In [3]:
```
# This Python 3 environment comes with many helpful analytics libraries ins
# It is defined by the kaggle/python Docker image: https://github.com/kaggl
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that
# You can also write temporary files to /kaggle/temp/, but they won't be sa
```

```
/kaggle/input/kills-csgo/esea_master_kills_demos.part2.csv
/kaggle/input/kills-csgo/esea_master_kills_demos.part1.csv
/kaggle/input/maps-data/esea_meta_demos_cbble.csv
/kaggle/input/maps-data/esea_meta_demos_inferno.csv
/kaggle/input/maps-data/esea_meta_demos_dust2.csv
/kaggle/input/maps-data/esea_meta_demos_nuke.csv
/kaggle/input/maps-data/esea_meta_demos_train.csv
/kaggle/input/maps-data/esea_meta_demos_cache.csv
/kaggle/input/maps-data/esea_meta_demos_overpass.csv
/kaggle/input/map-model-csgo/map_data.csv
/kaggle/input/map-model-csgo/esea_meta_demos.part2.csv
/kaggle/input/map-model-csgo/esea_meta_demos.part1.csv
/kaggle/input/map-model-csgo/esea_master_dmg_demos.part2.csv
/kaggle/input/map-model-csgo/esea_master_dmg_demos.part1.csv
/kaggle/input/maps-images/de_mirage.png
/kaggle/input/maps-images/de_cache.png
/kaggle/input/gren-data/mm_grenades_demos.csv
/kaggle/input/mm-master-demos/mm_master_demos.csv
/kaggle/input/de-overpass/de_overpass.png
```

VIDEO GAME ANALYTICS

Video game analytics refers to the process of collecting and analyzing data generated by players' interactions within video games. This data can encompass a wide range of metrics, including player behavior, in-game transactions, engagement patterns, and performance metrics. The primary goal of video game analytics is to gain insights into player behavior and preferences, which can then be used to improve various aspects of the game, such as game design, monetization strategies, and marketing campaigns.

USE OF MACHINE LEARNING

Machine learning (ML) plays a crucial role in video game analytics by enabling developers and publishers to extract meaningful insights from large volumes of data.

MOTTO OF THE PROJECT

Our project dives into the world of Counter-Strike: Global Offensive (CS:GO) to uncover the secrets of success on three iconic maps: Overpass, Mirage, and Cache. By analyzing player data, we reveal the most effective strategies and popular tactics used by players. From pinpointing where battles heat up to understanding which weapons reign supreme, we provide valuable insights for both players looking to up their game and developers seeking to enhance map design. Join us as we explore the strategies that define CS:GO gameplay and illuminate the path to victory in one of the most beloved esports titles of our time.

DATASETS DESCRIPTION

Master_damage Dataset:

- round: The round number in the match.
- tick: The tick at which the damage occurred.
- seconds: The time in seconds when the damage occurred.
- att_team: The team that initiated the attack.
- vic_team: The team that received the damage.
- att_side: The side (Terrorist or Counter-Terrorist) of the attacking team.
- vic_side: The side (Terrorist or Counter-Terrorist) of the victim team.
- hp_dmg: The amount of damage dealt to the victim's health points.
- arm_dmg: The amount of damage dealt to the victim's armor points.
- esea_match_13770997.dem: Identifier for the demo file.

Meta_Demos Dataset:

- file: Identifier for the demo file.
- map: The map being played in the match.
- round: The round number in the match.
- start_seconds: The starting time of the round in seconds.
- end_seconds: The ending time of the round in seconds.
- winner_team: The team that won the round.
- winner_side: The side (Terrorist or Counter-Terrorist) of the winning team.
- round_type: The type of round (e.g., PISTOL_ROUND).
- ct_eq_val: The value of equipment for Counter-Terrorists.
- t_eq_val: The value of equipment for Terrorists.

MM_Master Dataset:

- round: The round number in the match.
- tick: The tick at which the event occurred.
- seconds: The time in seconds when the event occurred.
- att_team: The team that initiated the action.
- vic_team: The team that received the action.
- att_side: The side (Terrorist or Counter-Terrorist) of the attacking team.
- vic_side: The side (Terrorist or Counter-Terrorist) of the victim team.
- hp_dmg: The amount of damage dealt to the victim's health points.

- arm_dmg: The amount of damage dealt to the victim's armor points.
- is_bomb_planted: Boolean indicating whether the bomb was planted (TRUE or FALSE).

MM_Grenades Dataset:

- file: Identifier for the demo file.
- map: The map being played in the match.
- round: The round number in the match.
- start_seconds: The starting time of the round in seconds.
- seconds: The time in seconds when the event occurred.
- end_seconds: The ending time of the round in seconds.
- att_team: The team that initiated the action.
- vic_team: The team that received the action.
- att_id: Identifier for the attacker.
- vic_id: Identifier for the victim.
- att_side: The side (Terrorist or Counter-Terrorist) of the attacking team.
- vic_side: The side (Terrorist or Counter-Terrorist) of the victim team.
- hp_dmg: The amount of damage dealt to the victim's health points.
- arm_dmg: The amount of damage dealt to the victim's armor points.
- is_bomb_planted: Boolean indicating whether the bomb was planted (TRUE or FALSE).
- bomb_site: The site where the bomb was planted.
- hitbox: The hitbox hit by the grenade.
- nade: Type of grenade used.
- winner_team: The team that won the round.
- winner_side: The side (Terrorist or Counter-Terrorist) of the winning team.
- att_rank: Rank of the attacker.
- vic_rank: Rank of the victim.
- att_pos_x, att_pos_y: Attacker's position coordinates.
- nade_land_x, nade_land_y: Grenade landing position coordinates.
- vic_pos_x, vic_pos_y: Victim's position coordinates.
- round_type: The type of round (e.g., PISTOL_ROUND).
- ct_eq_val: The value of equipment for Counter-Terrorists.
- t_eq_val: The value of equipment for Terrorists.
- avg_match_rank: Average rank of all players in the match.

In [4]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [5]:
```
df = pd.concat([pd.read_csv('/kaggle/input/map-model-csgo/esea_master_dmg_d
meta = pd.concat([pd.read_csv('/kaggle/input/map-model-csgo/esea_meta_demos
```

In [6]:
```
meta
```

Out[6]:

	file	map	round	start_seconds	end_seconds	winner_t
0	esea_match_13770997.dem	de_overpass	1	94.30782	160.9591	H Hooli
1	esea_match_13770997.dem	de_overpass	2	160.95910	279.3998	H Hooli
2	esea_match_13770997.dem	de_overpass	3	279.39980	341.0084	H Hooli
3	esea_match_13770997.dem	de_overpass	4	341.00840	435.4259	H Hooli
4	esea_match_13770997.dem	de_overpass	5	435.42590	484.2398	Animal
...	...	...	...	...	...	
161705	esea_match_13829173.dem	de_mirage	18	1647.20600	1694.6550	Te
161706	esea_match_13829173.dem	de_mirage	19	1694.65500	1753.4380	Te
161707	esea_match_13829173.dem	de_mirage	20	1753.43800	1803.6580	Te
161708	esea_match_13829173.dem	de_mirage	21	1803.65800	1856.0850	Te
161709	esea_match_13829173.dem	de_mirage	22	1856.08500	1905.8660	Te

