Tabular Kaggle Project

Guideline for steps for the Kaggle Tabular Project. You will "turn in" a GitHub repository, modeled after Project Template on the day of the final, Wednesday, Dec 11 at 11 – 1:30 p.m. During the final period we will have about 5 minutes to go over your project and your results.

You can find a list of possible Tabular datasets here on Excel File in Teams. You are not limited to these datasets. If you find a Kaggle challenge not listed that you would like to attempt, please go check with Dr. Farbin to make sure it is viable.

This notebook outlines the steps you should follow. The file(s) in the GitHub repository should contain these steps. Note that you will be only considering classification projects.

Define Project

- Provide Project link.
- Short paragraph describing the challenge.
- Briefly describe the data.

Project link: https://www.kaggle.com/competitions/widsdatathon2024-challenge1/rules

Paragraph: The task is to predict whether a tumor is malignant or benign (metastatic cancer diagnosis) based on a set of features. This is a binary classification problem where the goal is to use the provided features to classify the tumors accurately. Data Description: We are using a real-world evidence dataset from Health Verity (HV), one of the largest healthcare data ecosystems in the US, as the main data source . The features include patient ID,patient race, patient age and patient gender and bmi.

Data Loading and Initial Look

- Load the data.
- Count the number of rows (data points) and features.
- Any missing values?
- Make a table, where each row is a feature or collection of features:
    - Is the feature categorical or numerical
    - What values?
        - e.g. for categorical: "0,1,2"
        - e.g. for numerical specify the range

  - How many missing values
  - Do you see any outliers?
    - Define outlier.
- For classification is there class imbalance?
- What is the target:
  - Classification: how is the target encoded (e.g. 0 and 1)?
  - Regression: what is the range?

```
from google.colab import files

uploaded=files.upload()
```

Choose Files train.csv

- **train.csv**(text/csv) - 61194 bytes, last modified: 12/10/2024 - 100% done

```
Saving train.csv to train.csv
```

```
import pandas as pd

# Assuming the dataset is a CSV file
df = pd.read_csv("train.csv")

# Display the first few rows of the dataset
df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833

Next steps: Generate code with df View recommended plots New interactive sheet

```
from google.colab import files

uploaded=files.upload()
```

Choose Files test.csv

- **test.csv**(text/csv) - 58872301 bytes, last modified: 12/10/2024 - 100% done

```
Saving test.csv to test.csv
```

```
import pandas as pd

# Assuming the dataset is a CSV file
df = pd.read_csv("test.csv")

# Display the first few rows of the dataset
df.head()
```

	id	cat0	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	...	cont4	cont5	c
0	0	B	B	B	C	B	B	A	E	E	...	0.476739	0.376350	0.33
1	5	A	B	A	C	B	C	A	E	C	...	0.285509	0.860046	0.79
2	15	B	A	A	A	B	B	A	E	D	...	0.697272	0.683600	0.40
3	16	B	B	A	C	B	D	A	E	A	...	0.719306	0.777890	0.73
4	17	B	B	A	C	B	C	A	E	C	...	0.313032	0.431007	0.39

5 rows × 25 columns

```
# Count the number of rows and features
df.shape
```

```
(200000, 25)
```

```
# Check for missing values in each column
df.isnull().sum()
```

	0
id	0
cat0	0
cat1	0
cat2	0
cat3	0
cat4	0
cat5	0
cat6	0
cat7	0
cat8	0
cat9	0
cont0	0
cont1	0
cont2	0
cont3	0
cont4	0
cont5	0
cont6	0
cont7	0
cont8	0
cont9	0
cont10	0
cont11	0
cont12	0
cont13	0

dtype: int64

```
# Creating a summary table for features
feature_info = {
    "Feature": df.columns,
    "Type": ["Categorical" if df[col].dtype == 'object' else "Numerical" for col in df.colum
```

```
        "Value Range / Categories": [df[col].unique() if df[col].dtype == 'object' else df[col].
        "Missing Values": [df[col].isnull().sum() for col in df.columns]
    }

    # Convert into a DataFrame for better visualization
    feature_table = pd.DataFrame(feature_info)
    feature_table
```

	Feature	Type	Value Range / Categories	Missing Values
0	id	Numerical	count 200000.000000 mean 249970.884580 ...	0
1	cat0	Categorical	[B, A]	0
2	cat1	Categorical	[B, A]	0
3	cat2	Categorical	[B, A]	0
4	cat3	Categorical	[C, A, D, B]	0
5	cat4	Categorical	[B, C, D, A]	0
6	cat5	Categorical	[B, C, D, A]	0
7	cat6	Categorical	[A, B, C, I, D, H, E, G]	0
8	cat7	Categorical	[E, G, B, D, F, A, C, I]	0
9	cat8	Categorical	[E, C, D, A, G, F, B]	0
10	cat9	Categorical	[I, H, K, N, F, G, A, J, M, B, L, C, O, E, D]	0
11	cont0	Numerical	count 200000.000000 mean 0.526858 ...	0
12	cont1	Numerical	count 200000.000000 mean 0.460882 ...	0
13	cont2	Numerical	count 200000.000000 mean 0.491686 ...	0
14	cont3	Numerical	count 200000.000000 mean 0.496263 ...	0
15	cont4	Numerical	count 200000.000000 mean 0.492200 ...	0
16	cont5	Numerical	count 200000.000000 mean 0.509944 ...	0
17	cont6	Numerical	count 200000.000000 mean 0.468050 ...	0
18	cont7	Numerical	count 200000.000000 mean 0.537617 ...	0
19	cont8	Numerical	count 200000.000000 mean 0.497587 ...	0
20	cont9	Numerical	count 200000.000000 mean 0.474630 ...	0
21	cont10	Numerical	count 200000.000000 mean 0.473625 ...	0
22	cont11	Numerical	count 200000.000000 mean 0.473589 ...	0
23	cont12	Numerical	count 200000.000000 mean 0.492756 ...	0
24	cont13	Numerical	count 200000.000000 mean 0.508303 ...	0

Next steps: Generate code with `feature_table` View recommended plots New interactive sheet

An outlier is an observation or data point that significantly differs from other observations in a dataset.

```
# Check the column names of the dataset
print(df.columns)
```

```
Index(['id', 'cat0', 'cat1', 'cat2', 'cat3', 'cat4', 'cat5', 'cat6', 'cat7',
       'cat8', 'cat9', 'cont0', 'cont1', 'cont2', 'cont3', 'cont4', 'cont5',
       'cont6', 'cont7', 'cont8', 'cont9', 'cont10', 'cont11', 'cont12',
       'cont13'],
      dtype='object')
```

```
# Strip any leading/trailing spaces in the column names
df.columns = df.columns.str.strip()

# Check again for the presence of the 'signal' column
print(df.columns)
```

```
Index(['id', 'cat0', 'cat1', 'cat2', 'cat3', 'cat4', 'cat5', 'cat6', 'cat7',
       'cat8', 'cat9', 'cont0', 'cont1', 'cont2', 'cont3', 'cont4', 'cont5',
       'cont6', 'cont7', 'cont8', 'cont9', 'cont10', 'cont11', 'cont12',
       'cont13'],
      dtype='object')
```

```
import numpy as np
from scipy import stats

# Calculate Z-scores for numerical columns
numerical_cols = df.select_dtypes(include=[np.number]).columns
z_scores = stats.zscore(df[numerical_cols].dropna())

# Define a threshold for outliers (usually 3 or higher)
outliers = np.abs(z_scores) > 3
print(outliers)
```

```
           id  cont0  cont1  cont2  cont3  cont4  cont5  cont6  cont7  cont8  \
0       False  False  False  False  False  False  False  False  False  False   
1       False  False  False  False  False  False  False  False  False  False   
2       False  False  False  False  False  False  False  False  False  False   
3       False  False  False  False  False  False  False  False  False  False   
4       False  False  False  False  False  False  False  False  False  False   
...       ...    ...    ...    ...    ...    ...    ...    ...    ...    ...   
199995  False  False  False  False  False  False  False  False  False  False   
199996  False  False  False  False  False  False  False  False  False  False   
199997  False  False  False  False  False  False  False  False  False  False   
199998  False  False  False  False  False  False  False  False  False  False   
199999  False  False  False  False  False  False  False  False  False  False   

        cont9  cont10  cont11  cont12  cont13  
0       False   False   False   False   False  
1       False   False   False   False   False  
```

