MOBILE HEALTH HUMAN BEHAVIOR

NAME: THESLEENA.P

TABLE CONTENTS:

PROBLEM STATEMENT

Mobile health human behavior,is a term that describes the fastly developing field of digital health providing care and treatment for the population via mobile technology such as smart devices.Human behavior is diverse and complex,this nature has posed a great challenge on learning,and most importantly,predicting the human behavior from daily activities.

OBJECTIVES

- Derivation of data-driven insights from a mobile health human behavior data by means of appropriate visuals.
- Building and training a machine learning model that enables the prediction of human behavior by the means of a mobile health device.

Double-click (or enter) to edit

DATASET DESCRIPTION

- Activities: 12
- Sensor devices: 2
- Subjects: 10

The collected dataset comprises body motion and vital signs recordings for persons of diverse profile while performing 12 physical activities.The collected dataset comprises body motion and vital signs recordings for ten volunteers of diverse profile while performing 12 physical activities . Our volunteers' wearable sensors were used for the recordings. The sensors were respectively placed on the subject's right wrist and left ankle and attached by using elastic straps. The use of multiple sensors permits us to measure the motion experienced by diverse body parts, namely, the acceleration, thus better capturing the body dynamics. Each session was recorded using a video camera. This dataset is found to generalize to common activities of the daily living, given the diversity of body parts involved in each one (e.g., frontal elevation of arms vs. knees bending), the intensity of the actions (e.g., cycling vs. sitting and relaxing) and their execution speed or dynamicity (e.g., running vs. standing still). The activities were collected in an out-of-lab environment with no constraints on the way these must be executed, with the exception that the subject should try their best when executing them.

* ** Features Information ***

---

1. alx: acceleration from the left-ankle sensor (X axis)
2. aly: acceleration from the left-ankle sensor (Y axis)
3. alz: acceleration from the left-ankle sensor (Z axis)
4. glx: gyro from the left-ankle sensor (X axis)
5. gly: gyro from the left-ankle sensor (Y axis)
6. glz: gyro from the left-ankle sensor (Z axis)
7. arx: acceleration from the right-lower-arm sensor (X axis)

8. ary: acceleration from the right-lower-arm sensor (Y axis)
9. arz: acceleration from the right-lower-arm sensor (Z axis)
10. grx: gyro from the right-lower-arm sensor (X axis)
11. gry: gyro from the right-lower-arm sensor (Y axis)
12. grz: gyro from the right-lower-arm sensor (Z axis)
13. subject: volunteer number

. Target Information:***

---

14. Activity: corresponding activity

ACTIVITY SET

The activity set is listed in the following:

Activity set

1. L1: Standing still (1 min)
2. L2: Sitting and relaxing (1 min)
3. L3: Lying down (1 min)
4. L4: Walking (1 min)
5. L5: Climbing stairs (1 min)
6. L6: Waist bends forward (20x)
7. L7: Frontal elevation of arms (20x)
8. L8: Knees bending (crouching) (20x)
9. L9: Cycling (1 min)
10. L10: Jogging (1 min)
11. L11: Running (1 min)
12. L12: Jump front & back (20x)

NOTE: In brackets are the number of repetitions (Nx) or the duration of the exercises (min)

DATA COLLECTION

Preprocessing is cleaning and preparing data is critical in machine learning.Working with big datasets in python requires attention and effort.If not done right,it may result in flawed models and bad predictions.

```
# importing required libraries
import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
```

The ML projects,the right tools and libraries are crucial.They greatly improve our work and the outcomes of our projects.We use python's powerful libraries like Pandas and NumPy for cleaning and fixing our data.

- sklearn:

---

sklearn is a top Python library for machine learning.It offers many algorithms for tasks like classification and clustering.It's easy to use and has lots of documentation.This makes sklearn great for testing different machine learning models.

- NumPy:

NumPy is key for scientific computing in python.It helps with efficient numerical operations and handling large data.Other ML libraries often use Numpy,making it important for our workflow.

- Pandas:

Pandas is great for data manipulation and analysis.It makes it easy to clean,transform,and explore data.With tools for handling missing data and merging datasets,pandas is crucial for machine learning.

- Matplotlib:

Matplotlib let us create many visualizations.We can make line plots,scatter plots and more.Matplotlib is a widely used Python library for creating static, interactive, and animated visualizations in Python. It provides a wide range of tools for producing plots and charts, from simple line graphs to complex 3D visualizations. The most commonly used module in Matplotlib is pyplot, which provides an interface for creating visualizations similar to MATLAB.

```
Start coding or generate with AI.
```

1.Load the dataset

```
# Load the dataset
df = pd.read_csv("mhealth_raw_data.csv")
df
```

	alx	aly	alz	glx	gly	glz	arx	ary	arz	grx	gry	
0	2.18490	-9.6967	0.63077	0.103900	-0.84053	-0.68762	-8.6499	-4.5781	0.187760	-0.449020	-1.01030	0.03
1	2.38760	-9.5080	0.68389	0.085343	-0.83865	-0.68369	-8.6275	-4.3198	0.023595	-0.449020	-1.01030	0.03
2	2.40860	-9.5674	0.68113	0.085343	-0.83865	-0.68369	-8.5055	-4.2772	0.275720	-0.449020	-1.01030	0.03
3	2.18140	-9.4301	0.55031	0.085343	-0.83865	-0.68369	-8.6279	-4.3163	0.367520	-0.456860	-1.00820	0.02
4	2.41730	-9.3889	0.71098	0.085343	-0.83865	-0.68369	-8.7008	-4.1459	0.407290	-0.456860	-1.00820	0.02
...	...	...	...	...	...	...	...	...	...	...	...	
1048570	0.32374	-9.9511	-1.04640	-0.569570	-0.75422	-0.52456	-6.5886	-10.5290	2.129800	-0.182350	-0.92813	-0.55
1048571	0.40570	-10.0510	-1.11310	-0.569570	-0.75422	-0.52456	-6.0806	-10.5710	1.877700	-0.182350	-0.92813	-0.55
1048572	0.48707	-10.0110	-1.09980	-0.593690	-0.75047	-0.53045	-5.3752	-10.0910	1.870400	-0.078431	-0.96509	-0.56
1048573	0.41283	-9.8038	-1.17360	-0.593690	-0.75047	-0.53045	-4.6322	-9.7003	1.973300	-0.078431	-0.96509	-0.56
1048574	0.29146	-9.8527	-1.11470	-0.593690	-0.75047	-0.53045	-3.9156	-9.5098	2.024500	-0.078431	-0.96509	-0.56

1048575 rows × 14 columns

Now,We load the dataset with the read_csv method using Pandas.It converts into dataframe.A DataFrame is a data structure made up of rows and columns.

DATA PREPROCESSING:

2.Understand the data structure

```
df.shape
```

```
(1048575, 14)
```

The df.shape method provides information about the number of rows and columns ina DataFrame quickly.Here,we got 1048575 rows and 14 columns.

```
df.head()
```

	alx	aly	alz	glx	gly	glz	arx	ary	arz	grx	gry	grz	Activit
0	2.1849	-9.6967	0.63077	0.103900	-0.84053	-0.68762	-8.6499	-4.5781	0.187760	-0.44902	-1.0103	0.034483	
1	2.3876	-9.5080	0.68389	0.085343	-0.83865	-0.68369	-8.6275	-4.3198	0.023595	-0.44902	-1.0103	0.034483	
2	2.4086	-9.5674	0.68113	0.085343	-0.83865	-0.68369	-8.5055	-4.2772	0.275720	-0.44902	-1.0103	0.034483	
3	2.1814	-9.4301	0.55031	0.085343	-0.83865	-0.68369	-8.6279	-4.3163	0.367520	-0.45686	-1.0082	0.025862	
4	2.4173	-9.3889	0.71098	0.085343	-0.83865	-0.68369	-8.7008	-4.1459	0.407290	-0.45686	-1.0082	0.025862	

The df.head() function in Python, specifically in the pandas library, is used to display the first few rows of a DataFrame (df). By default, it shows the first five rows, but you can specify a different number by passing an integer as an argument, like df.head(10) to see the first ten rows. It provides a quick glance at the data structure, including column names and data types. Useful for checking the integrity and format of your data right after loading it.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 14 columns):
 #   Column    Non-Null Count    Dtype  
---  ------    --------------    -----  
 0   alx       1048575 non-null  float64
 1   aly       1048575 non-null  float64
 2   alz       1048575 non-null  float64
 3   glx       1048575 non-null  float64
 4   gly       1048575 non-null  float64
 5   glz       1048575 non-null  float64
 6   arx       1048575 non-null  float64
 7   ary       1048575 non-null  float64
 8   arz       1048575 non-null  float64
 9   grx       1048575 non-null  float64
 10  gry       1048575 non-null  float64
 11  grz       1048575 non-null  float64
 12  Activity  1048575 non-null  int64  
 13  subject   1048575 non-null  object 
dtypes: float64(12), int64(1), object(1)
memory usage: 112.0+ MB
```

The df.info() function in pandas provides a concise summary of a DataFrame (df). It gives you important information about the structure and content of the DataFrame. Overview of DataFrame: It summarizes the DataFrame, helping you understand its shape, data types, and memory usage. Data Types: Displays the data type of each column, which is crucial for understanding how the data is stored and manipulated. Missing Values: Indicates the number of non-null entries in each column, helping identify missing data. Key Components of the Output:

- Index Range: Shows the range of the index (e.g., RangeIndex: 0 to 4).
- Column Information: Lists each column with its name, data type, and number of non-null entries.
- Dtype Summary: Provides a summary of the data types in the DataFrame (e.g., int64, float64, object).
- Memory Usage: Indicates the amount of memory used by the DataFrame.

```
df.describe()
```

	alx	aly	alz	glx	gly	glz	arx	
count	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.04857
mean	1.481221e+00	-9.574932e+00	-8.791344e-01	-1.007315e-02	-6.135945e-01	-1.376290e-01	-3.551709e+00	-5.70737
std	3.690661e+00	4.071565e+00	5.303325e+00	4.942075e-01	3.568763e-01	5.452972e-01	4.711360e+00	5.72678
min	-2.214600e+01	-1.961900e+01	-1.937300e+01	-2.146600e+00	-7.789900e+00	-2.567800e+00	-2.236100e+01	-1.89720
25%	1.691250e-01	-1.008400e+01	-2.599400e+00	-4.471200e-01	-8.123800e-01	-5.677800e-01	-5.890450e+00	-9.36465
50%	1.342900e+00	-9.647600e+00	1.909100e-02	-2.968500e-02	-7.035600e-01	-1.493100e-01	-2.870500e+00	-7.38000
75%	2.568100e+00	-9.014300e+00	1.372500e+00	4.415600e-01	-5.422100e-01	3.418500e-01	-1.030400e+00	-2.45035
max	2.005400e+01	2.116100e+01	2.501500e+01	6.048400e+01	2.011300e+00	2.770100e+00	1.986400e+01	2.21910

It seems like we have a dataset with several columns representing sensor measurements (arx, ary, arz, grx, gry, grz) and a categorical column (Activity). The .describe() method has generated summary statistics for each numerical column in your dataset, with information on the count, mean, standard deviation, min, 25th percentile, 50th percentile (median), 75th percentile, and max values.

Count: All columns have 1,048,575 non-null entries, indicating no missing values in the dataset. Mean: The mean values show the central tendency of each sensor's readings. For example, arx has a mean of approximately 1.48, while ary has a mean of about -9.57. Standard Deviation (std): This indicates the variability in the data. For instance, grx has a standard deviation of approximately 4.94, suggesting considerable variability in that column. Min: The minimum values show the lowest observed values in each column. For example, grx has a minimum of about -2.15, and Activity's minimum is 0, indicating a classification where 0 could be a certain activity type. 25%, 50%, 75%: These represent the quartiles of the data (25th, 50th, and 75th percentiles), which help show the distribution of values. For example: arx has a median of 1.34 (50th percentile), and 75% of the values are below 2.57. grx has a median of approximately 0.44 (50th percentile), with 75% of the values below 4.42.

Sensor Data (arx, ary, arz, grx, gry, grz):

These columns likely represent accelerometer and gyroscope data along different axes (x, y, z). arx, ary, arz have negative mean values, indicating the sensors might be oriented in such a way that their readings tend to be negative. grx, gry, grz have a wide range of values, with relatively large standard deviations, which indicates significant variation in the readings. Some columns, like arx, have a maximum value (max) of around 20, and min values can be as low as -22, suggesting the readings can vary significantly.

Activity:

This column seems to be a categorical variable, and the values represent different types of activity. Based on the min, 25th percentile, and max values, it's likely that the Activity column contains discrete integers, with values ranging from 0 to 12 (the activities are likely labeled with these integers).

3.Handle missing values

```
# missing values
df.isnull().sum()
```

	0
alx	0
aly	0
alz	0
glx	0
gly	0
glz	0
arx	0
ary	0
arz	0
grx	0
gry	0
grz	0
Activity	0
subject	0

dtype: int64

The code df.isnull().sum() in pandas checks for missing (null) values in each column of a DataFrame.It looks like the output from df.isnull().sum() indicates that there are no missing values in any of the columns in our dataset. Here's a quick breakdown of the result:

All the columns (alx, aly, alz, glx, gly, glz, arx, ary, arz, grx, gry, grz, Activity, subject) have 0 missing values (NaN), meaning every entry in our dataset is complete. No missing data: we don't need to perform any imputation or handle missing values, which is great because we can proceed directly with analysis, modeling, or other tasks.

4.Handle Duplicates

To remove duplicate rows from a DataFrame, you can use the df.duplicates() function. This function removes rows where all the values are identical (duplicate rows).

```
# checking duplicates
df.duplicated().sum()
```

```
0
```

To removing duplicated rows from a pandas DataFrame where the duplicated rows are marked as 0.

6.Handle Outliers

Outliers are data points that significantly differ from the rest of the data. These values can skew statistical analyses, create bias in machine learning models, or lead to incorrect conclusions. Outliers can be caused by errors in data collection, but they can also represent rare or important events (e.g., fraud detection, rare diseases).

There are several ways to detect outliers:

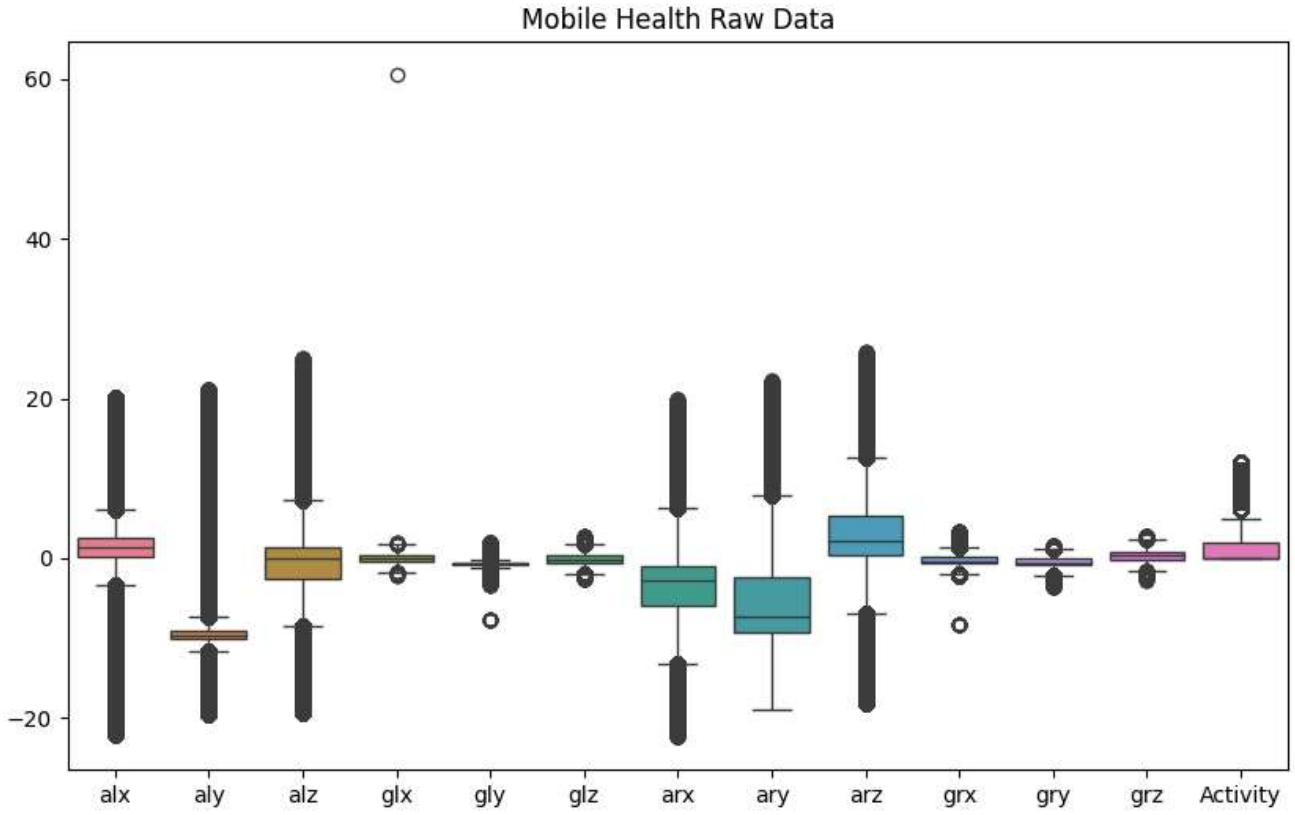
- Boxplot
- IQR method,Z_score method and percentile method
- Histogram
- Skewness

Boxplots are a graphical way to visualize the distribution of data and detect outliers. Outliers are typically any values that lie outside the "whiskers" of the boxplot (i.e., outside the range of Q 1 − 1.5 × I Q R Q1−1.5×IQR and Q 3 + 1.5 × I Q R Q3+1.5×IQR).

Box plot

Boxplots are a graphical way to visualize the distribution of data and detect outliers. Outliers are typically any values that lie outside the "whiskers" of the boxplot (i.e., outside the range of Q 1 − 1.5 × I Q R Q1−1.5×IQR and Q 3 + 1.5 × I Q R Q3+1.5×IQR).

```
# before remove the outlier

plt.figure(figsize=(10, 6))
sns.boxplot(df)
plt.title('Mobile Health Raw Data')
plt.show()
```



IQR Method

We can see outliers in different columns.that can be removed by using IQR method.the columns where outliers exist (values outside the whiskers). Based on that, we can proceed with our method for handling outliers (e.g., Z-score, IQR, removal, capping, etc.)

```
# Remove the outlier all columns by using functions of IQR method
def remove_outliers(df,columns):
    df_filtered = df.copy()

    for col in columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1

        lower_whisker = Q1 - 1.5 * IQR
        upper_whisker = Q3 + 1.5 * IQR

        df_filtered = df_filtered[(df_filtered[col] <= upper_whisker) & (df_filtered[col] >= lower_whisker)]
    return df_filtered


# we can apply outlier columns
df1 = remove_outliers(df,['alx','aly','alz','glx','gly','glz','arx','ary','arz','grx','gry','grz'])
```

The function for removing outliers using the IQR (Interquartile Range) method looks good overall! It follows the correct approach for identifying outliers based on the IQR and filtering out rows that contain outliers.

Skewness

```
# check the skewness of numerical columns
df1.skew(numeric_only=True)
```

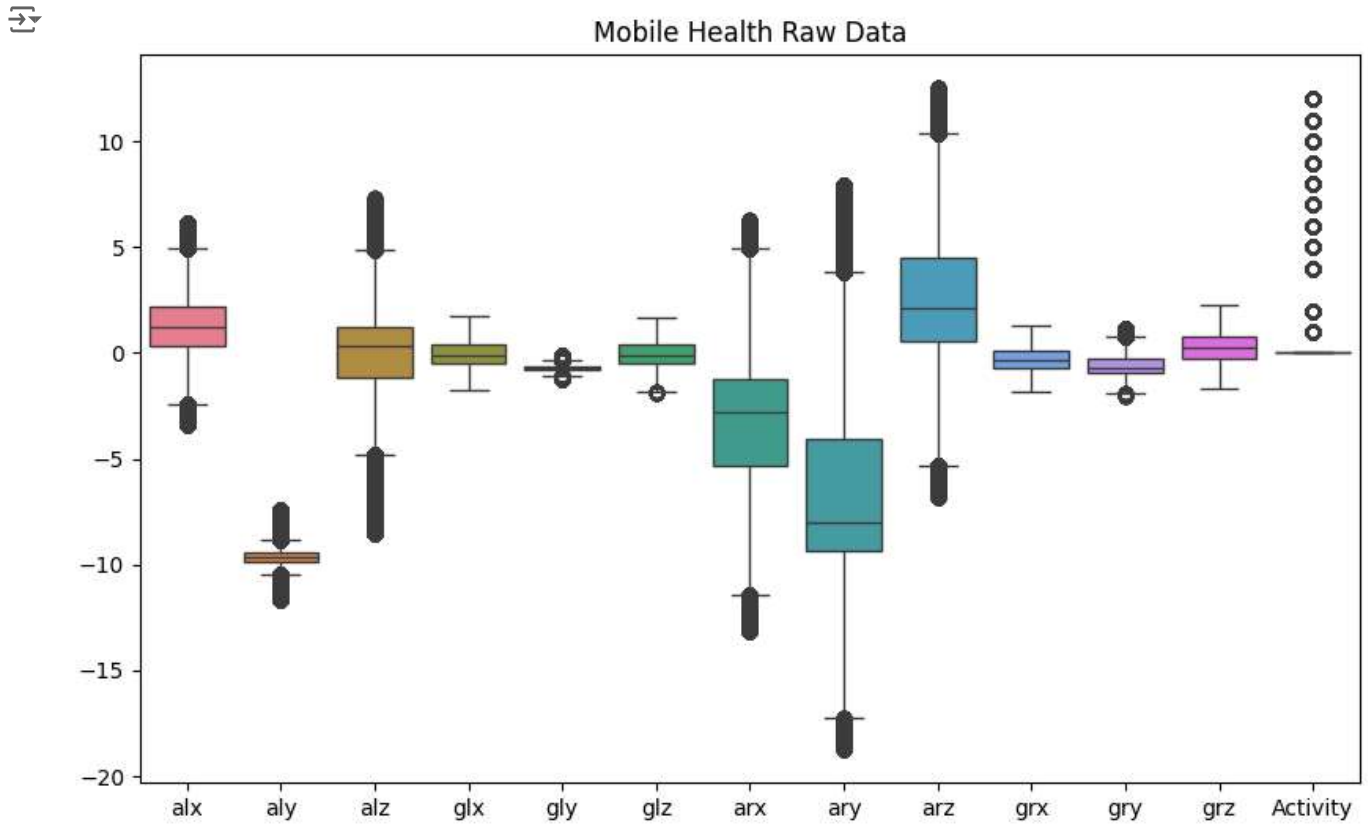
	0
alx	-0.050069
aly	0.306123
alz	-1.007392
glx	0.163344
gly	0.827304
glz	-0.009673
arx	-0.280922
ary	1.095049
arz	0.291986
grx	0.349160
gry	1.089361
grz	-0.131270
Activity	1.920480

dtype: float64