377629 rows × 10 columns

In [7]:
```
meta.groupby('map')['file'].count()
```

Out[7]:
```
map
de_cache       128024
de_cbble         5792
de_dust2        20225
de_inferno      52876
de_mirage      132862
de_nuke          1549
de_overpass     23697
de_train        12604
Name: file, dtype: int64
```

DATA LINKING DONE USING JOIN FUNCTION

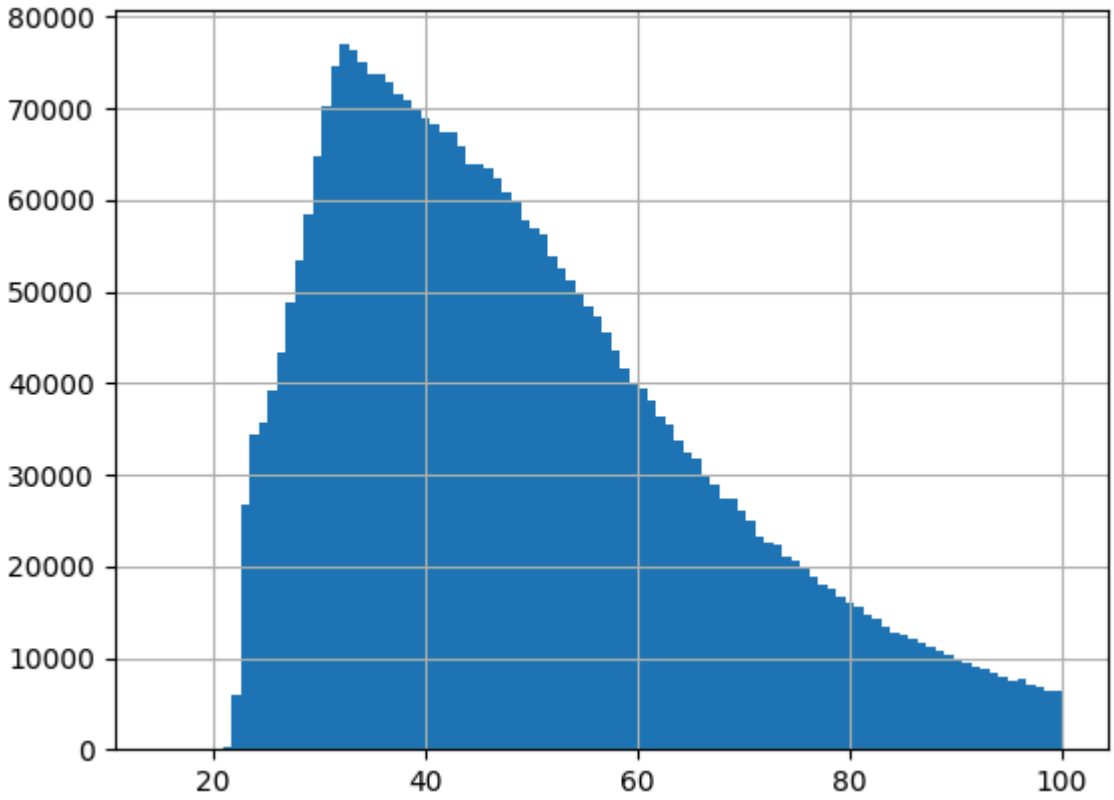
This line of code filters and selects specific rows from the original DataFrame df based on matches played on the map and with a specific index structure, and then joins this filtered DataFrame with another DataFrame meta based on matching index values.

In [8]:
```
meta = meta[meta['map'] == 'de_mirage'].set_index(['file', 'round'])
df_mirage = df.set_index(['file', 'round']).join(meta, how = 'inner')
```

In [9]:
```
df_mirage['s'] = df_mirage['seconds'] - df_mirage['start_seconds']
df_mirage = df_mirage[(df_mirage['s'] < 100) & (df_mirage['s'] > 0)]
```

In [10]:
```
df_mirage['s'].hist(bins=100)
```

Out[10]: <Axes: >



INTERPRETATION

This histogram is showed to find the maximum number of seconds a match happened in the specific map and from this we can see that the maximum is between 7000 to 8000 seconds in hours is upto 2 hours and the lowest is under 1000 seconds is roughly 15 to 20 minutes which is a minimum for a match according to the game standards

In [11]:
```
df_mirage['bin'] = pd.cut(df_mirage['s'], 20, labels=list(range(0,20))).ast
```

In [12]:
```
ddf = df_mirage.groupby(['file', 'round', 'bin', 'att_side', 'vic_side'])[[
```

ALGORITHM USED

KNN

K-Nearest Neighbors (KNN) is a simple yet powerful supervised learning algorithm used for classification and regression tasks. The main idea behind KNN is to classify or predict the label of a new data point by looking at the 'k' closest labeled data points and taking a majority vote (for classification) or averaging (for regression).

PROCESS

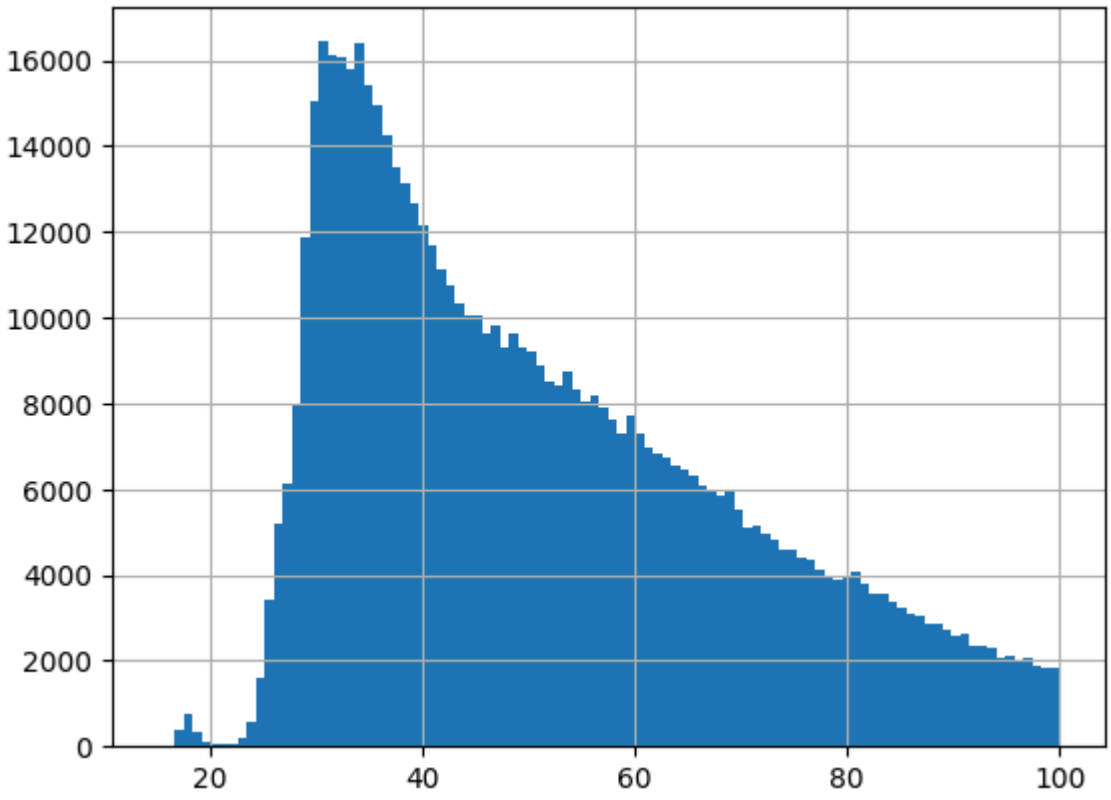
- Filtering: It extracts rows from a DataFrame where a specific column ('bin') has a certain value ('2') and creates a new DataFrame ('df1').
- Data Loading: It loads data from a CSV file ('map_data.csv') containing map bounds information.
- Data Rescaling: It rescales position coordinates in 'df1' to match the map bounds of a specific map ('de_mirage').
- DataFrame Creation: It constructs a new DataFrame ('df2') by combining subsets of 'df1' related to player positions and sides. Additionally, it adds a binary column indicating whether the side is CounterTerrorist or not.
- Model Initialization: It sets up a KNN classifier with specified parameters (100 neighbors and 'distance' weights) and trains it using position coordinates and binary labels from 'df2'.
- Prediction: It generates a meshgrid covering the entire map area and predicts labels for each point in the mesh using the trained KNN classifier.
- Visualization: It plots a map image ('de_mirage.png') using Matplotlib, overlays scatter plots for CounterTerrorist and Terrorist positions, and adds contour lines based on KNN predictions to visualize the decision boundary between CounterTerrorist and Terrorist areas.

