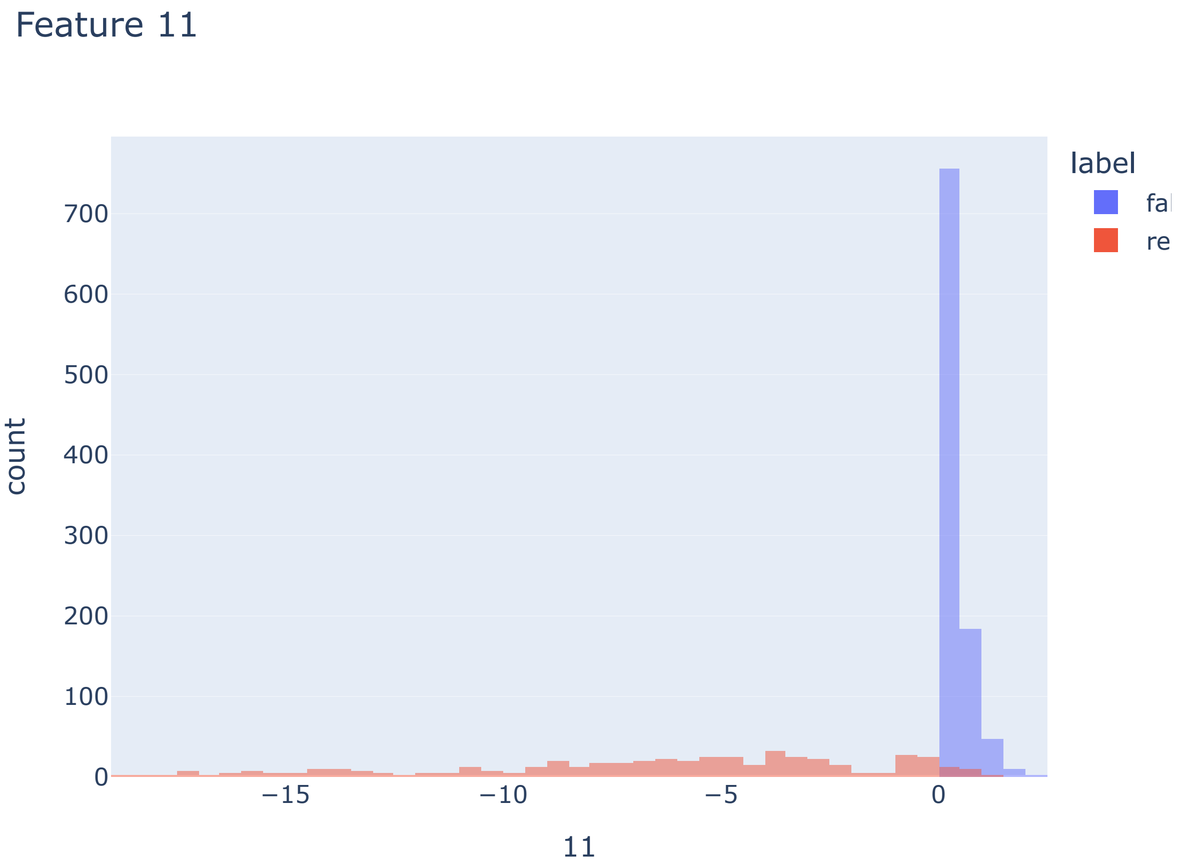

Feature 6
count
800
700
600
500
400
300
200
100
0
−40
−30
−20
−10
0
6
label
fal
re

