

AI Usage Declaration Form

Referencing the Output of Artificial Intelligence Language Models

Output text of an artificial intelligence language model (such as a ChatBot) for written assessment items of this unit will not be treated as your own work.

- If any output text is used, it must be quoted in italics, and the app or website must be referenced appropriately (AI language model Name/Chatbot Name, Developer or Organisation, date accessed).*
- If any output ideas are used and paraphrased, the app or website must be referenced appropriately (AI language model Name/Chatbot Name, Developer or Organisation, date accessed).*
- If the output of an artificial intelligence language model is prescribed by your Lecturer as an integral part of an assessment item, the app or website must be referenced appropriately (AI language model Name/Chatbot Name, Developer or Organisation, date accessed).*

(excerpt from Unit Information)

Step 1: Select the declaration statement that best describes the level of AI tool usage in this assignment.

Table 1. Declaration Statement

Tick (✓) only 1 statement that applies	Declaration statements
	Declaration 1: We do not use any content generated by AI tools to be presented as our work.
	Declaration 2: We acknowledge the use of AI tool(s) to generate materials that aid with the process of completing this assessment (e.g. for background research or self-study). None of these materials are presented as our work.
✓	Declaration 3: We acknowledge the use of AI tool(s) to generate materials that are included in our work in modified forms.

Step 2: Complete the following form if you selected either Declaration 2 or Declaration 3. You do not need to complete this section if you select Declaration 1.

Table 2. Details of AI Usage

Reference to one or more AI tools. Example: https://shorturl.at/8UT25	Response: AI tools used: ChatGpt 5
--	---

	<div>llama 4 Scout</div> <div>GitHub Co-pilot</div>
<div>List all prompts you have used.</div>	<div>Response:</div> <div><div>Duck.ai</div><div>Prompt 1: what does update layout function do line by line</div><div><div><div>what does update layout function do line by line</div><div><div>GPT-4o mini</div><div>The <code>update_layout</code> function in Plotly is used to modify the layout properties of a figure. Here's a breakdown of what each line does in the context of the provided code:</div><div><div><div><> python</div><div>Copy Code</div></div><pre>fig.update_layout(title='Distribution of bat_landing_to_food with Thresholds 1 to 10', xaxis_title='bat_landing_to_food', yaxis_title='Probability Density', barmode='overlay', template='plotly_white')</pre></div><div><div>Line-by-Line Breakdown:</div><div><div>1. <code>title='Distribution of bat_landing_to_food with Thresholds 1 to 10'</code> :<ul style="list-style-type: none">This line sets the main title of the plot. It appears at the top of the figure and provides context about what the plot represents.</div><div><div>2. <code>xaxis_title='bat_landing_to_food'</code> :<ul style="list-style-type: none">This line sets the title for the x-axis. It labels the x-axis with "bat_landing_to_food," indicating what data is being represented along this axis.</div><div><div>3. <code>yaxis_title='Probability Density'</code> :<ul style="list-style-type: none">This line sets the title for the y-axis. It labels the y-axis with "Probability Density," which describes the type of measurement being represented on this axis.</div><div><div>4. <code>barmode='overlay'</code> :<ul style="list-style-type: none">This line specifies how the bars in the histogram should be displayed. Setting <code>barmode</code> to <code>'overlay'</code> means that the bars for different traces (in this case, different thresholds) will be overlaid on top of each other rather than stacked. This allows for easier comparison of distributions.</div><div><div>5. <code>template='plotly_white'</code> :<ul style="list-style-type: none">This line applies a predefined template to the figure. The <code>'plotly_white'</code> template provides a clean, white background for the plot, enhancing visibility and aesthetics. It also applies default styling for fonts, colors, and grid lines.</div></div></div></div></div></div></div></div></div></div></div>

Prompt 2: and describe add trace function

and describe add trace function

GPT-4o mini

The `add_trace` function in Plotly is used to add a new data series (or trace) to a figure. Each trace represents a different set of data or a different type of visualization. In the context of the provided code, the `add_trace` function is used to add histograms for the distribution of the `bat_landing_to_food` values. Here's a breakdown of how it works:

```
pythonfig.add_trace(go.Histogram(  
    x=x,  
    name=f'Threshold = {threshold}',  
    opacity=0.75,  
    histnorm='probability density',  
    marker=dict(line=dict(width=1))  
))
```

Copy Code

Breakdown of Parameters:

- `go.Histogram(...)`:
 - This specifies that the trace being added is a histogram. The `go` module refers to the `plotly.graph_objects` library, which provides a way to create various types of plots.
- `x=x`:
 - This parameter defines the data to be plotted on the x-axis of the histogram. In this case, `x` is the data from the `bat_landing_to_food` column of the DataFrame. Each value in `x` will contribute to the histogram's bars.
- `name=f'Threshold = {threshold}'`:
 - This sets the name of the trace, which will appear in the legend of the plot. The `f'Threshold = {threshold}'` syntax uses an f-string to dynamically insert the current threshold value into the name. This helps differentiate the histograms for each threshold in the overlay.
- `opacity=0.75`:
 - This parameter controls the transparency of the bars in the histogram. An opacity of `0.75` means the bars will be slightly transparent, allowing overlapping bars from different traces to be visible. This is particularly useful when multiple histograms are overlaid.
- `histnorm='probability density'`:
 - This parameter specifies how the histogram should be normalized. Setting `histnorm` to `'probability density'` means that the area under the histogram will sum to 1, allowing for a comparison of distributions regardless of the number of data points. This is useful for visualizing the relative frequency of values.
- `marker=dict(line=dict(width=1))`:
 - This parameter customizes the appearance of the bars in the histogram. The `marker` dictionary allows you to define properties of the bars. In this case, `line=dict(width=1)` specifies that the outline of each bar should have a width of 1 unit, enhancing visibility and separation between bars.