In [13]:
```
df1 = ddf[ddf['bin'] == '2']
df1['map'] = 'de_mirage'

map_bounds = pd.read_csv('/kaggle/input/map-model-csgo/map_data.csv', index
md = map_bounds.loc[df1['map']]
md[['att_pos_x', 'att_pos_y', 'vic_pos_x', 'vic_pos_y']] = (df1.set_index('
md['att_pos_x'] = (md['ResX']*(md['att_pos_x']-md['StartX']))/(md['EndX']-m
md['att_pos_y'] = (md['ResY']*(md['att_pos_y']-md['StartY']))/(md['EndY']-m
md['vic_pos_x'] = (md['ResX']*(md['vic_pos_x']-md['StartX']))/(md['EndX']-m
md['vic_pos_y'] = (md['ResY']*(md['vic_pos_y']-md['StartY']))/(md['EndY']-m
df1[['att_pos_x', 'att_pos_y', 'vic_pos_x', 'vic_pos_y']] = md[['att_pos_x'

df2 = pd.concat([df1[['att_side', 'att_pos_x', 'att_pos_y']].rename(columns
                 df1[['vic_side', 'vic_pos_x', 'vic_pos_y']].rename(columns=
df2['z'] = df2['side'] == 'CounterTerrorist'

from sklearn import neighbors

clf = neighbors.KNeighborsClassifier(100, weights='distance')
clf.fit(df2[['x', 'y']], df2['z'])
xx, yy = np.meshgrid(np.arange(0, 1024, 1),
                     np.arange(0, 1024, 1))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

import matplotlib.pyplot as plt
import numpy as np

# Assuming you have defined xx, yy, Z already for contour plot

# Read the image using matplotlib
bg = plt.imread('/kaggle/input/maps-images/de_mirage.png')
fig, ax1 = plt.subplots(1,1,figsize=(18,16))
ax1.grid(True, which='major', color='w', linestyle='--', alpha=0.25)  # Fix

# Display the image
ax1.imshow(bg, zorder=0, extent=[0.0, 1024, 0., 1024])

plt.xlim(0,1024)
plt.ylim(0,1024)

# Assuming df2 is defined
df2[df2['side'] == 'CounterTerrorist'].plot(x='x', y='y', kind='scatter', a
df2[df2['side'] == 'Terrorist'].plot(x='x', y='y', kind='scatter', ax=ax1,

# Overlay contours
contour = ax1.contour(xx, yy, Z, cmap=plt.cm.Paired)

# Show plot
plt.show()
```

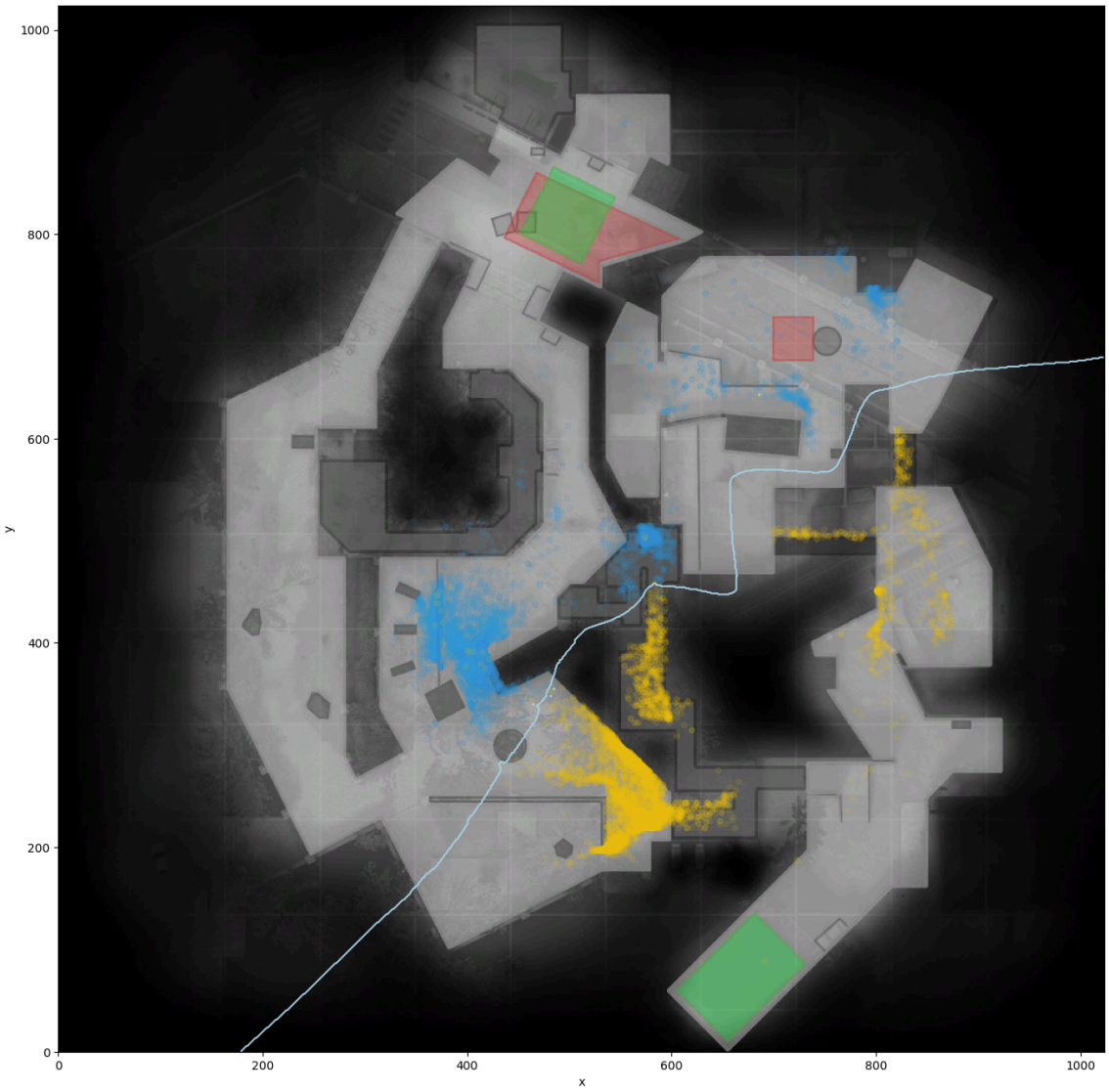
In [14]:
```
Z
```

Out[14]:
```
array([[ True,  True,  True, ..., False, False, False],
       [ True,  True,  True, ..., False, False, False],
       [ True,  True,  True, ..., False, False, False],
       ...,
       [ True,  True,  True, ..., False, False, False],
       [ True,  True,  True, ..., False, False, False],
       [ True,  True,  True, ..., False, False, False]])
```