```
2       False   False   False   False   False
3       False   False   False   False   False
4       False   False   False   False   False
...       ...     ...     ...     ...     ...
199995  False   False   False   False   False
199996  False   False   False   False   False
199997  False   False   False   False   False
199998  False   False   False   False   False
199999  False   False   False   False   False

[200000 rows x 15 columns]
```

```
# Check all column names again
print(df.columns)

# Check the data types of each column to see if there are any categorical columns
print(df.dtypes)
```

```
Index(['id', 'cat0', 'cat1', 'cat2', 'cat3', 'cat4', 'cat5', 'cat6', 'cat7',
       'cat8', 'cat9', 'cont0', 'cont1', 'cont2', 'cont3', 'cont4', 'cont5',
       'cont6', 'cont7', 'cont8', 'cont9', 'cont10', 'cont11', 'cont12',
       'cont13'],
      dtype='object')
id          int64
cat0       object
cat1       object
cat2       object
cat3       object
cat4       object
cat5       object
cat6       object
cat7       object
cat8       object
cat9       object
cont0     float64
cont1     float64
cont2     float64
cont3     float64
cont4     float64
cont5     float64
cont6     float64
cont7     float64
cont8     float64
cont9     float64
cont10    float64
cont11    float64
cont12    float64
cont13    float64
dtype: object
```

```
# Assuming 'cat0' is the target column
class_distribution = df['cat0'].value_counts()
print(class_distribution)
```

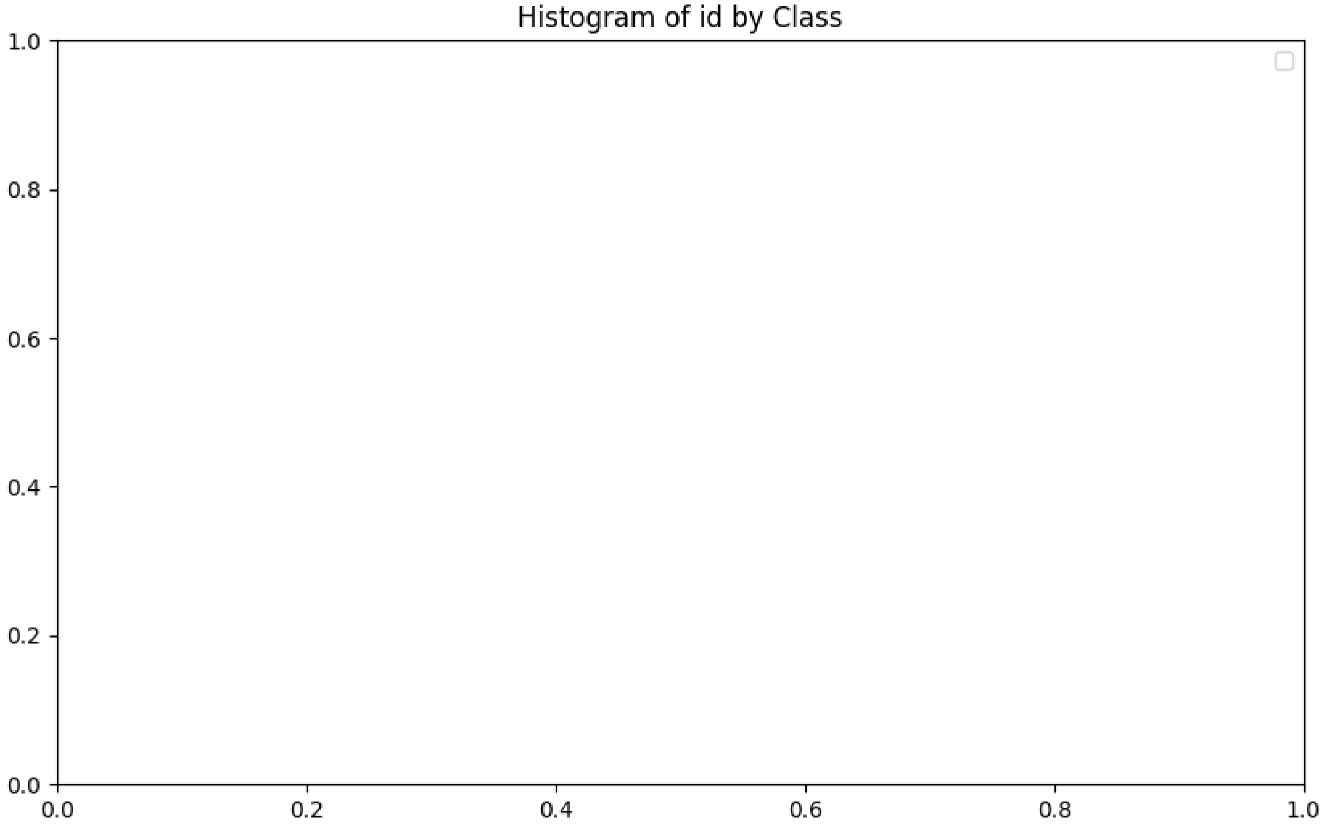
```
cat0
A    128830
B     71170
Name: count, dtype: int64
```

Data Visualization

- For classification: compare histogram every feature between the classes. Lots of examples of this in class.
- For regression:
    - Define 2 or more class based on value of the regression target.
        - For example: if regression target is between 0 and 1:
            - 0.0-0.25: Class 1
            - 0.25-0.5: Class 2
            - 0.5-0.75: Class 3
            - 0.75-1.0: Class 4
    - Compare histograms of the features between the classes.
- Note that for categorical features, often times the information in the histogram could be better presented in a table.
- Make comments on what features look most promising for ML task.

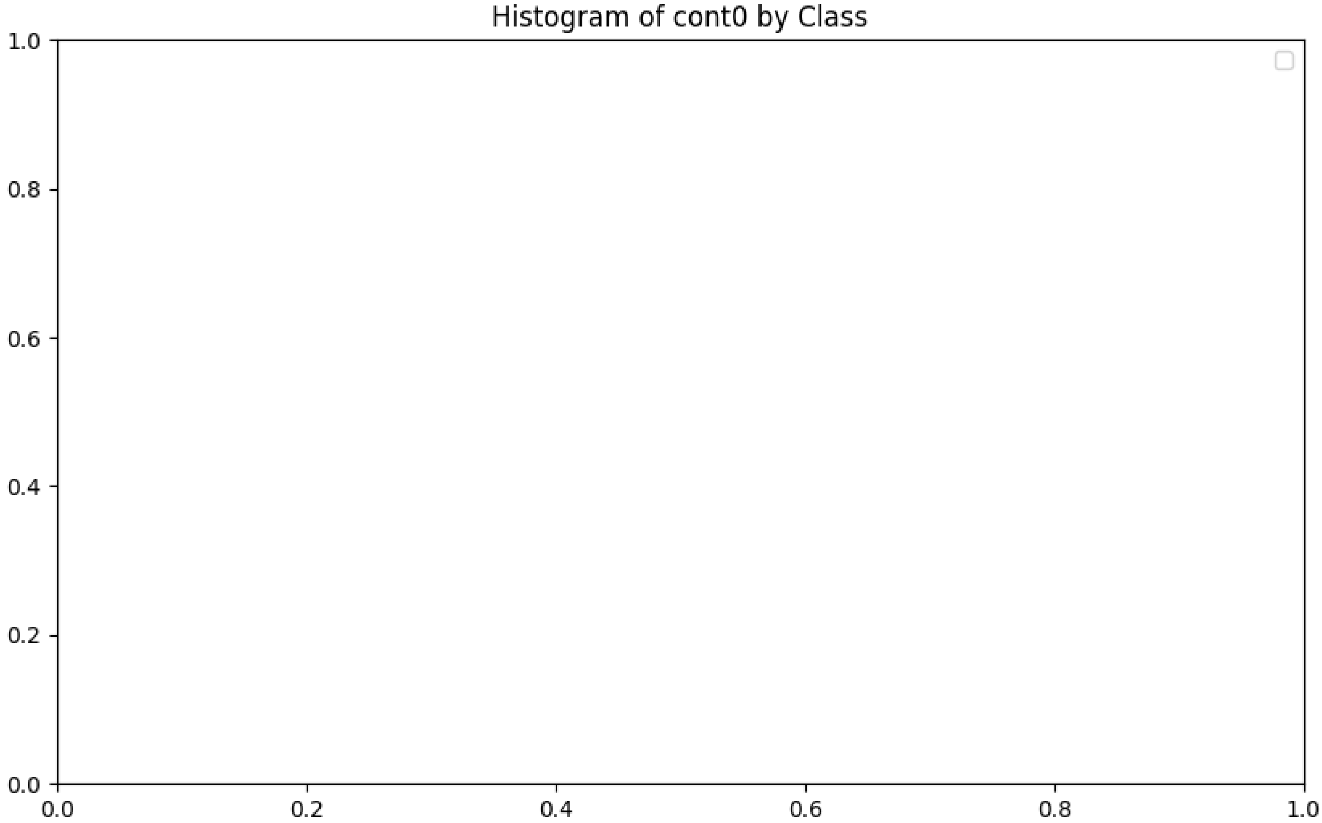
```
import matplotlib.pyplot as plt
import seaborn as sns