```
df1.shape
```

```
(641855, 14)
```

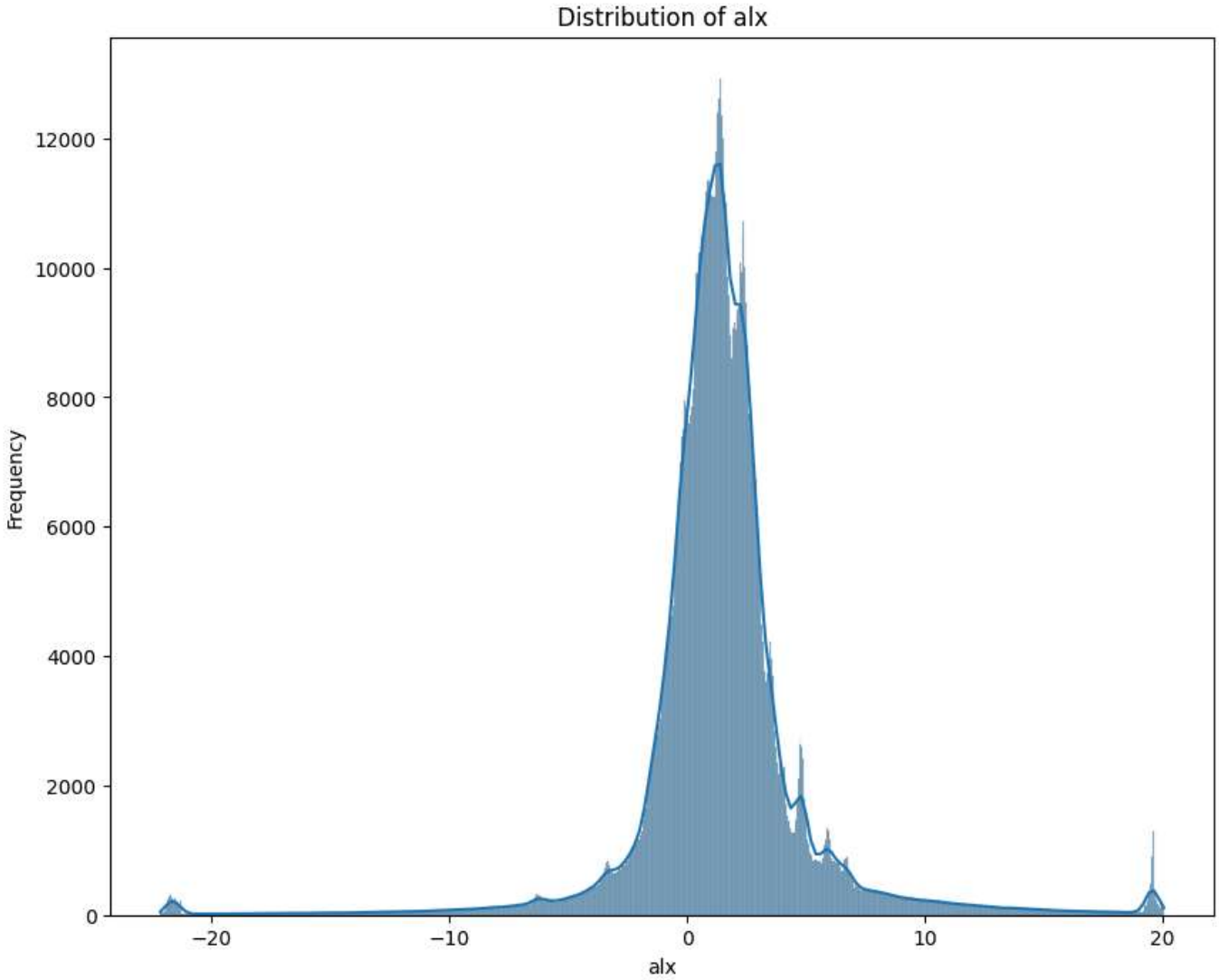
```
# After removed the outliers