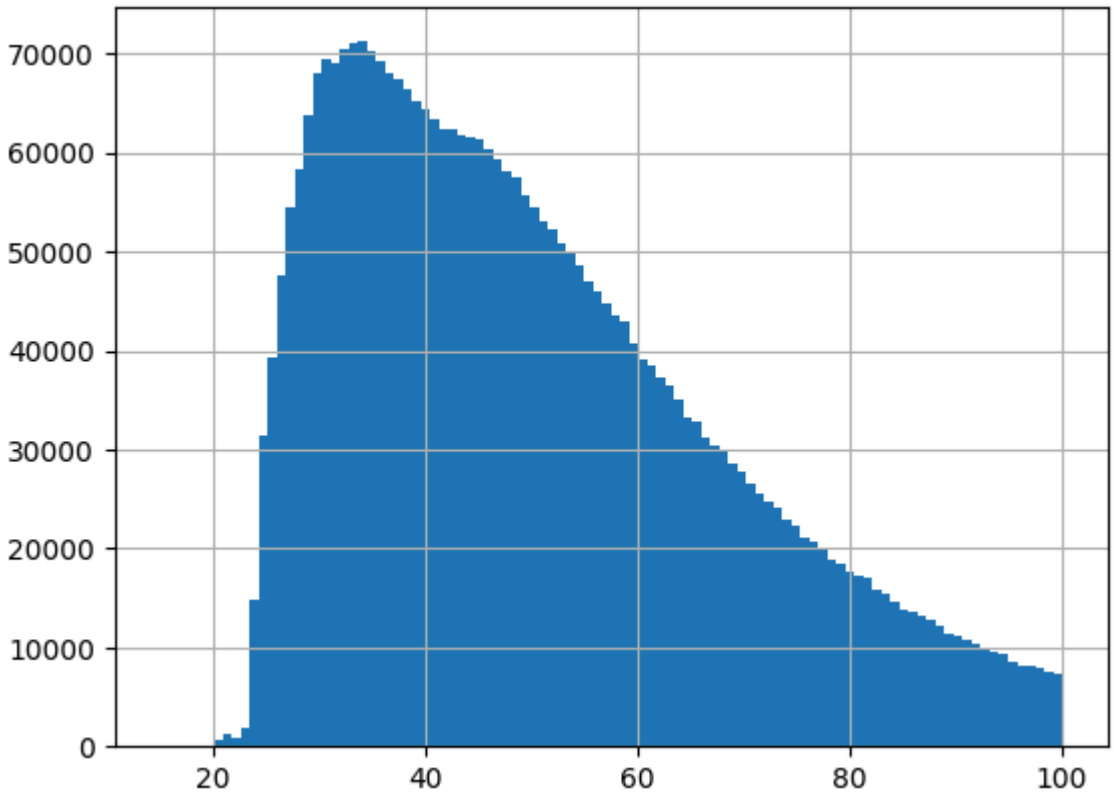
In [15]:
```
meta = pd.read_csv("/kaggle/input/maps-data/esea_meta_demos_overpass.csv")
meta1 = meta[meta['map'] == 'de_overpass'].set_index(['file', 'round'])

# Assuming df is defined elsewhere
df_overpass = df.join(meta1, how='inner', on=['file', 'round'])

df_overpass['s'] = df_overpass['seconds'] - df_overpass['start_seconds']
df_overpass = df_overpass[(df_overpass['s'] < 100) & (df_overpass['s'] > 0)

# Assuming df_cruise is defined elsewhere
df_overpass['s'].hist(bins=100)

df_overpass['bin'] = pd.cut(df_overpass['s'], 20, labels=list(range(0,20)))

ddf = df_overpass.groupby(['file', 'round', 'bin', 'att_side', 'vic_side'])
```