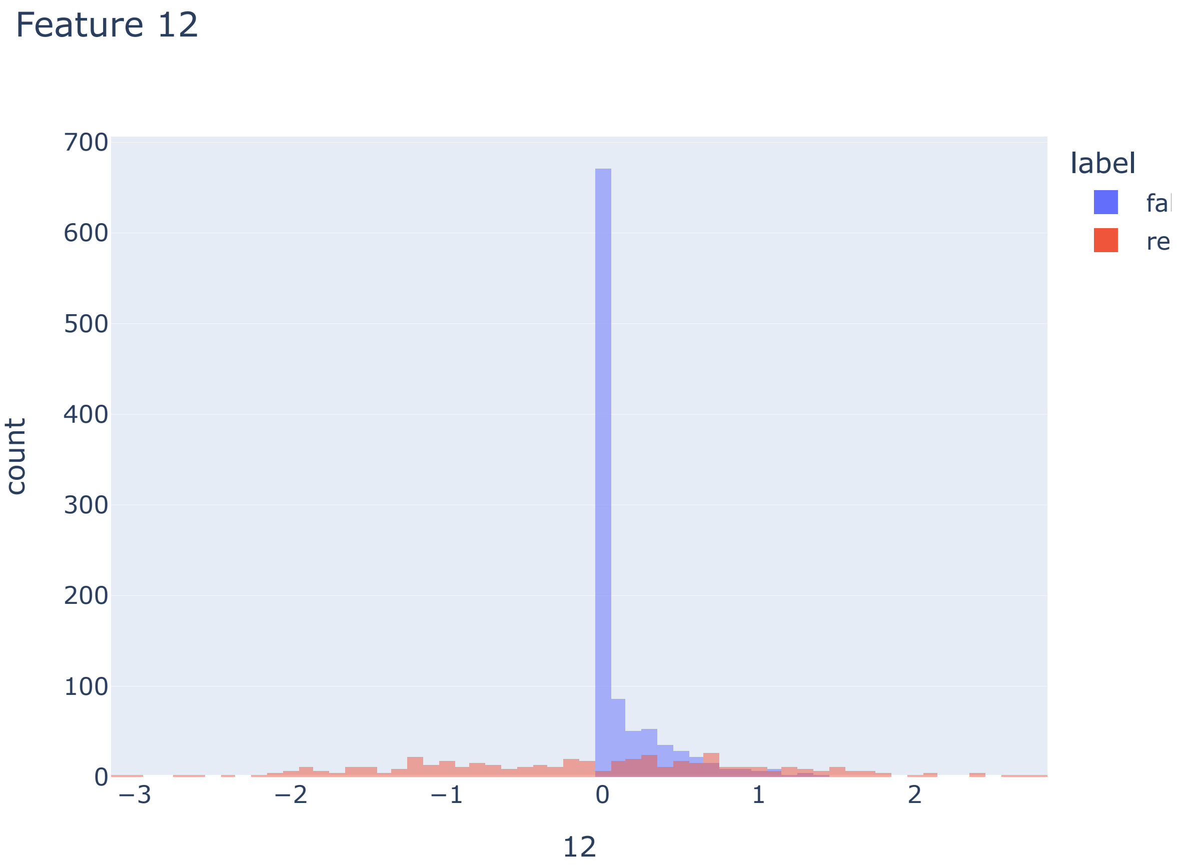

Feature 7
label
fal
re
count
600
500
400
300
200
100
0
−40
−30
−20
−10
0
10
20
7

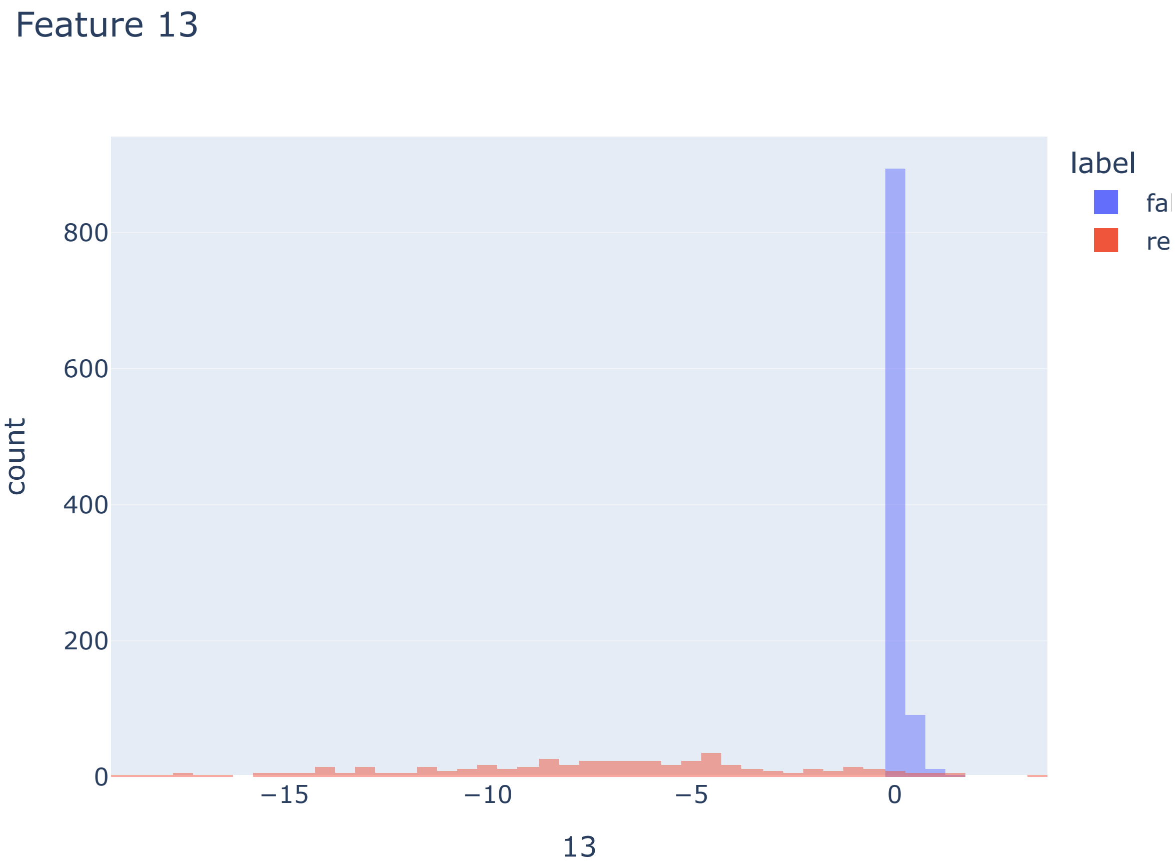

Feature 8
count
500
400
300
200
100
0
−10
−5
0
8
label
fa
re

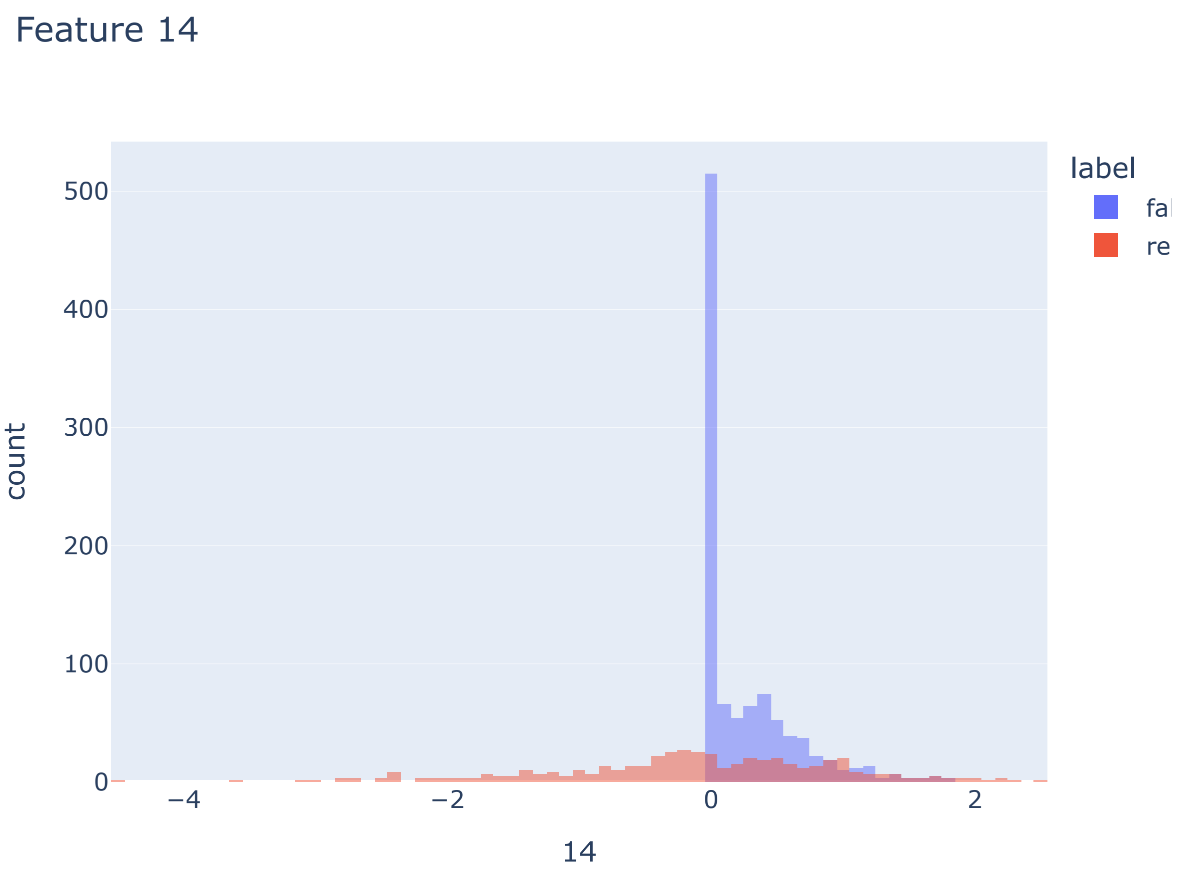

Feature 9
count
600
500
400
300
200
100
0
−25
−20
−15
−10
−5
0
9
label
fal
re