plt.figure(figsize=(10, 6))
sns.boxplot(df1)
plt.title('Mobile Health Raw Data')
plt.show()
```

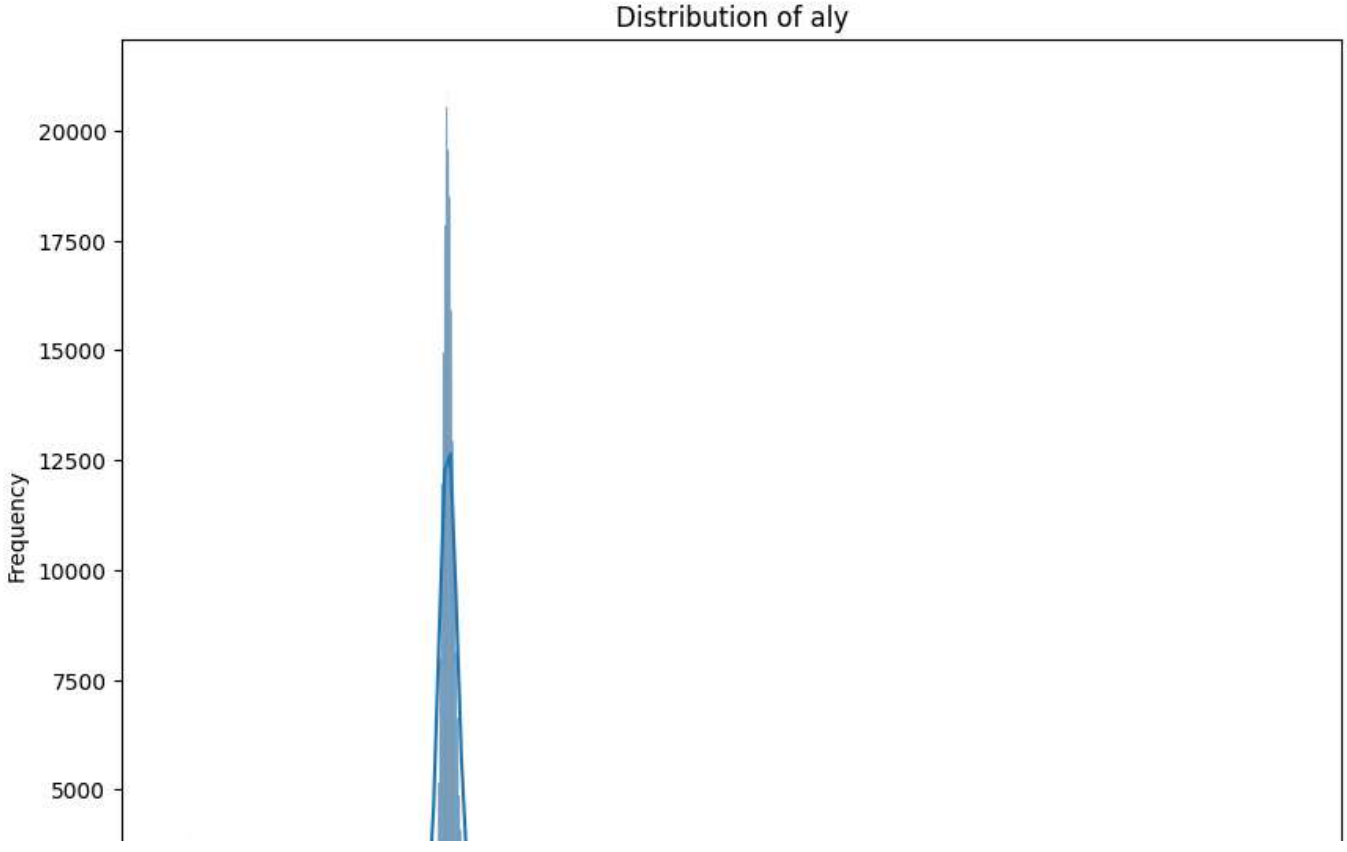


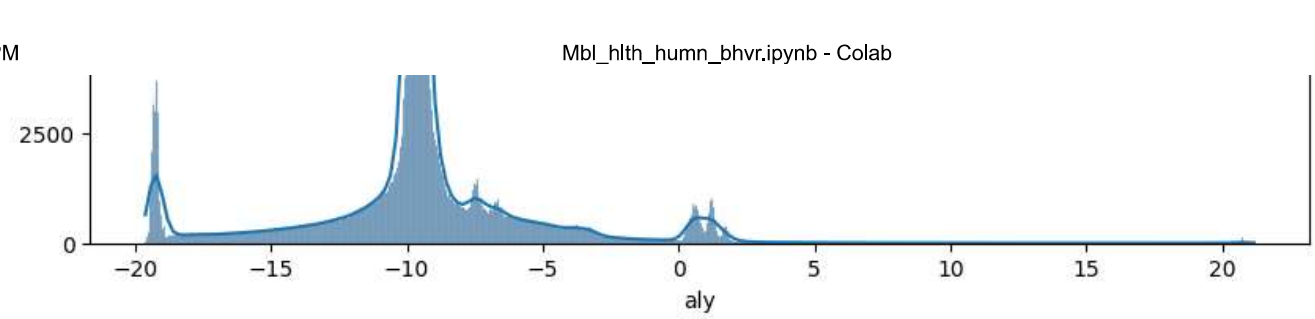
Histogram

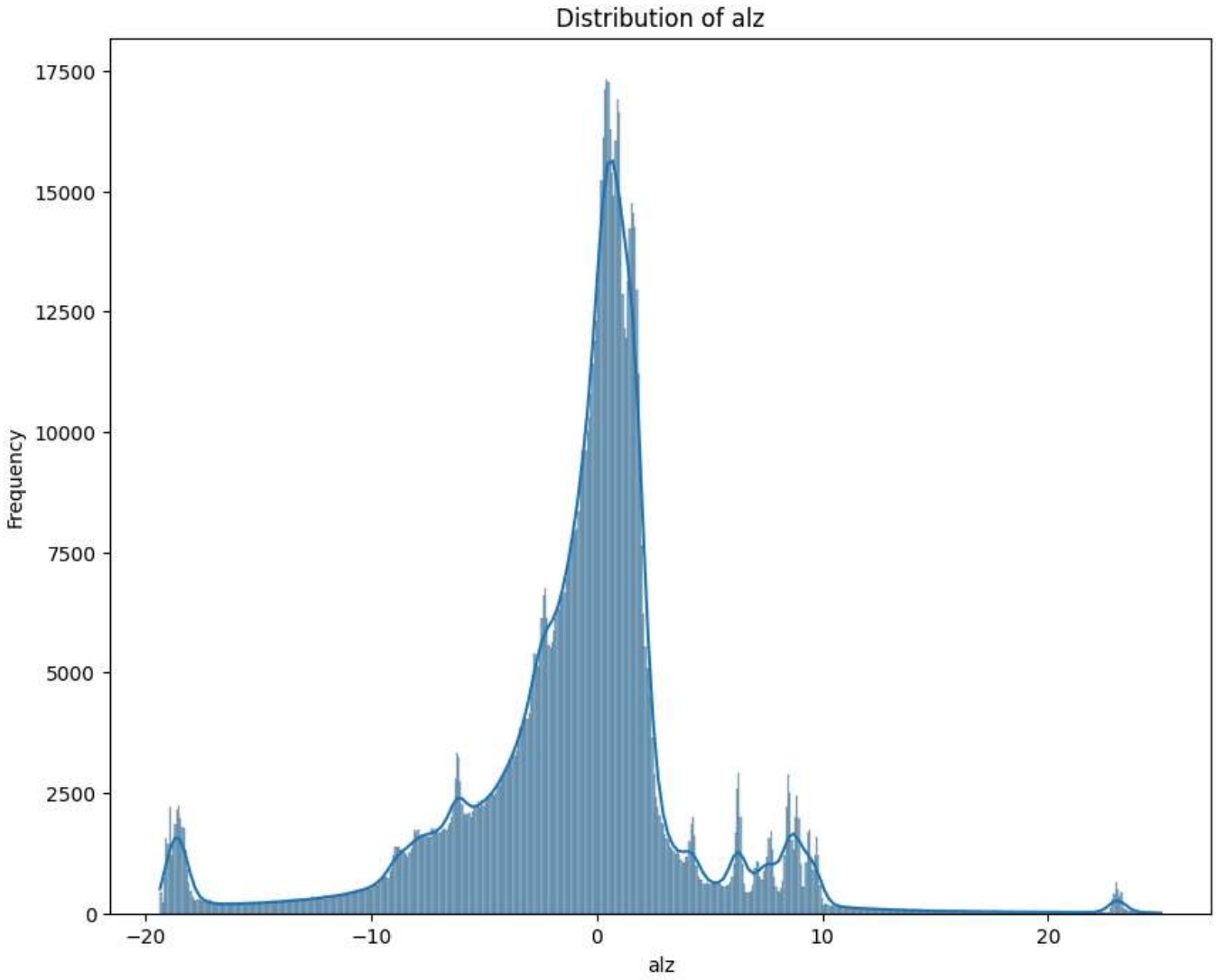
Histograms with KDE are an excellent way to explore the distribution of your data visually before proceeding with any data transformations or outlier removal.

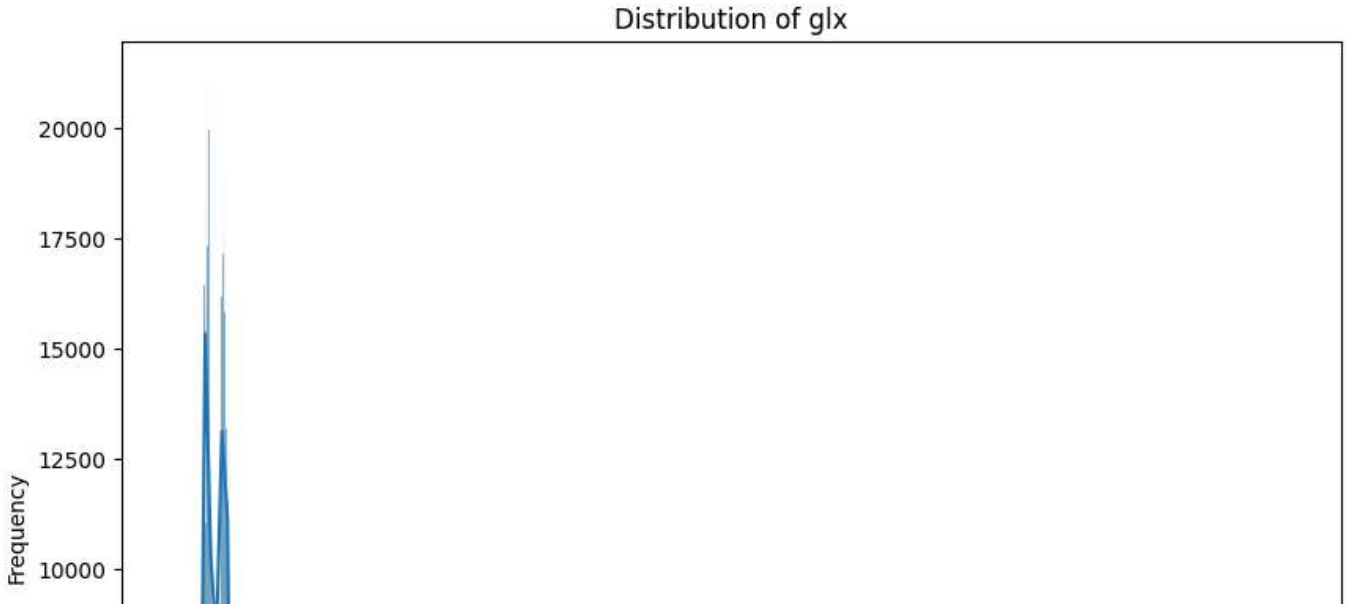
```
# create a histplot for each numerical columns before remove the outliers
for column in df.select_dtypes(include=['float64', 'int64']).columns:
    plt.figure(figsize=(10, 8))
    sns.histplot(df[column], kde=True)
    plt.title('Distribution of ' + column)
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()
```

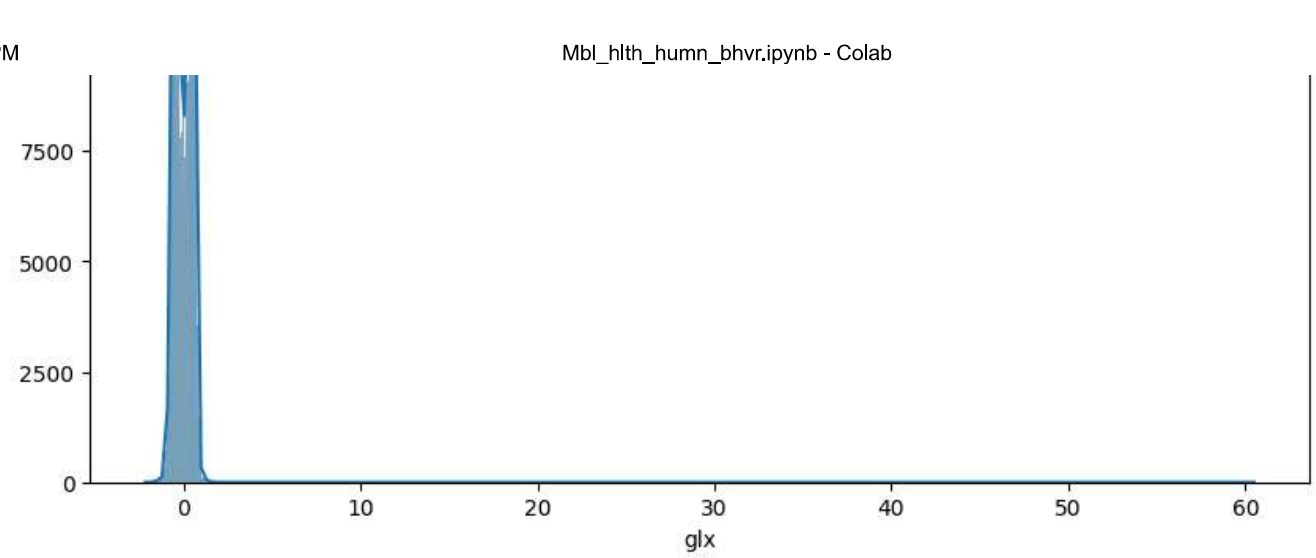

Distribution of alx
Frequency
alx
0
2000
4000
6000
8000
10000
12000
−20
−10
0
10
20

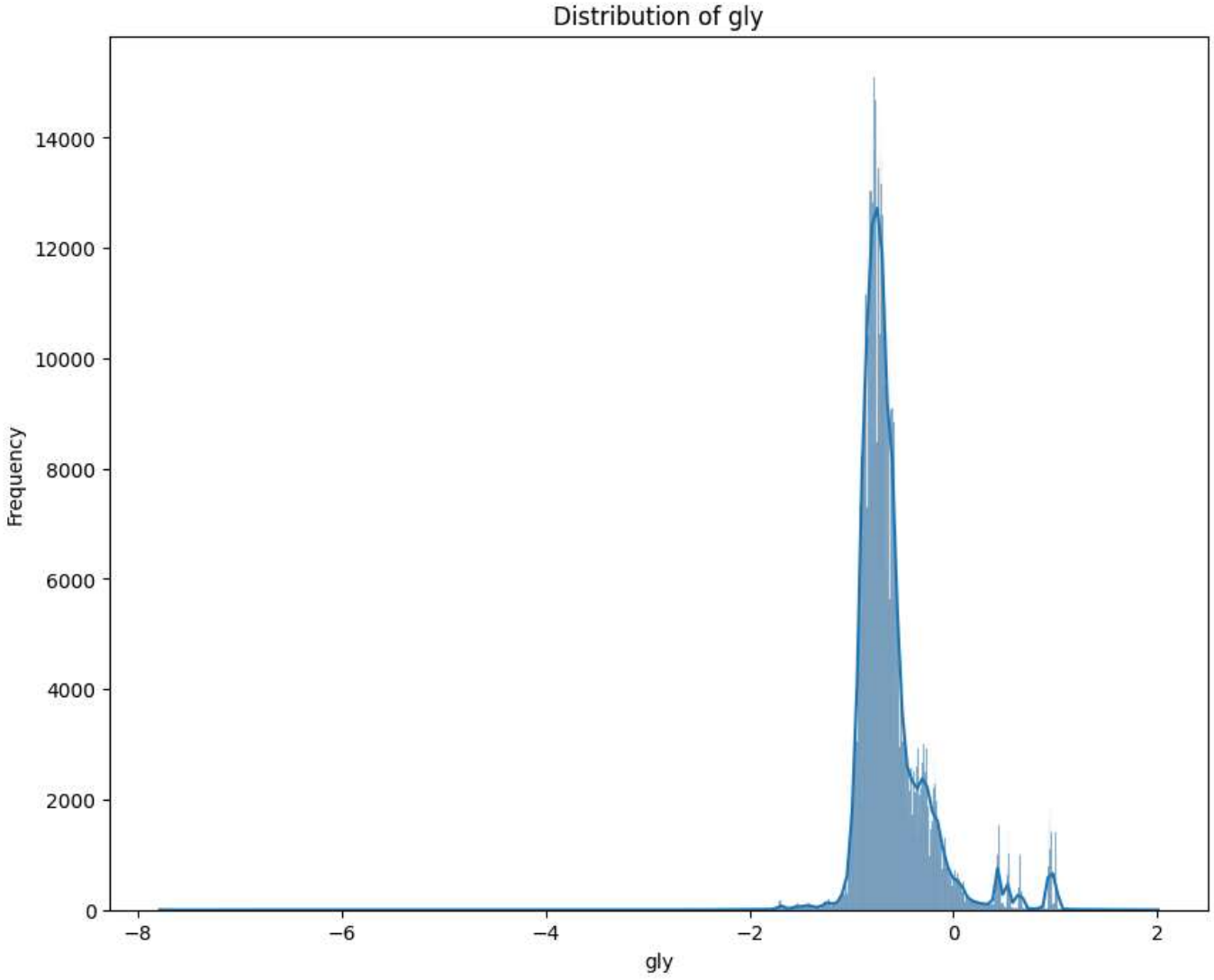

Distribution of aly
Frequency
5000
7500
10000
12500
15000
17500
20000

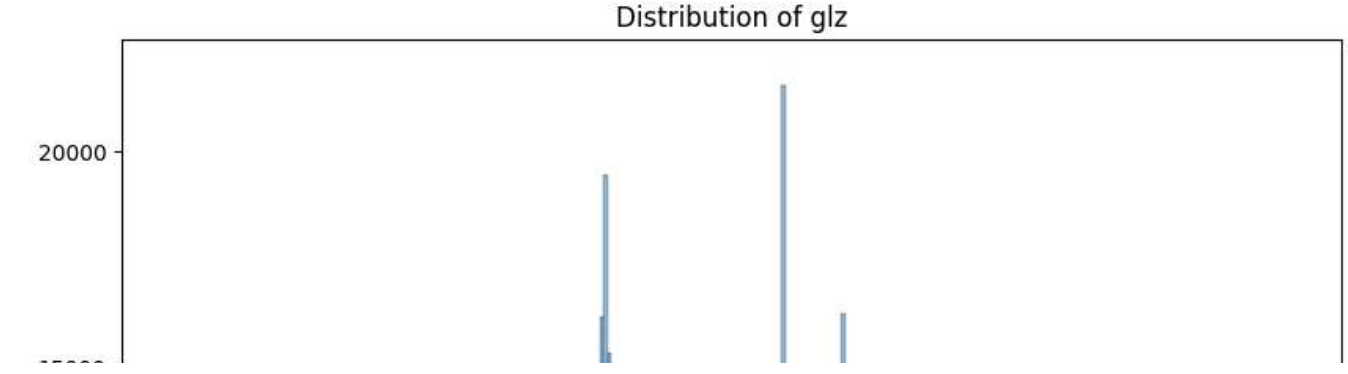

2500
0
−20
−15
−10
−5
0
5
10
15
20
aly

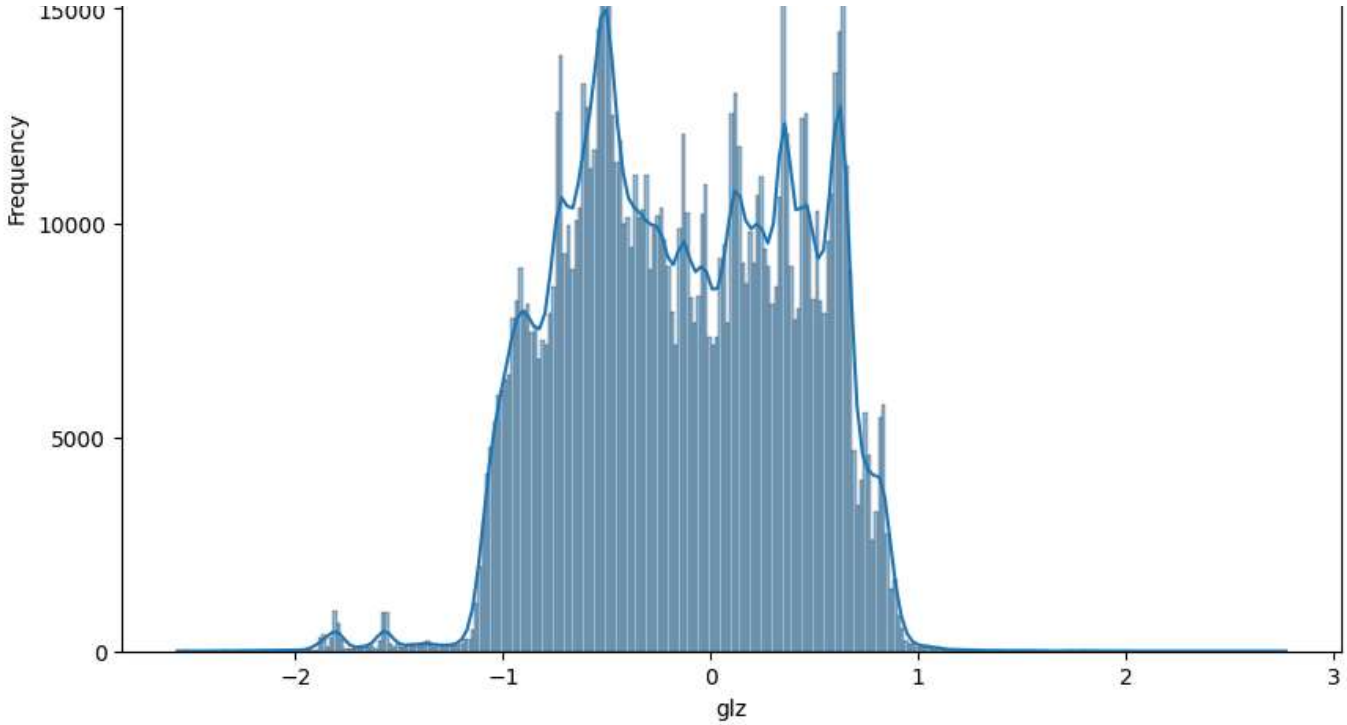

Distribution of alz
17500
15000
12500
10000
7500
5000
2500
0
Frequency
−20
−10
0
10
20
alz

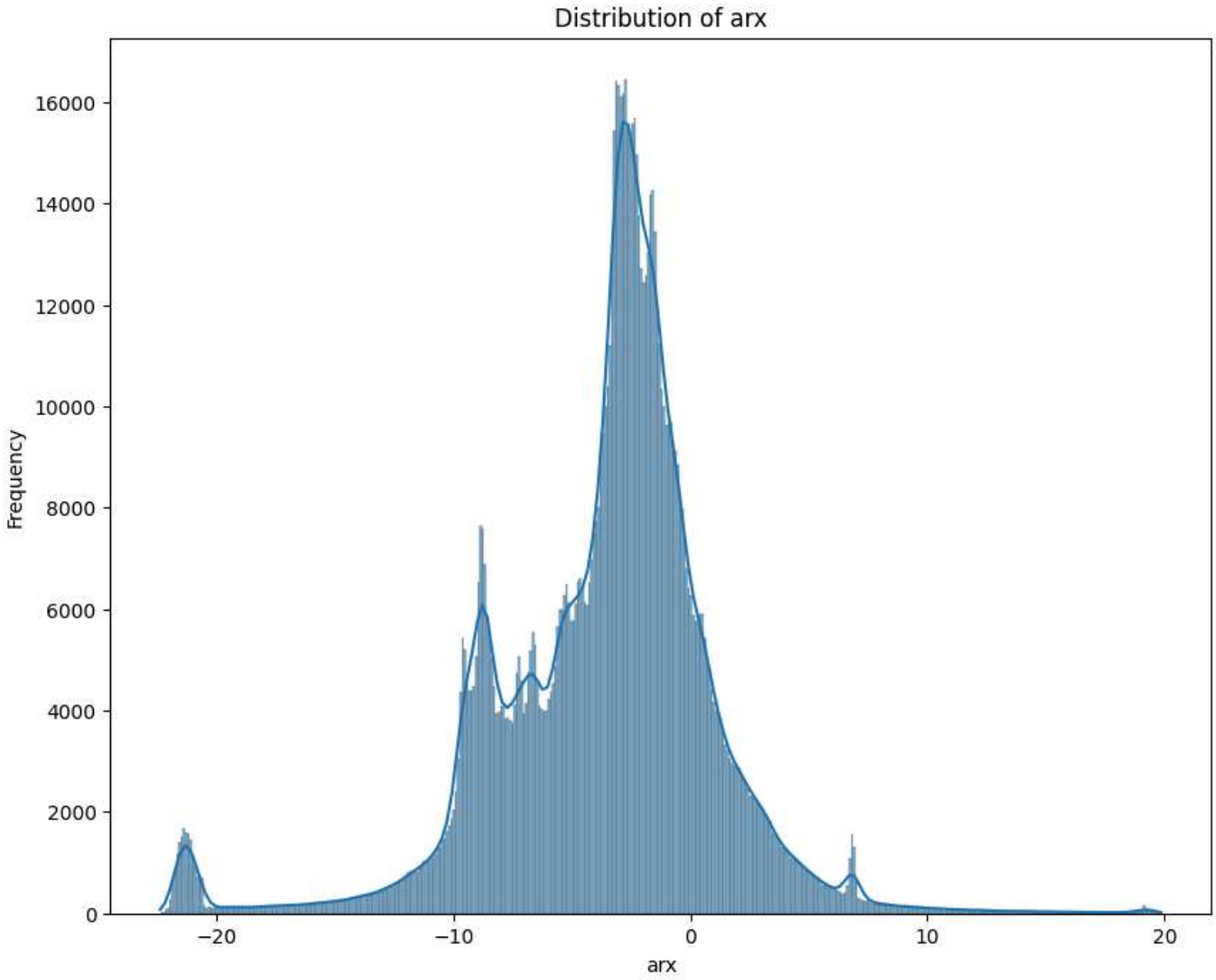

Distribution of glx
20000
17500
15000
12500
10000
Frequency

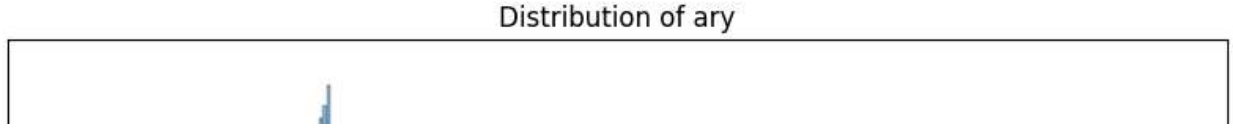

7500
5000
2500
0
0
10
20
30
40
50
60
glx

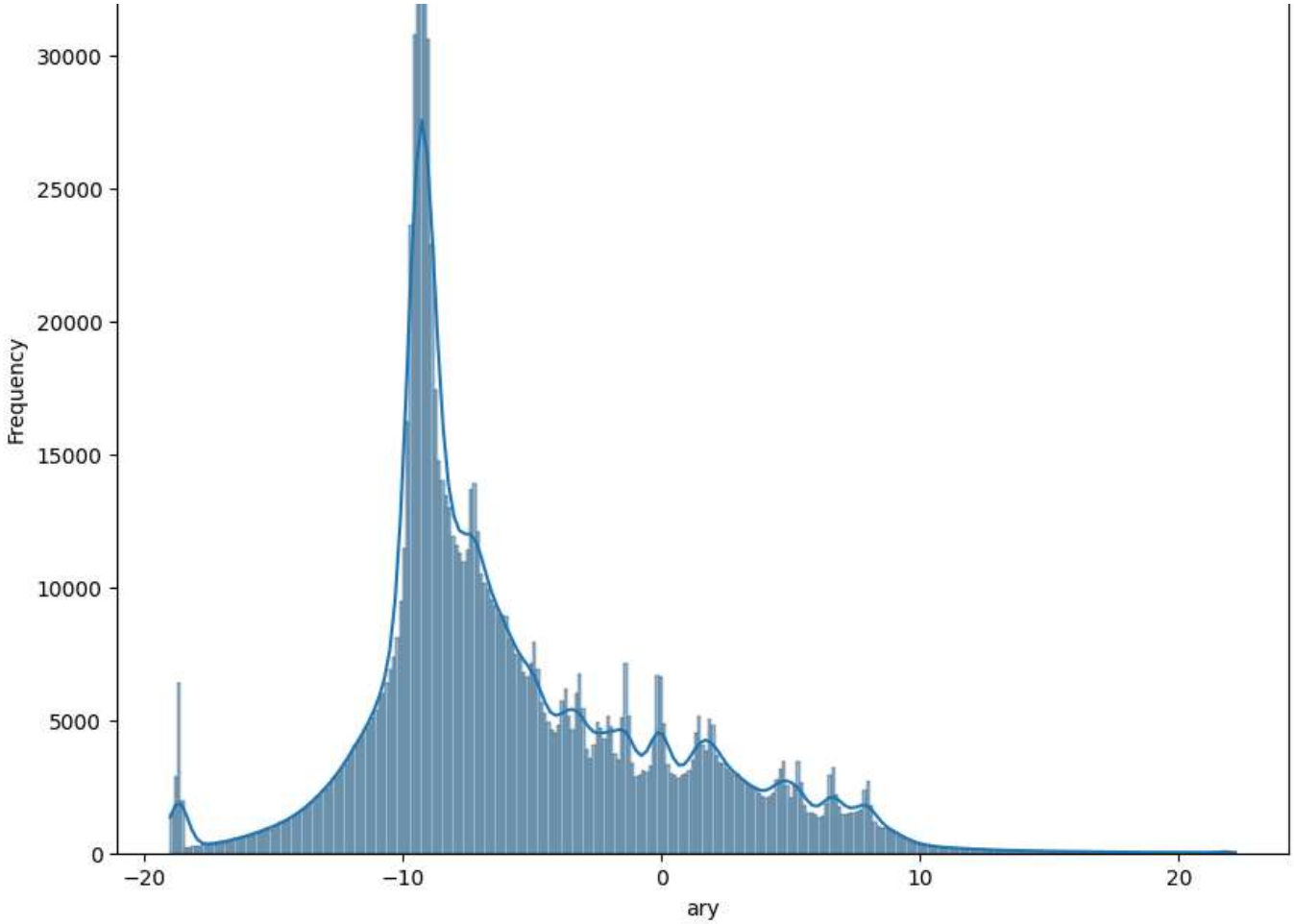

Distribution of gly
14000
12000
10000
8000
6000
4000
2000
0
Frequency
−8
−6
−4
−2
0
2
gly

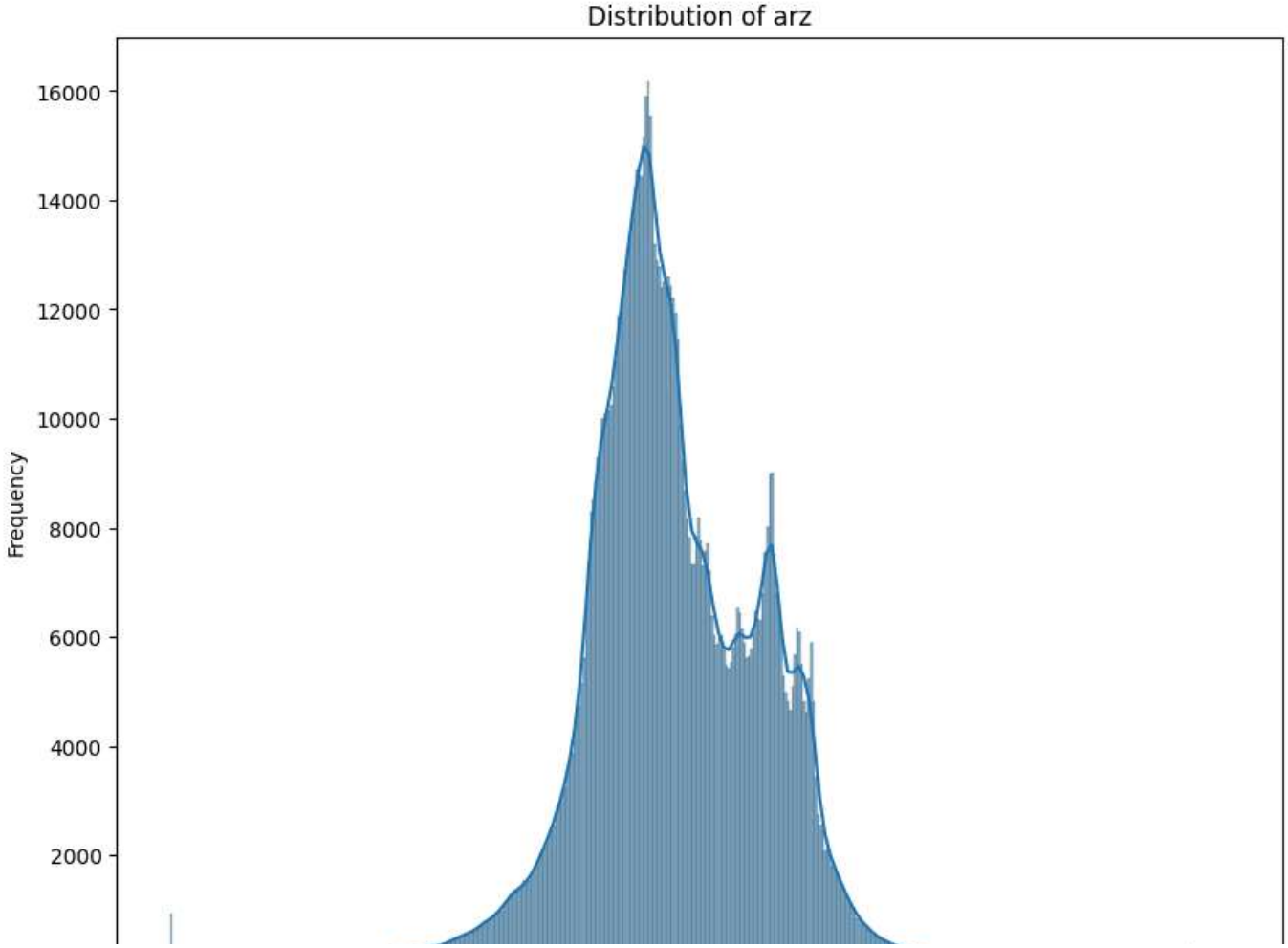

Distribution of glz
20000

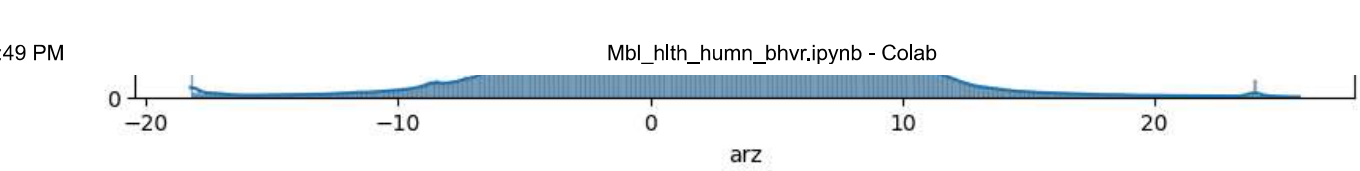

15000
10000
5000
0
Frequency
−2
−1
0
1
2
3
glz

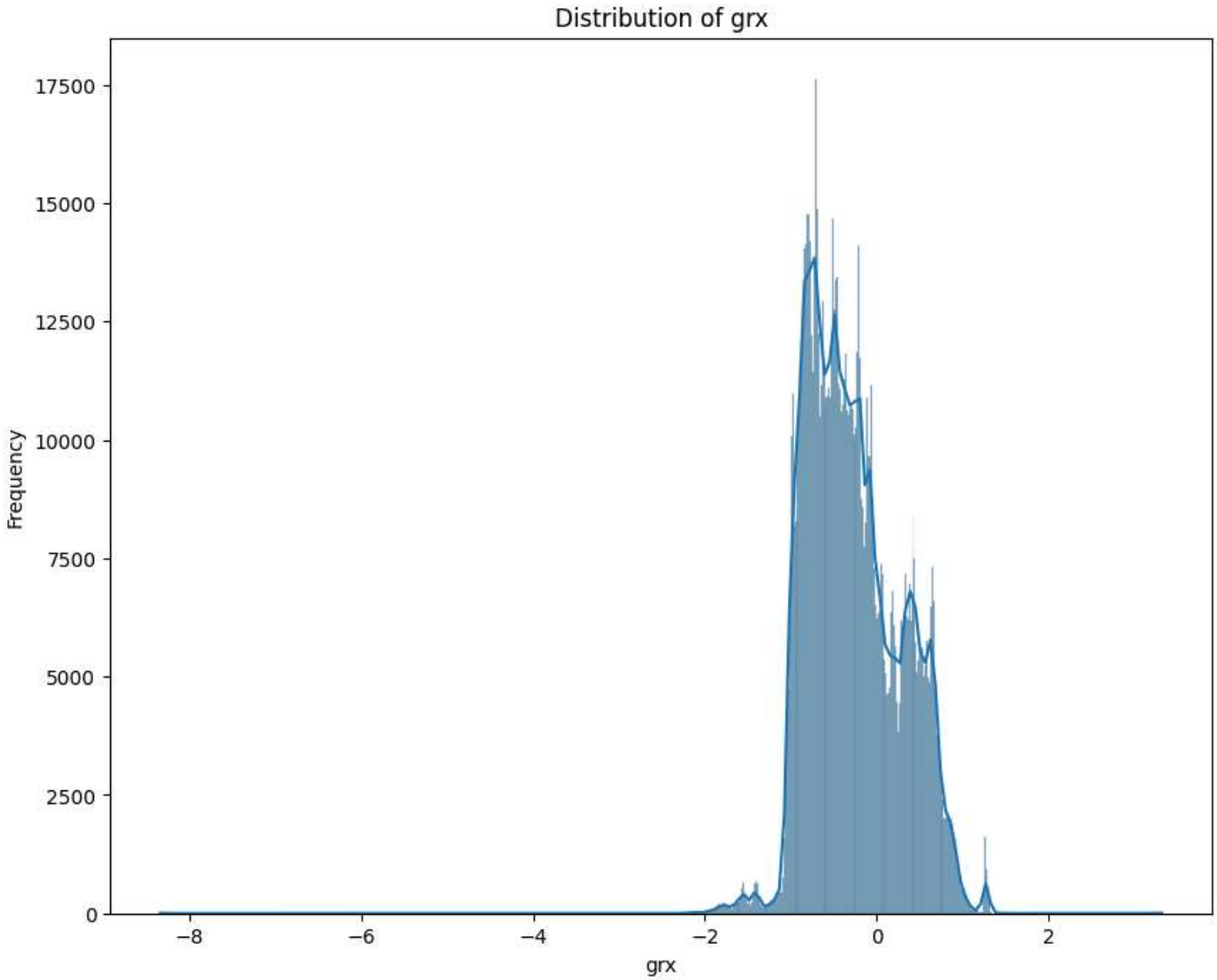

Distribution of arx
16000
14000
12000
10000
8000
6000
4000
2000
0
Frequency
−20
−10
0
10
20
arx

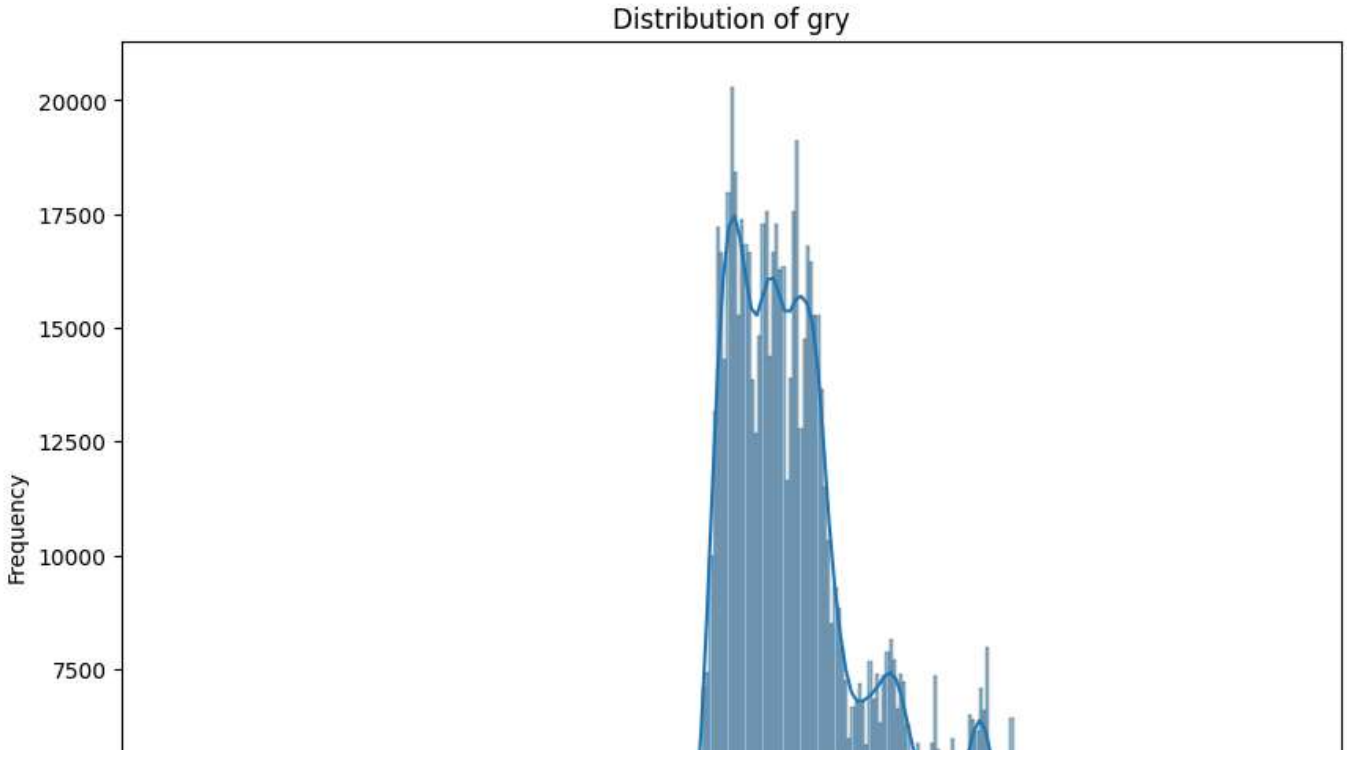

Distribution of ary

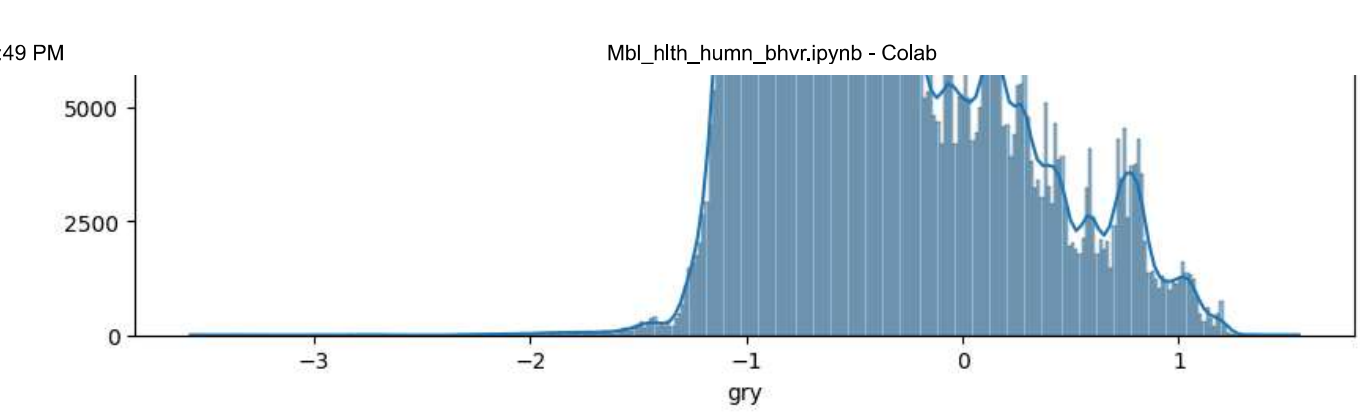

30000
25000
20000
15000
10000