	<p>Prompt 3:</p> <pre>import numpy as np from scipy.stats import binomtest # Initialize an array to store p-values p_values = [] # Loop through threshold values from 1 to 10 for threshold in range(1, 5): # Create a new column based on the current threshold df5['above_seconds'] = (df5['bat_landing_to_food'] > threshold).astype(int) # Count how many 1s successes = df5['above_seconds'].sum() # Total number of trials n = len(df5) # Perform one-sided binomial test result = binomtest(successes, n, p=0.5, alternative='greater') # Store the p-value in the array p_values.append(result.pvalue.round(10)) # Print the results for the current threshold print(f"Threshold: {threshold}, Number of ones: {successes}, Number of trials: {n}, p-value: {result.pvalue.round(10)}") # Convert p_values to a NumPy array for further analysis if needed p_values_array = np.array(p_values)</pre> <p>can you make it increment it by 0.2</p>
--	---

can you make it increment it by 0.2

GPT-4o mini

Copy to clipboard

Certainly! To modify the code so that the threshold increments by 0.2 instead of whole numbers, you can use `numpy.arange()` to generate the threshold values. Here's the updated code:

```
python
import numpy as np
from scipy.stats import binomtest

# Initialize an array to store p-values
p_values = []

# Loop through threshold values from 1 to 5 in increments of 0.2
for threshold in np.arange(1, 5.2, 0.2): # 5.2 to include 5.0
    # Create a new column based on the current threshold
    dfs['above_seconds'] = (dfs['bat_landing_to_food'] > threshold).astype(int)

    # Count how many 1s
    successes = dfs['above_seconds'].sum()

    # Total number of trials
    n = len(dfs)

    # Perform one-sided binomial test
    result = binomtest(successes, n, p=0.5, alternative='greater')

    # Store the p-value in the array
    p_values.append(result.pvalue.round(10))

    # Print the results for the current threshold
    print(f'Threshold: {threshold:.1f}, Number of ones: {successes}, Number of trials: {n}, p-value: {result.pvalue.round(10)}')

# Convert p_values to a NumPy array for further analysis if needed
p_values_array = np.array(p_values)
```

Key Changes:

- The loop now uses `np.arange(1, 5.2, 0.2)` to generate threshold values starting from 1 to 5 (inclusive) in increments of 0.2.
- The printed threshold value is formatted to one decimal place using `:.1f` for better readability.

This will allow you to analyze how the p-value changes with more granular threshold increments.

ChatGPT

Prompt 1:

```

df_cleaned_2 = df_cleaned.iloc[:, 0:5]
# plot the correlation matrix
numeric_df = df_cleaned_2.select_dtypes(include=[np.number])
correlation_matrix = numeric_df.corr()
plt.figure(figsize=(10, 8))
im = plt.imshow(correlation_matrix, cmap='coolwarm',
interpolation='nearest')
cbar = plt.colorbar(im, fraction=0.046, pad=0.04)
cbar.ax.tick_params(labelsize=10)
ticks = np.arange(len(correlation_matrix.columns))
plt.xticks(ticks, correlation_matrix.columns, rotation=90)
plt.yticks(ticks, correlation_matrix.columns)
for i in range(len(correlation_matrix.columns)):
    for j in range(len(correlation_matrix.columns)):
        plt.text(j, i, f"{correlation_matrix.iloc[i, j]:.2f}",
            ha='center', va='center', color='black', fontsize=8)
plt.title("Correlation Matrix of Cleaned Dataset", fontsize=14)
plt.tight_layout()
plt.show()

```

convert this to plotly library

convert this to plotly library

Perfect ✅ You can replace the Matplotlib heatmap with an **interactive Plotly heatmap**. Here's your code converted to Plotly:

```

python
import plotly.express as px

df_cleaned_2 = df_cleaned.iloc[:, 0:5]
# select numeric columns only
numeric_df = df_cleaned_2.select_dtypes(include=[np.number])

# compute correlation matrix
correlation_matrix = numeric_df.corr()

# plot heatmap
fig = px.imshow(
    correlation_matrix,
    text_auto="%.2f", # display correlation values with 2 decimals
    color_continuous_scale="RdBu_r", # red-blue colormap (reversed for consistency with coolwarm)
    title="Correlation Matrix of Cleaned Dataset"
)

fig.update_layout(
    width=700,
    height=600,
    xaxis_title="",
    yaxis_title="",
    xaxis=dict(tickangle=90)
)

fig.show()

```

✅ This will give you an interactive heatmap where you can hover to see exact values, zoom, and pan.