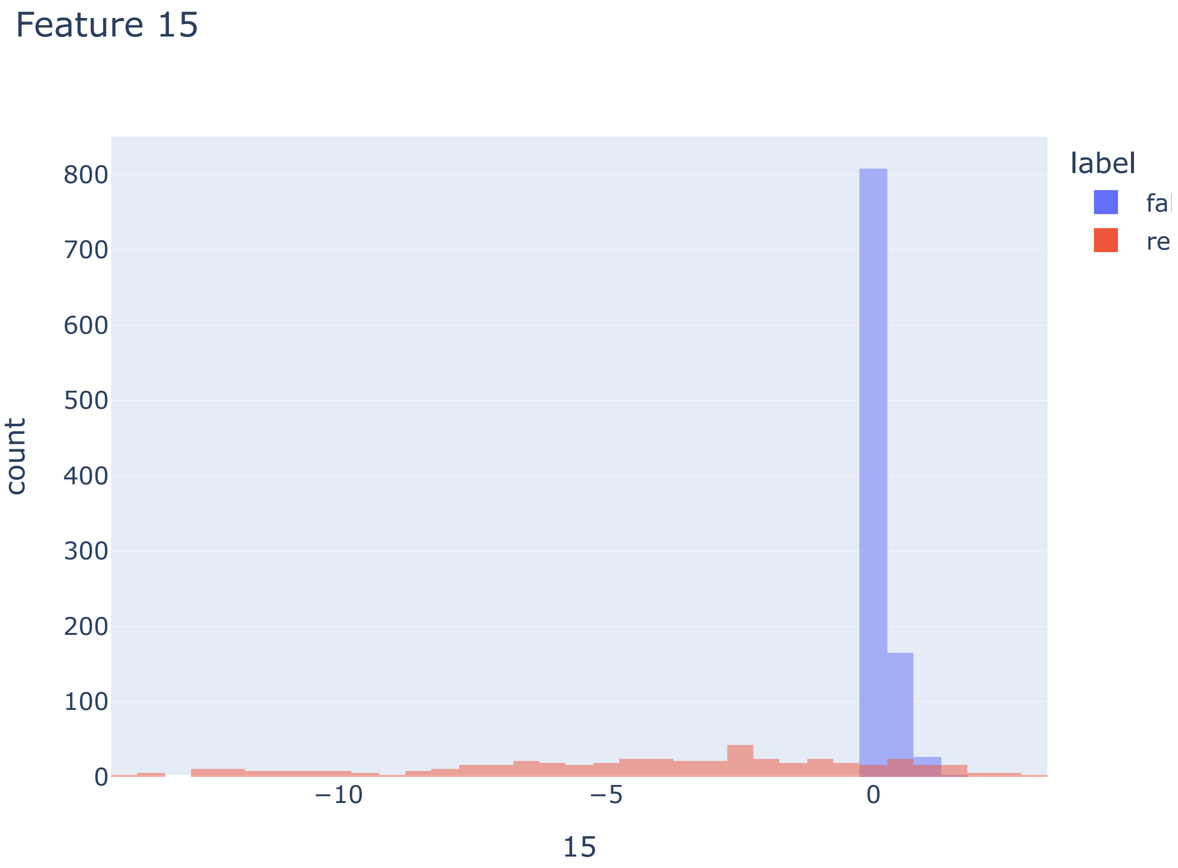

Feature 10
label
🟦 fal
🟥 re
count
500
400
300
200
100
0
0
5
10
10

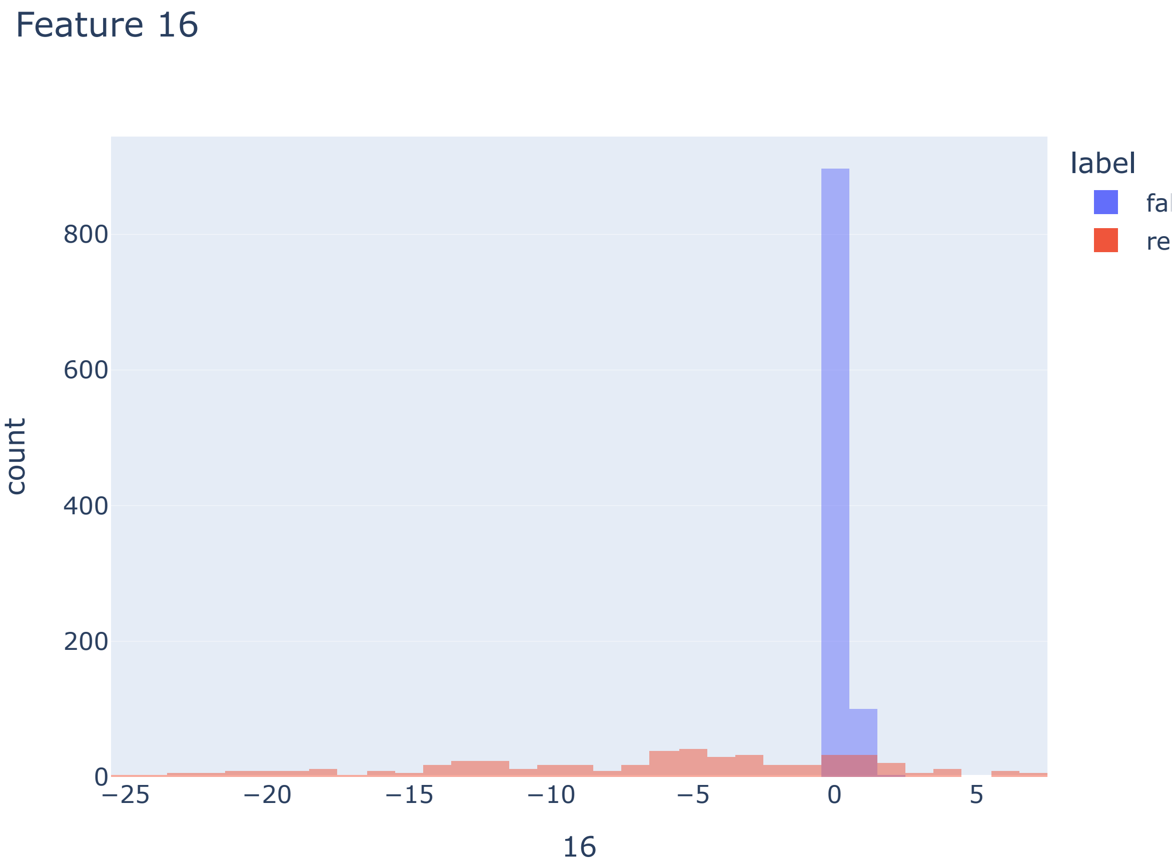

Feature 11
label
fal
re
count
700
600
500
400
300
200
100
0
−15
−10
−5
0
11

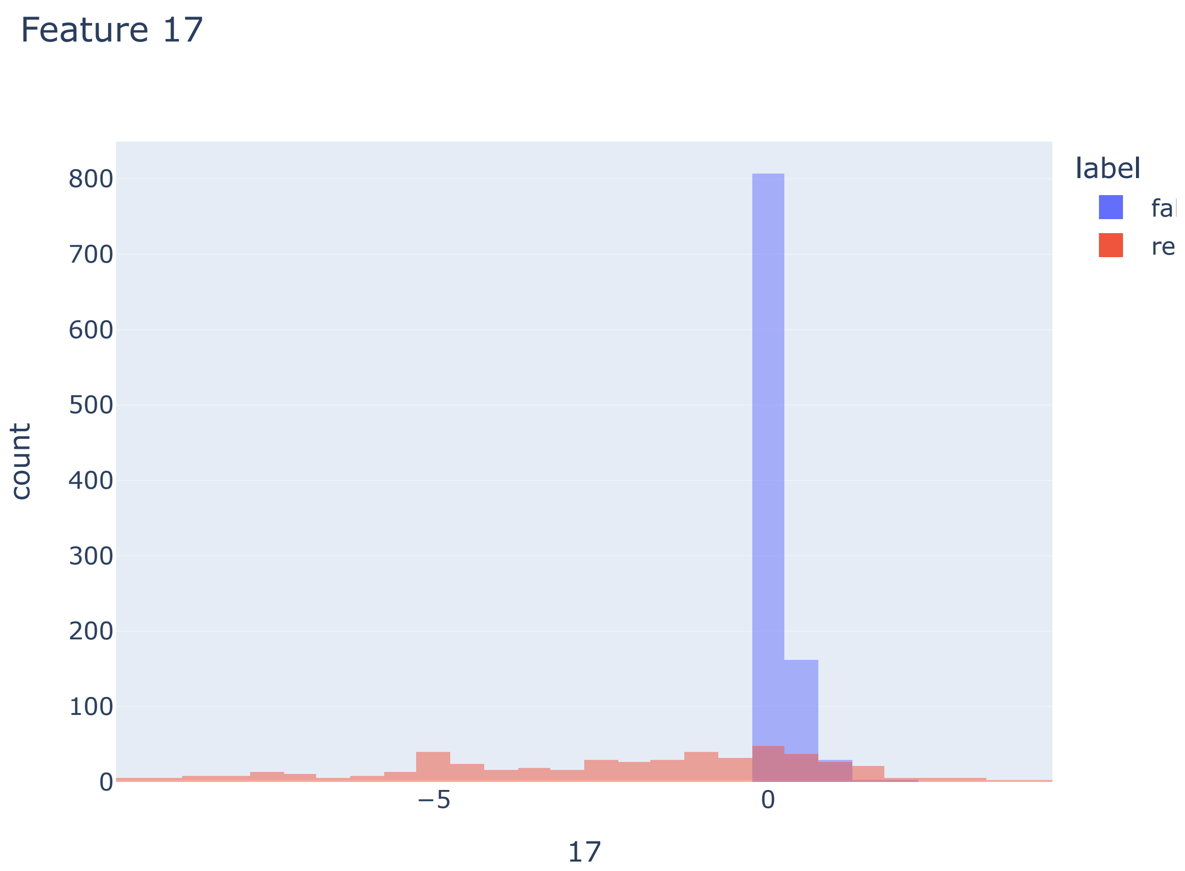

Feature 12
label
fal
re
count
700
600
500
400
300
200
100
0
−3
−2
−1
0
1
2
12

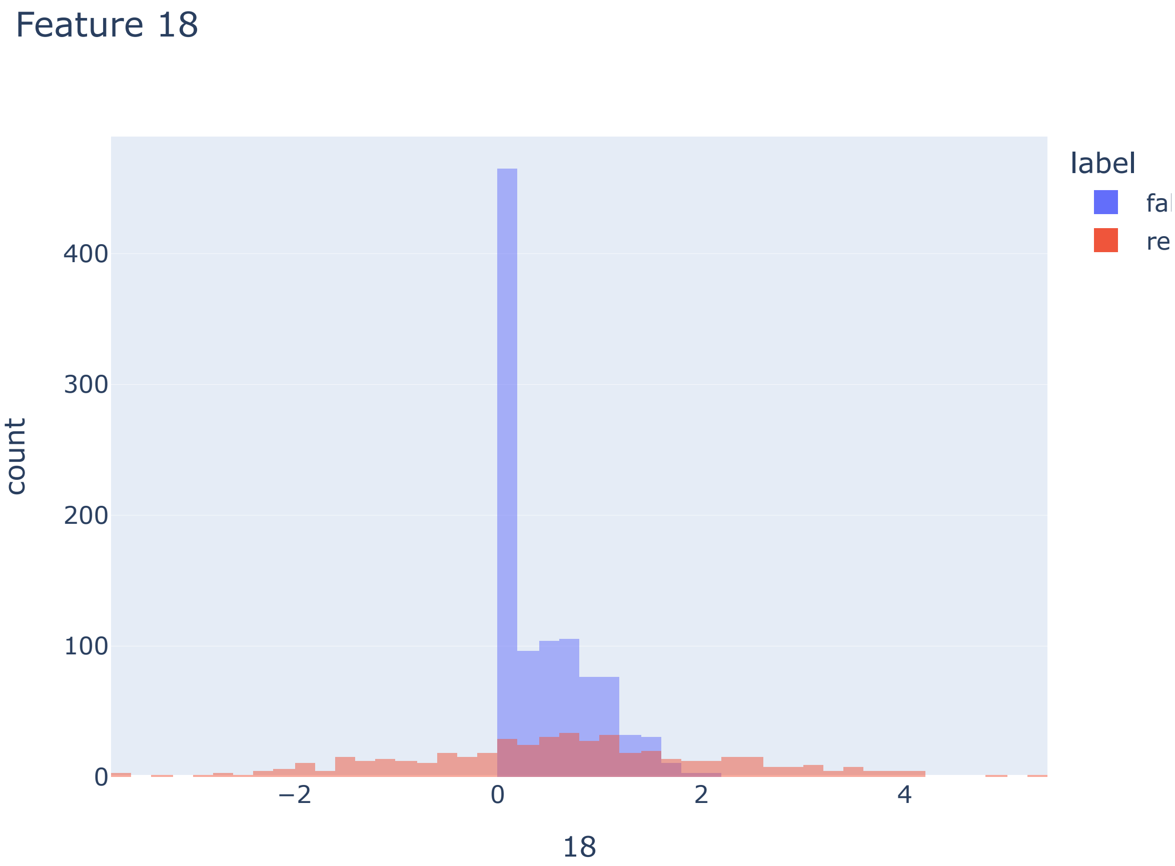