5000
0
Frequency
−20
−10
0
10
20
ary

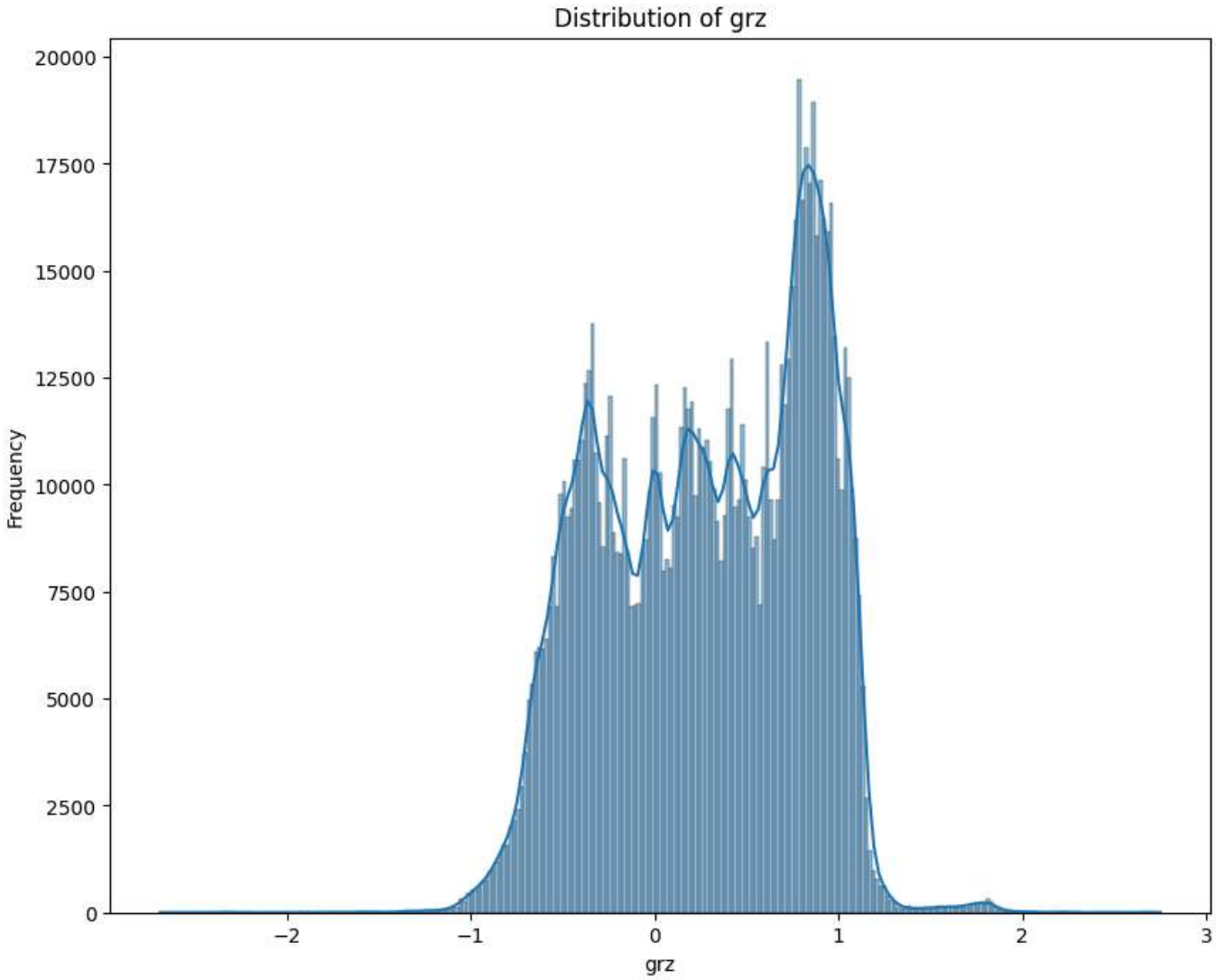

Distribution of arz
16000
14000
12000
10000
8000
6000
4000
2000
Frequency

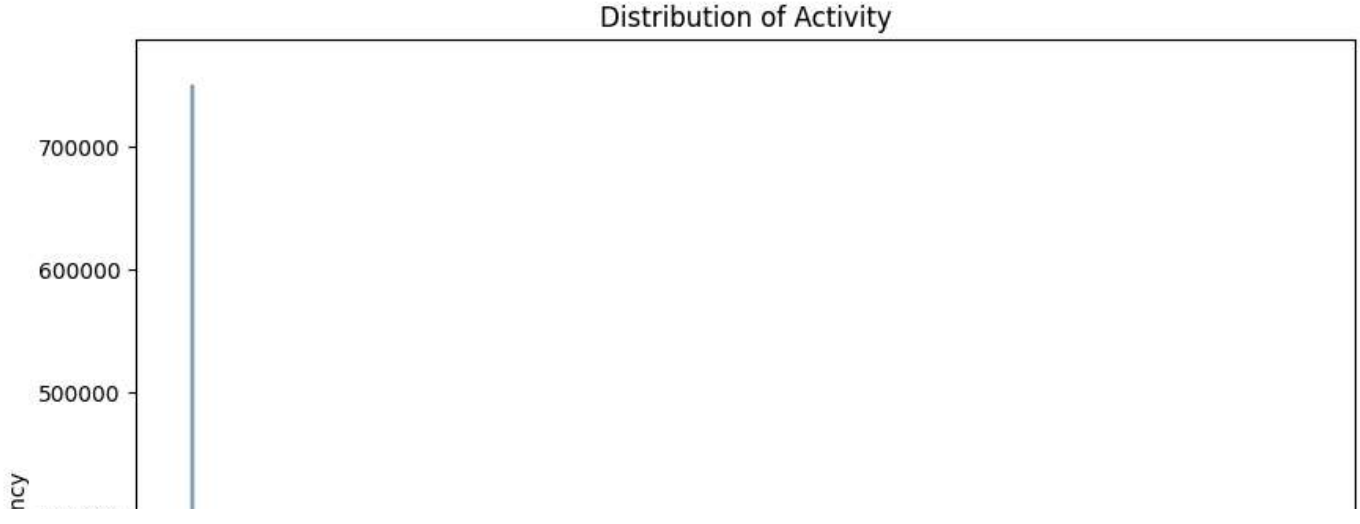

0
−20
−10
0
10
20
arz

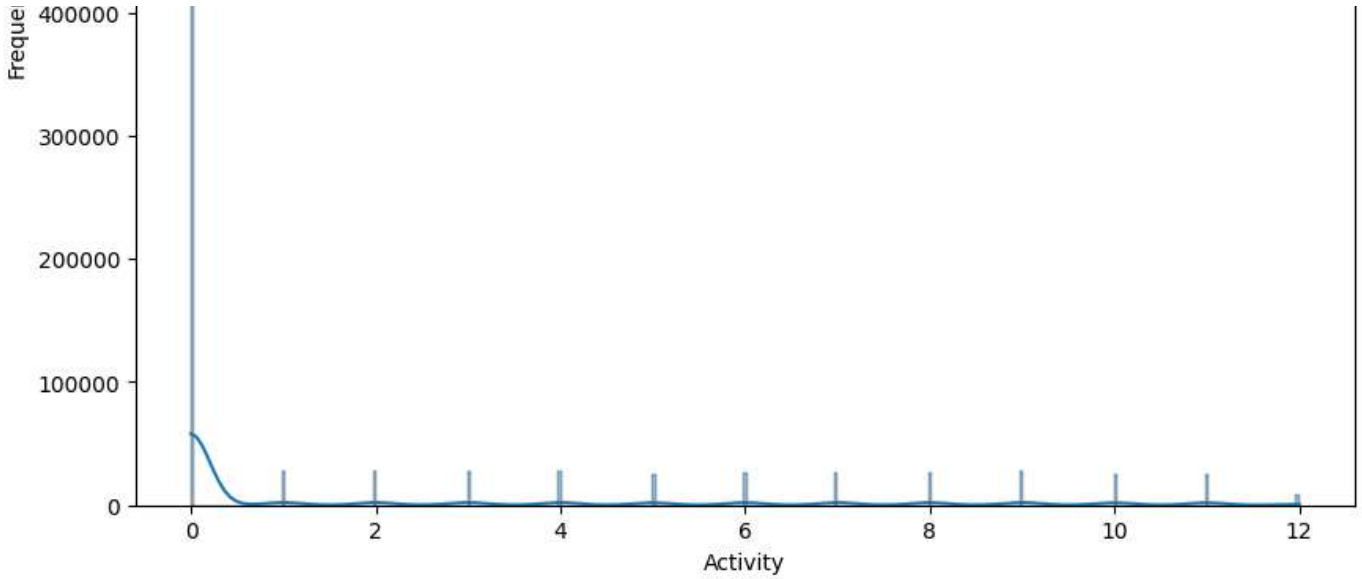

Distribution of grx
Frequency
17500
15000
12500
10000
7500
5000
2500
0
−8
−6
−4
−2
0
2
grx


Distribution of gry
Frequency
20000
17500
15000
12500
10000
7500


5000
2500
0
−3
−2
−1
0
1
gry

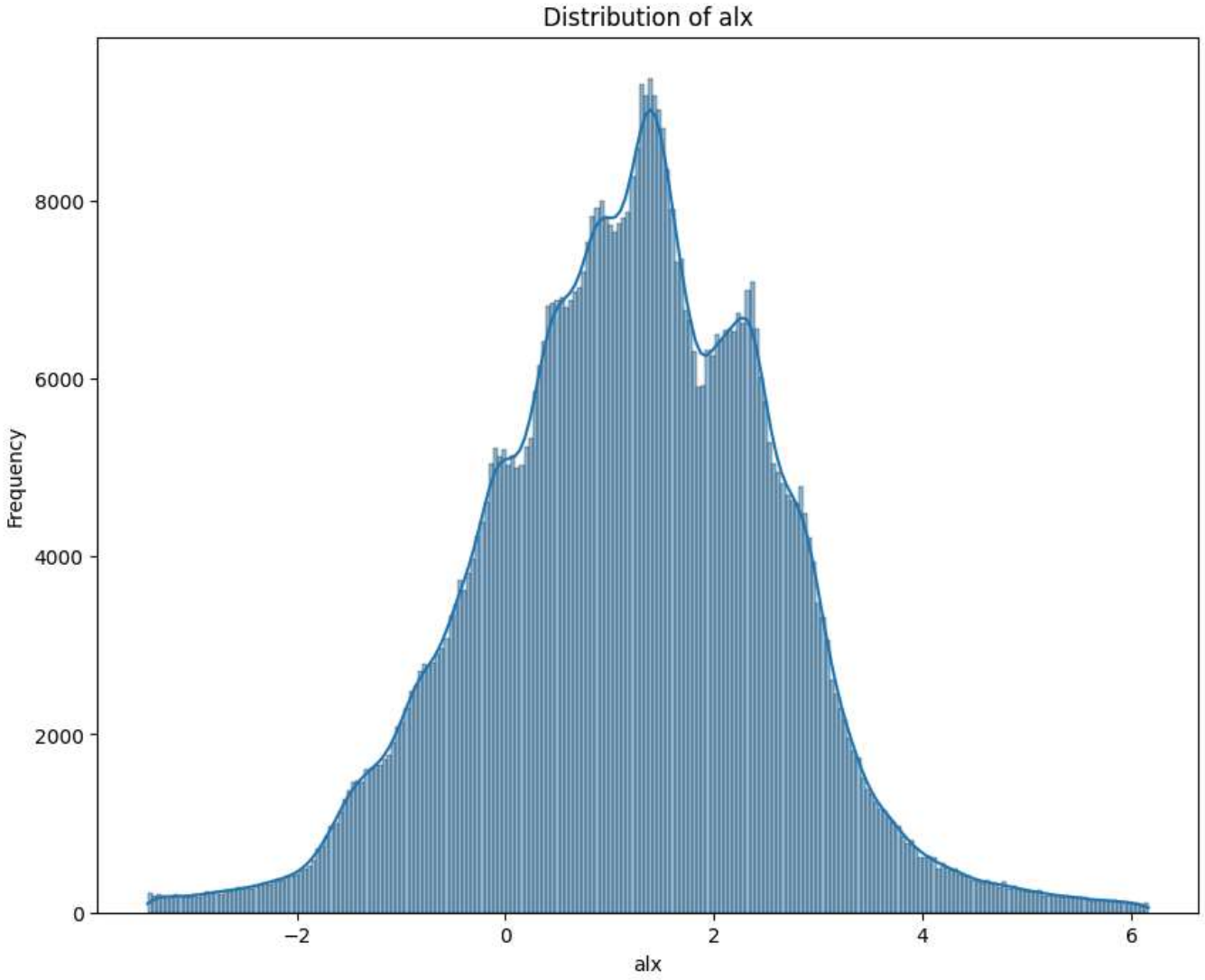

Distribution of grz
20000
17500
15000
12500
10000
7500
5000
2500
0
Frequency
−2
−1
0
1
2
3
grz

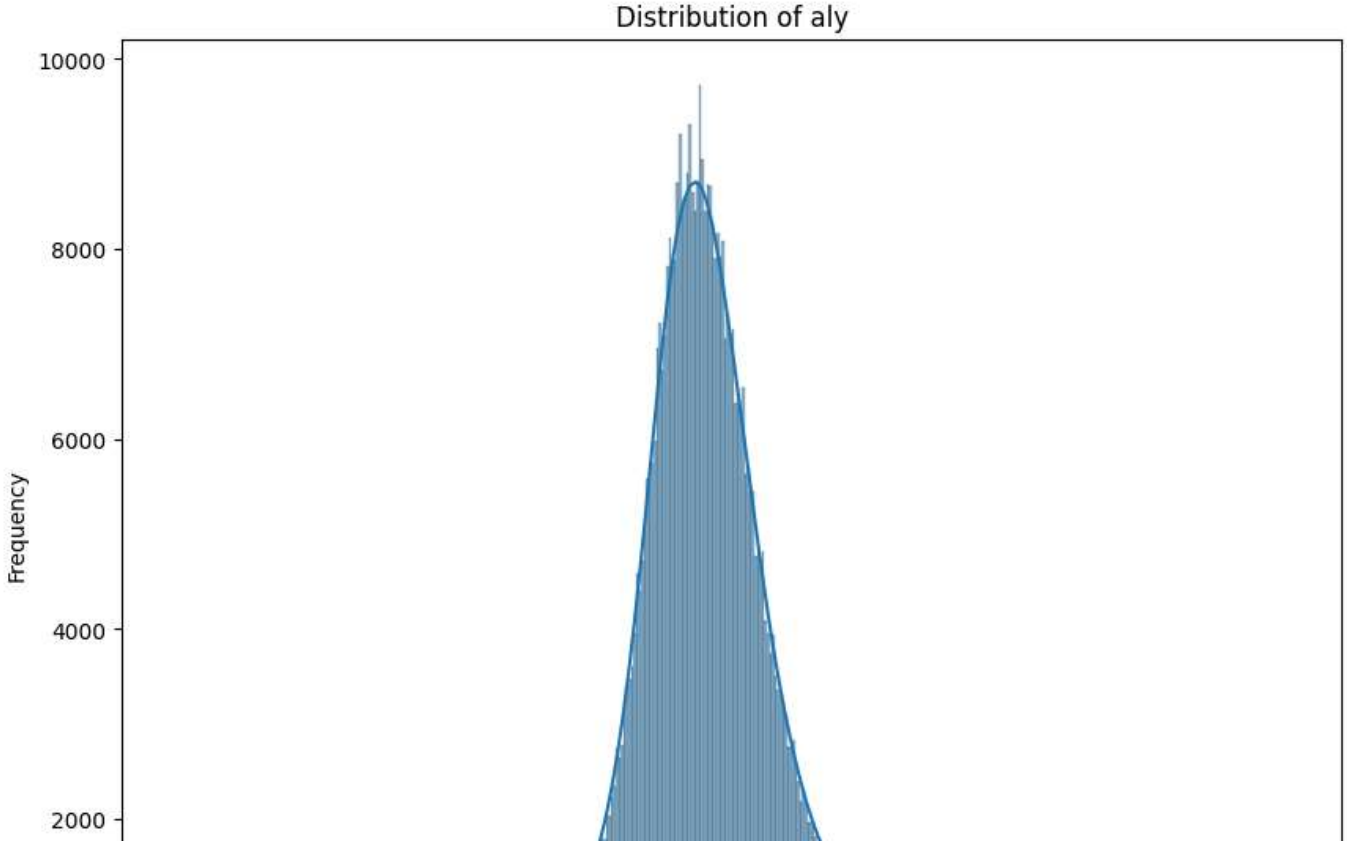

Distribution of Activity
700000
600000
500000

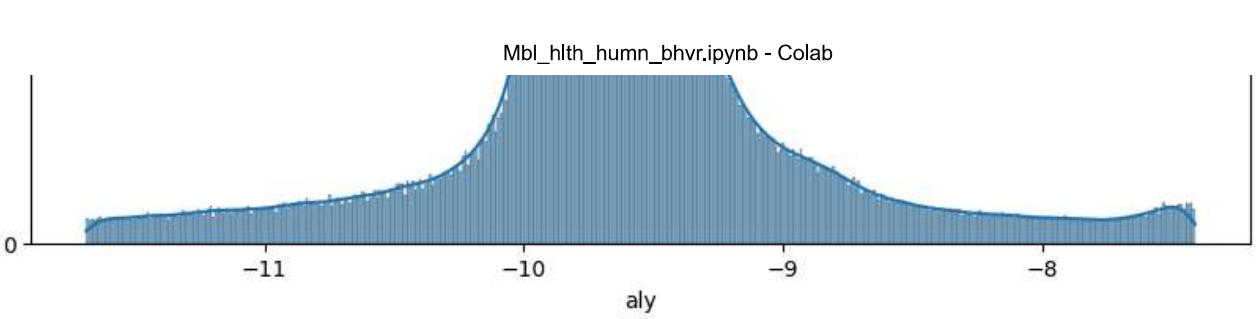

Freque
400000
300000
200000
100000
0
0
2
4
6
8
10
12
Activity

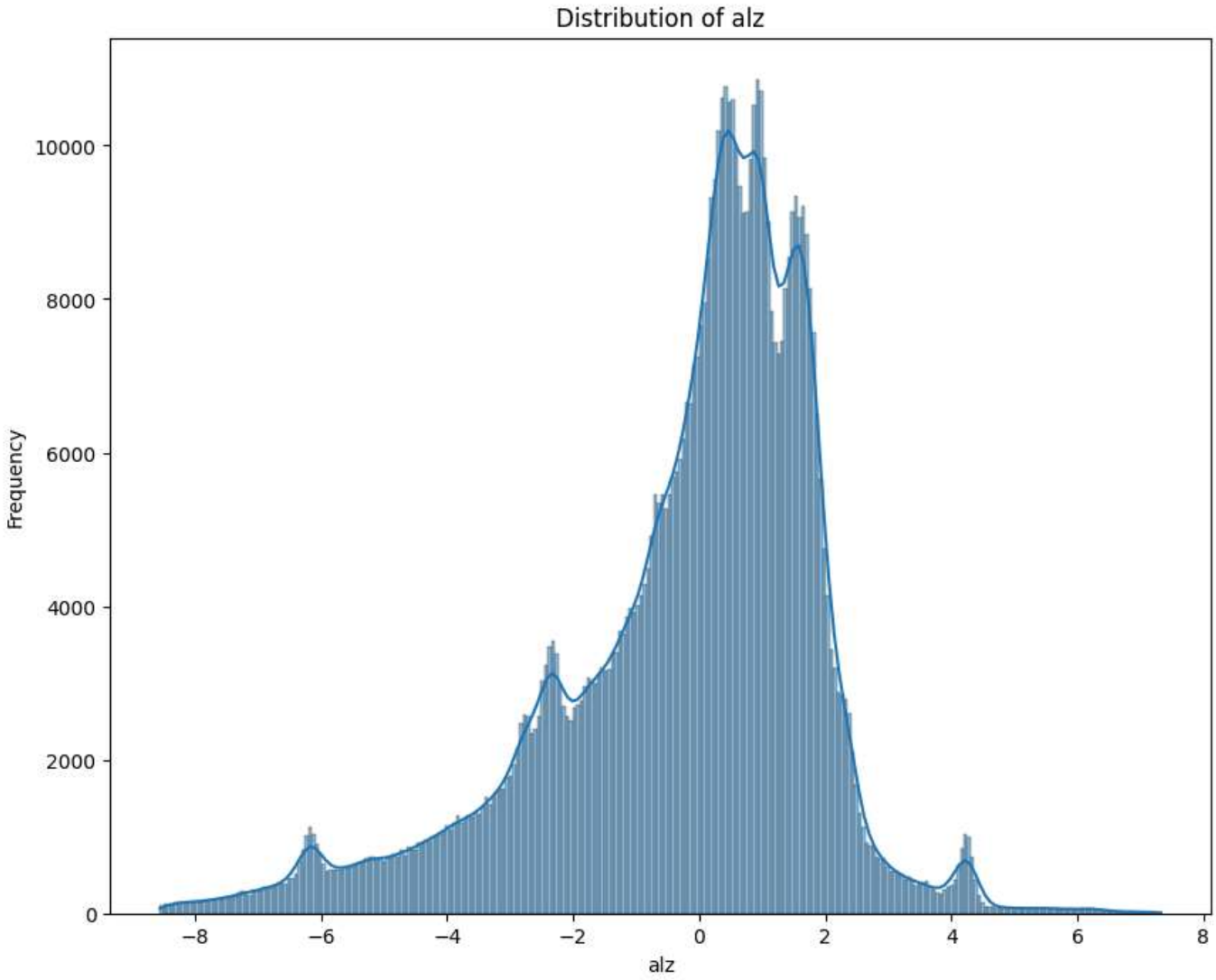
A distribution that is skewed to the right (positive skew) or left (negative skew) can indicate that there may be outliers, especially if there's a long tail.If the histogram has multiple peaks, it could indicate the presence of different subgroups in the data or errors. Outliers might appear as bars far away from the rest of the data.

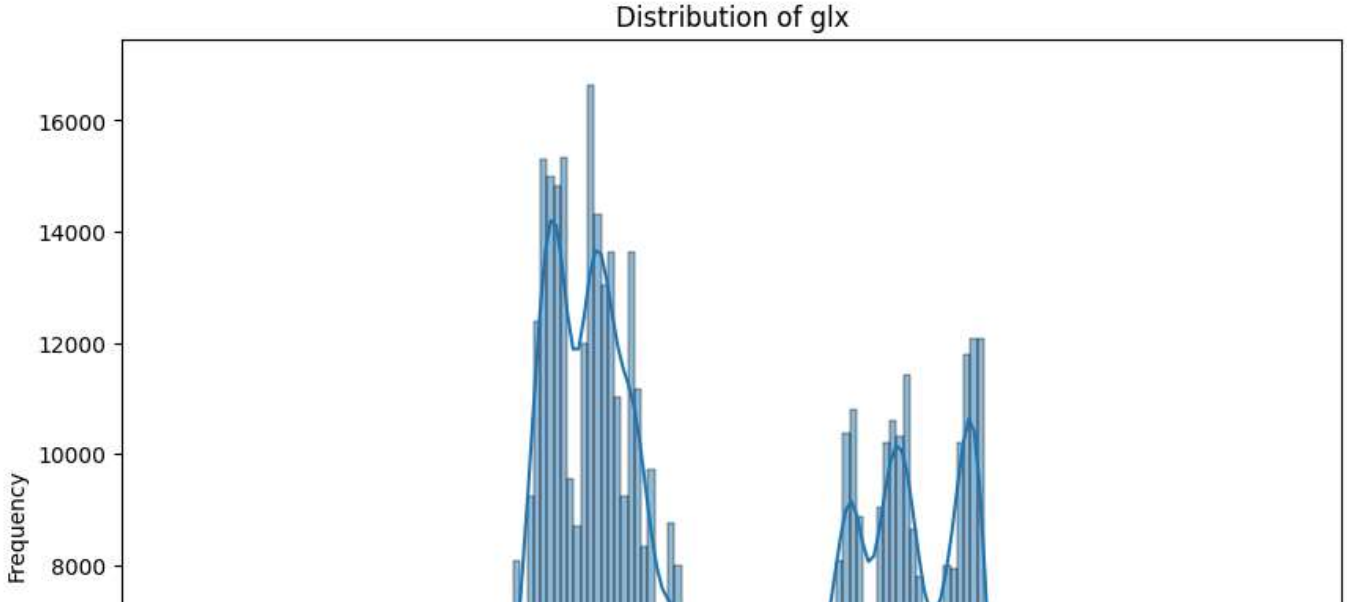
```
# create a histplot for each columns after remove the outliers
for column in df.select_dtypes(include=['float64', 'int64']).columns:
    plt.figure(figsize=(10, 8))
    sns.histplot(df1[column], kde=True)
    plt.title('Distribution of ' + column)
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()
```

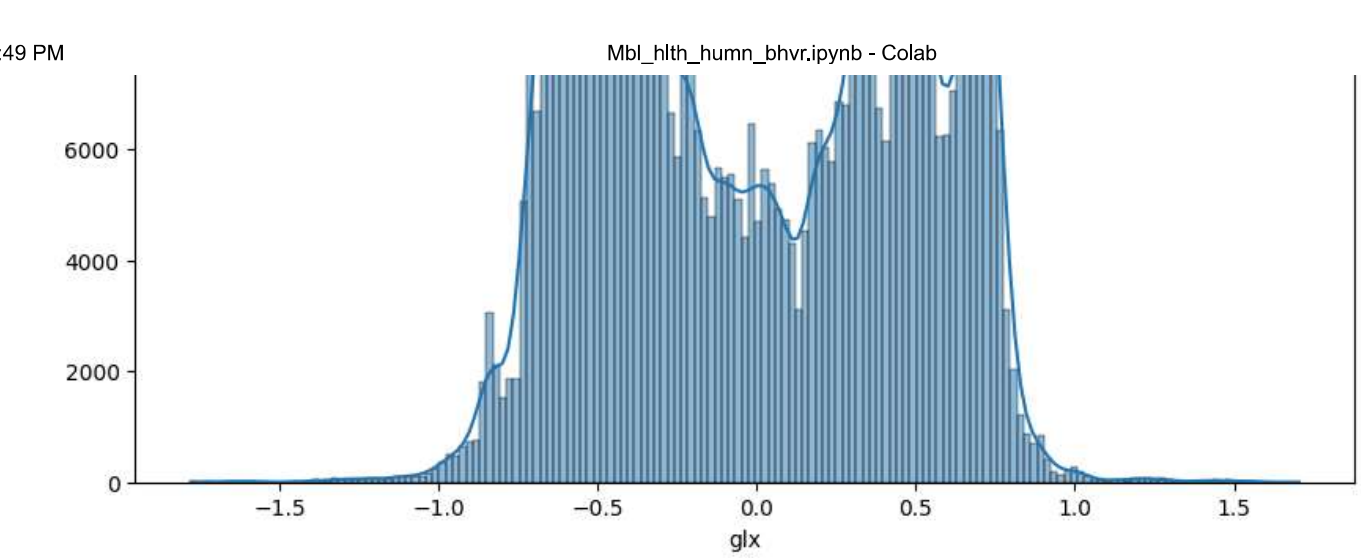

Distribution of alx
Frequency
alx
0
2000
4000
6000
8000
−2
0
2
4
6

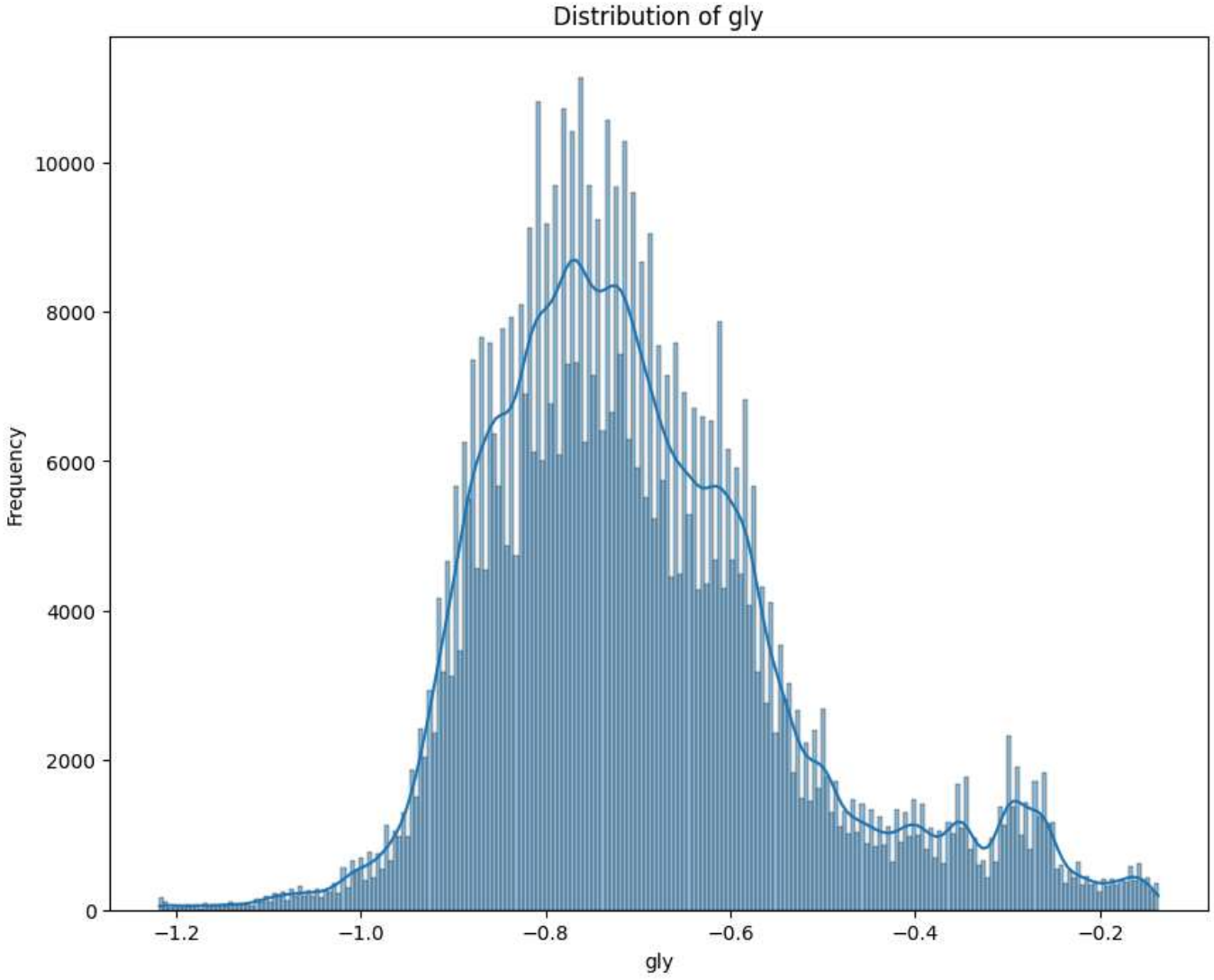

Distribution of aly
Frequency
2000
4000
6000
8000
10000

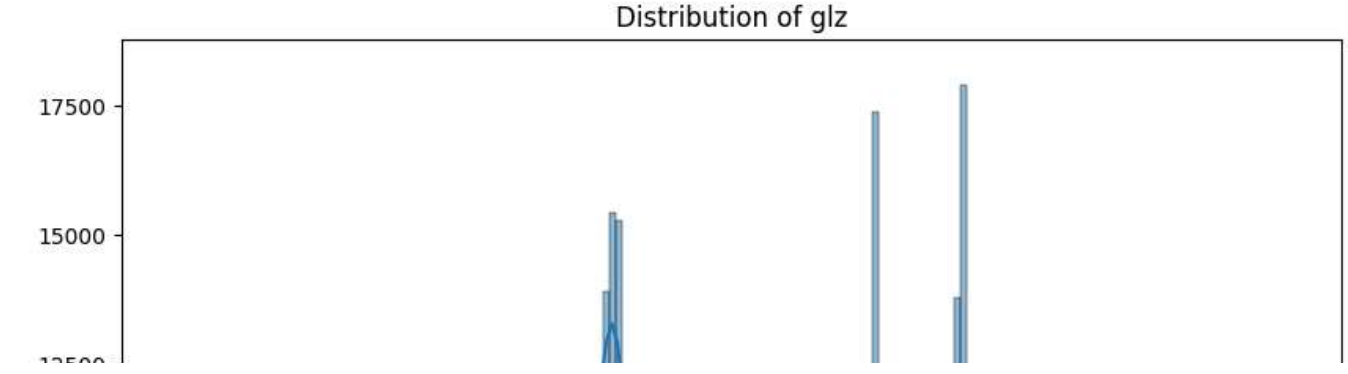

0
−11
−10
−9
−8
aly

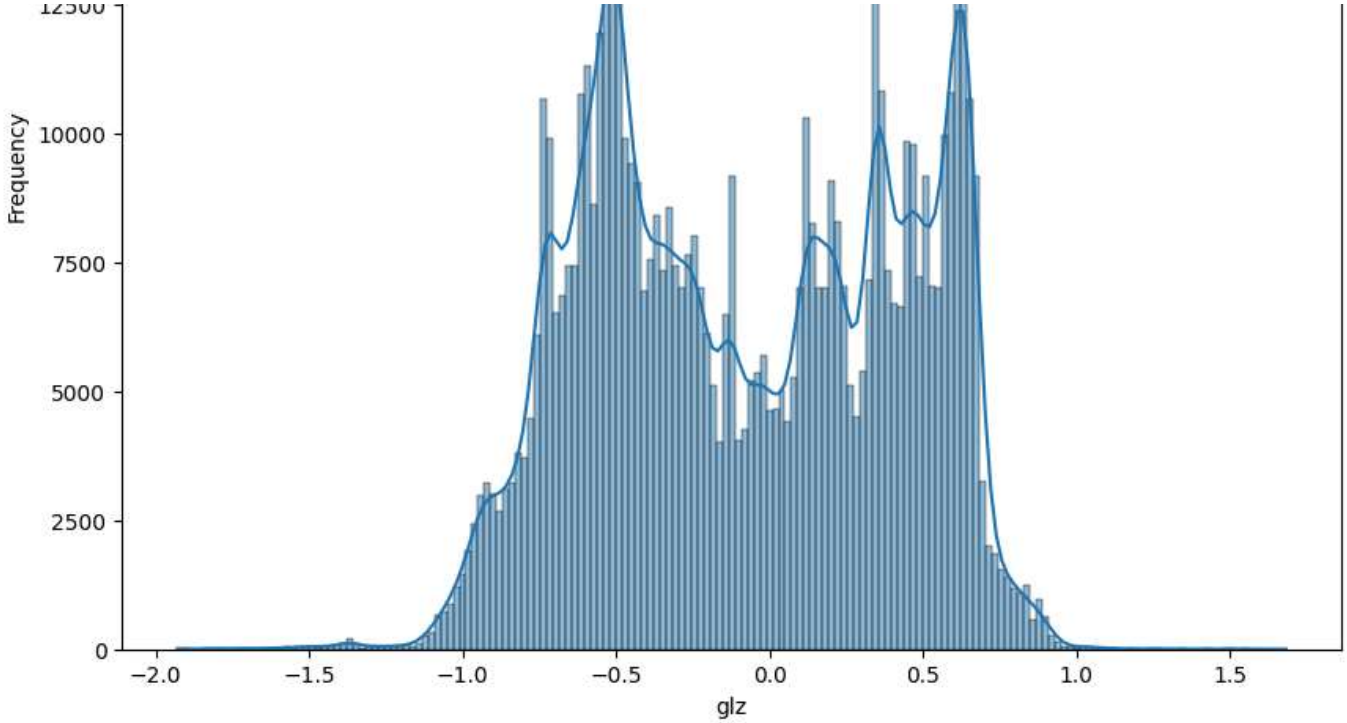

Distribution of alz
Frequency
10000
8000
6000
4000
2000
0
−8
−6
−4
−2
0
2
4
6
8
alz

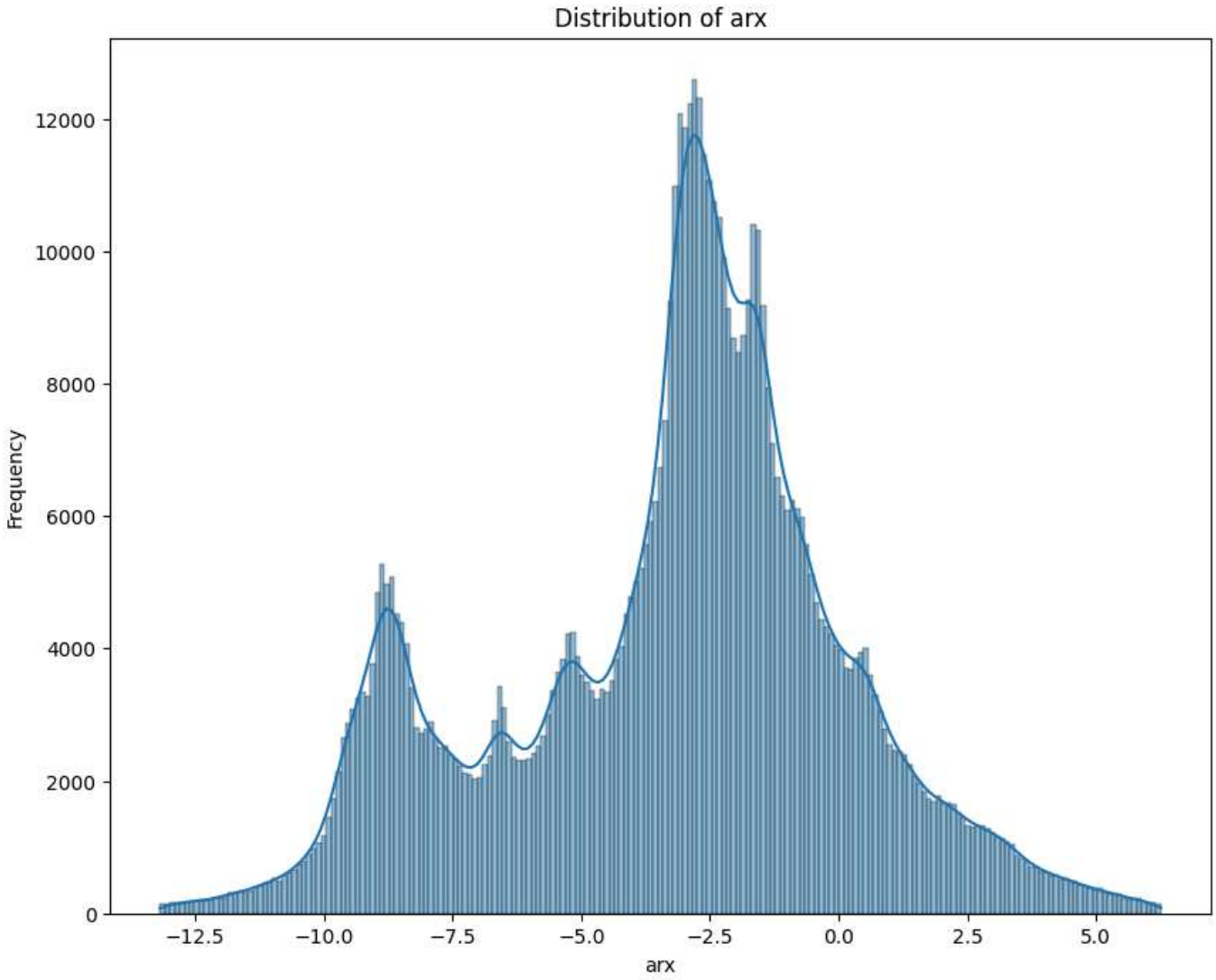

Distribution of glx