# Separate the data by target class (assuming target column is 'cat0')
class_0 = df[df['cat0'] == 0]
class_1 = df[df['cat0'] == 1]

# Plot histograms for each numerical feature
numerical_columns = df.select_dtypes(include=['float64', 'int64']).columns

for column in numerical_columns:
    plt.figure(figsize=(10, 6))
    sns.histplot(class_0[column], kde=True, color='blue', label='Class 0', bins=30)
    sns.histplot(class_1[column], kde=True, color='red', label='Class 1', bins=30)
    plt.title(f'Histogram of {column} by Class')
    plt.legend()
    plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that arti
```



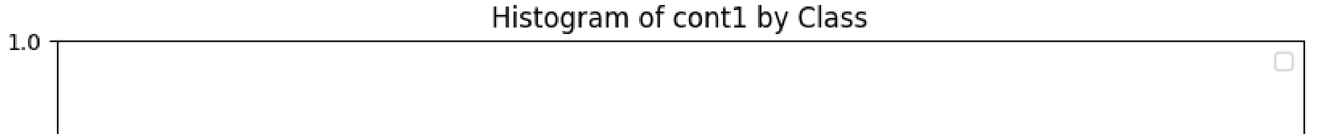
Histogram of id by Class

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that arti
```

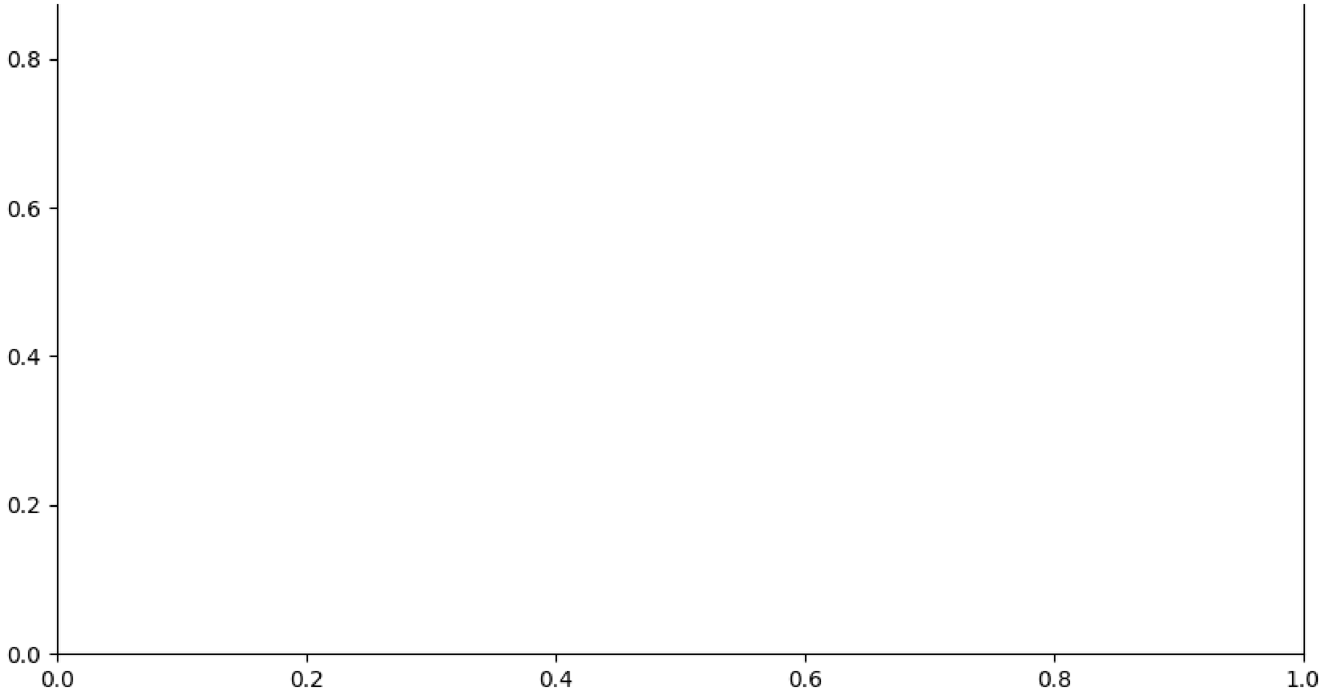


Histogram of cont0 by Class

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that arti
```

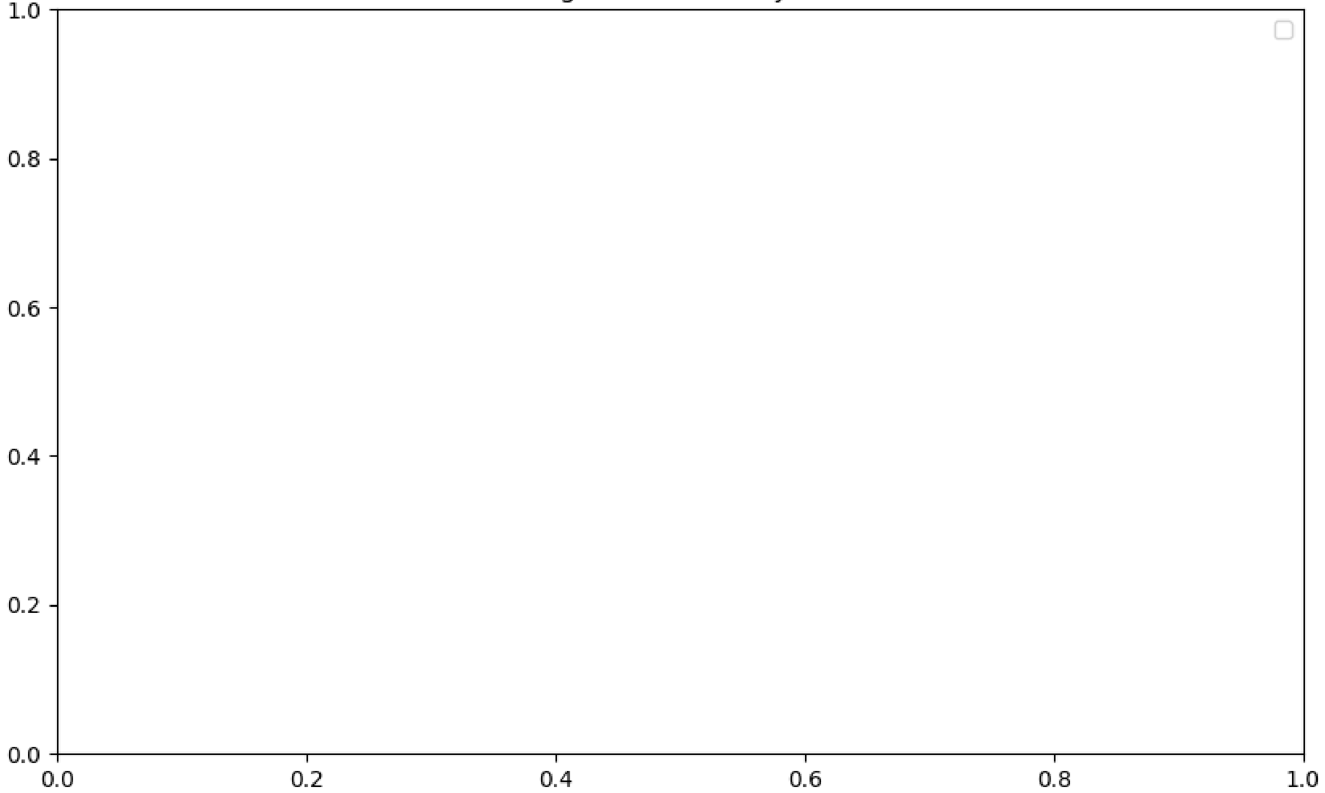


Histogram of cont1 by Class



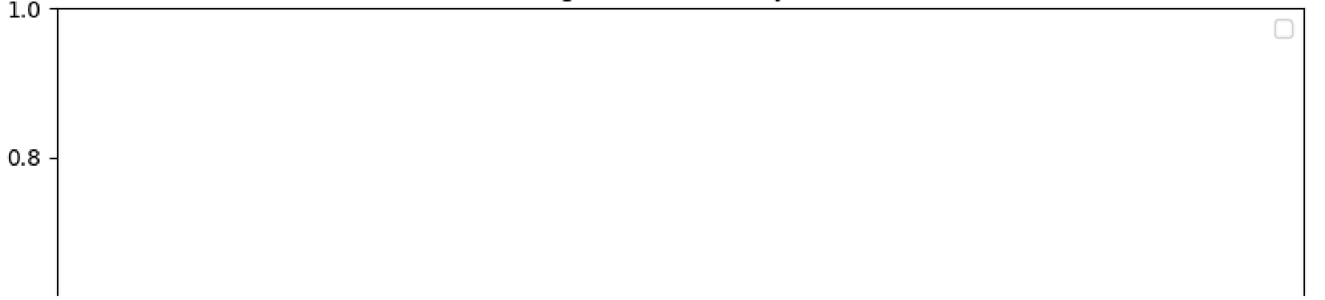
```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that arti
```

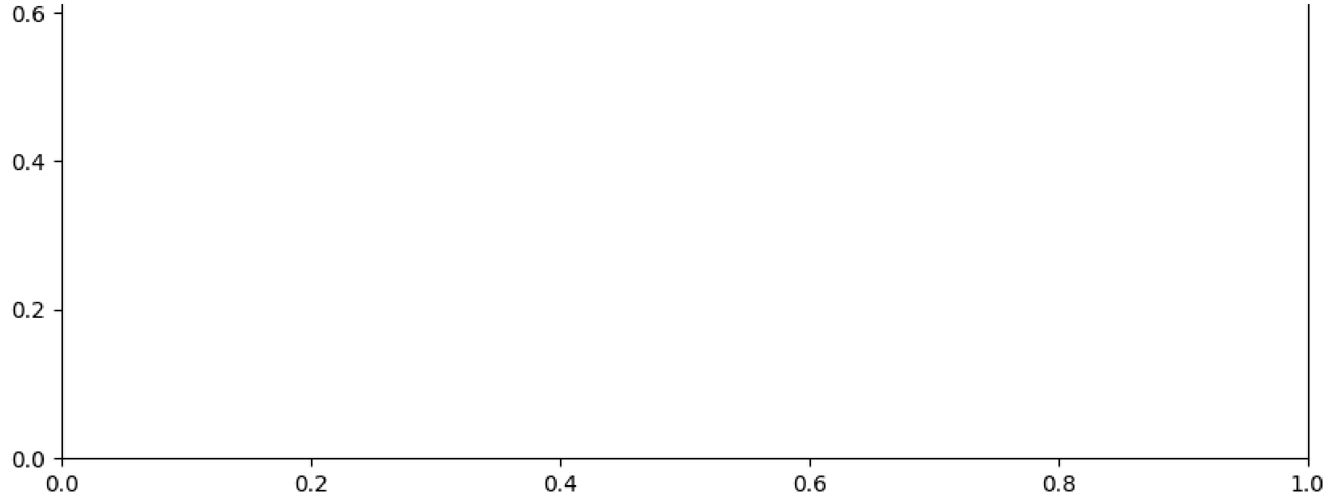
Histogram of cont2 by Class



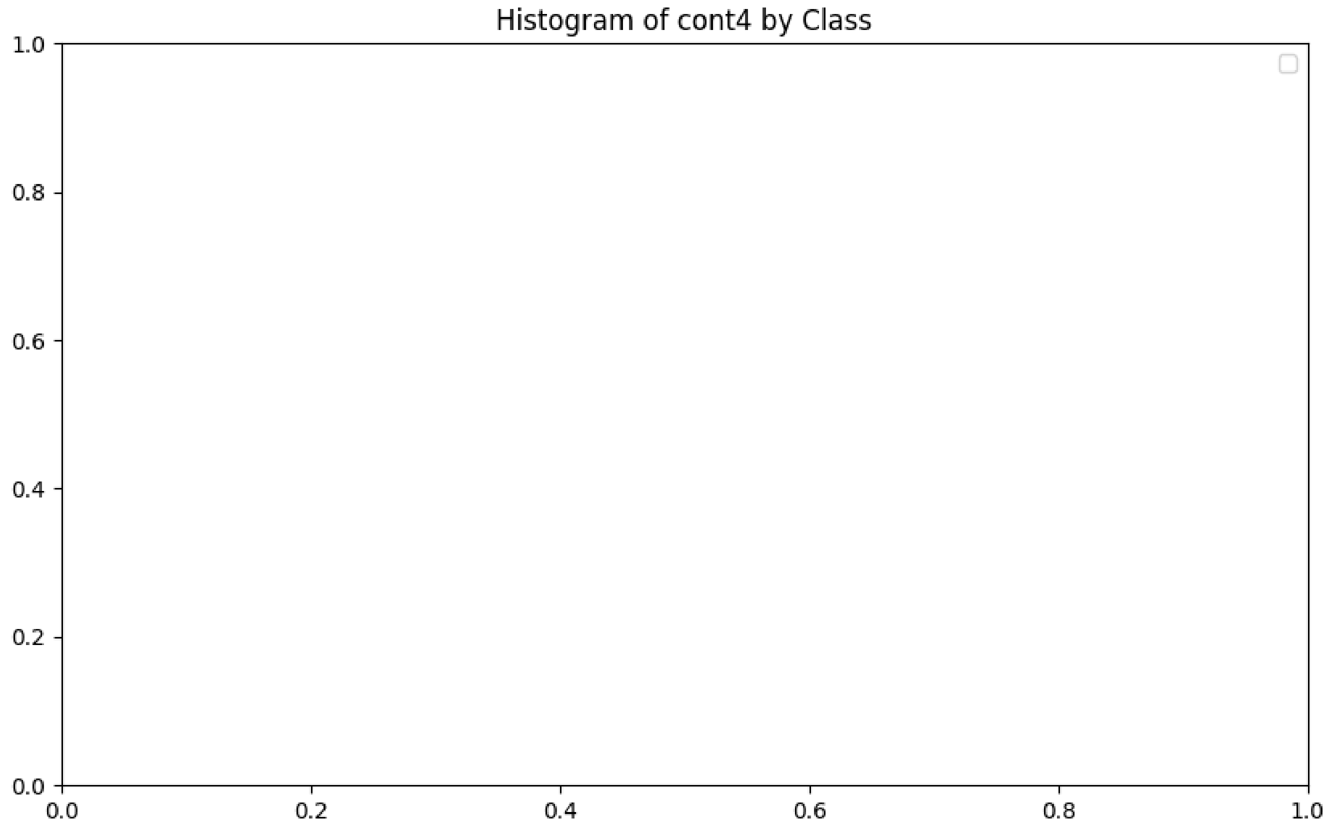
```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that arti
```

Histogram of cont3 by Class

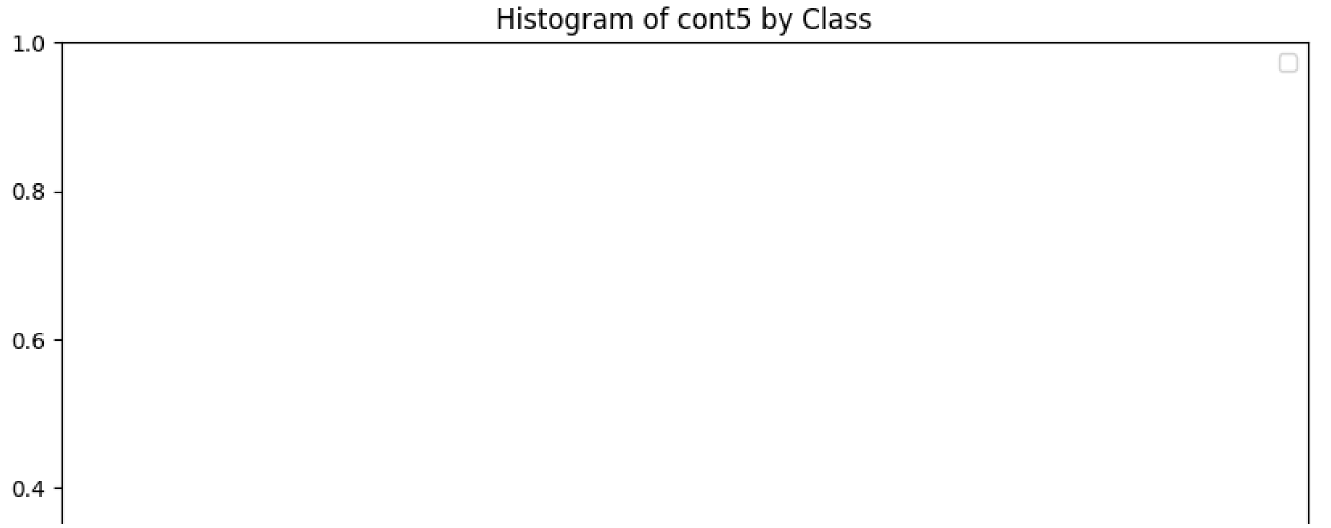


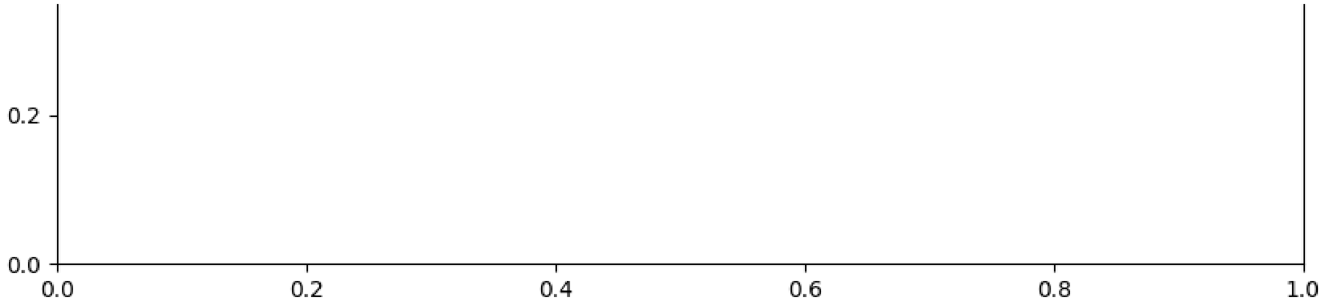