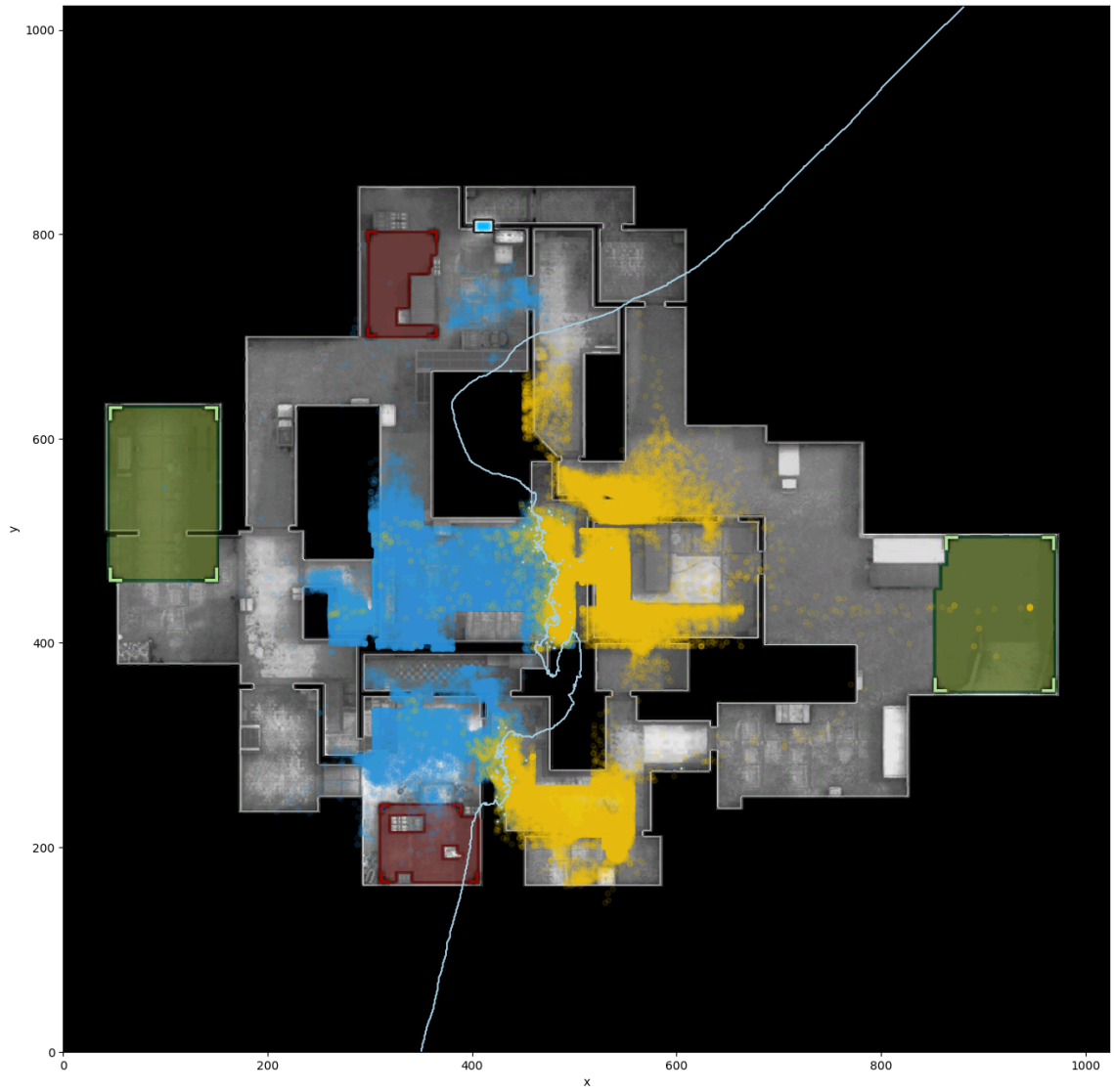
In [16]:
```
df1 = ddf[ddf['bin'] == '2']
df1['map'] = 'de_overpass'

map_bounds = pd.read_csv('/kaggle/input/map-model-csgo/map_data.csv', index
md = map_bounds.loc[df1['map']]
md[['att_pos_x', 'att_pos_y', 'vic_pos_x', 'vic_pos_y']] = (df1.set_index('
md['att_pos_x'] = (md['ResX']*(md['att_pos_x']-md['StartX']))/(md['EndX']-m
md['att_pos_y'] = (md['ResY']*(md['att_pos_y']-md['StartY']))/(md['EndY']-m
md['vic_pos_x'] = (md['ResX']*(md['vic_pos_x']-md['StartX']))/(md['EndX']-m
md['vic_pos_y'] = (md['ResY']*(md['vic_pos_y']-md['StartY']))/(md['EndY']-m
df1[['att_pos_x', 'att_pos_y', 'vic_pos_x', 'vic_pos_y']] = md[['att_pos_x'

df2 = pd.concat([df1[['att_side', 'att_pos_x', 'att_pos_y']].rename(columns
                 df1[['vic_side', 'vic_pos_x', 'vic_pos_y']].rename(columns=
df2['z'] = df2['side'] == 'CounterTerrorist'

from sklearn import neighbors

clf = neighbors.KNeighborsClassifier(84, weights='distance')
clf.fit(df2[['x', 'y']], df2['z'])
xx, yy = np.meshgrid(np.arange(0, 1024, 1),
                     np.arange(0, 1024, 1))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

import matplotlib.pyplot as plt
import numpy as np

# Assuming you have defined xx, yy, Z already for contour plot

# Read the image using matplotlib
bg = plt.imread('/kaggle/input/de-overpass/de_overpass.png')
fig, ax1 = plt.subplots(1,1,figsize=(18,16))
ax1.grid(True, which='major', color='w', linestyle='--', alpha=0.25)  # Fix

# Display the image
ax1.imshow(bg, zorder=0, extent=[0.0, 1024, 0., 1024])

plt.xlim(0,1024)
plt.ylim(0,1024)

# Assuming df2 is defined
df2[df2['side'] == 'CounterTerrorist'].plot(x='x', y='y', kind='scatter', a
df2[df2['side'] == 'Terrorist'].plot(x='x', y='y', kind='scatter', ax=ax1,

# Overlay contours
contour = ax1.contour(xx, yy, Z, cmap=plt.cm.Paired)

# Show plot
plt.show()
```


1000
800
600
400
200
0
y
0
200
400
600
800
1000
x

In [17]:
```
meta = pd.read_csv("/kaggle/input/maps-data/esea_meta_demos_cache.csv")
meta2 = meta[meta['map'] == 'de_cache'].set_index(['file', 'round'])

# Assuming df is defined elsewhere
df_cache = df.join(meta2, how='inner', on=['file', 'round'])

df_cache['s'] = df_cache['seconds'] - df_cache['start_seconds']
df_cache = df_cache[(df_cache['s'] < 100) & (df_cache['s'] > 0)]

# Assuming df_cruise is defined elsewhere
df_cache['s'].hist(bins=100)

df_cache['bin'] = pd.cut(df_cache['s'], 20, labels=list(range(0,20))).astyp

ddf = df_cache.groupby(['file', 'round', 'bin', 'att_side', 'vic_side'])[['
```



In [18]:
```
df1 = ddf[ddf['bin'] == '2']
df1['map'] = 'de_cache'

map_bounds = pd.read_csv('/kaggle/input/map-model-csgo/map_data.csv', index
md = map_bounds.loc[df1['map']]
md[['att_pos_x', 'att_pos_y', 'vic_pos_x', 'vic_pos_y']] = (df1.set_index('
md['att_pos_x'] = (md['ResX']*(md['att_pos_x']-md['StartX']))/(md['EndX']-m
md['att_pos_y'] = (md['ResY']*(md['att_pos_y']-md['StartY']))/(md['EndY']-m
md['vic_pos_x'] = (md['ResX']*(md['vic_pos_x']-md['StartX']))/(md['EndX']-m
md['vic_pos_y'] = (md['ResY']*(md['vic_pos_y']-md['StartY']))/(md['EndY']-m
df1[['att_pos_x', 'att_pos_y', 'vic_pos_x', 'vic_pos_y']] = md[['att_pos_x'

df2 = pd.concat([df1[['att_side', 'att_pos_x', 'att_pos_y']].rename(columns
                 df1[['vic_side', 'vic_pos_x', 'vic_pos_y']].rename(columns=
df2['z'] = df2['side'] == 'CounterTerrorist'

from sklearn import neighbors

clf = neighbors.KNeighborsClassifier(84, weights='distance')
clf.fit(df2[['x', 'y']], df2['z'])
xx, yy = np.meshgrid(np.arange(0, 1024, 1),
                     np.arange(0, 1024, 1))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

import matplotlib.pyplot as plt
import numpy as np

# Assuming you have defined xx, yy, Z already for contour plot

# Read the image using matplotlib
bg = plt.imread('/kaggle/input/maps-images/de_cache.png')
fig, ax1 = plt.subplots(1,1,figsize=(18,16))
ax1.grid(True, which='major', color='w', linestyle='--', alpha=0.25)  # Fix

# Display the image
ax1.imshow(bg, zorder=0, extent=[0.0, 1024, 0., 1024])

plt.xlim(0,1024)
plt.ylim(0,1024)

# Assuming df2 is defined
df2[df2['side'] == 'CounterTerrorist'].plot(x='x', y='y', kind='scatter', a
df2[df2['side'] == 'Terrorist'].plot(x='x', y='y', kind='scatter', ax=ax1,

# Overlay contours
contour = ax1.contour(xx, yy, Z, cmap=plt.cm.Paired)

# Show plot
plt.show()
```



INTERPRETATION:

Using the KNN algorithm we found some insights.By plotting player positions as scatter points on the map, the code provides a visual representation of player activity and hotspots, enabling insights into strategic areas, chokepoints, and common routes used by players during gameplay on the 'de_mirage','de_overpass','de_cache' map in CS:GO. Using these points we can see whether we can prefer an alternative route other than the highlighted positions and then we can understand and skip out on the concentrated points and prefer alternatives or prepare some trap in the locations which can cover wide range and area

In [19]:
```
# Read the two parts of the dataset into Pandas DataFrames
part1_df = pd.read_csv("/kaggle/input/kills-csgo/esea_master_kills_demos.pa
part2_df = pd.read_csv("/kaggle/input/kills-csgo/esea_master_kills_demos.pa

# Concatenate the two DataFrames vertically
dataset= pd.concat([part1_df, part2_df], ignore_index=True)
```

SECOND INDENTIFICATION STRATEGY

ALGORITHM USED

LOGISTIC REGRESSION

Logistic Regression is a statistical method used for binary classification tasks, where the target variable has two possible outcomes. It's widely used in various fields, including healthcare, finance, and marketing, for predicting the probability of a certain event occurring.