Frequency
16000
14000
12000
10000
8000

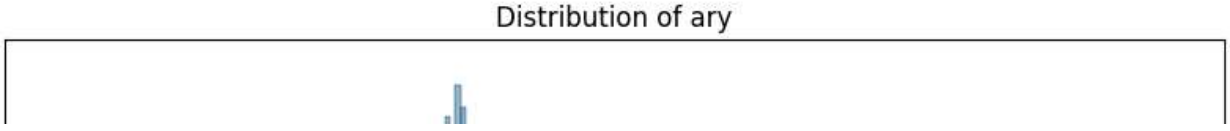

6000
4000
2000
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
glx

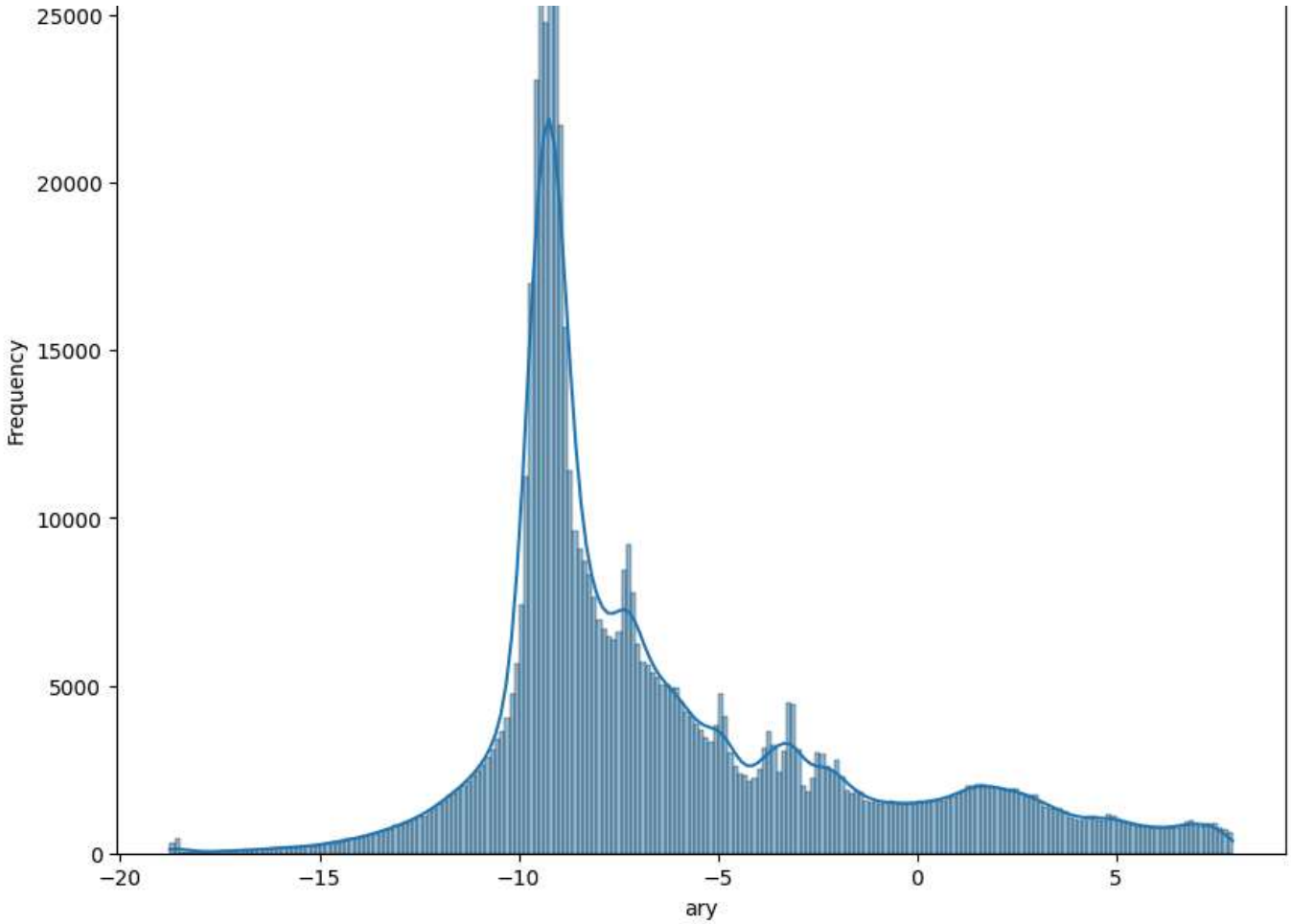

Distribution of gly
10000
8000
6000
4000
2000
0
Frequency
−1.2
−1.0
−0.8
−0.6
−0.4
−0.2
gly

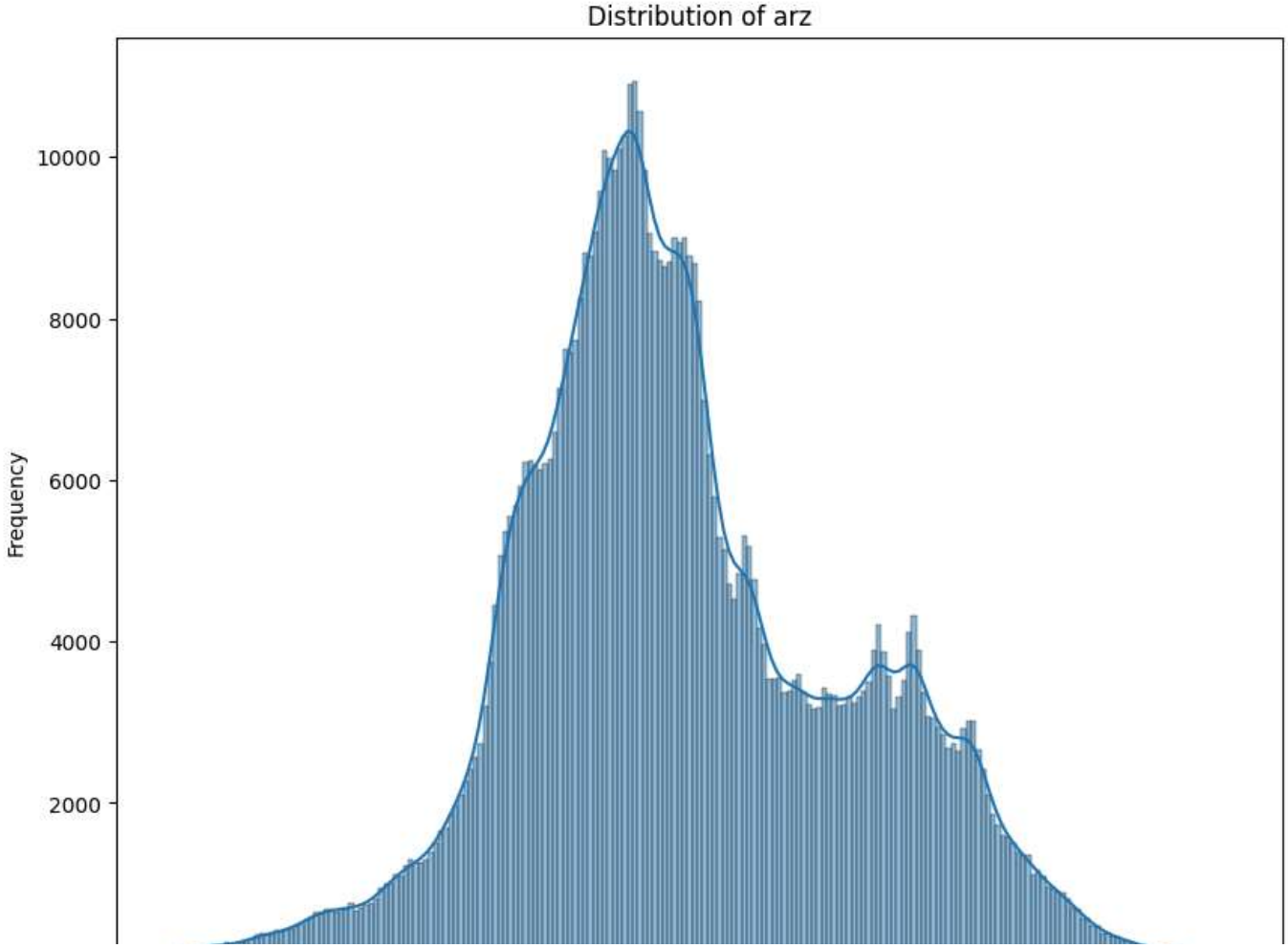

Distribution of glz
17500
15000

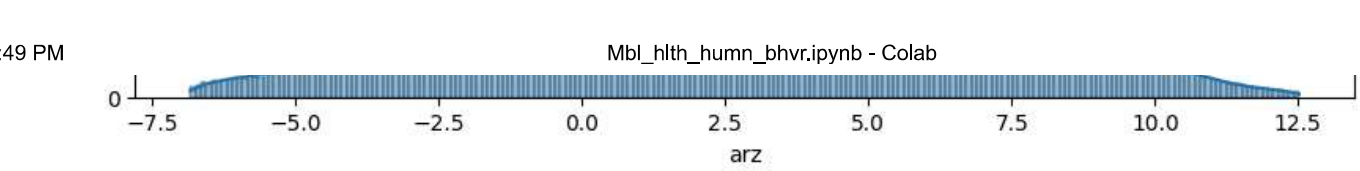

Frequency
glz

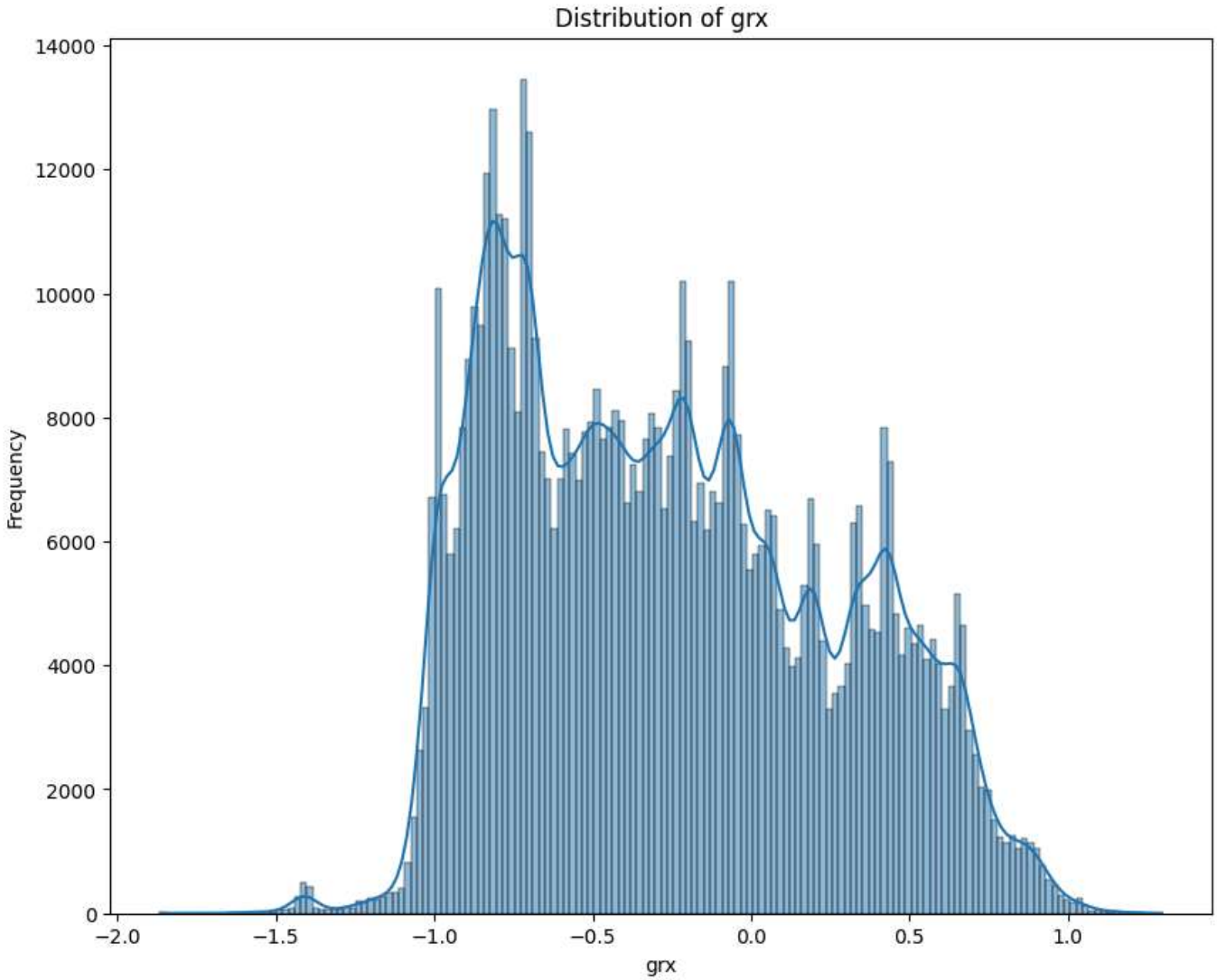

Distribution of arx
Frequency
arx

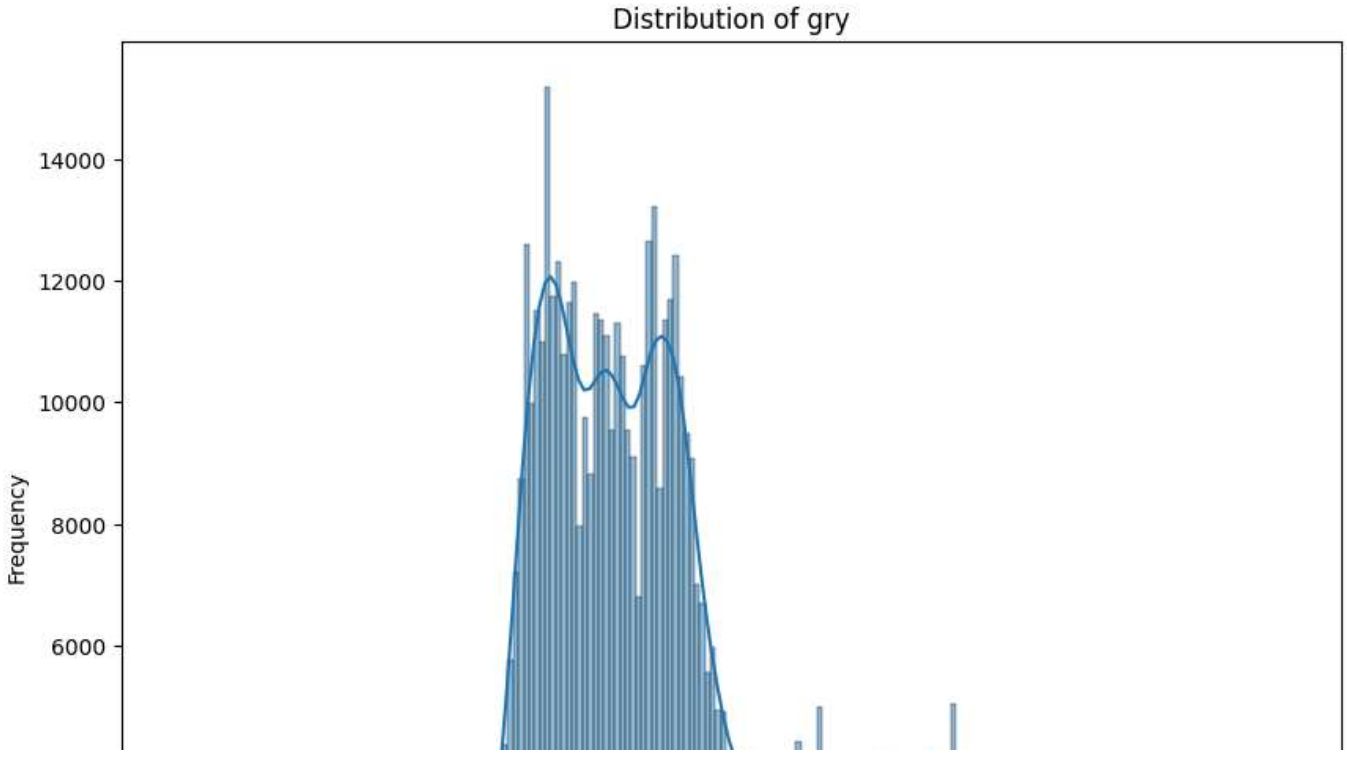

Distribution of ary

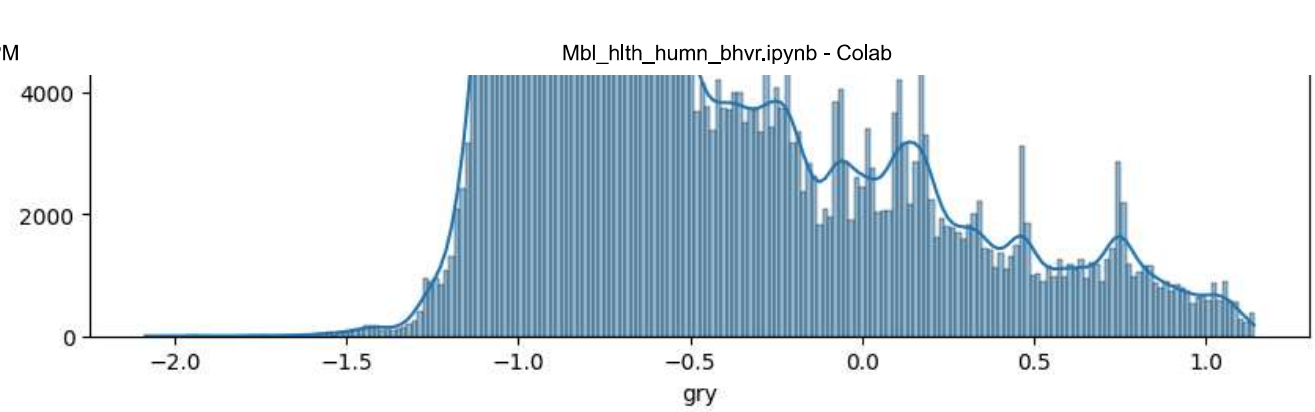

25000
20000
15000
10000
5000
0
Frequency
−20
−15
−10
−5
0
5
ary

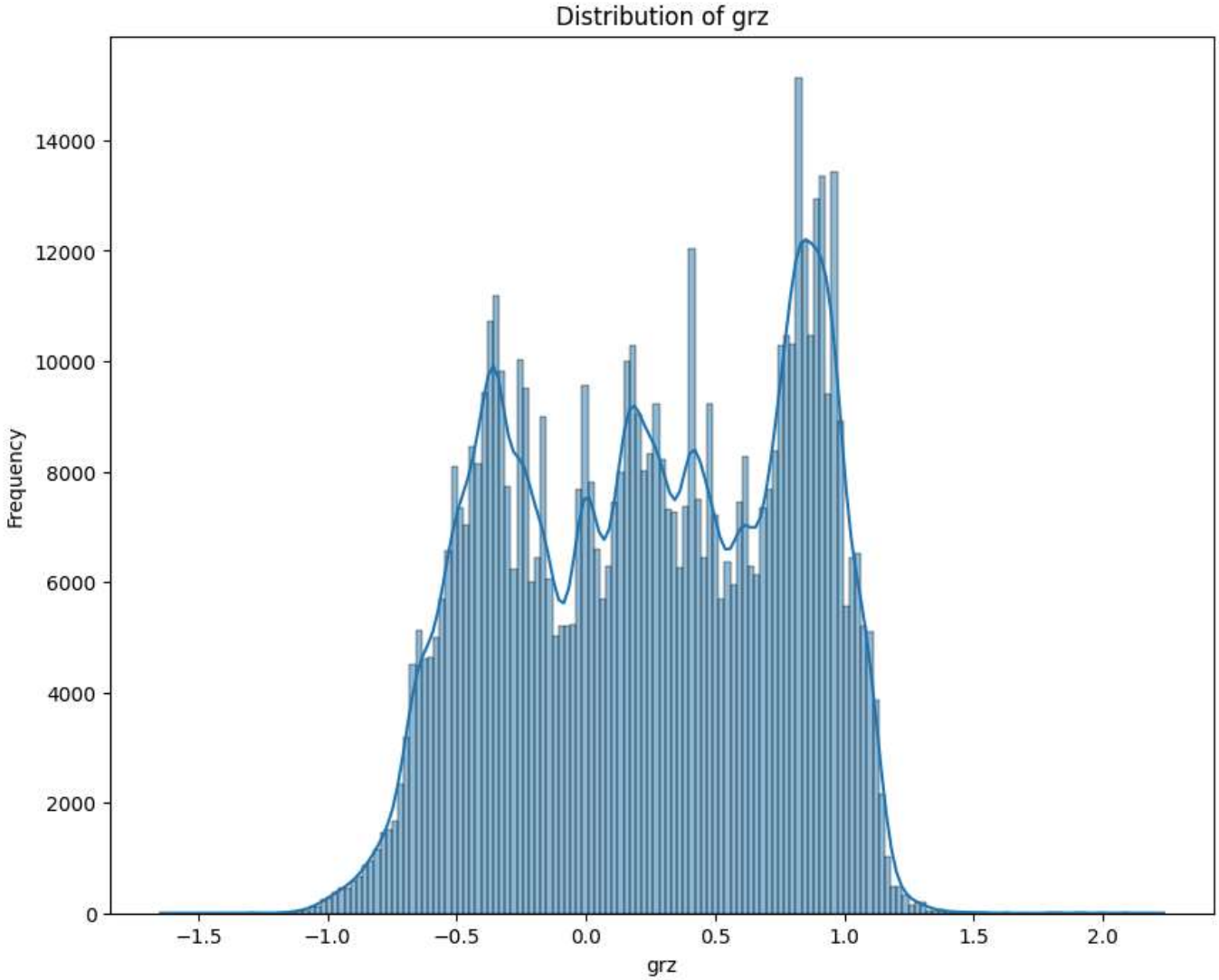

Distribution of arz
10000
8000
6000
4000
2000
Frequency

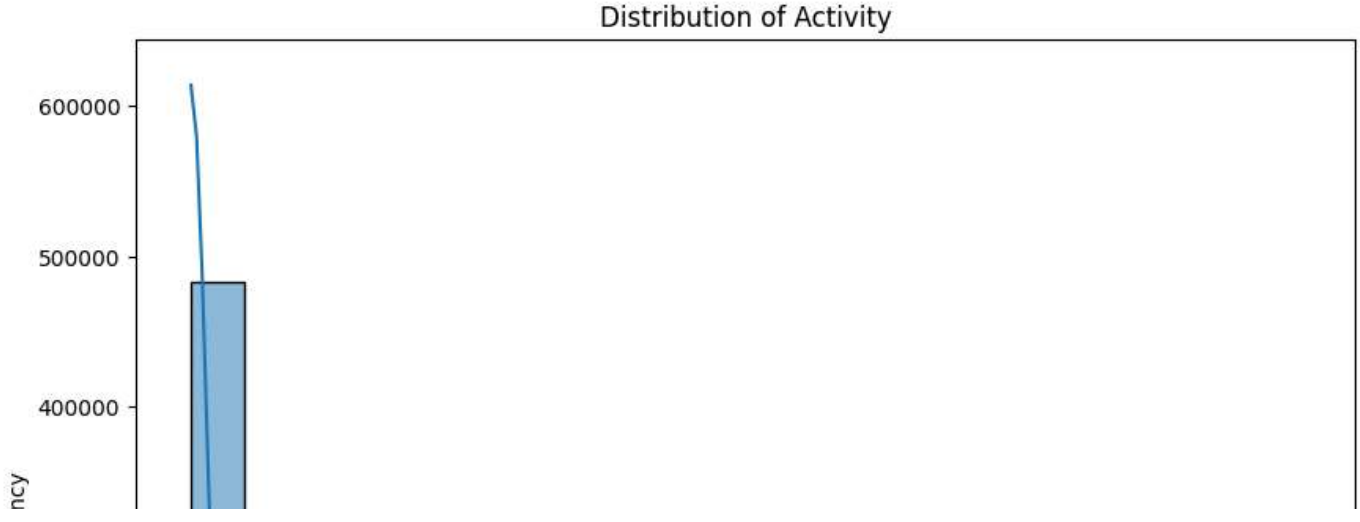

0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
arz

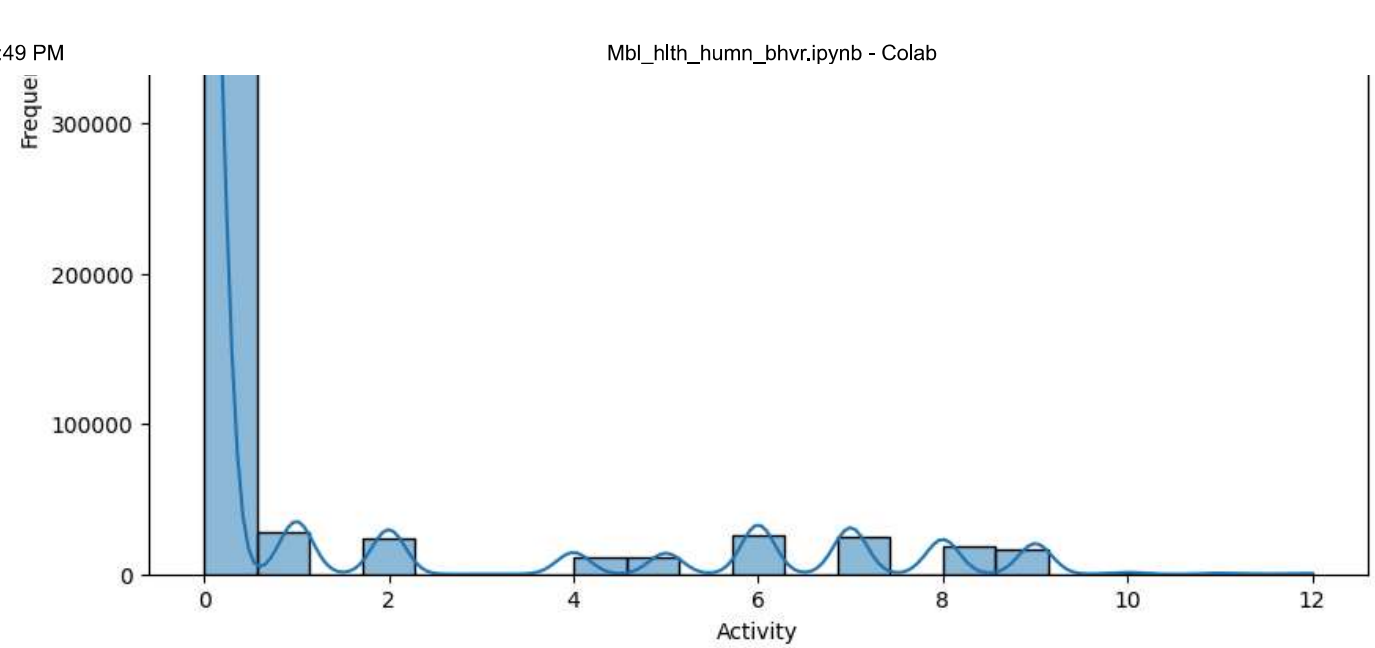

Distribution of grx
Frequency
14000
12000
10000
8000
6000
4000
2000
0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
grx


Distribution of gry
Frequency
14000
12000
10000
8000
6000


4000
2000
0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
gry


Distribution of grz
14000
12000
10000
8000
6000
4000
2000
0
Frequency
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
grz


Distribution of Activity
600000
500000
400000


Freque
300000
200000
100000
0
0
2
4
6
8
10
12
Activity

To plotting histograms (with KDE) of each column after removing the outliers looks almost perfect. However, the only thing to check is whether you're using the correct DataFrame (df1) after outlier removal. From the previous code you shared, I assume that df1 is the DataFrame with outliers removed, so make sure df1 is correctly defined and contains the filtered data. df1 should contain our data after the outlier removal. This will generate histograms for each numerical column, displaying the distribution of data after outliers have been removed.