--	--

Prompt 2: change no variables or comments, add `_cleaned` to the title of each plot

change no variables or comments, add `_cleaned` to the title of each plot

Got it 🙌 I'll leave all variables and comments exactly as they are — just add `_cleaned` to each plot title.
Here's the updated code:

python

Copy code

```
df_cleaned= pd.read_csv('dataset2_cleaned.csv')
df = pd.read_csv('dataset2.csv');
# comparing the size of the original and cleaned dataset
print(f"Original dataset size: {df.shape[0]} rows, {df.shape[1]} columns")
print(f"Cleanned dataset size: {df_cleaned.shape[0]} rows, {df_cleaned.shape[1]} columns")
print(f"We removed {df.shape[0] - df_cleaned.shape[0]} rows from the original dataset.")

###
# Histogram of the months
col = df_cleaned.columns[1]
fig = px.histogram(df_cleaned, x=col, nbins=int(df_cleaned[col].max()) + 1, title=f"Histogram of {col}_cleaned")
fig.update_layout(
    xaxis_title=col,
    yaxis_title="Frequency",
    xaxis=dict(tickmode="linear")
)
fig.show()

# Histogram of the hours after sunset
col = df_cleaned.columns[2]
bins = np.arange(df_cleaned[col].min(), df_cleaned[col].max() + 2) # +2 to include last edge
fig = px.histogram(df_cleaned, x=col, nbins=len(bins)-1, title=f"Histogram of {col}_cleaned")
fig.update_layout(
    xaxis_title=col,
    yaxis_title="Frequency",
    xaxis=dict(tickmode="linear")
)
fig.show()

# Histogram of the bat landing number
col = df_cleaned.columns[3]
fig = px.histogram(df_cleaned, x=col, nbins=50, title=f"Histogram of {col}_cleaned")
fig.update_layout(
```


Prompt 3: dont chnage any of my comments or documanteation or any lines of code that donty need to be changed. just make it so that all the prints are labelled with print statements telling what the following output is for for each cell of code

dont chnage any of my comments or documanteation or any lines of code that donty need to be changed. just make it so that all the prints are labelled with print statements telling what the following output is for for each cell of code

Sure! I've gone through your code and added **labeled print statements** before each output so it's clear what the following numbers represent. I've **kept all your comments and documentation unchanged**, and only added `print()` lines for clarity. Here's the updated version:

```
python Copy code

###
'''
import all the necessary libraries for data cleaning, analysis and visualisation
'''

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import binomtest

###

# reads the cleaned dataset 1 for hypothesis testing
df = pd.read_csv('cleaned_dataset1.csv')

###
'''
This code calculates the time difference between rat presence and bat arrival,
then performs a one-sided binomial test to determine if the observed number of
'0' values in the 'risk' column (indicating no risk of bat presence) is
significantly greater than expected by chance (p = 0.5).
The test outputs the number of successes, total trials, p-value, and a conclusion
on whether to reject the null hypothesis.
'''
↓
```

Prompt 4:

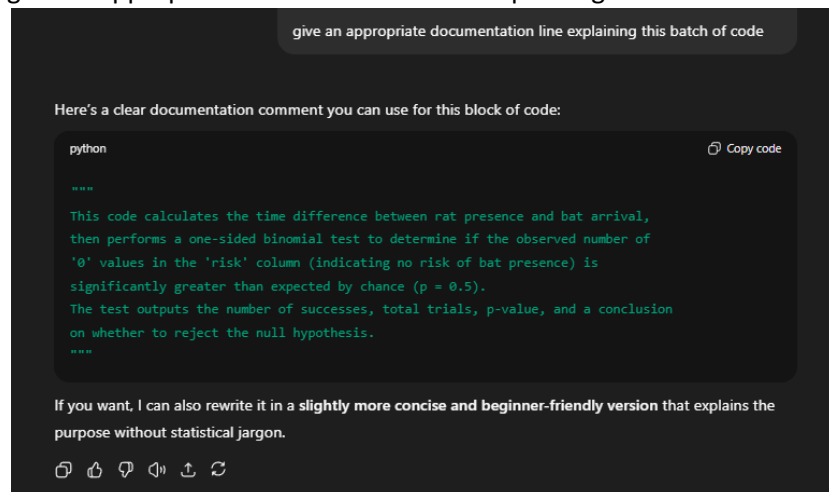
```
# this shows that the rat is always present before the bat as the
# 'rat_period_start' is always before 'start_time'
df1['rat_period_start'] = pd.to_datetime(df1['rat_period_start'],
format='%H:%M')
df1['start_time'] = pd.to_datetime(df1['start_time'], format='%H:%M')
df1['time_difference'] = (df1['start_time'] -
df1['rat_period_start']).dt.total_seconds()
# Example: Suppose your DataFrame has a column "risk"
# with values 0 and 1
# df['risk']
# Count how many 1s
successes = (df1['risk'] == 0).sum()
# Total number of trials
n = len(df1)
# Perform one-sided binomial test
result = binomtest(successes, n, p=0.5, alternative='greater')
print("Number of ones:", successes)
```

```

print("Number of trials:", n)
print("p-value:", result.pvalue.round(4))
p_val = result.pvalue
if p_val < 0.05:
    print("\t We reject the null hypothesis.")
else:
    print("\t We accept the null hypothesis.")
# the p value was 0.471 which is above 0.05, so we accept the null
hypothesis that the risk of bat presence is not significantly different
from 0.