```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that arti
```

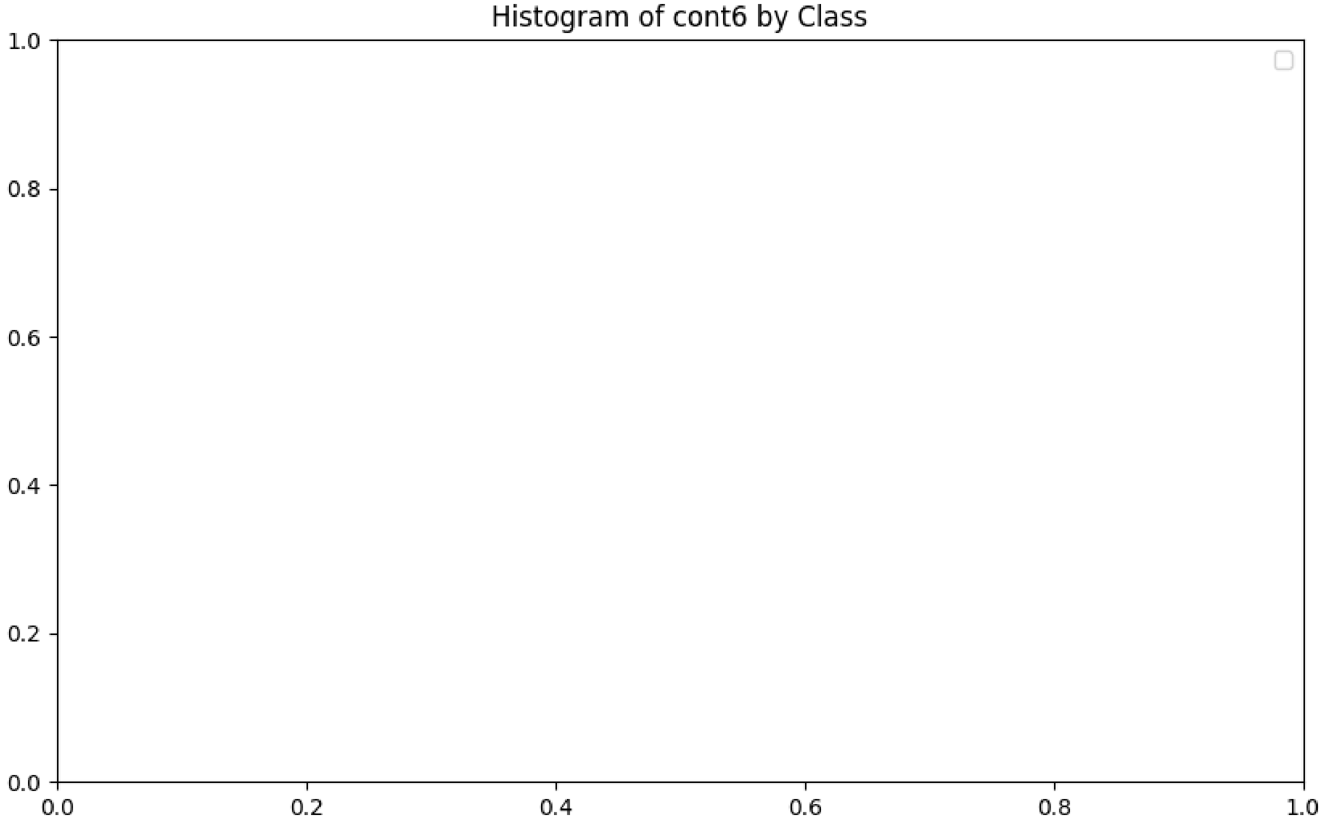