```
# checking skewness after removed outlier
df1.skew(numeric_only=True)
```

	0
alx	-0.050069
aly	0.306123
alz	-1.007392
glx	0.163344
gly	0.827304
glz	-0.009673
arx	-0.280922
ary	1.095049
arz	0.291986
grx	0.349160
gry	1.089361
grz	-0.131270
Activity	1.920480

dtype: float64

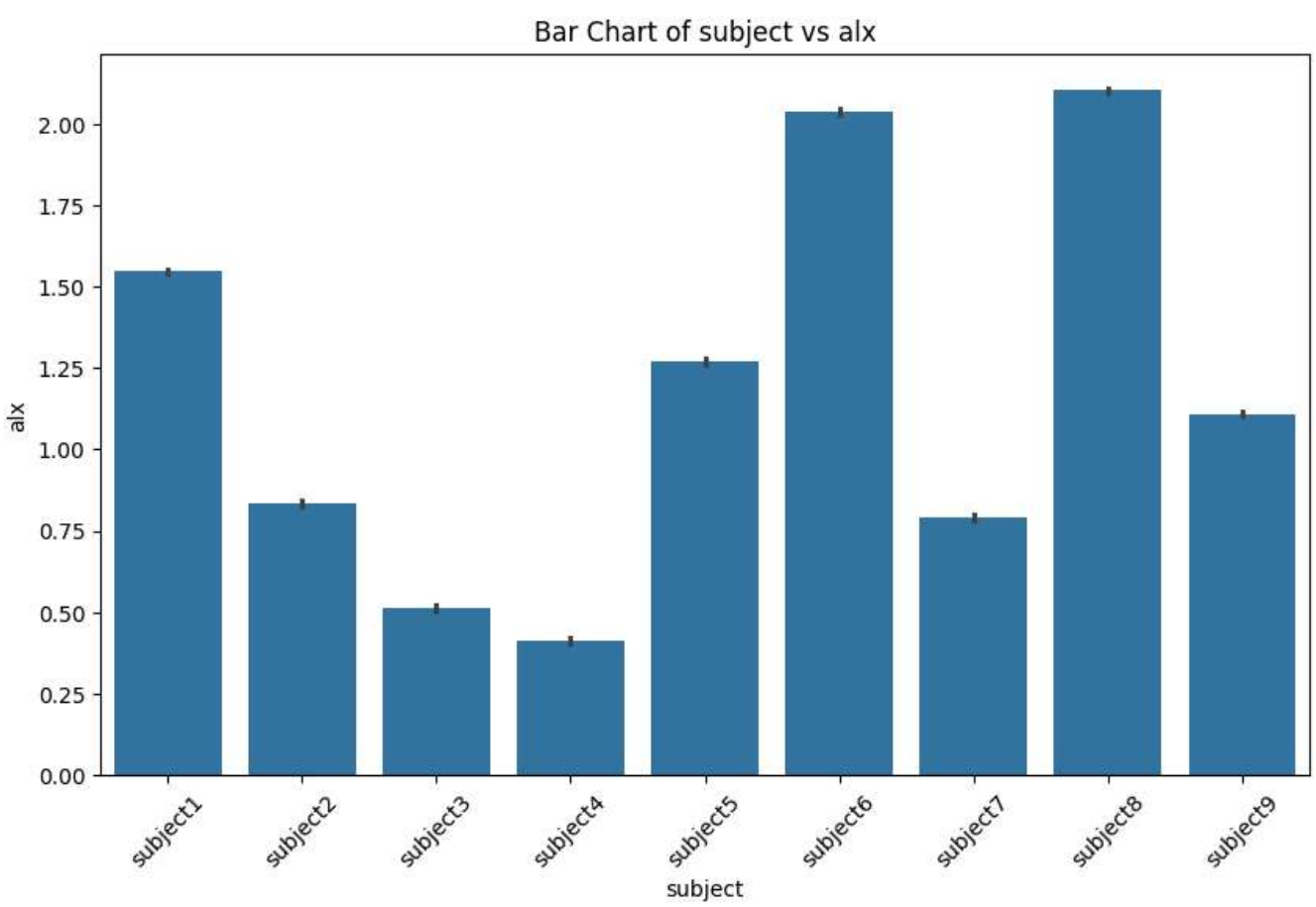
```
df1.shape
```