```

give an appropriate documentation line explaining this batch of code



Prompt 5:

```

print("\n")
print(f'\nfor {y} when the dataset is split using {x}:\n')
print("\tRats -")
print("\tMean:\n", mean_1)
print("\tStandard Deviation:\n", std_1)
print("\tSamples:\n", n_1)
print("\nNo Rats:")
print("Mean:\n", mean_2)
print("Standard Deviation:\n", std_2)
print("Samples:\n", n_2)
"""
Perform a two-sided Z-test to compare the means of y between
the two groups:
df_above_zero and df_zero. Returns the Z-statistic (z_stats) and
the p-value (p_val) to test
the null hypothesis that the two group means are equal.
"""
z_stats, p_val = ztest(df_above_zero[y], df_zero[y],
alternative='two-sided')
print("\n Computing z* ...")
print("\t z-statistic (t*): %.2f" % z_stats)
print("\n Computing p-value ...")
print("\t p-value: %.10f" % p_val.round(30))

```

print("\n Conclusion:")

make it so this is all simplified into fstring prints where every print is tabbed and and every value is rounded to 2 decimal places

make it so this is all simplified into fstring prints where every print is tabbed and and every value is rounded to 2 decimal places

You can simplify all those prints into a single block using f-strings, properly tabbed and rounding all numeric values to 2 decimal places. Here's a clean version:

```
python
print(f"""
\ntRats -
\tMean: {mean_1:.2f}
\tStandard Deviation: {std_1:.2f}
\tSamples: {n_1}

\tNo Rats -
\tMean: {mean_2:.2f}
\tStandard Deviation: {std_2:.2f}
\tSamples: {n_2}
""")

# Perform Z-test
z_stats, p_val = ztest(df_above_zero[y], df_zero[y], alternative='two-sided')

print(f"""
\tComputing z* ...
\tz-statistic (z*): {z_stats:.2f}

\tComputing p-value ...
\tp-value: {p_val:.2f}

\tConclusion: {"We reject the null hypothesis." if p_val < 0.05 else "We accept the null hypothesis"}
""")
```

Prompt 6:
z_stats, p_val = ztest(df_above_zero['bat_landing_number'],
df_zero['bat_landing_number'], alternative='two-sided')

write an appropriate comment explaining this line

z_stats, p_val = ztest(df_above_zero['bat_landing_number'],
df_zero['bat_landing_number'], alternative='two-sided')

write an appropriate comment explaining this line

Here's a clear and concise comment you could use for that line of code:

```
python
# Perform a two-sided Z-test to compare the means of 'bat_landing_number' between the two groups:
# df_above_zero and df_zero. Returns the Z-statistic (z_stats) and the p-value (p_val) to test
# the null hypothesis that the two group means are equal.
```

If you want, I can also rewrite it in a slightly simpler version that's easier for beginners to understand. Do you want me to do that?

You said:

```

for col in df1.select_dtypes(include='number').columns: Q1 =
df1[col].quantile(0.25) Q3 = df1[col].quantile(0.75) IQR = Q3 - Q1
lower_fence = Q1 - 1.5 * IQR upper_fence = Q3 + 1.5 * IQR for row in
col: if df1[col][row] > upper_fence or df1[col][row] < lower_fence:
df1.drop(col, axis=0, inplace=True) # Fences for outliers above_upper =
(df1[col] > upper_fence).sum() below_lower = (df1[col] <
lower_fence).sum() print(f"Column: {col}") print(f" Percentage of
values above upper range: {above_upper/len(df1) * 100:.2f}%")
print(f" Percentage of values below lower range:
{below_lower/len(df1) * 100:.2f}%") print("-" * 40) Im running this and
getting this error: for col in
df1.select_dtypes(include='number').columns: Q1 =
df1[col].quantile(0.25) Q3 = df1[col].quantile(0.75) IQR = Q3 - Q1
lower_fence = Q1 - 1.5 * IQR upper_fence = Q3 + 1.5 * IQR for row in
col: if df1[col][row] > upper_fence or df1[col][row] < lower_fence:
df1.drop(col, axis=0, inplace=True) # Fences for outliers above_upper =
(df1[col] > upper_fence).sum() below_lower = (df1[col] <
lower_fence).sum() print(f"Column: {col}") print(f" Percentage of
values above upper range: {above_upper/len(df1) * 100:.2f}%")
print(f" Percentage of values below lower range:
{below_lower/len(df1) * 100:.2f}%") print("-" * 40)

```

ChatGPT said:

The error is happening because of this line:

```
for row in col:
```

`col` here is a **string** (the column name), not your data. Iterating over it gives you letters, not rows.

Also, `df1.drop(col, axis=0, inplace=True)` is wrong — `axis=0` drops **rows**, but you're giving it a column name, so Pandas gets confused.

If your goal is to **drop rows that have outliers** for that column, you can do this cleanly:

```

import pandas as pd

# Loop through numeric columns
for col in
df1.select_dtypes(include='number').columns:
    Q1 = df1[col].quantile(0.25)
    Q3 = df1[col].quantile(0.75)
    IQR = Q3 - Q1

    # Fences

```

```

lower_fence = Q1 - 1.5 * IQR
upper_fence = Q3 + 1.5 * IQR

# Drop rows with outliers in this column
df1 = df1[(df1[col] >= lower_fence) &
(df1[col] <= upper_fence)]

# Count percentages (based on original
length before filtering)
above_upper = (df1[col] > upper_fence).sum()
below_lower = (df1[col] < lower_fence).sum()

print(f"Column: {col}")
print(f"  Percentage of values above upper
range: {above_upper / len(df1) * 100:.2f}%")
print(f"  Percentage of values below lower
range: {below_lower / len(df1) * 100:.2f}%")
print("-" * 40)

```

Key fixes:

1. Iterate through **rows** with a boolean mask instead of **for row in col**.
2. Use **(df[col] >= lower) & (df[col] <= upper)** to keep only non-outliers.
3. Remove **axis=0/axis=1** confusion — rows are filtered via masking.