```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that arti
```

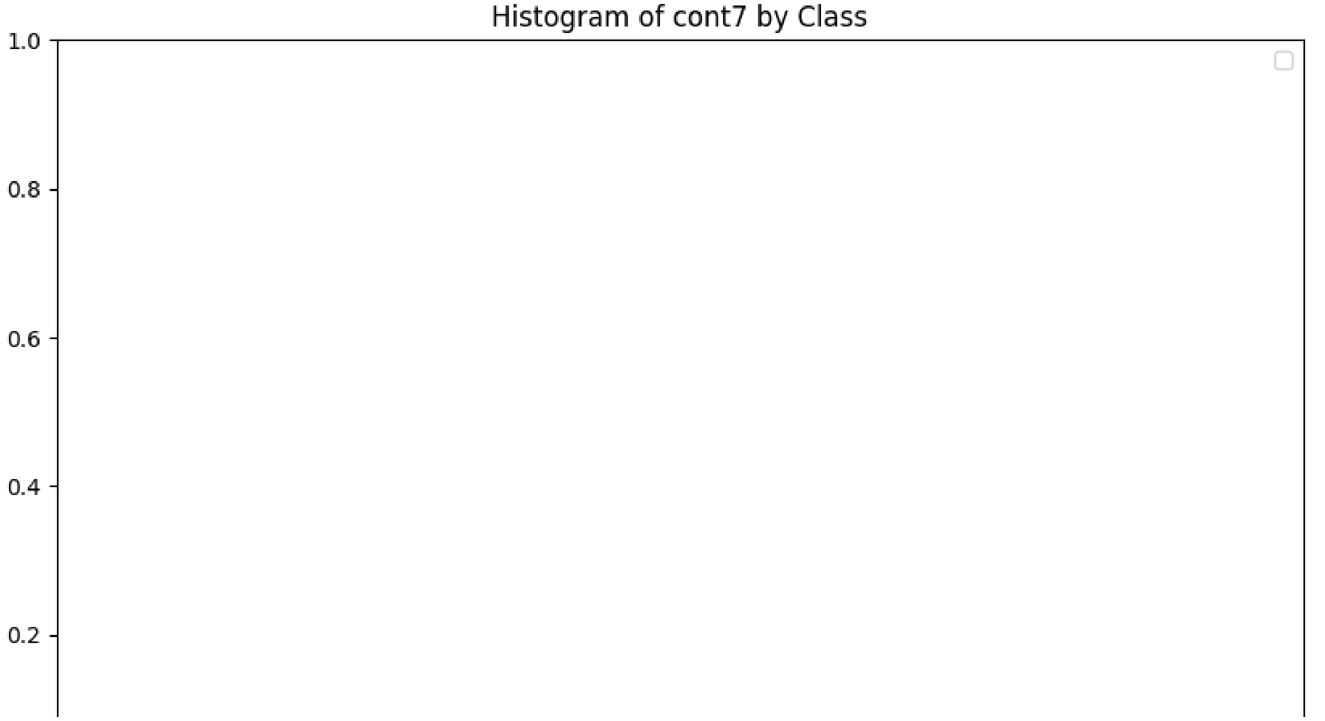


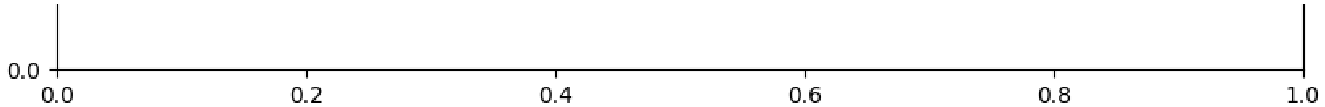


```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that arti
```

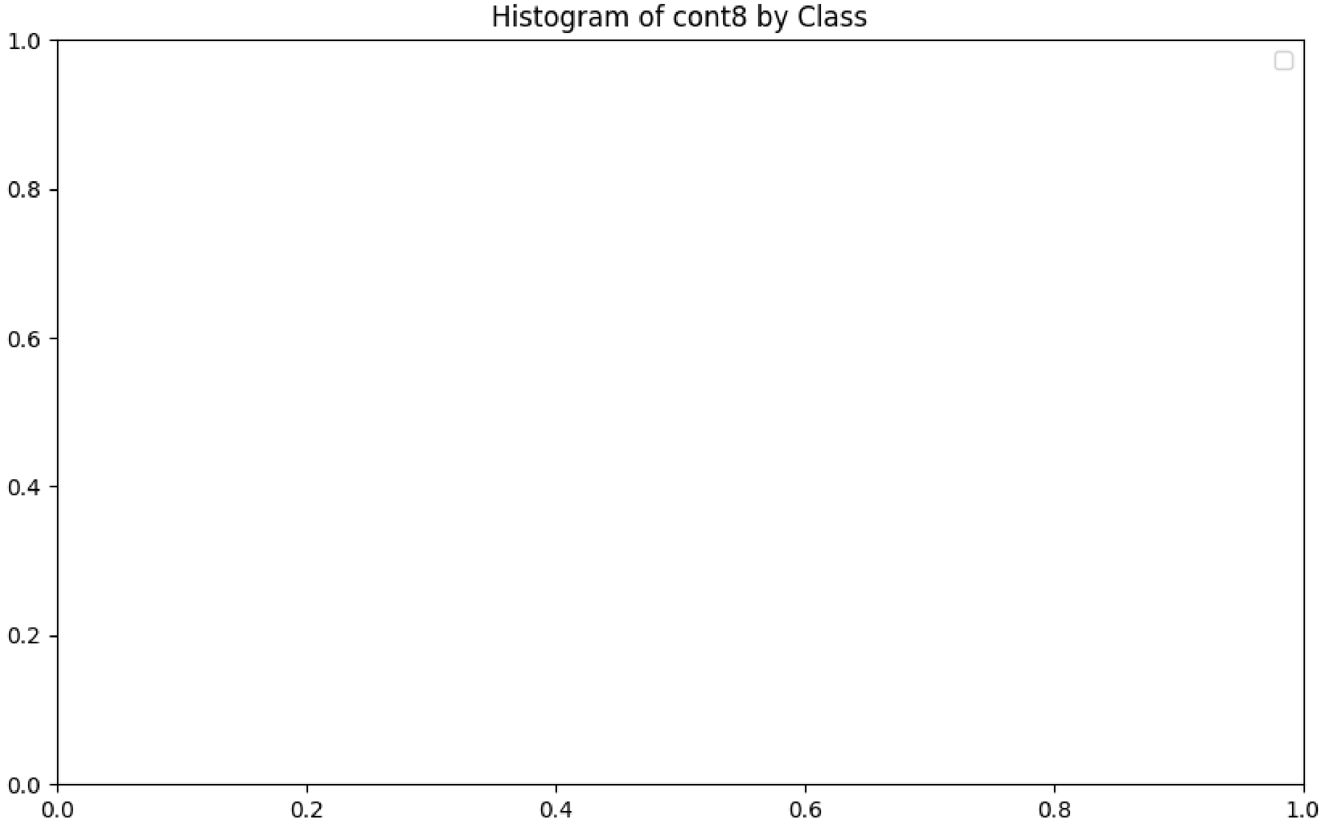

Histogram of cont6 by Class

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that arti
```