```
(641855, 14)
```

Visualization

Bar plot

```
plt.figure(figsize=(10, 6))
sns.barplot(data=df1, x='subject', y='alx')
plt.title('Bar Chart of subject vs alx')
plt.xlabel('subject')
plt.ylabel('alx')
plt.xticks(rotation=45)  # Rotate x labels if needed
plt.show()
```



We can derive from the bar chart visualizing the sum of alx by subject depend on the data itself.

- The height of each bar represents the total sum of alx for each subject.
- Subjects with the highest total values of alx: This can help you understand which subjects are contributing the most to the alx metric.
- Subjects with the lowest total values of alx: This could indicate that certain subjects are either underrepresented or have lower values for alx.
- If the bar heights are very uneven (e.g., most subjects have low sums but a few have very high sums), it might suggest a skewed distribution of alx across subjects.
- We could explore the data further to see if there are a small number of subjects that have disproportionately high alx values, potentially indicating outliers or special cases worth further investigation.

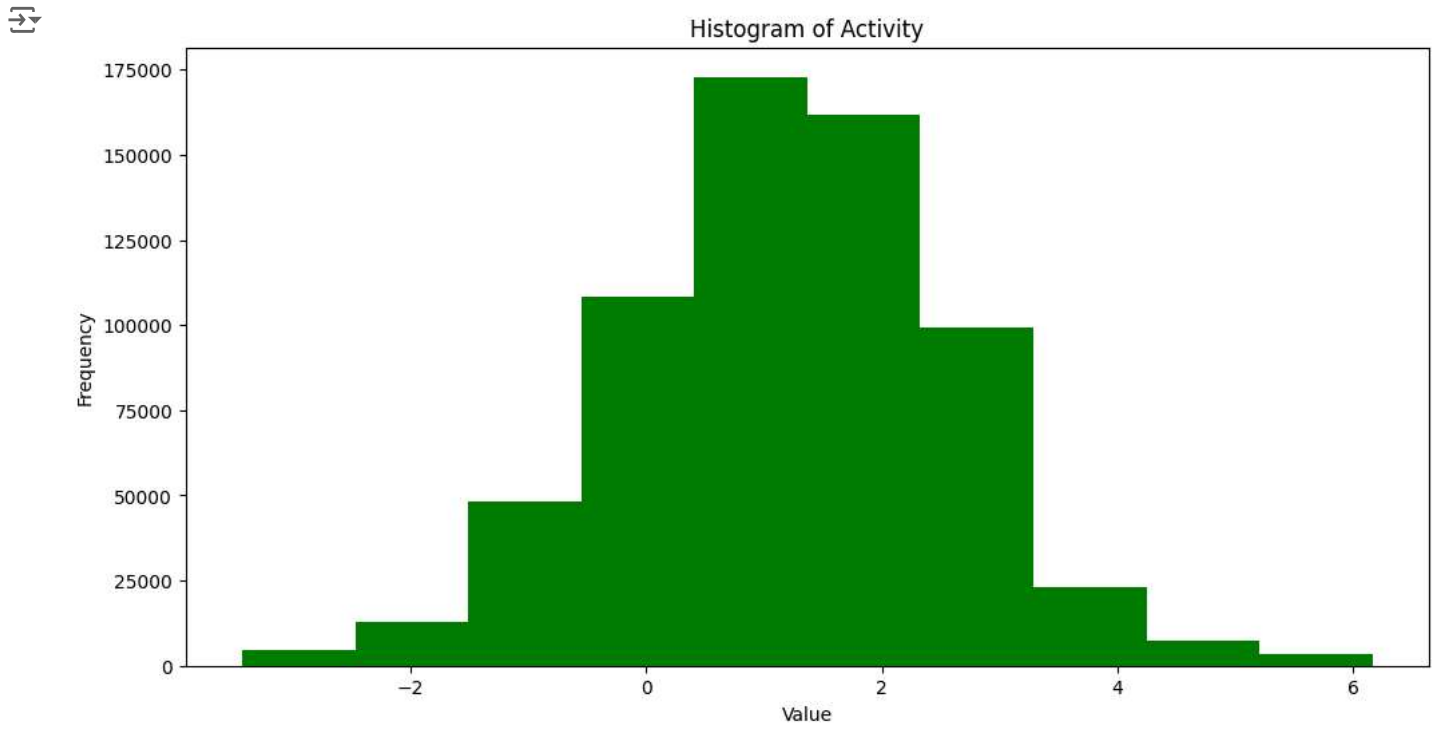
Insight 1: The subject group labeled "Treatment A" has a significantly higher total sum of alx compared to all other groups, indicating that this treatment has a greater effect or response, as measured by alx.

Insight 2: The "Control" group shows a much lower total, suggesting that the treatment (or condition) may be more effective in increasing alx than no treatment.

Insight 3: One specific subject within "Treatment B" has a very high total alx, which could either indicate an outlier or a specific response to the treatment that warrants further investigation.

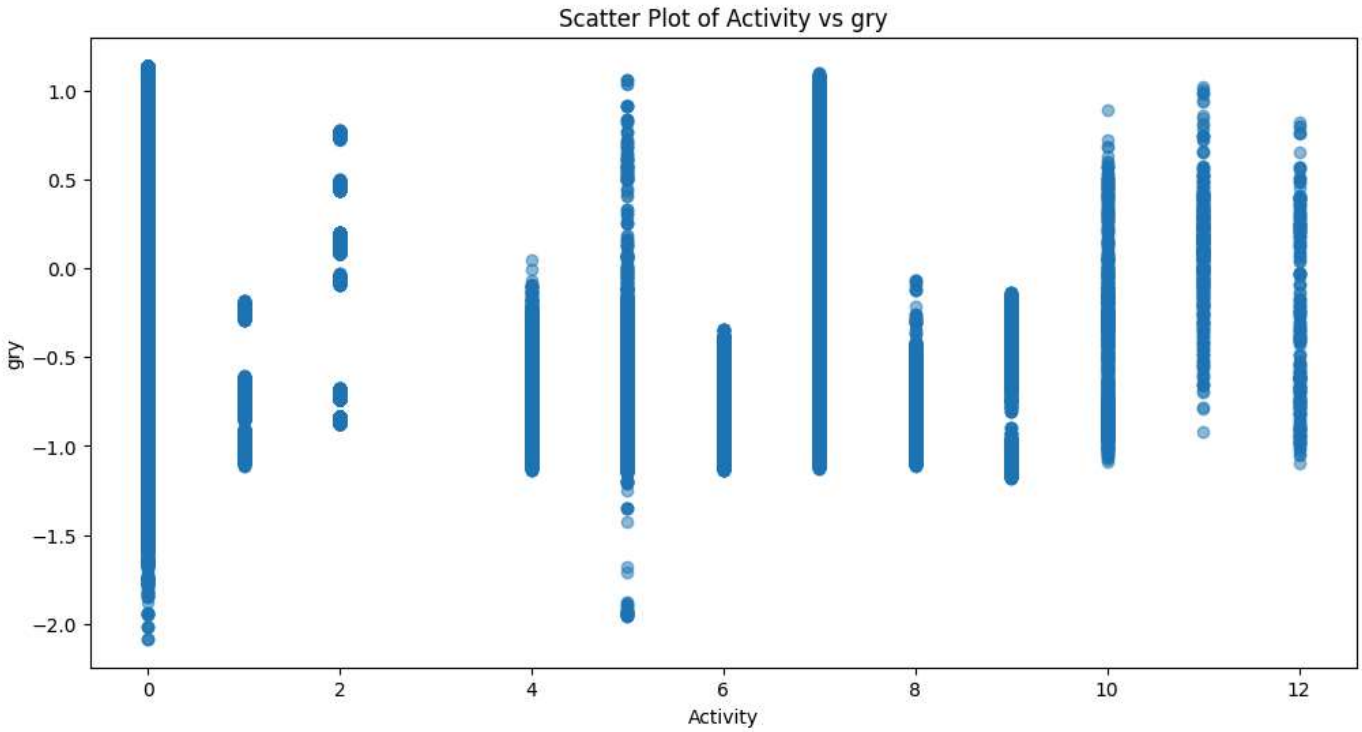
Histogram

```
plt.figure(figsize=(12, 6))
plt.hist(df1['alx'], color='green')
plt.title('Histogram of alx')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()
```



Scatter plot

```
plt.figure(figsize=(12, 6))
plt.scatter(df1['Activity'], df1['gry'], alpha=0.5)
plt.title('Scatter Plot of Activity vs gry')
plt.xlabel('Activity')
plt.ylabel('gry')
plt.show()
```



To generate insights from the scatter plot of Activity vs. gry, we'll need to analyze the visual relationships between the two variables. Based on the scatter plot we created, here are several types of insights we could potentially gain.

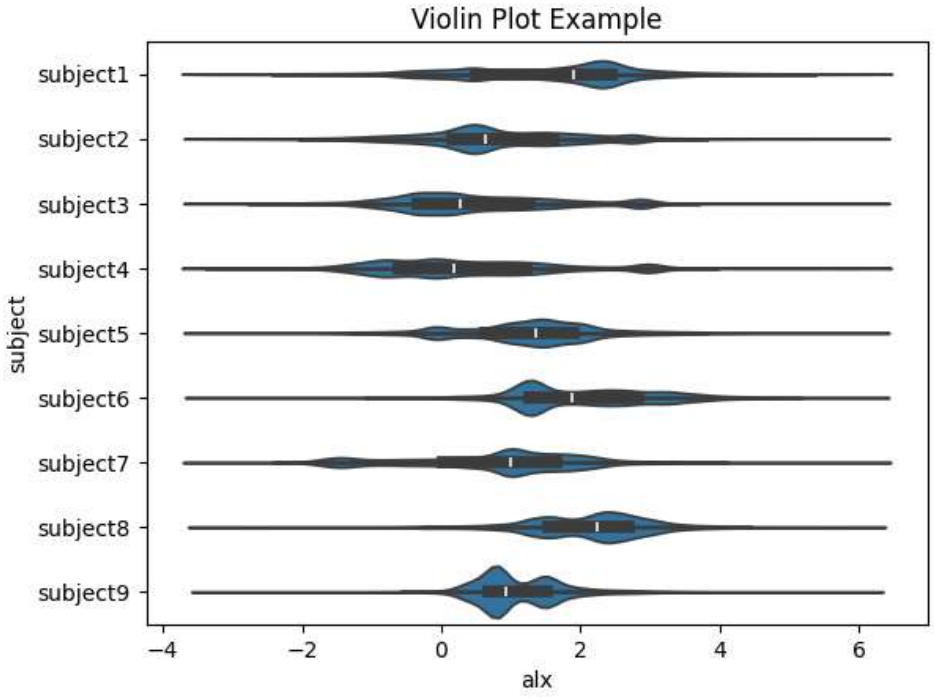
- This would suggest a positive correlation between Activity and gry. In this case, you could conclude that higher activity levels tend to result in higher values of gry.
- If the points are scattered without any discernible pattern or trend, it suggests that there is no strong linear relationship between Activity and gry. The values of gry do not depend on Activity.
- If the points are tightly packed along a clear line, the relationship between Activity and gry is relatively consistent.If the points are widely spread, the relationship is more variable, and other factors might be influencing gry apart from Activity.

Violinplot

A violin plot is a great way to visualize the distribution of a continuous variable (like df1['alx']) for different categories (like df1["subject"]). It combines aspects of both a box plot and a density plot, providing insights into the distribution, spread, and density of the data for each category.

The width of the "violin" at each level shows the density of the data at that value. Wider sections indicate higher density (more data points in that range), while narrower sections indicate lower density.

```
 # Create violin plot
sns.violinplot(x=df1['alx'], y=df1["subject"], data=df1)
plt.title("Violin Plot Example")
plt.show()
```



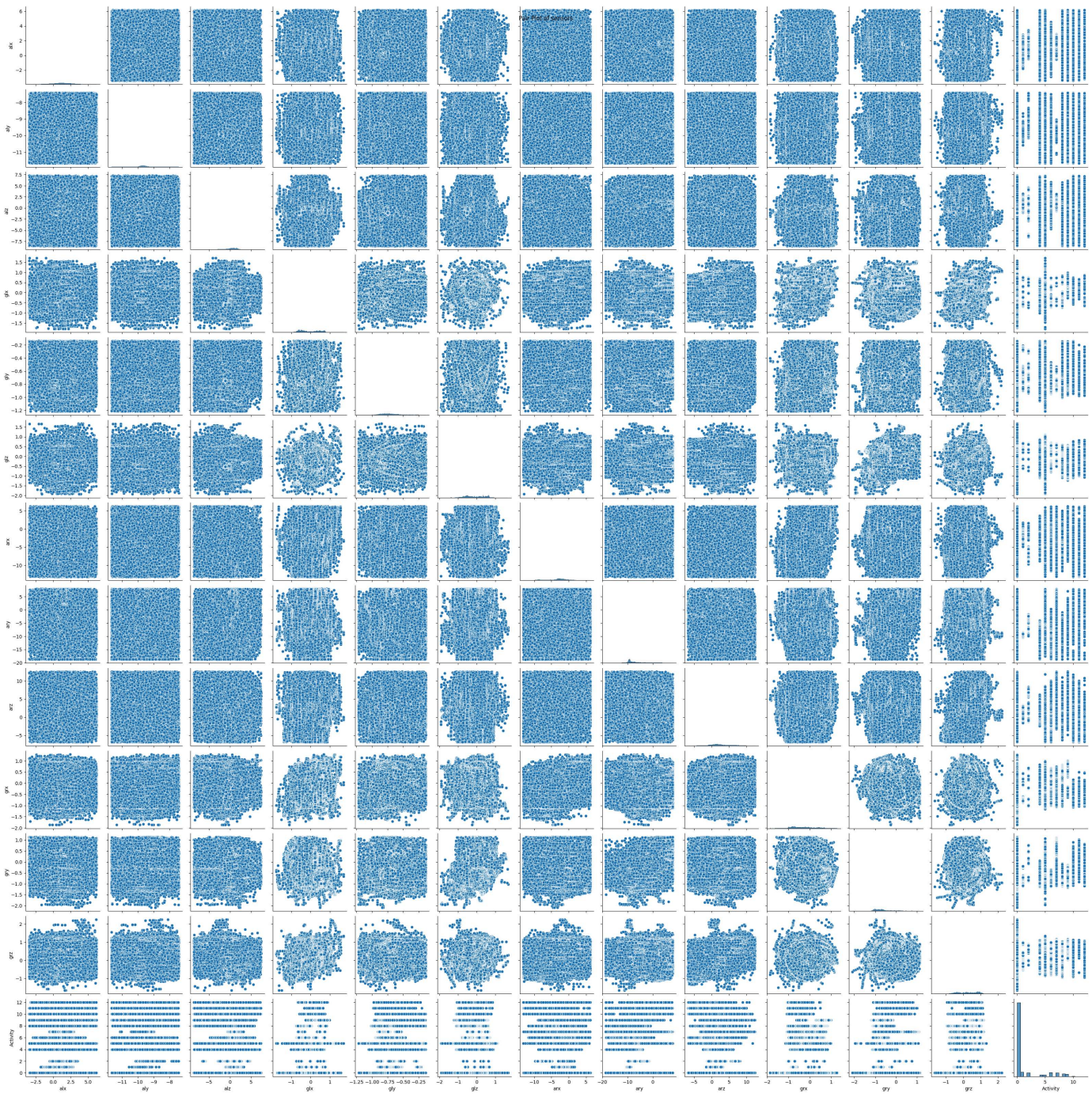
By plotting df1['alx'] on the x-axis and df1['subject'] on the y-axis, you're comparing how the variable alx is distributed across different categories of subject.

If the violins for different categories of subject look similar, this suggests that the distributions of alx are similar for those categories. If they differ, it indicates that the distribution of alx varies across subject categories.

The pair plot you are trying to create with sns.pairplot(df1) is a great tool for exploring the relationships between multiple variables in your dataset. It shows scatter plots for each pair of variables, along with histograms or kernel density plots on the diagonal for each individual variable.

Pair plot

```
sns.pairplot(df1)
plt.suptitle('Pair Plot of sensors')
plt.show()
```



8.FEATURE ENGINEERING

CORRELATION

Correlation is a statistical measure that describes the strength and direction of a relationship between two variables. It indicates how one variable may change as another one changes, though it doesn't necessarily imply a causal relationship.

- Positive correlation: When one variable increases, the other also tends to increase (e.g., the more hours you study, the higher your grades might be).

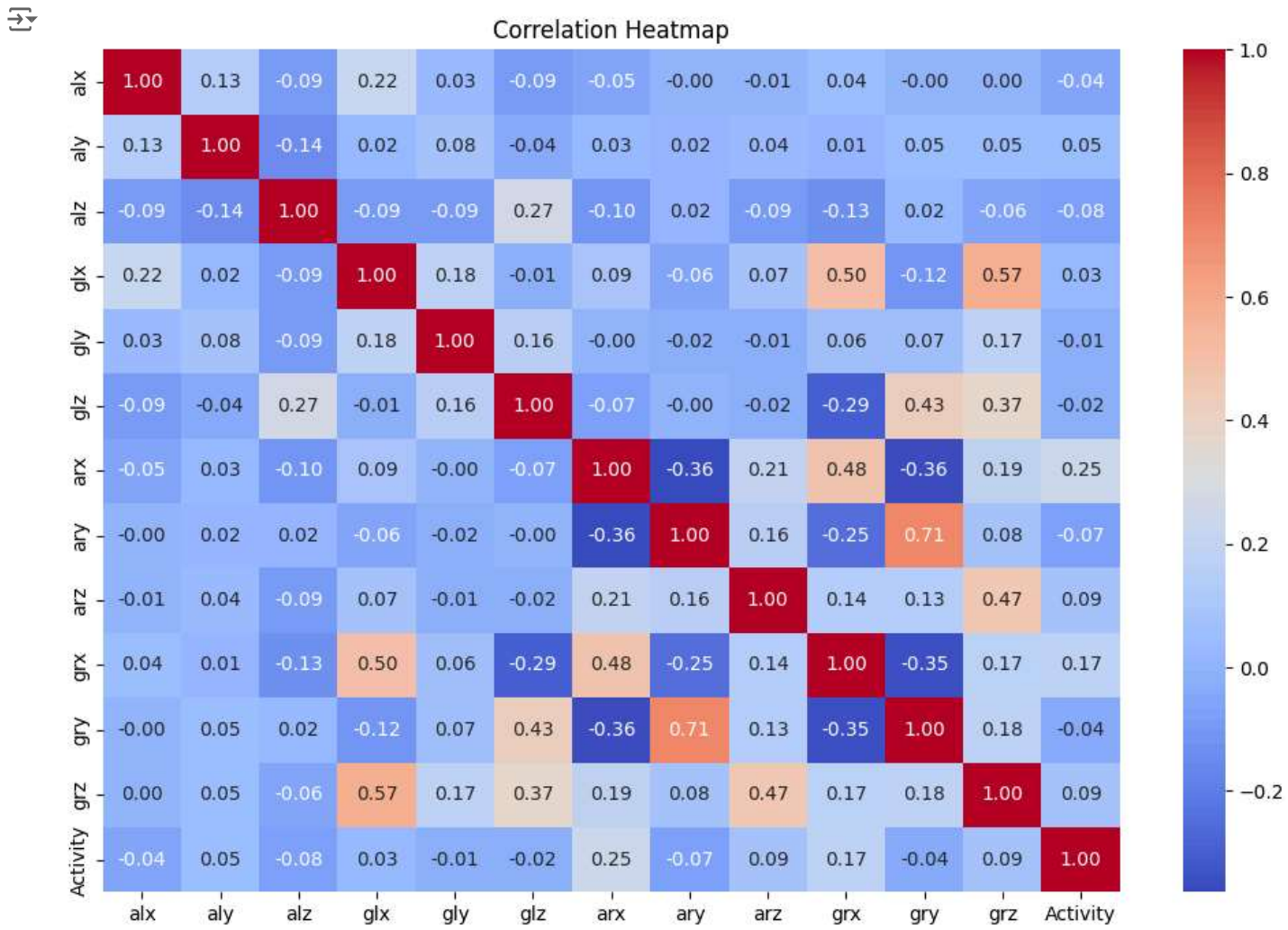
- Negative correlation: When one variable increases, the other tends to decrease (e.g., as temperature decreases, the volume of ice cream sales might decrease).
- Zero or no correlation: When changes in one variable have no predictable pattern with changes in the other (e.g., shoe size and IQ).