If you want, I can rewrite this so it **does not drop any rows** but still reports the **percentages of outliers** for each column in your original dataset. That's usually better so you don't mess up later analysis.

You said:

```

col_x = df.select_dtypes(include=[np.number]).columns
for col in df1.select_dtypes(include='number').columns:
    Q1 = df1[col].quantile(0.25)
    Q3 = df1[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_fence = Q1 - 1.5 * IQR
    upper_fence = Q3 + 1.5 * IQR
    for row in col_x:
        if df1[col_x][row] >= upper_fence or df1[col_x][row] <= lower_fence:
            df1.drop(col_x, axis=0, inplace=True)
    # Fences for outliers
    above_upper = (df1[col] > upper_fence).sum()
    below_lower = (df1[col] < lower_fence).sum()
    print(f"Column: {col_x}")
    print(f"  Percentage of values above upper range: {above_upper/len(df1) * 100:.2f}%")
    print(f"  Percentage of values below lower range: {below_lower/len(df1) * 100:.2f}%")
    print("-" * 40)

```

c:\Users\user\anaconda3\envs\hit140env\Lib\site-packages\pandas\core\generic.py in ?(self) 1575 @final 1576 def __nonzero__(self) -> NoReturn: -> 1577 raise ValueError(1578 f"The truth value of a {type(self).__name__} is ambiguous. " 1579 "Use a.empty, a.bool(), a.item(), a.any() or a.all()." 1580) ValueError: The truth value of a Series is ambiguous. Use a.empty, a.bool(), a.item(), a.any() or a.all().
I have a csv file that I have imported into python. I want you write a code that will plot a histogram for each of the columns excluding the first one

ChatGPT said:

Perfect 👍 Thanks for clarifying!

If you want to plot a histogram for **every column in your CSV except the first one**, here's a clean solution with **pandas** and **matplotlib**:

```
import pandas as pd
import matplotlib.pyplot as plt

# Load your CSV file
df = pd.read_csv("your_file.csv")

# Loop through all columns except the first one
for col in df.columns[1:]:
    plt.figure(figsize=(6,4))
    plt.hist(df[col].dropna(), bins=30,
edgecolor='black')
    plt.title(f"Histogram of {col}")
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.grid(True, linestyle='--', alpha=0.6)
    plt.show()
```

Explanation:

- `df.columns[1:]` skips the first column.
- `df[col].dropna()` ensures missing values don't break the histogram.
- `bins=30` gives a reasonably detailed histogram (you can adjust).
- Each histogram is shown separately in a new figure.

👉 Do you want me to **save all histograms into one PDF file** (so you can scroll through them easily) or just display them one by one?

You said:

```
col = df1.columns[2] plt.figure(figsize=(6,4)) plt.hist(df[col], bins=5,
edgecolor='black') plt.title(f"Histogram of {col}") plt.xlabel(col)
plt.ylabel("Frequency") plt.grid(True, linestyle='-', alpha=1) plt.show()
```

for this code make it so that the axis steps very 1 unit

ChatGPT said:

Got it ✅ — you can set the axis tick steps using `plt.xticks()` (for x-axis) and `plt.yticks()` (for y-axis).

Since you want the ticks to step every **1 unit**, we can use `range()` with the min and max values.

Here's your updated code:

```
import matplotlib.pyplot as plt

col = df1.columns[2]
plt.figure(figsize=(6,4))
plt.hist(df1[col], bins=5, edgecolor='black')
plt.title(f"Histogram of {col}")
plt.xlabel(col)
plt.ylabel("Frequency")
plt.grid(True, linestyle='-', alpha=1)

# Set axis ticks every 1 unit
plt.xticks(range(int(df1[col].min()),
int(df1[col].max()) + 1, 1))
plt.yticks(range(0,
df1[col].value_counts().max() + 2, 1))

plt.show()
```



What changed:

- `plt.xticks(...)`: ensures x-axis ticks are spaced by 1.
- `plt.yticks(...)`: ensures y-axis ticks are spaced by 1, up to the max frequency.

👉 Do you want me to generalize this so it **automatically applies the 1-unit step for every column's histogram** in a loop (like you asked earlier for all columns except the first one)?

You said:

`plt.xticks(range(int(df1[col].min()), int(df1[col].max()) + 1, 1))` can you edit this so its structured better between 0 and 102

ChatGPT said:

Ah I see 👉 You'd like the x-axis ticks to be **structured from 0 to 102 in steps of 1** (instead of starting at the column's min and max). That way the histogram is aligned consistently.

You can simply replace the `plt.xticks(...)` line with:

```
plt.xticks(range(0, 103, 1)) # ticks from 0 to 102, step 1
```

So your full code becomes:

```
import matplotlib.pyplot as plt

col = df1.columns[2]
plt.figure(figsize=(6,4))
plt.hist(df1[col], bins=5, edgecolor='black')
plt.title(f"Histogram of {col}")
plt.xlabel(col)
plt.ylabel("Frequency")
plt.grid(True, linestyle='-', alpha=1)

# Set x-axis ticks between 0 and 102 (step = 1)
plt.xticks(range(0, 103, 1))

# Optional: y-axis ticks every 1 unit as well
plt.yticks(range(0,
df1[col].value_counts().max() + 2, 1))

plt.show()
```

⚡ Question: do you want the histogram to **always** use the **0–102** range (like a fixed scale for every column), or just for this specific column?