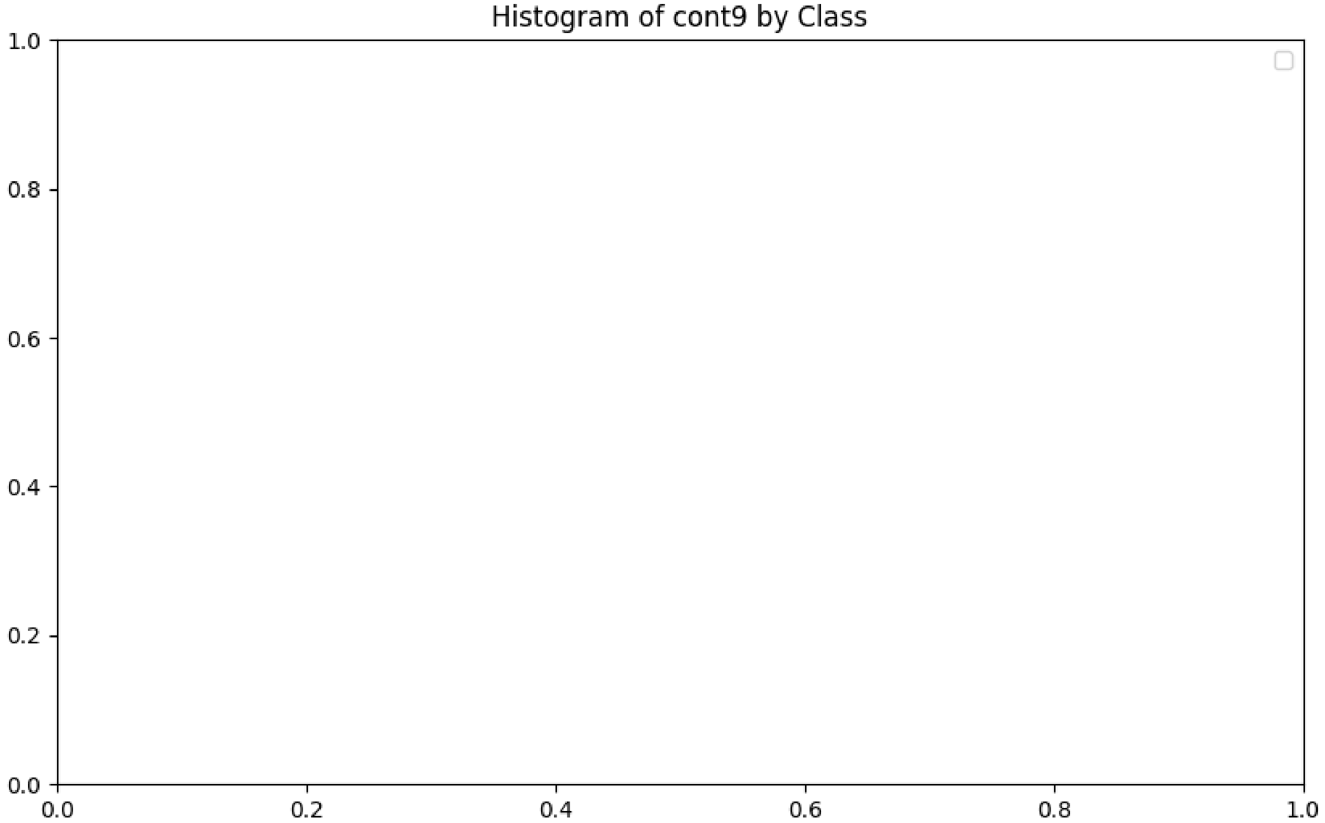

Histogram of cont7 by Class



```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that arti
```

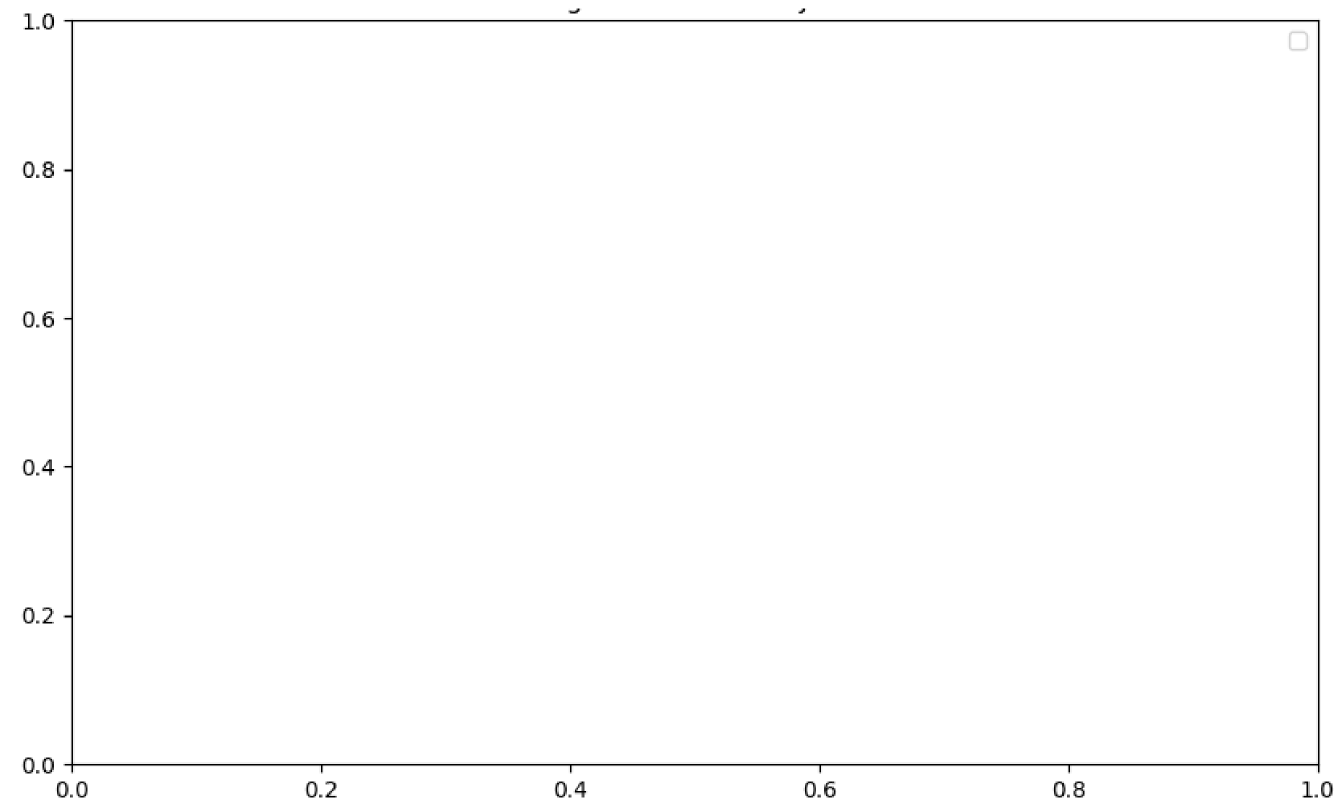


```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that arti
```



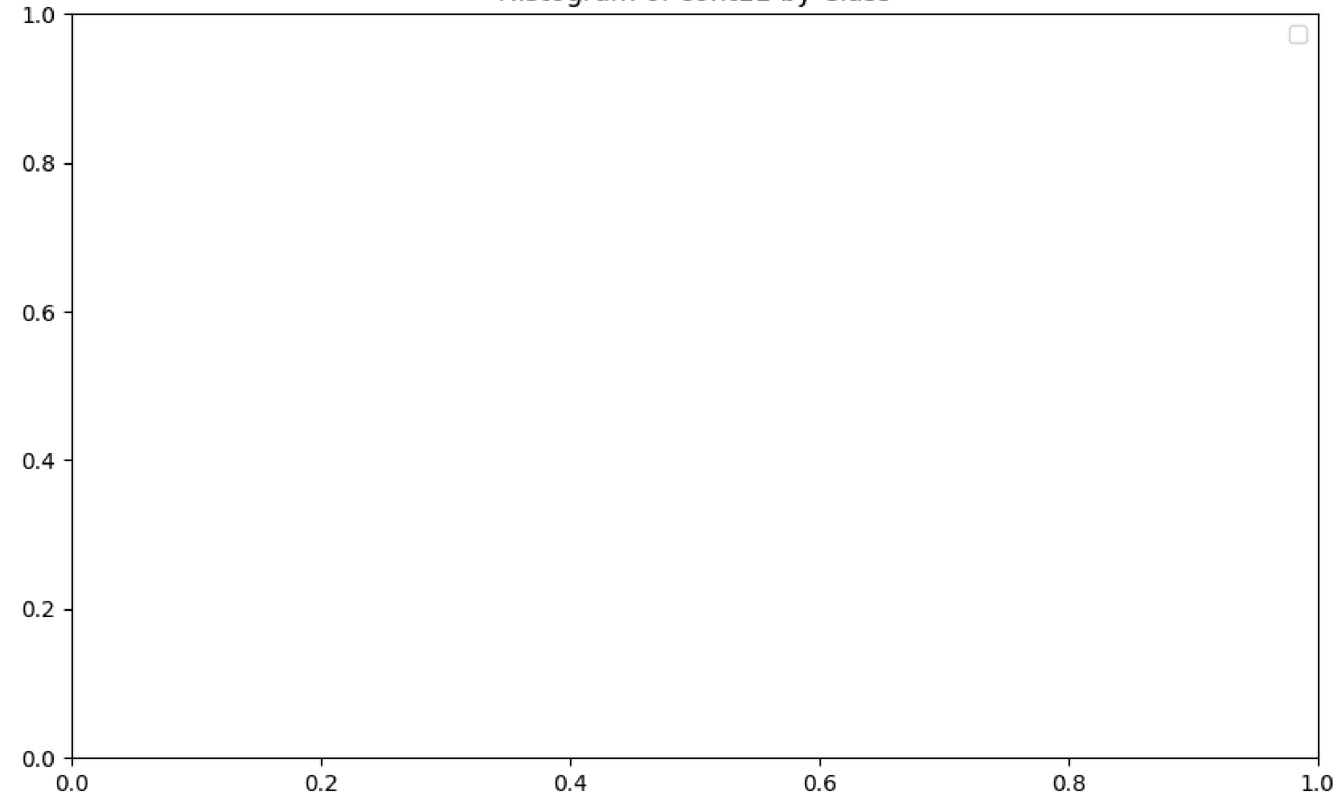
```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that arti
```