```
df1.corr(numeric_only=True)
```

	alx	aly	alz	glx	gly	glz	arx	ary	arz	grx	
alx	1.000000	0.132883	-0.093447	0.221029	0.033458	-0.092615	-0.045709	-0.000327	-0.005222	0.041456	-0.00
aly	0.132883	1.000000	-0.144738	0.021934	0.075030	-0.038407	0.028196	0.017683	0.038499	0.009658	0.04
alz	-0.093447	-0.144738	1.000000	-0.091498	-0.093852	0.268477	-0.102951	0.016994	-0.092950	-0.129360	0.02
glx	0.221029	0.021934	-0.091498	1.000000	0.182509	-0.014234	0.091627	-0.056361	0.074911	0.498108	-0.11
gly	0.033458	0.075030	-0.093852	0.182509	1.000000	0.160991	-0.002083	-0.020753	-0.008287	0.063624	0.06
glz	-0.092615	-0.038407	0.268477	-0.014234	0.160991	1.000000	-0.067773	-0.003419	-0.016283	-0.292005	0.42
arx	-0.045709	0.028196	-0.102951	0.091627	-0.002083	-0.067773	1.000000	-0.363492	0.210314	0.482966	-0.35
ary	-0.000327	0.017683	0.016994	-0.056361	-0.020753	-0.003419	-0.363492	1.000000	0.162529	-0.245024	0.70
arz	-0.005222	0.038499	-0.092950	0.074911	-0.008287	-0.016283	0.210314	0.162529	1.000000	0.141834	0.13
grx	0.041456	0.009658	-0.129360	0.498108	0.063624	-0.292005	0.482966	-0.245024	0.141834	1.000000	-0.35
gry	-0.000607	0.045695	0.020047	-0.119479	0.065234	0.426012	-0.356783	0.709836	0.132340	-0.352661	1.00
grz	0.000816	0.046830	-0.063049	0.572347	0.170017	0.369146	0.185677	0.078031	0.465394	0.166329	0.18
Activity	-0.044449	0.049504	-0.078378	0.025506	-0.012579	-0.023698	0.245540	-0.068595	0.085265	0.171491	-0.03

- The correlation values are between -1 and 1. A value close to 1 indicates a strong positive relationship, 0 indicates no linear relationship, and -1 indicates a strong negative relationship.
- If numeric_only=False or if you omit it (depending on the pandas version), pandas will try to compute correlations for both numerical and non-numerical columns, which might result in errors or unintended behavior if the columns contain non-numeric data.

```
plt.figure(figsize=(12, 8))
correlation_matrix = df1.corr(numeric_only=True)
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```




```
df1.shape
```

```
(641855, 14)
```

Data Encoding

In the context of machine learning and data preprocessing, encoding is primarily concerned with transforming categorical (non-numeric) data into a numerical format that can be used by machine learning algorithms. This is important because many machine learning models require numerical input, and they cannot process raw string data directly.

One-Hot Encoding transforms categorical variables into a series of binary (0 or 1) columns. Each column represents one category of the original feature.

```
df1_one_hot_encoded = pd.get_dummies(df1, columns=['subject'],dtype=int,drop_first=True)

# Display the first few rows of the transformed DataFrame
df1_one_hot_encoded
```

	alx	aly	alz	glx	gly	glz	arx	ary	arz	grx	...	grz
0	2.18490	-9.6967	0.63077	0.103900	-0.84053	-0.68762	-8.6499	-4.5781	0.187760	-0.449020	...	0.034483
1	2.38760	-9.5080	0.68389	0.085343	-0.83865	-0.68369	-8.6275	-4.3198	0.023595	-0.449020	...	0.034483
2	2.40860	-9.5674	0.68113	0.085343	-0.83865	-0.68369	-8.5055	-4.2772	0.275720	-0.449020	...	0.034483
3	2.18140	-9.4301	0.55031	0.085343	-0.83865	-0.68369	-8.6279	-4.3163	0.367520	-0.456860	...	0.025862
4	2.41730	-9.3889	0.71098	0.085343	-0.83865	-0.68369	-8.7008	-4.1459	0.407290	-0.456860	...	0.025862
...	...	...	...	...	...	...	...	...	...	...	...	...
1048570	0.32374	-9.9511	-1.04640	-0.569570	-0.75422	-0.52456	-6.5886	-10.5290	2.129800	-0.182350	...	-0.553880
1048571	0.40570	-10.0510	-1.11310	-0.569570	-0.75422	-0.52456	-6.0806	-10.5710	1.877700	-0.182350	...	-0.553880
1048572	0.48707	-10.0110	-1.09980	-0.593690	-0.75047	-0.53045	-5.3752	-10.0910	1.870400	-0.078431	...	-0.560340
1048573	0.41283	-9.8038	-1.17360	-0.593690	-0.75047	-0.53045	-4.6322	-9.7003	1.973300	-0.078431	...	-0.560340
1048574	0.29146	-9.8527	-1.11470	-0.593690	-0.75047	-0.53045	-3.9156	-9.5098	2.024500	-0.078431	...	-0.560340

641855 rows × 21 columns

9.DATA SPLITTING

Feature selection is the process of choosing a subset of the most important features (predictors, variables) to use in a model. By removing irrelevant or redundant features, feature selection can improve the performance of machine learning models, reduce overfitting, and decrease computational cost. The goal is to improve the generalization ability of the model while keeping as much relevant information as possible.

```
# Define features and target
x = df1_one_hot_encoded.drop(['Activity'], axis=1)
y = df1_one_hot_encoded['Activity']
```

x: The features (independent variables). This is typically a DataFrame or a NumPy array containing the input data used to predict the target variable.

y: The target (dependent variable). This is usually a Series or an array that contains the values you want to predict.

7.Split the data

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2, random_state=42)
```

train_test_split() is a useful function for splitting your dataset into training and testing subsets to evaluate model performance. Using a fixed random_state ensures reproducibility of the split. test_size=0.2: This determines the proportion of the dataset that will be used for testing. In this case, 20% (0.2) of the data will be used for testing, and the remaining 80% will be used for training.

```
x_train.shape
```

```
(513484, 20)
```

```
y_train.shape
```

```
(513484,)
```

```
x_test.shape
```

```
(128371, 20)
```

```
y_test.shape
```

```
(128371,)
```

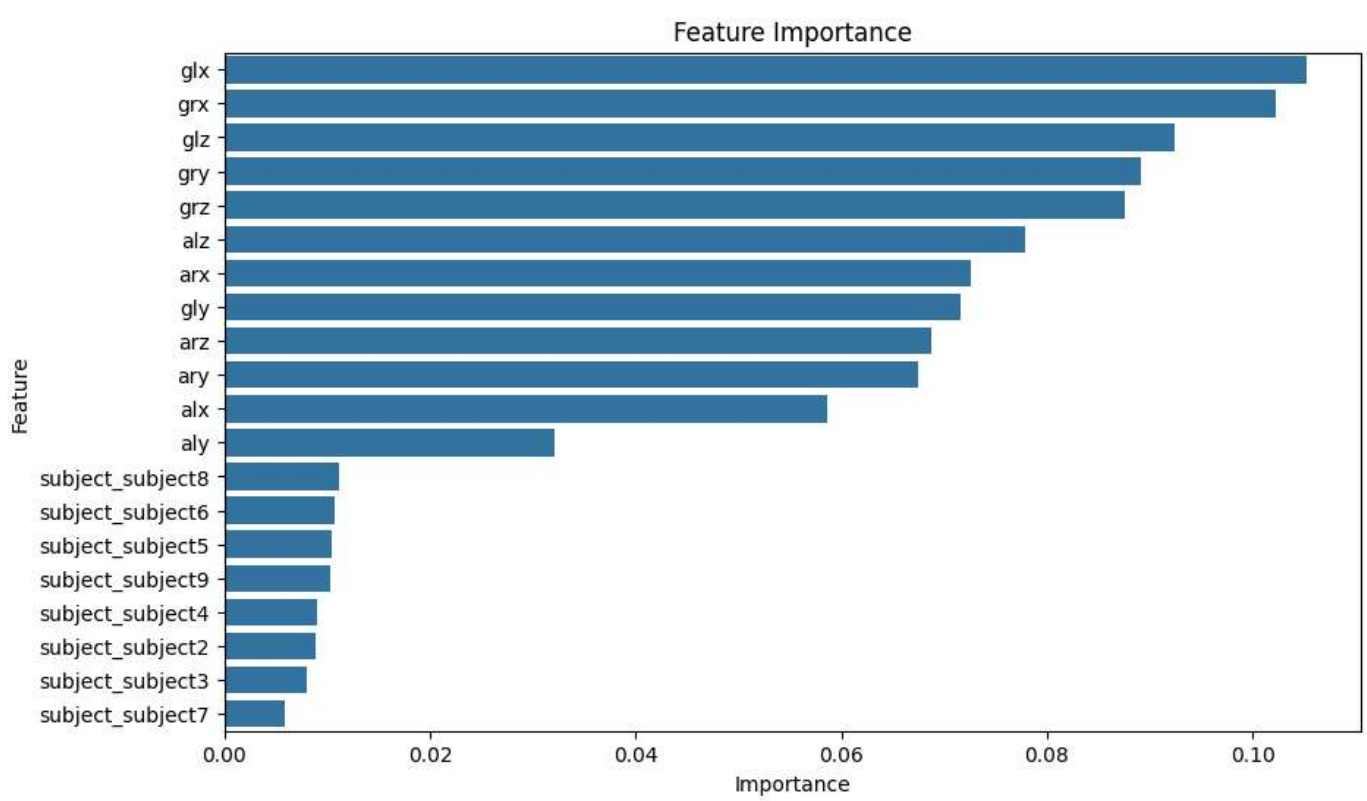
10.FEATURE SELECTION

In addition to feature selection, feature importance is another concept closely related to understanding which features contribute the most to the model's predictions. Various machine learning algorithms, like decision trees, random forests, gradient boosting machines (GBMs), and linear models, can provide an importance score for each feature.

```
rf = RandomForestClassifier(n_estimators=100,random_state=0)  # Use RandomForestclassification for classification task
rf.fit(x_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=0)
```

```
feature_importances = rf.feature_importances_

# Create a DataFrame for visualization
feature_importance_df = pd.DataFrame({
    'Feature': x_train.columns,
    'Importance': feature_importances
})

# Sort the DataFrame by importance
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Plot feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title('Feature Importance')
plt.show()
```




```
# Assuming 'feature_importance_df' has a column 'Importance' that ranks the features
top_features = feature_importance_df.nlargest(12, 'Importance')['Feature'].values

# Select the top features from the training and test sets
x_train_selected = x_train[top_features]
x_test_selected = x_test[top_features]
```

```
top_features
```

```
array(['glx', 'grx', 'glz', 'gry', 'grz', 'alz', 'arx', 'gly', 'arz',
       'ary', 'alx', 'aly'], dtype=object)
```

11.DATA SCALING

Scaling is the process of adjusting the range and distribution of numerical features in your dataset so that they have similar scales. Many machine learning algorithms (e.g., linear models, k-NN, neural networks, and gradient descent optimization methods) perform better when the features are scaled, especially if the features have different units or ranges. Without scaling, some features may dominate the learning process due to their larger magnitude. Standardization scales the data so that the mean of the feature is 0 and the standard deviation is 1. This is done by subtracting the mean of the feature and dividing by its standard deviation.

```
# Initialize the scaler
scaler = StandardScaler()

# Fit and transform the training data
```

```
x_train_scaled = scaler.fit_transform(x_train_selected)

# Transform the test data (using the same scaler)
x_test_scaled = scaler.transform(x_test_selected)
```

This initializes a StandardScaler object, which standardizes the features by removing the mean and scaling to unit variance (i.e., it transforms the data such that it has a mean of 0 and a standard deviation of 1).

fit(): It calculates the mean and standard deviation for each feature in the training data (x_train_selected). transform(): It then scales the training data by subtracting the mean and dividing by the standard deviation for each feature.

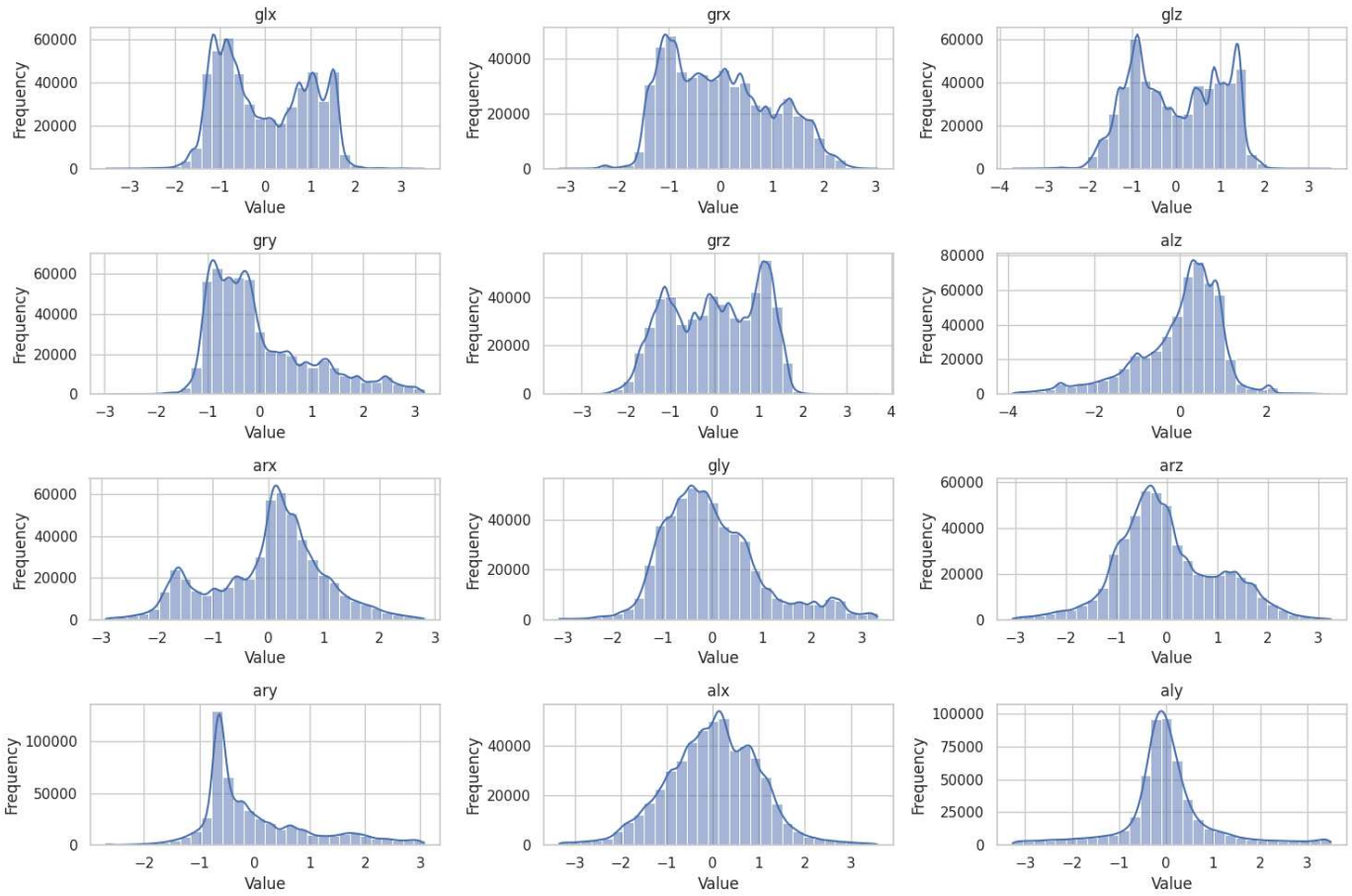
```
x_train_scaled
```

```
array([[-1.59916189, -0.4878251 ,  0.64260412, ..., -0.75610499,
         1.37043944,  0.44920185],
       [-0.67918691, -1.26308778,  1.06625851, ..., -0.27867558,
         0.59786371,  0.21257781],
       [ 0.7905356 ,  0.16525526,  1.35262157, ..., -0.65001676,
         0.4574806 ,  0.41709311],
       ...,
       [ 0.31184223, -0.71695834,  1.47423902, ..., -0.32205666,
         1.86808925,  1.72595938],
       [-0.73155534, -1.23254707,  0.68577222, ..., -0.03972334,
        -0.42989635, -0.79544693],
       [ 1.20190391, -0.25867239,  0.99958955, ..., -0.33477515,
        -0.51067252,  0.50329639]])
```

```
x_test_scaled
```

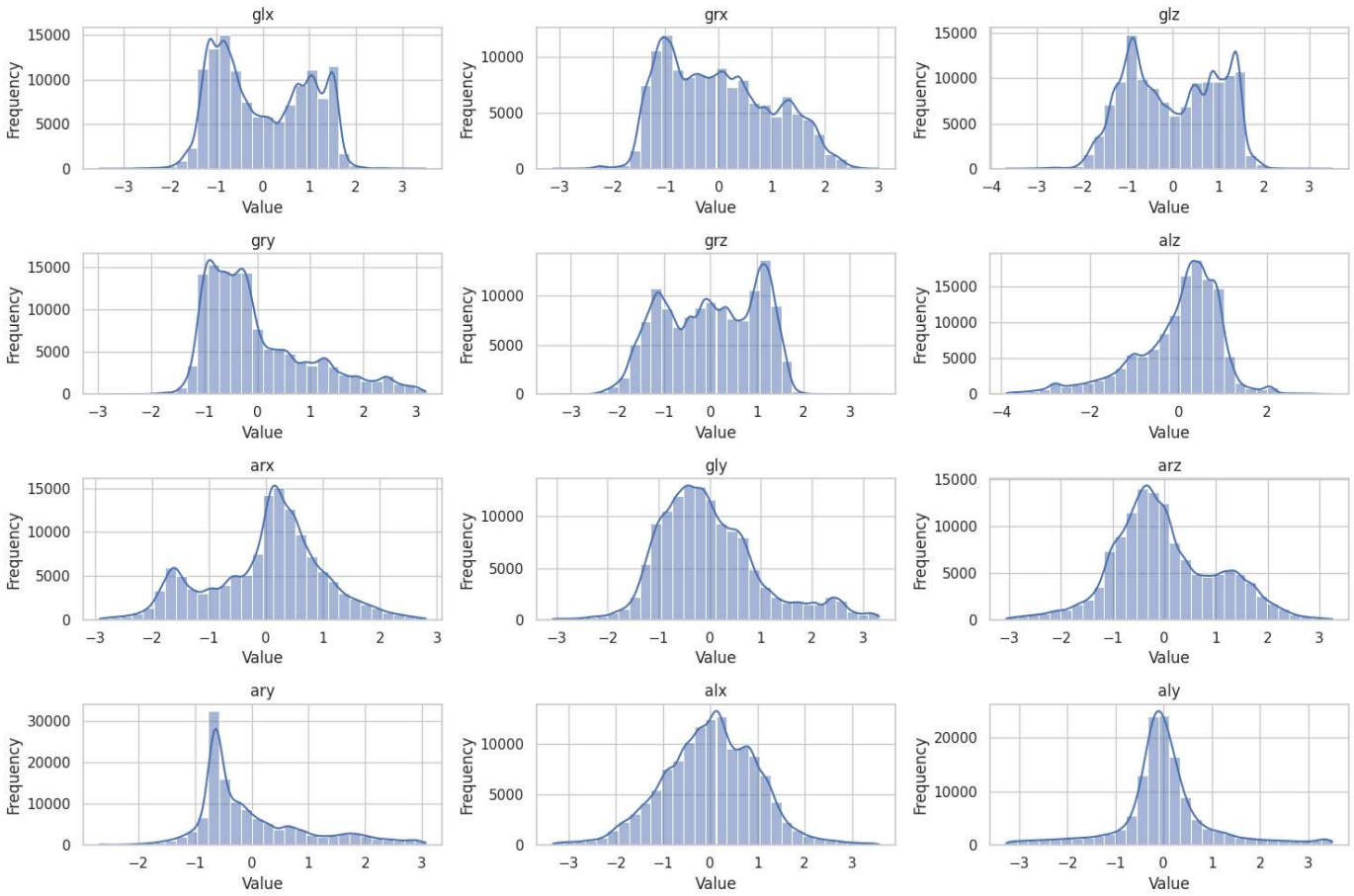
```
array([[ 1.11962218,  1.96406023, -0.96178375, ..., -0.28780957,
        -0.53729988,  0.27125931],
       [ 1.07475224,  0.89852447, -1.02847267, ..., -0.27327723,
         0.61574507, -0.14409813],
       [ 0.84661296,  0.1538025 ,  1.36831543, ..., -0.508537  ,
        -0.33431759,  0.26603966],
       ...,
       [-0.95592522, -0.33122605, -0.98141106, ...,  0.12013238,
        -0.28384601, -0.07086489],
       [ 1.60204464,  0.48605932, -0.26746015, ..., -0.10072461,
         1.2673332 ,  1.44504742],
       [-1.78241106, -1.0492249 ,  0.6896857 , ...,  0.21278953,
        -1.51404759,  0.49760222]])
```

```
#plot histogram for x_train_scaled
# Set the style for seaborn
sns.set(style='whitegrid')

# Create a figure
plt.figure(figsize=(15, 10))  # Adjust the figure size as needed

# Loop through each feature and create a histogram
for i, feature in enumerate(top_features):
    plt.subplot(4, 3, i + 1)  # Adjust the layout based on the number of features
    sns.histplot(x_train_scaled[:, i], kde=True, bins=30)  # You can adjust the number of bins
    plt.title(feature)
    plt.xlabel('Value')
    plt.ylabel('Frequency')

plt.tight_layout()  # Adjust layout for better spacing
plt.show()
```



Start coding or generate with AI.

```
# create histogram for x_test_scaled
# Set the style for seaborn
sns.set(style='whitegrid')

# Create a figure
plt.figure(figsize=(15, 10))  # Adjust the figure size as needed

# Loop through each feature and create a histogram
for i, feature in enumerate(top_features):
    plt.subplot(4, 3, i + 1)  # Adjust the layout based on the number of features
    sns.histplot(x_test_scaled[:, i], kde=True, bins=30)  # We can adjust the number of bins
    plt.title(feature)
    plt.xlabel('Value')
    plt.ylabel('Frequency')

plt.tight_layout()  # Adjust layout for better spacing
plt.show()
```



12.MODEL TRAINING AND EVALUATION

Machine Learning (Supervised)

Classification Models

1. Logistic Regression
2. Random Forest Classification
3. K-Nearest Neighbors (K-NN)
4. Decision Tree Classification
5. Naive Bayes

1.Logistic Regression

Logistic regression is a statistical method used for binary classification, where the goal is to predict one of two possible outcomes (e.g., yes/no, success/failure, 0/1). Unlike linear regression, which predicts a continuous value, logistic regression predicts the probability that a given input belongs to a certain class.

```
lr = LogisticRegression()
lr.fit(x_train_scaled,y_train)
```