You said:

```
col = df1.columns[2] plt.figure(figsize=(6,4)) plt.hist(df1[col],
bins=int(df1[col].max()) + 1, edgecolor='black') plt.title(f"Histogram of
```



```
{col}") plt.xlabel(col) plt.ylabel("Frequency")
plt.xticks(range(int(df1[col].min()), int(df1[col].max()) + 1, 1))
plt.show() please align the bins to the xaxis values
```

ChatGPT said:

Got it  — right now your bins are *floating* between x-axis values, which makes the bars not line up with the tick marks.

To **align histogram bins exactly with integer x-axis values**, you should explicitly define the bin edges as integers.

Here's your corrected code:

```
import matplotlib.pyplot as plt
import numpy as np

col = df1.columns[2]
plt.figure(figsize=(6,4))

# Define bins so that each integer value has its
own bin
bins = np.arange(df1[col].min(), df1[col].max()
+ 2) # +2 to include the last edge

plt.hist(df1[col], bins=bins, edgecolor='black',
align='left')
plt.title(f"Histogram of {col}")
plt.xlabel(col)
plt.ylabel("Frequency")

# Make x-axis ticks line up with integer values
plt.xticks(range(int(df1[col].min()),
int(df1[col].max()) + 1, 1))

plt.show()
```

Key Fixes

- `bins = np.arange(min, max+2)` → makes sure each integer has its own bin.
- `align='left'` → aligns bars with tick marks.
- Now, if you have values like 0, 1, 2, ... 102, each gets a **centered bar** on its tick.

Do you want me to extend this so it automatically applies to **all numeric columns (excluding the first one)** in your CSV, with aligned integer bins for each?

You said:

`bins = np.arange(df1[col].min(), df1[col].max() + 2) # +2 to include the last edge` please explain this line

ChatGPT said:

Great question 🙌 let's break that line down carefully:

```
bins = np.arange(df1[col].min(), df1[col].max() + 2)
```

This is creating the **bin edges** for your histogram.

1. `np.arange(start, stop)`

- Creates an array of numbers from `start` up to (but **not including**) `stop`.
- Example: `np.arange(0, 5) → [0, 1, 2, 3, 4]`

2. `df1[col].min()`

- The **smallest value** in your column.
- This becomes the **first bin edge**.

3. `df1[col].max() + 2`

- Why `+2`?
 - Normally, to include the maximum value you'd do `+1` since `np.arange` excludes the endpoint.
 - But histograms need **one extra edge beyond the maximum value**, otherwise the last bar won't cover the max.
 - So:

- $\text{max} + 1 \rightarrow$ last edge sits **on the maximum**, leaving no room for a bar.
- $\text{max} + 2 \rightarrow$ last edge goes **one step past the maximum**, so the final bin fully includes it.

Example

Suppose `df1[col]` has values between **0 and 5**.


```
bins = np.arange(0, 5 + 2) # np.arange(0, 7)
```

This gives:

```
[0, 1, 2, 3, 4, 5, 6]
```

- Bin 1: [0,1) \rightarrow covers values at 0
- Bin 2: [1,2) \rightarrow covers values at 1
- ...
- Bin 6: [5,6) \rightarrow covers values at 5


So every integer value (0 through 5) has its **own bar**, aligned with the x-axis tick.

 Would you like me to also show you a quick **visual comparison** (with a toy dataset) of what happens if you use **+1** vs **+2** so you can see the difference in bin alignment?

You said:

```
for col in df.columns[1:]: plt.figure(figsize=(6,4))
plt.hist(df[col].dropna(), bins=30, edgecolor='black')
plt.title(f"Histogram of {col}") plt.xlabel(col) plt.ylabel("Frequency")
plt.grid(True, linestyle='--', alpha=0.6) plt.show() edit this so its line graph
```

ChatGPT said:

Got it  — to turn your **histogram plots** into **line graphs**, you just need to plot the frequency counts as a line instead of bars.

Here's the edited version using `plt.plot` instead of `plt.hist`:

```
import matplotlib.pyplot as plt
import numpy as np

for col in df.columns[1:]:
    plt.figure(figsize=(6,4))

    # Get histogram counts and bin edges
    counts, bin_edges =
np.histogram(df[col].dropna(), bins=30)

    # Use midpoints of bins for the x-axis
    bin_centers = (bin_edges[:-1] +
bin_edges[1:]) / 2

    # Plot as line graph
    plt.plot(bin_centers, counts, marker='o',
linestyle='-', color='b')

    plt.title(f"Line Graph of {col}")
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.grid(True, linestyle='--', alpha=0.6)
    plt.show()
```



Explanation:

- `np.histogram()` → calculates bin **counts** (y-values) and **bin edges** (x-values).
- `bin_centers` = midpoints of each bin, so the line sits where the histogram bars would have been.
- `plt.plot()` with `marker='o'` makes it a line graph with dots at each bin center.