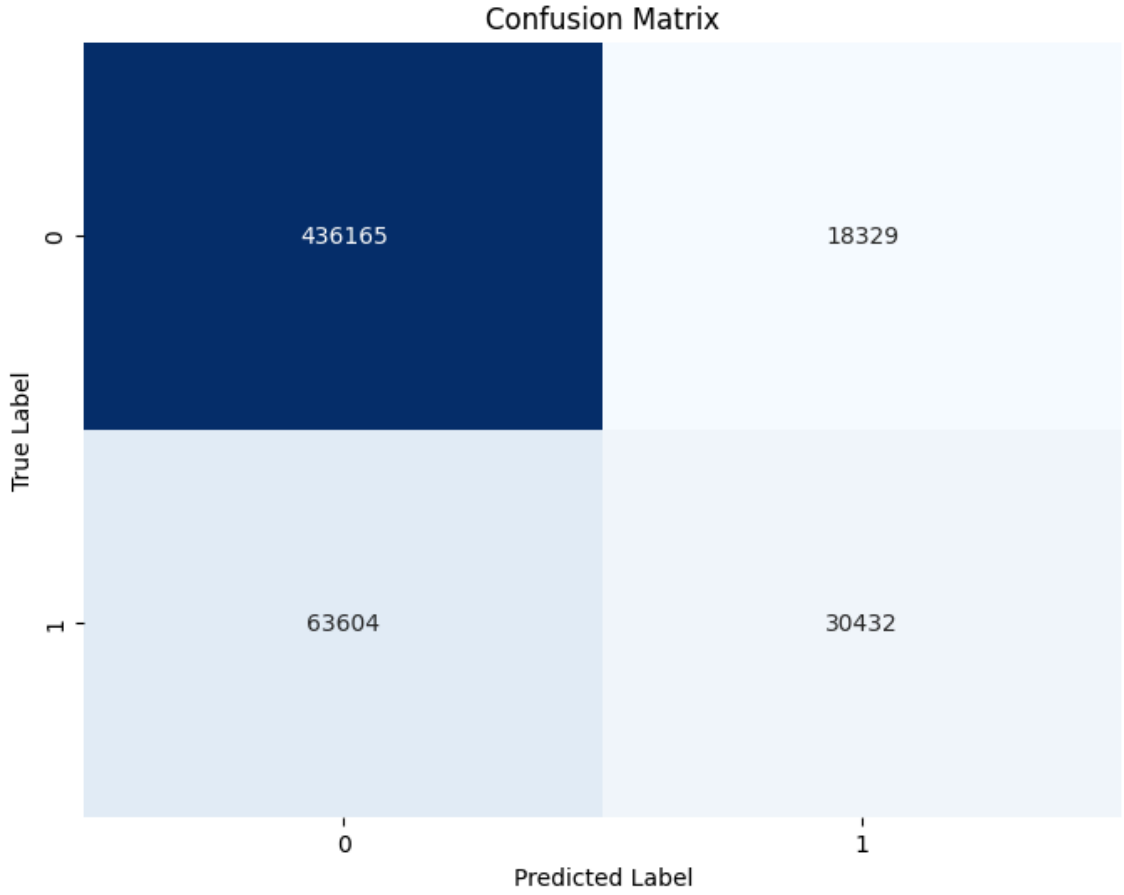
PROCEDURE

- Data Preparation:Unnecessary columns are dropped, and categorical variables are encoded.
- Feature Engineering:Features (X) are defined by dropping the target variable.
- Handling Missing Values:Missing values are replaced with the mean of each column.
- Data Splitting:The dataset is split into training and testing sets.
- Model Training:Logistic regression model is initialized and trained on the training data.
- Model Evaluation:Predictions are made on the test set, and accuracy is calculated. Confusion matrix is generated to evaluate model performance.

In [20]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt
import seaborn as sns

# Drop unnecessary columns
# Drop unnecessary columns if they exist
if 'file' in dataset.columns:
    dataset.drop(columns=['file'], inplace=True)  # Assuming 'file' column

# Encode categorical variables (if any)
# Encode categorical variables (if any)
dataset = pd.get_dummies(dataset)

# Define features (X) by dropping the target variable
X = dataset.drop(columns=['is_bomb_planted'])

# Handle missing values
# Replace missing values with the mean of each column
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)

# Define binary target variable based on 'is_bomb_planted'
y = dataset['is_bomb_planted']  # Target variable

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y, test_size

# Initialize and train the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 0.8506316883306292
Confusion Matrix:
[[436165  18329]
 [ 63604  30432]]
```

In [21]:
```
# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



INTERPRETATION

- Accuracy: The model achieved an accuracy of approximately 85.06%, indicating that it correctly predicted whether a bomb would be planted or not in around 85.06% of cases.

Confusion Matrix:

- The top-left cell (True Negative, TN) represents the number of instances where the model correctly predicted that a bomb would not be planted (436,165).
- The top-right cell (False Positive, FP) represents the number of instances where the model incorrectly predicted that a bomb would be planted when it wasn't (18,329).
- The bottom-left cell (False Negative, FN) represents the number of instances where the model incorrectly predicted that a bomb would not be planted when it was (63,604).
- The bottom-right cell (True Positive, TP) represents the number of instances where the model correctly predicted that a bomb would be planted (30,432).

THIRD IDENTIFICATION

ALGORITHM USED

APRIORI ALGORITHM

The Apriori algorithm is a classic algorithm used in association rule learning and data mining. Its main objective is to discover frequent itemsets in transactional databases and generate association rules based on these itemsets.

PROCEDURE

- Support Calculation: The algorithm begins by scanning the database to calculate the support of each item. Support refers to the frequency of occurrence of an item or itemset in the database.
- Frequent Itemset Generation: After calculating the support, the algorithm generates frequent itemsets, which are sets of items that have support greater than or equal to a specified minimum support threshold.
- Join Step: In this step, the algorithm joins pairs of frequent itemsets to form candidate itemsets of larger sizes. This process is repeated iteratively to generate candidate itemsets of increasing sizes.
- Prune Step: After generating candidate itemsets, the algorithm prunes the candidate itemsets that do not meet the minimum support threshold. This pruning step helps reduce the search space and computational complexity.
- Association Rule Generation: Finally, the algorithm generates association rules from the frequent itemsets. An association rule consists of an antecedent (left-hand side) and a consequent (right-hand side), indicating a relationship between sets of items.