Feature 13
count
800
600
400
200
0
−15
−10
−5
0
13
label
fal
re

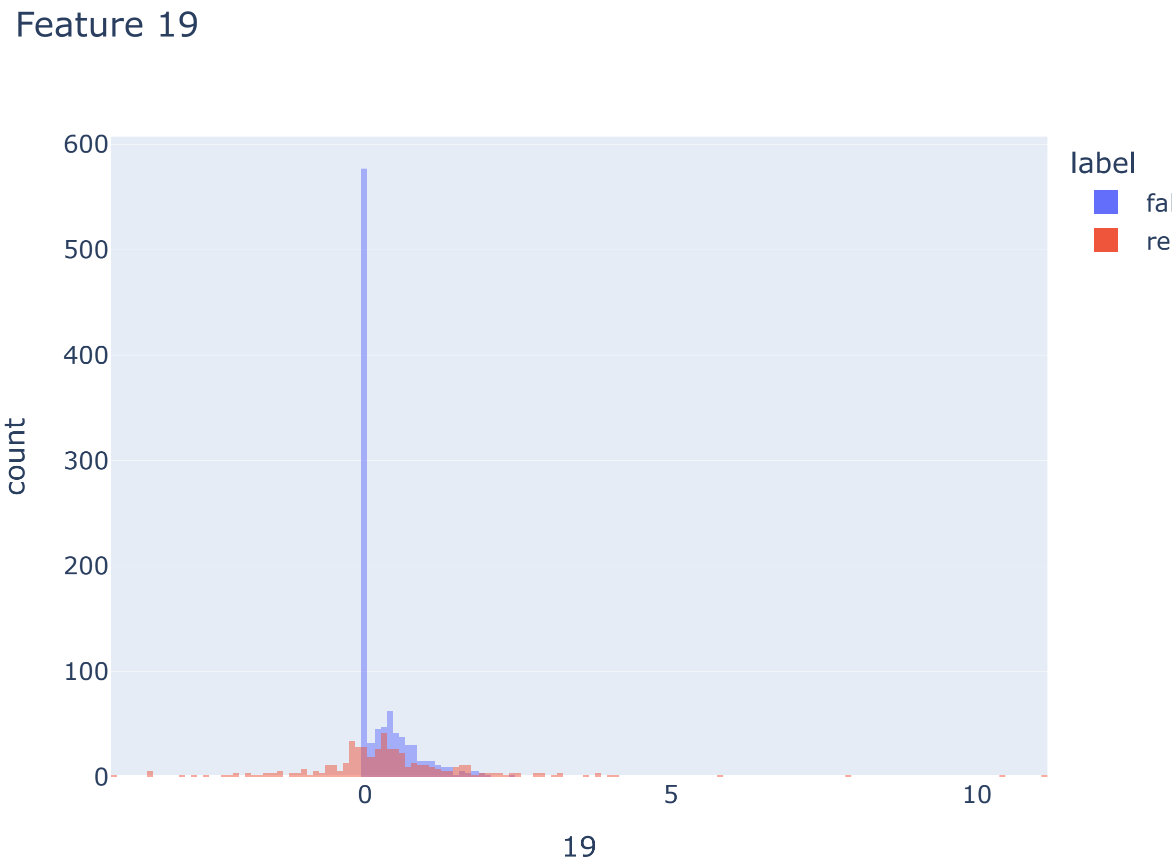

Feature 14
label
fal
re
count
500
400
300
200
100
0
−4
−2
0
2
14


Feature 15
label
fal
re
count
800
700
600
500
400
300
200
100
0
−10
−5
0
15

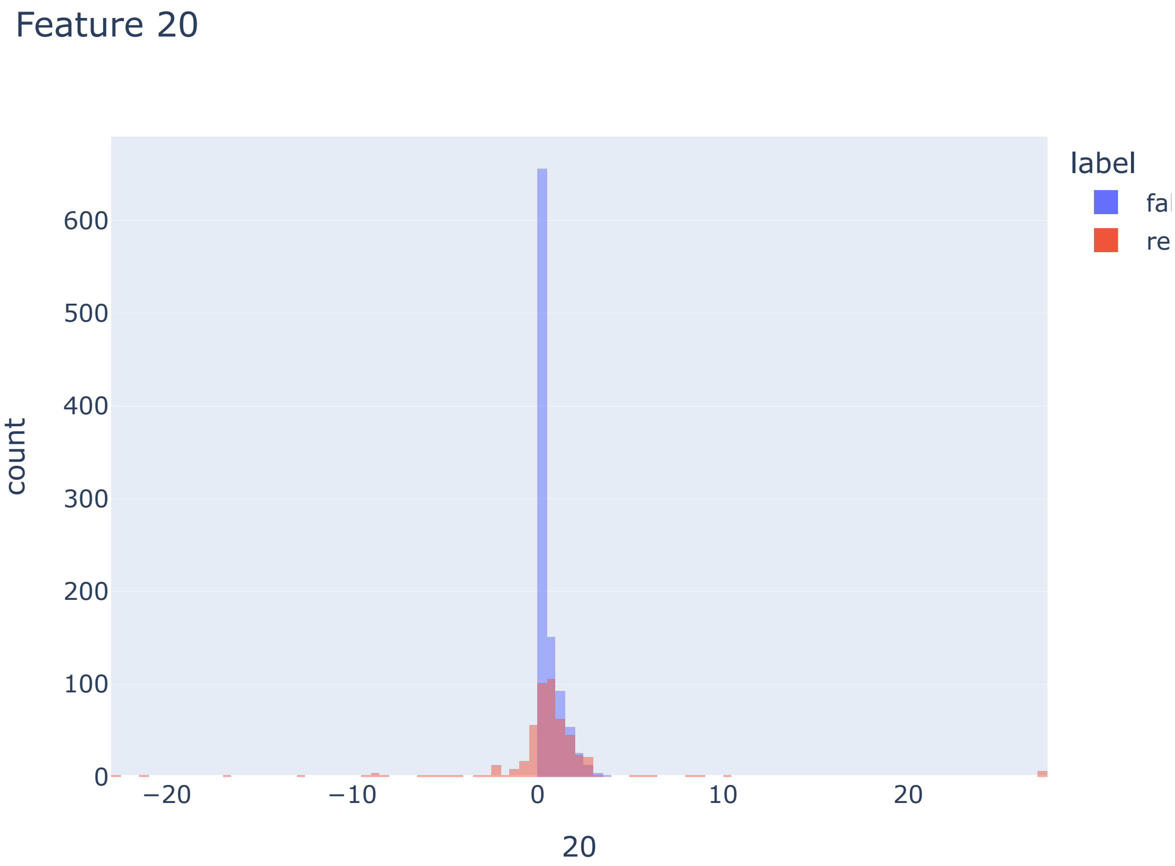

Feature 16
label
fal
re
count
800
600
400
200
0
−25
−20
−15