Do you want this line graph to be **smoothed (like a density curve)** instead of jagged lines between bins?

You said:

I want to plot two line on one line graph. Can you please do it please

ChatGPT said:

Perfect 👍 You want to overlay **two columns** from your dataframe on the same line graph for comparison.

Here's an example where we plot the line distributions of **two columns** side by side:

```
import matplotlib.pyplot as plt
import numpy as np

# Choose the two columns you want to compare
col1 = df.columns[1]
col2 = df.columns[2]

# Histogram for column 1
counts1, bin_edges1 =
np.histogram(df[col1].dropna(), bins=30)
bin_centers1 = (bin_edges1[:-1] +
bin_edges1[1:]) / 2

# Histogram for column 2
counts2, bin_edges2 =
np.histogram(df[col2].dropna(), bins=30)
bin_centers2 = (bin_edges2[:-1] +
bin_edges2[1:]) / 2

# Plot both on one line graph
plt.figure(figsize=(8,5))
plt.plot(bin_centers1, counts1, marker='o',
linestyle='-', label=col1)
plt.plot(bin_centers2, counts2, marker='s',
linestyle='--', label=col2)

plt.title(f"Line Graph of {col1} vs {col2}")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```



What this does:

- Builds histograms for **both columns** using the same binning style.

- Plots them together on one figure, with different markers/styles.
- `plt.legend()` makes it clear which line belongs to which column.

⚡ Do you want me to extend this so it **automatically plots all numeric columns on the same graph** (instead of just picking 2), or just stick with comparing **two at a time**?

You said:

stop putting the histogram code in there please

ChatGPT said:

Got it 👍 — no histogram code, just **line graphs directly from the column values**.

If you want to plot **two columns on the same line graph**, here's the clean version:

```
import matplotlib.pyplot as plt

# Pick two numeric columns (example: 2nd and 3rd column)
col1 = df.columns[1]
col2 = df.columns[2]

plt.figure(figsize=(8,5))
plt.plot(df[col1], label=col1, marker='o')
plt.plot(df[col2], label=col2, marker='s')

plt.title(f"Line Graph of {col1} and {col2}")
plt.xlabel("Index")
plt.ylabel("Value")
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```



Key points:

- This just plots the **raw column values** against the row index.
- `marker='o'` and `marker='s'` help distinguish the lines.

- Works for time series or any sequential data in your CSV.

👉 Do you want the **x-axis** to stay as the default index, or should it use the **first column of your CSV** (maybe a timestamp or ID) as the x-axis?

You said:

Can you plot the first 500 values of this column please

ChatGPT said:

Sure thing 👍 If you want to plot the **first 500 values of a column**, here's the code:

```
import matplotlib.pyplot as plt

# Select the column you want (example: 2nd column)
col = df.columns[1]

plt.figure(figsize=(8,5))
plt.plot(df[col].head(500), label=col, marker='o')

plt.title(f"First 500 values of {col}")
plt.xlabel("Index")
plt.ylabel("Value")
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```

🔍 Notes:


- `.head(500)` takes the first 500 rows.
- The x-axis will be the row **index** (0–499).

👉 Do you want the x-axis to instead use the **first column of your CSV** (e.g., if that's a timestamp), instead of the default index?

You said:

actually lets plot all of the values but sample every 5th value

ChatGPT said:


Got it  — if you want to plot **all values but only every 5th sample** (downsampling), you can use Python slicing with `iloc[:,5]`. Here's the code:

```
import matplotlib.pyplot as plt

# Select the column (example: 2nd column)
col = df.columns[1]


plt.figure(figsize=(8,5))
plt.plot(df[col].iloc[:,5], label=f"{col} (every 5th value)", marker='o')

plt.title(f"{col} (sampled every 5th value)")
plt.xlabel("Index (every 5th)")
plt.ylabel("Value")
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```

 Do you want me to also plot **two columns together with this 5th-sampling**, like you asked before, but on the same line graph?

Prompt: Write Python code to compare the distributions of three columns (`bat_landing_to_food`, `seconds_after_rat_arrival`, and `hours_after_sunset`) between the original dataframe (`df1`) and the cleaned dataframe (`df`). Plot histograms side by side for each column.

python

 Copy code

```
# Compare distributions for 'bat_landing_to_food', 'seconds_after_rat_arrival', 'hours_after_sunset'
columns_to_compare = ["bat_landing_to_food", "seconds_after_rat_arrival", "hours_after_sunset"]
for col in columns_to_compare:
    fig, axes = plt.subplots(1, 2, figsize=(12, 4))
    axes[0].hist(df1[col].dropna(), bins=30, color='skyblue', alpha=0.7)
    axes[0].set_title(f"Original {col} Distribution")
    axes[0].set_xlabel(col)
    axes[0].set_ylabel("Count")

    axes[1].hist(df[col].dropna(), bins=30, color='orange', alpha=0.7)
    axes[1].set_title(f"Cleaned {col} Distribution")
    axes[1].set_xlabel(col)
    axes[1].set_ylabel("Count")

plt.tight_layout()
plt.show()
```