In [26]:
```
from mlxtend.frequent_patterns import apriori, association_rules
import pandas as pd

# Assuming 'dataset' is your DataFrame containing the dataset
dataset=pd.read_csv("/kaggle/input/map-model-csgo/esea_meta_demos.part1.csv
# Filter relevant columns for analysis
association_data = dataset[['round_type', 'winner_side', 'winner_team', 'ct

# One-hot encode the data for the Apriori algorithm
association_data_encoded = pd.get_dummies(association_data, columns=['round

# Convert ct_eq_val and t_eq_val to boolean values
association_data_encoded['ct_eq_val'] = association_data_encoded['ct_eq_val
association_data_encoded['t_eq_val'] = association_data_encoded['t_eq_val']

# Perform Apriori algorithm to find frequent itemsets
frequent_itemsets = apriori(association_data_encoded, min_support=0.1, use_

# Generate association rules
association_rules_df = association_rules(frequent_itemsets, metric="lift",

# Display the top association rules
print("Top association rules between equipment values and other columns:")
print(association_rules_df.head(10))
```

```
Top association rules between equipment values and other columns:
                      antecedents                     consequents  \
0                     (ct_eq_val)                      (t_eq_val)
1                      (t_eq_val)                     (ct_eq_val)
2                     (ct_eq_val)                (round_type_ECO)
3                (round_type_ECO)                     (ct_eq_val)
4          (round_type_FORCE_BUY)                     (ct_eq_val)
5                     (ct_eq_val)          (round_type_FORCE_BUY)
6             (round_type_NORMAL)                     (ct_eq_val)
7                     (ct_eq_val)             (round_type_NORMAL)
8                     (ct_eq_val)  (winner_side_CounterTerrorist)
9  (winner_side_CounterTerrorist)                     (ct_eq_val)

   antecedent support  consequent support   support  confidence  lift  \
0            1.000000            1.000000  1.000000    1.000000   1.0
1            1.000000            1.000000  1.000000    1.000000   1.0
2            1.000000            0.341781  0.341781    0.341781   1.0
3            0.341781            1.000000  0.341781    1.000000   1.0
4            0.107614            1.000000  0.107614    1.000000   1.0
5            1.000000            0.107614  0.107614    0.107614   1.0
6            0.411955            1.000000  0.411955    1.000000   1.0
7            1.000000            0.411955  0.411955    0.411955   1.0
8            1.000000            0.487558  0.487558    0.487558   1.0
9            0.487558            1.000000  0.487558    1.000000   1.0

   leverage  conviction  zhangs_metric
0       0.0         inf            0.0
1       0.0         inf            0.0
2       0.0         1.0            0.0
3       0.0         inf            0.0
4       0.0         inf            0.0
5       0.0         1.0            0.0
6       0.0         inf            0.0
7       0.0         1.0            0.0
8       0.0         1.0            0.0
9       0.0         inf            0.0
```

INTERPRETATION

Association between CT Equipment Value and T Equipment Value:

- Antecedents: CT Equipment Value (ct_eq_val)
- Consequents: T Equipment Value (t_eq_val)
- Support: 100%
- Confidence: 100%
- Lift: 1.0
- Interpretation: This rule indicates a strong association between CT equipment value and T equipment value, with all CT equipment values also being T equipment values and vice versa.

Association between CT Equipment Value and Eco Rounds:

- Antecedents: CT Equipment Value (ct_eq_val)
- Consequents: Eco Rounds (round_type_ECO)
- Support: 34.18%
- Confidence: 34.18%
- Lift: 1.0
- Interpretation: This rule suggests that in matches where CTs have high equipment value, there's a 34.18% likelihood that the round type is an Eco round.

Association between CT Equipment Value and Normal Rounds:

- Antecedents: CT Equipment Value (ct_eq_val)
- Consequents: Normal Rounds (round_type_NORMAL)
- Support: 41.20%
- Confidence: 41.20%
- Lift: 1.0
- Interpretation: This rule indicates that in matches where CTs have high equipment value, there's a 41.20% likelihood that the round type is a Normal round.

Association between CT Equipment Value and CounterTerrorist Side:

- Antecedents: CT Equipment Value (ct_eq_val)
- Consequents: CounterTerrorist Side (winner_side_CounterTerrorist)
- Support: 48.76%
- Confidence: 48.76%
- Lift: 1.0
- Interpretation: This rule suggests that in matches where CTs have high equipment value, there's a 48.76% likelihood that the winning side is CounterTerrorist.

FOURTH IDENTIFICATION

ALGORITHM USED

MEAN SHIFT CLUSTERING

Mean Shift clustering is a density-based clustering algorithm that aims to discover clusters in a dataset by identifying the modes or peaks in its underlying probability density function. Unlike traditional clustering algorithms like K-means, Mean Shift does not require the number of clusters to be specified in advance, making it particularly useful when the number of clusters is unknown.

PROCEDURE

- Initialization: Choose a bandwidth parameter and initialize cluster centers.
- Density Estimation: Estimate the density for each data point using a kernel function.
- Mean Shift Iteration: Compute mean shift vectors and update data point positions iteratively.

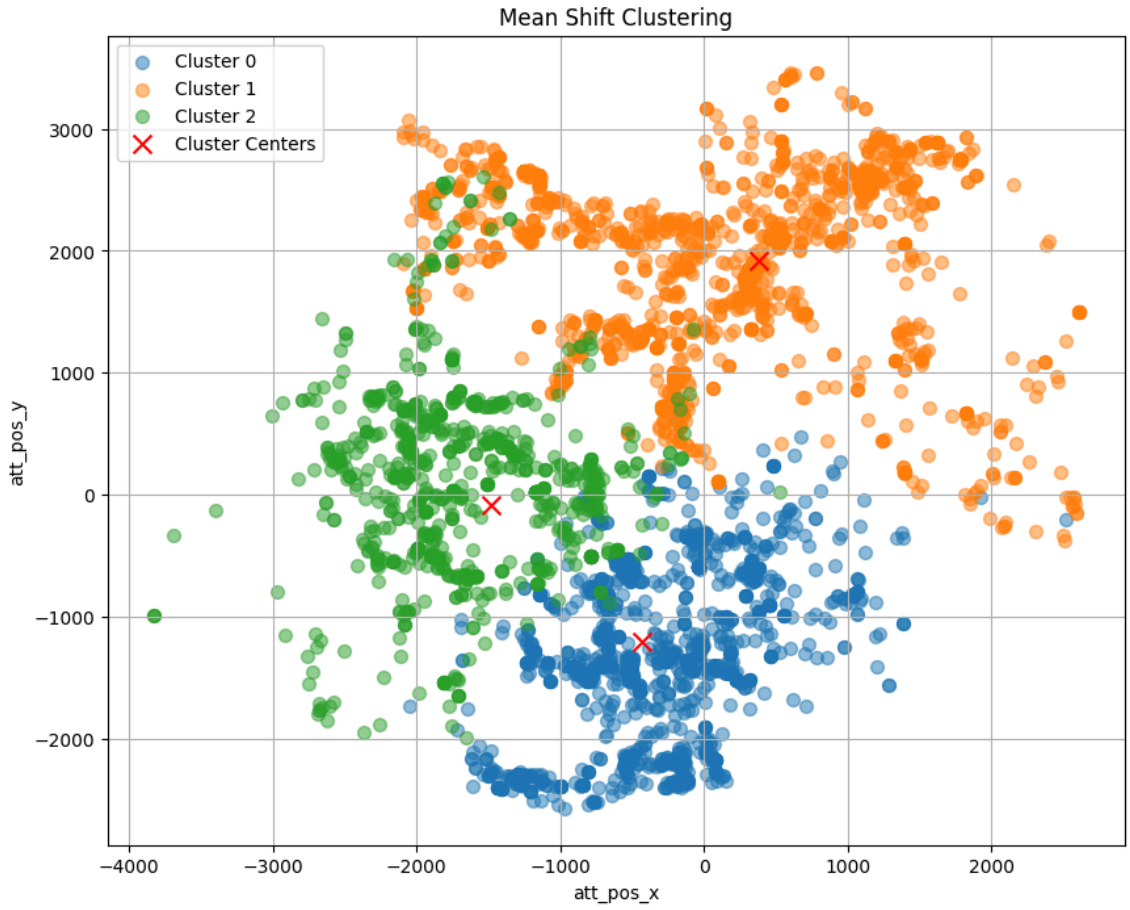
- Cluster Assignment: Assign data points to clusters based on convergence points.
- Output: Return cluster centers and assign data points to clusters.