−10
−5
0
5
16

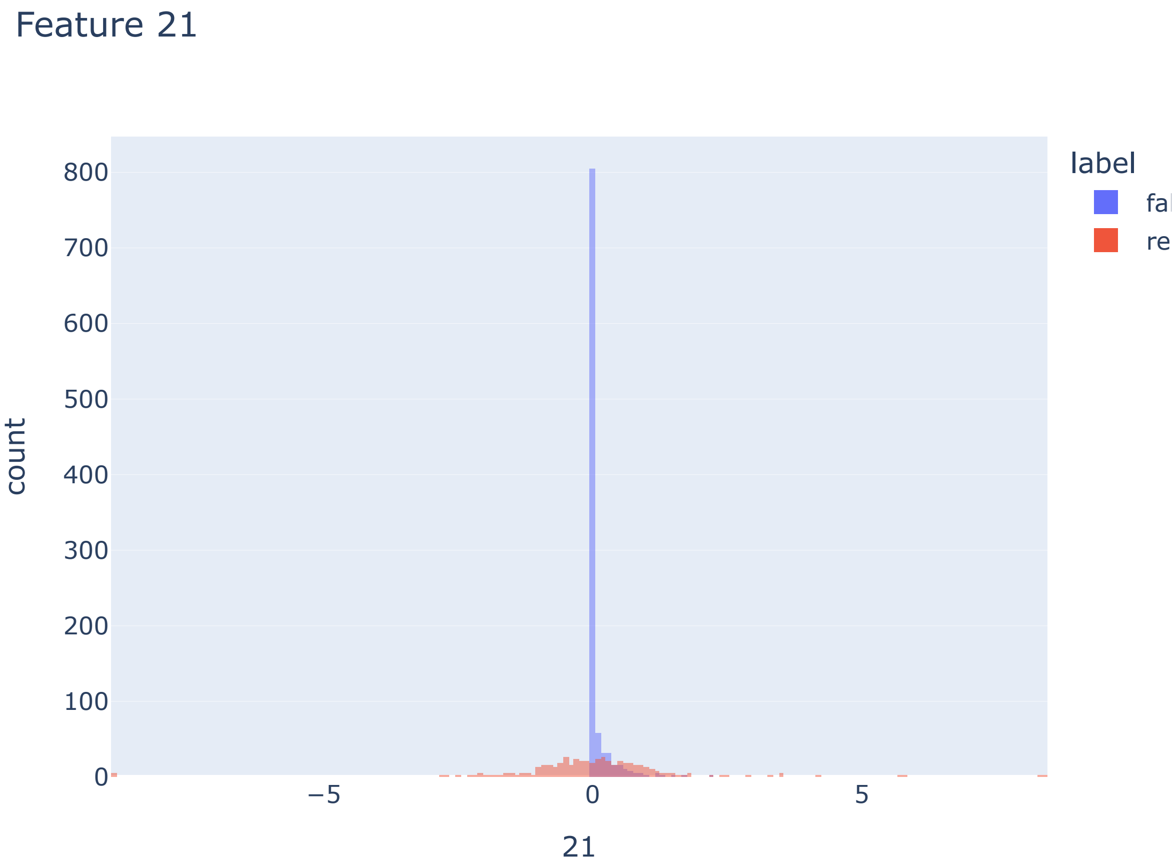

Feature 17
count
800
700
600
500
400
300
200
100
0
−5
0
17
label
fal
re

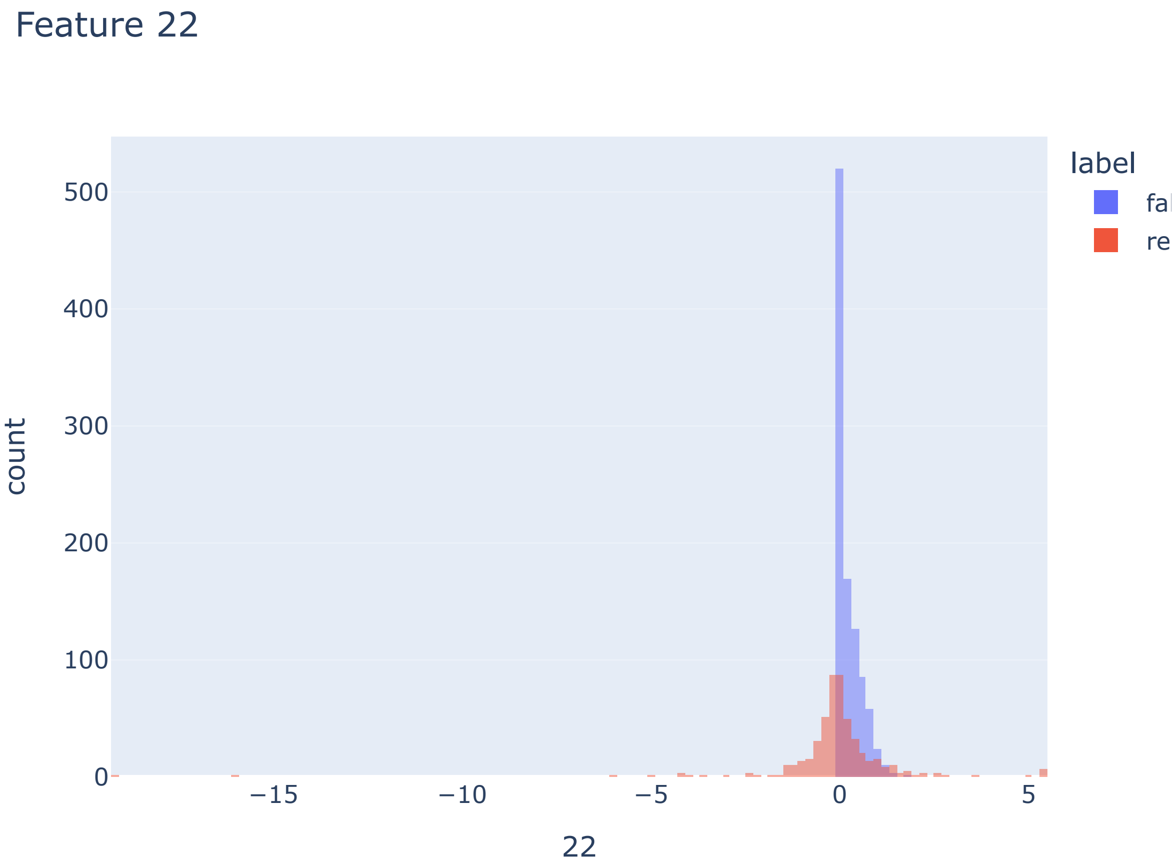

Feature 18
count
400
300
200
100
0
−2
0
2
4
18
label
fal
re

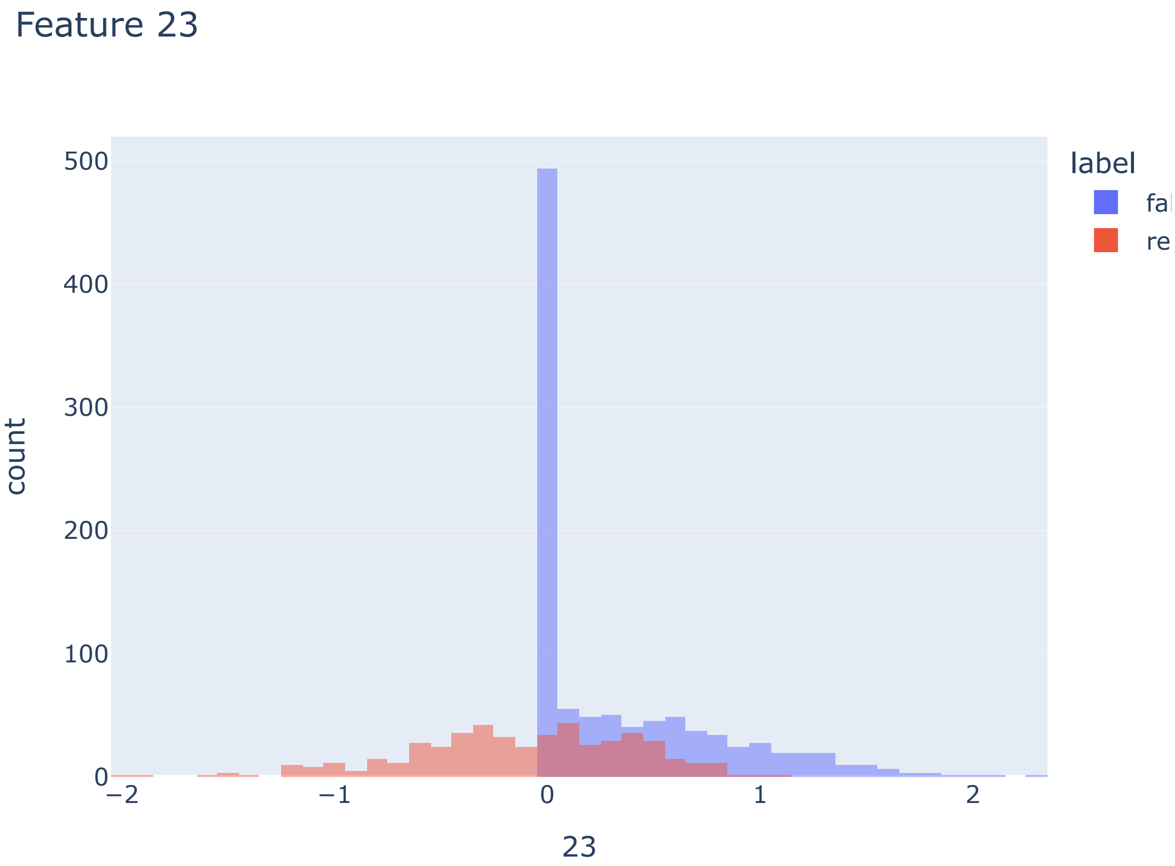


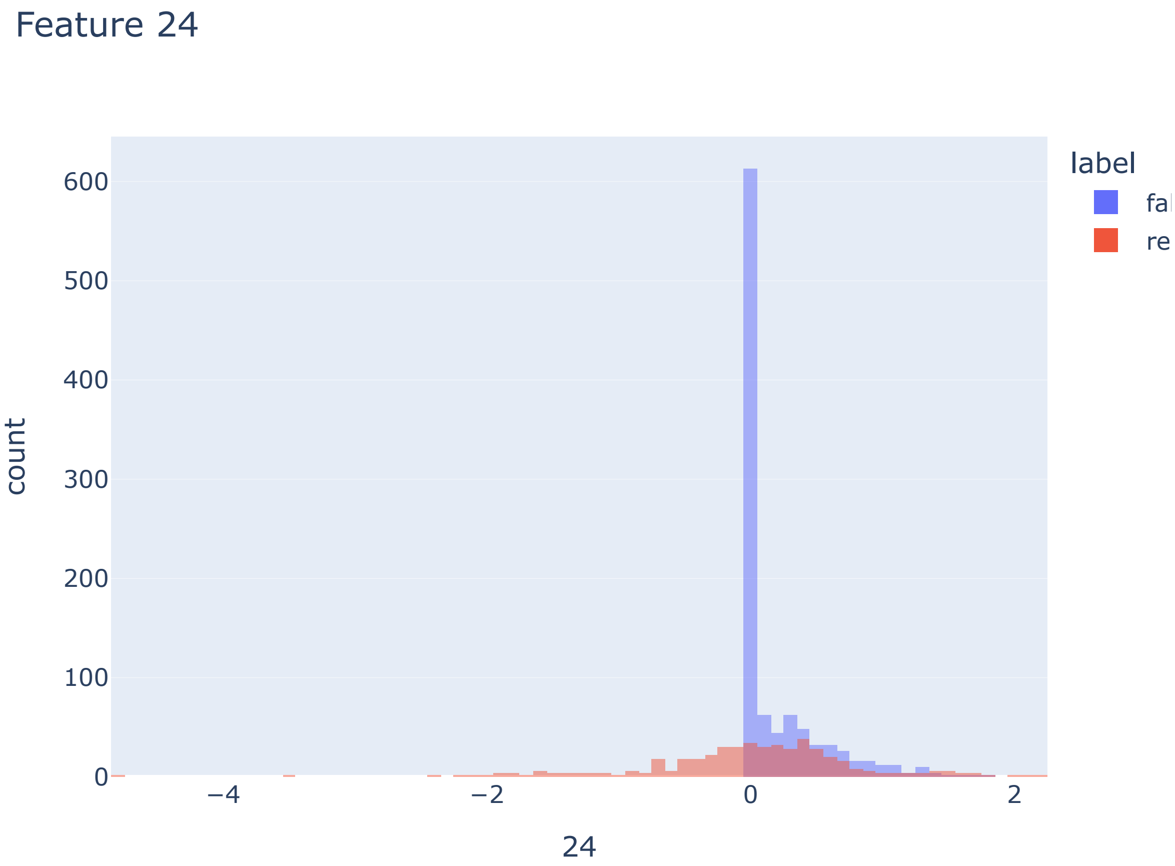
```
/var/folders/w5/_kz_512d0bd76w88dw4ym66r0000gn/T/ipykernel_14370/3618524353.
py:2: RuntimeWarning:

More than 20 figures have been opened. Figures created through the pyplot in
terface (`matplotlib.pyplot.figure`) are retained until explicitly closed an
d may consume too much memory. (To control this warning, see the rcParam `fi
gure.max_open_warning`). Consider using `matplotlib.pyplot.close()`.
```