Histogram of cont10 by Class



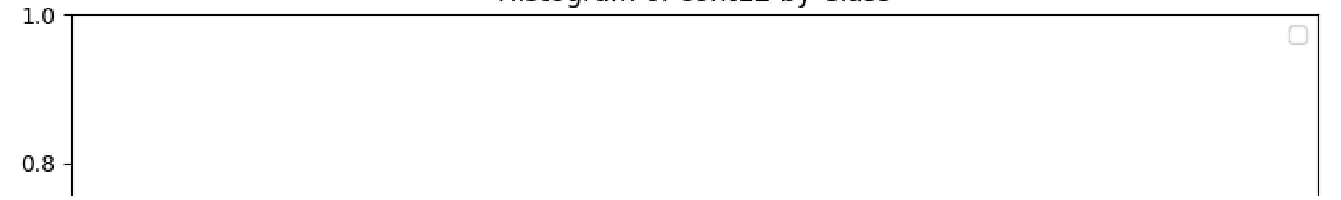
```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that arti
```

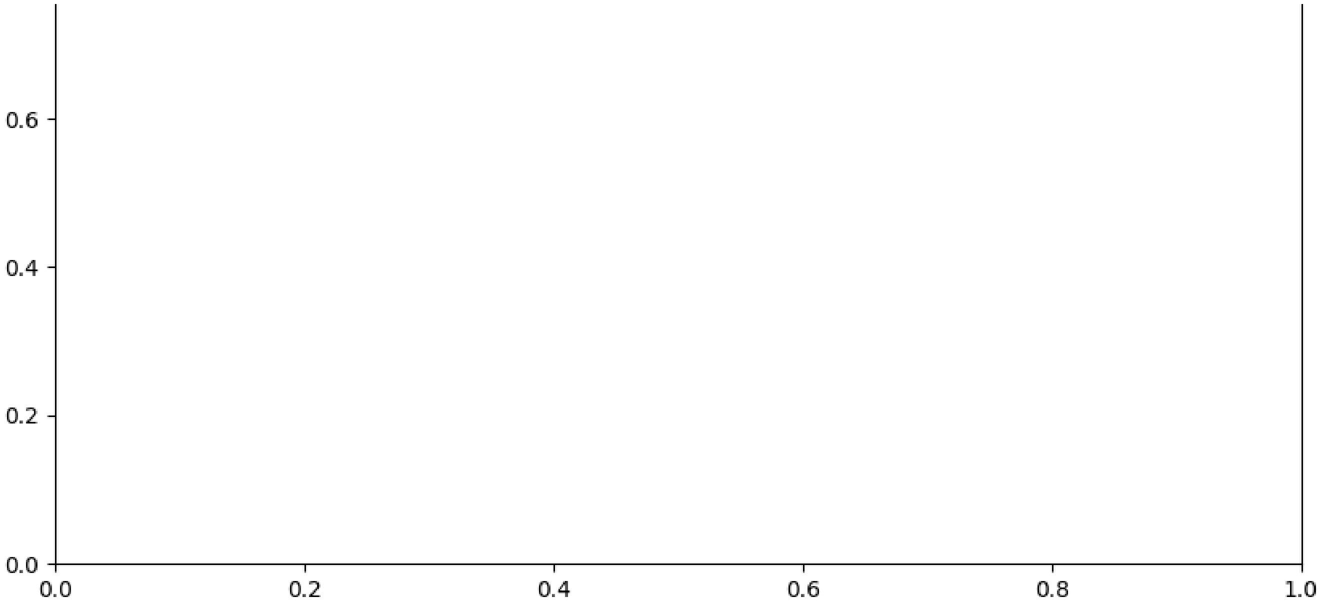
Histogram of cont11 by Class



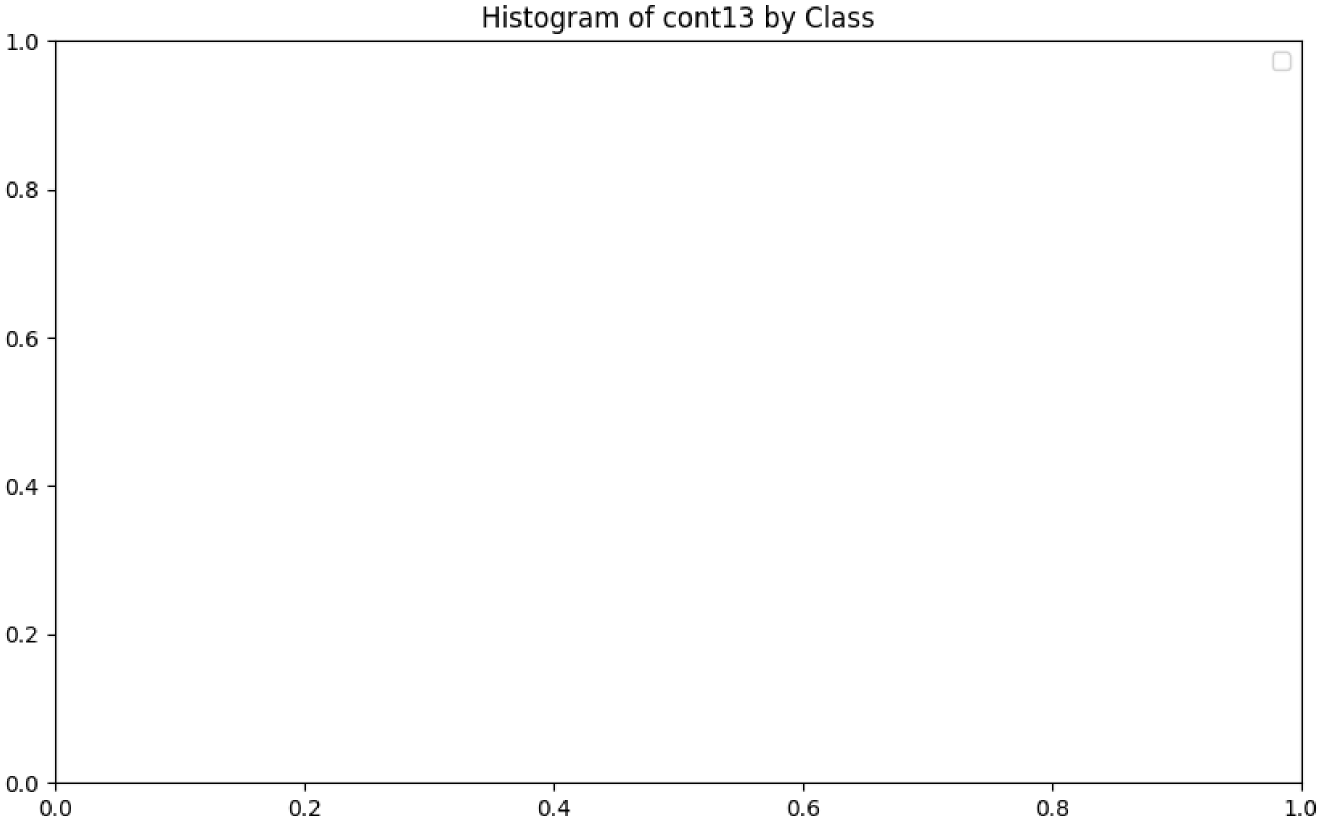
```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that arti
```

Histogram of cont12 by Class





```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that arti
```



```
# Categorical features columns
categorical_columns = df.select_dtypes(include=['object']).columns

for column in categorical_columns:
    plt.figure(figsize=(10, 6))
    sns.countplot(x=column, hue='cat0', data=df)
    plt.title(f'Distribution of {column} by Class')
    plt.show()
```