In [23]:
```
import pandas as pd
from sklearn.cluster import MeanShift, estimate_bandwidth
import numpy as np

# Load the dataset
dataset = pd.read_csv("/kaggle/input/gren-data/mm_grenades_demos.csv")

# Drop rows with missing values
dataset.dropna(inplace=True)

# Extract relevant features
features = dataset[['att_pos_x', 'att_pos_y', 'vic_pos_x', 'vic_pos_y']]

# Estimate bandwidth using sklearn's built-in function
bandwidth = estimate_bandwidth(features, quantile=0.2, n_samples=500)

# Apply Mean Shift clustering
ms = MeanShift(bandwidth=bandwidth, bin_seeding=True)
ms.fit(features)

# Retrieve cluster centers
cluster_centers = ms.cluster_centers_

# Number of clusters
num_clusters = len(np.unique(ms.labels_))

print(f"Number of clusters: {num_clusters}")
print("Cluster centers:")
print(cluster_centers)
```

```
/opt/conda/lib/python3.10/site-packages/pandas/core/algorithms.py:1743: De
precationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  return lib.map_infer(values, mapper, convert=convert)
/opt/conda/lib/python3.10/site-packages/pandas/core/algorithms.py:1743: De
precationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  return lib.map_infer(values, mapper, convert=convert)
/opt/conda/lib/python3.10/site-packages/pandas/core/algorithms.py:1743: De
precationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  return lib.map_infer(values, mapper, convert=convert)
/opt/conda/lib/python3.10/site-packages/pandas/core/algorithms.py:1743: De
precationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  return lib.map_infer(values, mapper, convert=convert)
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:605: D
eprecationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(pd_dtype):
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:614: D
eprecationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(pd_dtype) or not is_extension_array_dtype(pd_dtype):
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:605: D
eprecationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(pd_dtype):
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:614: D
eprecationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(pd_dtype) or not is_extension_array_dtype(pd_dtype):
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:605: D
eprecationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(pd_dtype):
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:614: D
eprecationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(pd_dtype) or not is_extension_array_dtype(pd_dtype):
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:605: D
eprecationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(pd_dtype):
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:614: D
eprecationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(pd_dtype) or not is_extension_array_dtype(pd_dtype):
/opt/conda/lib/python3.10/site-packages/pandas/core/algorithms.py:1743: De
precationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  return lib.map_infer(values, mapper, convert=convert)
/opt/conda/lib/python3.10/site-packages/pandas/core/algorithms.py:1743: De
precationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  return lib.map_infer(values, mapper, convert=convert)
/opt/conda/lib/python3.10/site-packages/pandas/core/algorithms.py:1743: De
precationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  return lib.map_infer(values, mapper, convert=convert)
/opt/conda/lib/python3.10/site-packages/pandas/core/algorithms.py:1743: De
```

```
precationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  return lib.map_infer(values, mapper, convert=convert)
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:605: D
eprecationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(pd_dtype):
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:614: D
eprecationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(pd_dtype) or not is_extension_array_dtype(pd_dtype):
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:605: D
eprecationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(pd_dtype):
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:614: D
eprecationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(pd_dtype) or not is_extension_array_dtype(pd_dtype):
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:605: D
eprecationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(pd_dtype):
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:614: D
eprecationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(pd_dtype) or not is_extension_array_dtype(pd_dtype):
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:605: D
eprecationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(pd_dtype):
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:614: D
eprecationWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(pd_dtype) or not is_extension_array_dtype(pd_dtype):

Number of clusters: 3
Cluster centers:
[[ -430.79932486 -1203.99102703  -416.10338699 -1345.10221608]
 [  382.031641    1919.88278764   327.8364458   2039.39162312]
 [-1477.26597442   -88.24452227 -1494.25223517  -150.03243306]]
```

In [25]:
```
# Plot the clusters
plt.figure(figsize=(10, 8))

# Plot data points
for cluster_label in set(ms.labels_):
    cluster_points = features[ms.labels_ == cluster_label]
    plt.scatter(cluster_points['att_pos_x'], cluster_points['att_pos_y'], l

# Plot cluster centers
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], c='red', marker='

plt.title('Mean Shift Clustering')
plt.xlabel('att_pos_x')
plt.ylabel('att_pos_y')
plt.legend()
plt.grid(True)
plt.show()
```



INTERPRETATION

We can see that Overlapping points in the clustering shows the grenade position where all the teams playing in the match throws the grenade this overlapping shows that the grenade positioning and overlapping shows that these overlapping positions are the hotspots in the game and the frequency of grenade is really high in this game mode

CUMMULATIVE INTERPRETATION

- Match Duration: The histogram reveals the distribution of match durations on specific maps, indicating the maximum and minimum durations observed. This information helps in understanding the typical length of matches and planning strategies accordingly.
- Player Positioning: By plotting player positions as scatter points on maps like 'de_mirage', 'de_overpass', and 'de_cache', the code visualizes player activity and identifies hotspots, strategic areas, chokepoints, and common routes. This visualization aids in strategic decision-making, route planning, and identifying potential trap locations.
- Bomb Plant Prediction: The KNN algorithm predicts whether a bomb will be planted during gameplay with an accuracy of approximately 85.06%. The confusion matrix further illustrates the model's performance, showing true positives, true negatives, false positives, and false negatives, enabling a deeper understanding of the predictive capability.
- Association Rules Analysis: Association rules between CT (CounterTerrorist) equipment values and various game aspects, such as round types and sides, provide insights into the relationship between equipment value and gameplay dynamics. For example, high CT equipment value is associated with normal rounds and playing on the CounterTerrorist side.
- Grenade Positioning: The clustering of grenade positions highlights hotspots where teams frequently throw grenades. Overlapping points indicate areas with high grenade frequency, suggesting key strategic locations and potential combat zones.

CONCLUSION

The game mode that was focused on was bomb planting and this was a cummulative interpretation using KNN we found out all the hotzones in the maps and then identified the method and strategy preffered in the maps which was trying to win the game without placing any bombs and eliminating the opponent players is the most preferred startegy in the dataset it was done using Logistic regression and then we found out how much the equipments affect the win ratio of the game and surprisingly compared to other studies we can see the rate of effect is slightly more in this dataset as there was extra focus on the round type and the wepons used and credits held in game and even though according to ML standards these values maybe considered insignificant but this can help to understand the association of the wepons and round type also plays a significant role and helps us understand the amount of skill needed for a player.The final finding was using Mean shift clustering which was used to find the overlapping positions on the grenade usage and the result was that there was lots of overlapping between the teams and we can say that these places are where friendly fire can also take place and the amount of grenade usage is really more compared to other game modes.