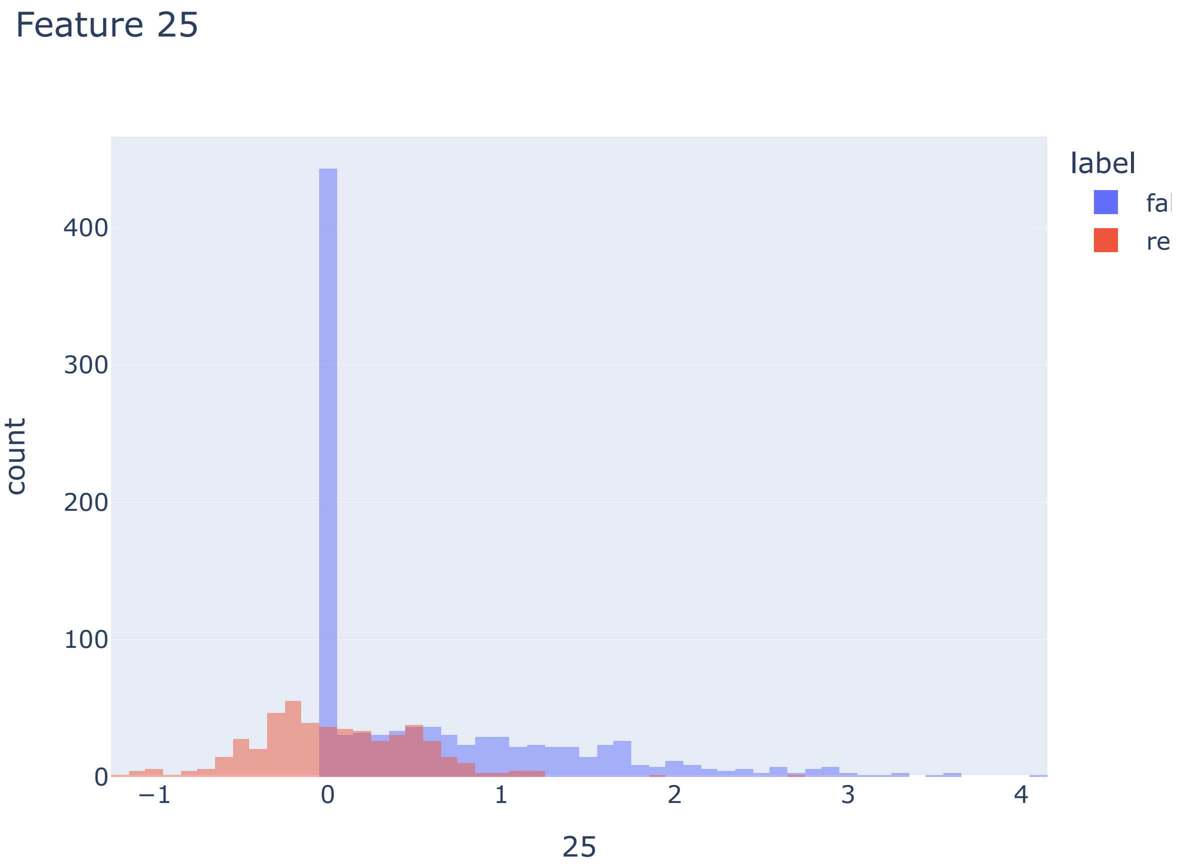

Feature 20
label
fal
re
count
600
500
400
300
200
100
0
−20
−10
0
10
20
20

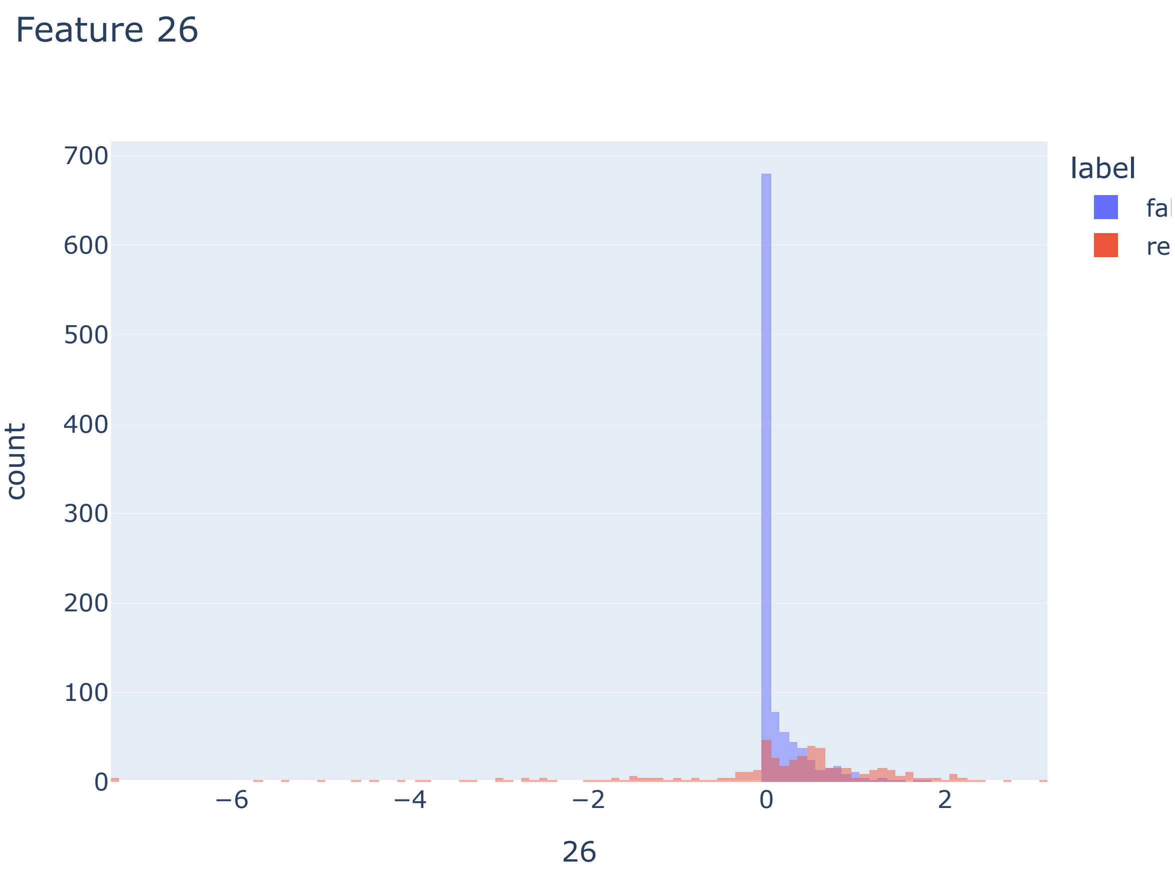

Feature 21
label
fal
re
count
800
700
600
500
400
300
200
100
0
−5
0
5
21

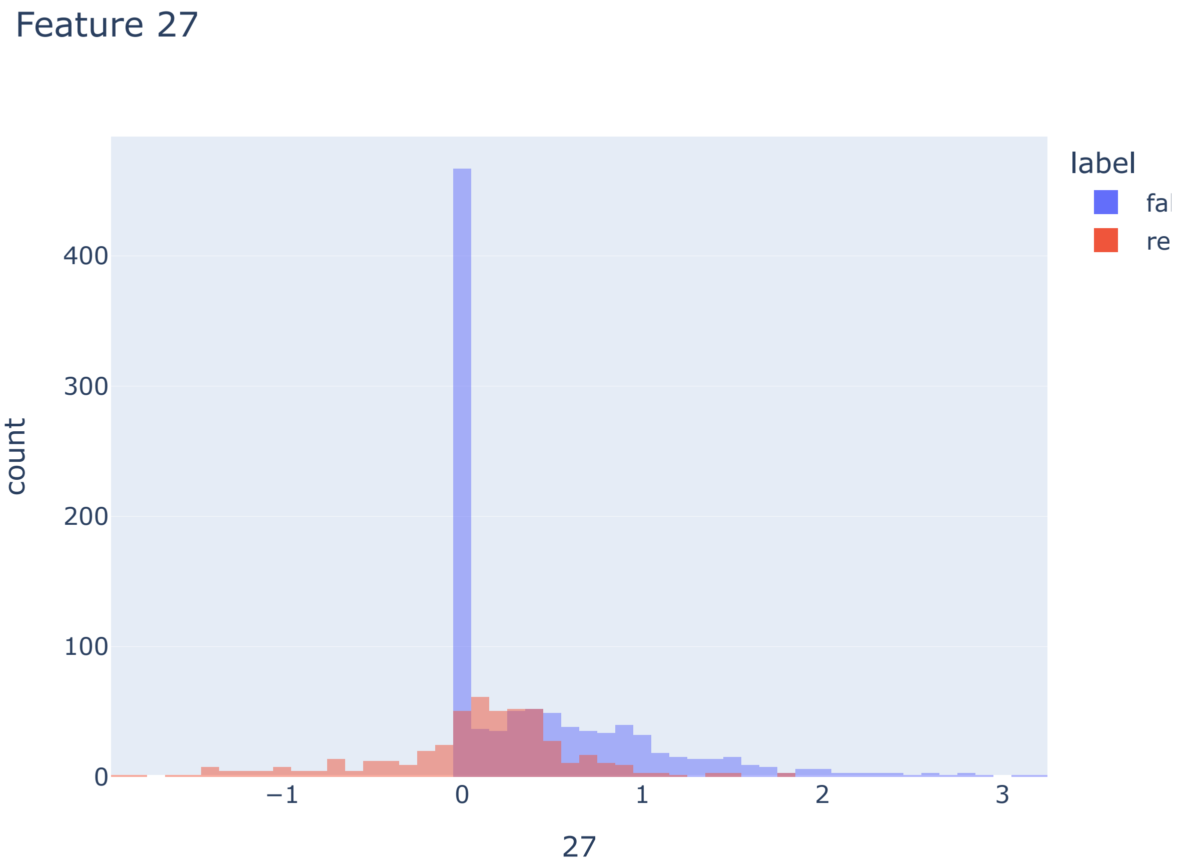

Feature 22
label
fal
re
500
400
300
200
100
0
count
−15
−10
−5
0
5
22

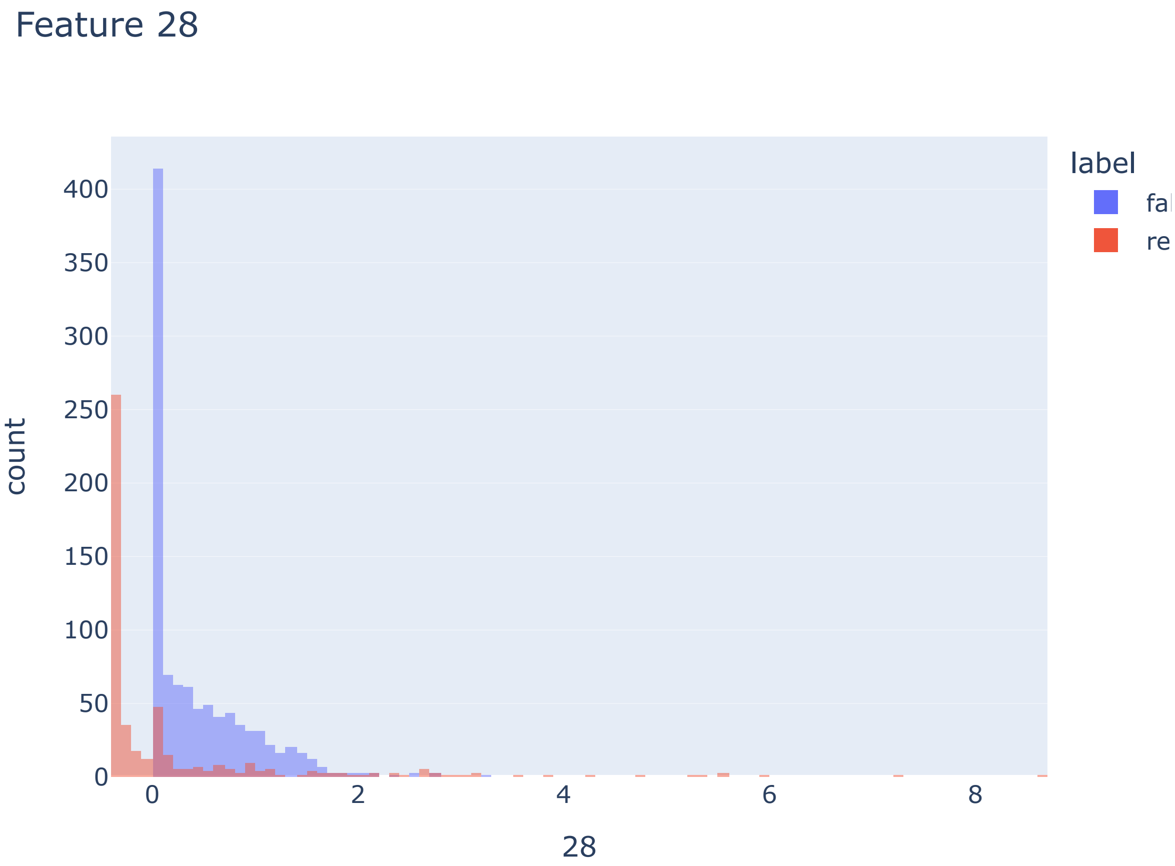

Feature 23
label
fal
re
count
500
400
300
200
100
0
−2
−1
0
1
2
23

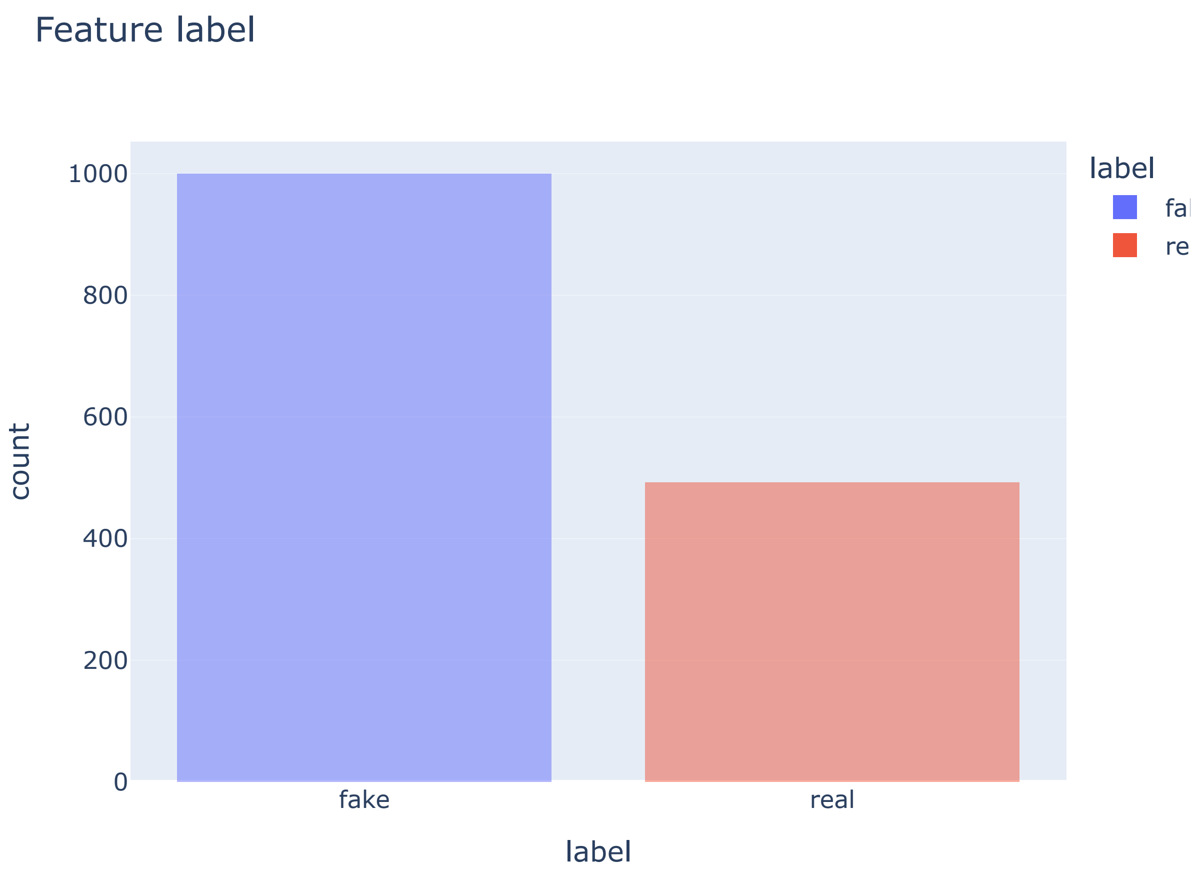

Feature 24
label
fal
re
count
600
500
400
300
200
100
0
−4
−2
0
2
24


Feature 25
count
400
300
200
100
0
−1
0
1
2
3
4
25
label
🟦 fal
🟥 re


Feature 26
700
600
500
400
300
200
100
0
count
−6
−4
−2
0
2
26
label
fal
re


Feature 27
label
fal
re
count
0
100
200
300
400
−1
0
1
2
3
27


Feature 28
label
fal
re
count
400
350
300
250
200
150
100
50
0
0
2
4
6
8
28


Feature label
count
1000
800
600
400
200
0
fake
real
label
label
fal
re

```
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
```

```
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
```

```
In [ ]:
```