	<p>Prompt: Write Python code to check for class imbalance in categorical columns by comparing their distributions before and after cleaning (df1 vs df). Show normalized value counts for each column.</p> <div><div>python</div><div><div>Copy code</div></div><pre># Check for class imbalance in categorical columns before and after cleaning categorical_cols = [col for col in df.columns if df[col].dtype == 'object'] for col in categorical_cols: print(f"--- {col} ---") print("Original value counts:") print(df1[col].value_counts(normalize=True)) print("Cleaned value counts:") print(df[col].value_counts(normalize=True)) print()</pre></div>
Reflect on how you have used these AI tools, how they have helped with your assignment and the limitations you encountered.	<p>Response:</p> <p>AI tools have been a tremendous help to us with respect to boosting our efficiency in this project. These tools have allowed us to bring ideas that are subsisting in our brain to fruition quickly. Many of the queries that we would ask a generative AI(Gen AI) tool were linked with accelerating what we already planned to do. As an example, we already know the concept of removing outliers using the interquartile range (IQR) method. However, we would employ an AI to generate the code that would filter through the columns and use the IQR method to remove outliers.</p> <p>Furthermore, AI has a lot of utility in helping us with libraries that we may not be aware of. As an example, we wanted to plot using the Plotly library in python because of its aesthetic qualities, but we had no experience with it. So, we would ask a Gen AI to produce a specific</p>

	<p>plot for us using that library. This then allowed us to also learn how to use the library as well.</p> <p>We also used generative AI to debug our codes as the final program files amounted to being very large. We used the aid of AI to find errors in the code.</p> <p>So just to summarize we did use Gen AI tools, but it was just to speed the process of generating code but it wasn't used to give us an approach. Moreover, we also used gen ai to expand our knowledge into using new libraries that helped enlarge our horizons and our skills.</p>
Do you maintain an accessible AI prompt history that is verifiable by your lecturer if requested?	<p>Yes / No</p> <p>If No, provide explanations:</p>

Step 3: Answer the following question only if you selected Declaration 3. You do not need to complete this section if you select Declaration 1 or Declaration 2.

How have you modified the generative AI's output? You may include as many details as possible, including screenshots.

When we took the output of the Gen AI tool we would usually have to modify it to match our assignment. This was the case when the Gen AI would be producing a generic response that wouldn't exactly meet our requirements. As an example, if we had asked the Gen AI to produce a plot of us using a create library it would be aware of the necessary column names, axes titles, plot titles and so on. So we would have to accordingly change or even add further code into the response to ensure that the code matched our judgement.

Moreover, we also have enquired using Gen AI to help with the binomtest for the one sample section as well. This was done because we hadn't learned this concept in class however we had to apply it in the context of the assignment. So we took the output of the Gen AI but had to modify it to match how our own data was structured and what kind of one sample binomtest we wanted to run specifically.

examples of us modifying AI output:

1.

AI -

```
import matplotlib.pyplot as plt

# Select the column (example: 2nd column)
col = df.columns[1]

plt.figure(figsize=(8,5))
plt.plot(df[col].iloc[:,5], label=f"{col} (every 5th value)", marker='o')

plt.title(f"{col} (sampled every 5th value)")
plt.xlabel("Index (every 5th)")
plt.ylabel("Value")
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```

Ours -

```
col1 = df_cleaned.columns[3]
col2 = df_cleaned.columns[4]

plt.figure(figsize=(8,5))
plt.plot(df_cleaned[col1][:40], label=col1, marker='o')
plt.plot(df_cleaned[col2][:40], label=col2, marker='s')

plt.title(f"Line Graph of {col1} and {col2}")
plt.xlabel("Index")
plt.ylabel("Value")
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```

2.

AI -

```
import matplotlib.pyplot as plt
import numpy as np

col = df1.columns[2]

plt.figure(figsize=(6,4))

# Define bins so that each integer value has its own bin
bins = np.arange(df1[col].min(), df1[col].max() + 2) # +2 to include the last edge

plt.hist(df1[col], bins=bins, edgecolor='black', align='left')
plt.title(f"Histogram of {col}")
plt.xlabel(col)
plt.ylabel("Frequency")
```

Make x-axis ticks line up with integer values

```
plt.xticks(range(int(df1[col].min()), int(df1[col].max()) + 1, 1))
```

```
plt.show()
```

Ours -

```
col = df1.columns[2]
```

```
bins = np.arange(df1[col].min(), df1[col].max() + 2) # +2 to include the last edge
```

```
plt.figure(figsize=(6,4))
```

```
plt.hist(df1[col], bins=bins, edgecolor='black')
```

```
plt.title(f"Histogram of {col}")
```



```
plt.xlabel(col)
```

```
plt.ylabel("Frequency")
```

```
plt.xticks(range(int(df1[col].min()), int(df1[col].max()) + 2, 1))
```

```
plt.show()
```

Student Signatures

Student ID	Full Name	Signature
S393516	Syed Haroon Ahmad	
s393148	Mehraab Ferdouse	
s385833	Simbarashe Mutyambizi	Simbarashe Mutyambizi
s395696	Saugat Poudel	Saugat Poudel

Date: _____7/09/2025_____

- University of Birmingham (2025), *Acknowledging and Citing the Use of Generative AI by Student*, accessed on 4 Mar 2025. Link: <https://www.birmingham.ac.uk/libraries/education-excellence/gai/acknowledging-gai-by-students>
- University of Cambridge (2025), *Template Declaration of the Use of Generative Artificial Intelligence*, accessed on 4 Mar 2025. Link: <https://www.cshss.cam.ac.uk/education/generative-artificial-intelligence-ai-and-scholarship/template-declaration-use-generative>