```
# Upgrading Seaborn
!pip install --upgrade seaborn
# Importing Libraries
import pandas as pd
import numpy as np
import scipy.stats as stats
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib
```

```
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /usr/local/lib/python3.11/dist-packages (from seaborn) (1.26.4)
Requirement already satisfied: pandas>=1.2 in /usr/local/lib/python3.11/dist-packages (from seaborn) (2.2.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /usr/local/lib/python3.11/dist-packages (from seaborn) (3.10.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.3.
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.5
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.2.
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.2->seaborn) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.2->seaborn) (2025.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->
```

Import Dataset

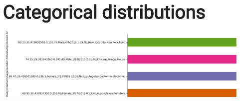
```
from google.colab import files
import io
import pandas as pd

uploaded = files.upload()
df = pd.read_csv(io.StringIO(uploaded[list(uploaded.keys())[0]].decode('utf-8')), sep=';')
df.head(5)
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving data.csv to data (2).csv
```

	Daily Time Spent on Site,Age,Area Income,Daily Internet Usage,Gender,Timestamp,Clicked on Ad,City,State,Category
0	68.95,35,432837300.0,256.09,Female,3/27/2016 0...
1	80.23,31,479092950.0,193.77,Male,4/4/2016 1:39...
2	69.47,26,418501580.0,236.5,Female,3/13/2016 20...
3	74.15,29,383643260.0,245.89,Male,1/10/2016 2:3...
4	68.37,35,517229930.0,225.58,Female,6/3/2016 3:...

Categorical distributions



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 11 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Unnamed: 0                1000 non-null   int64
 1   Daily Time Spent on Site  978 non-null    float64
 2   Age                       1000 non-null   int64
 3   Area Income               983 non-null    float64
 4   Daily Internet Usage      979 non-null    float64
 5   Male                      975 non-null    object
 6   Timestamp                 1000 non-null   object
 7   Clicked on Ad             1000 non-null   object
 8   city                      1000 non-null   object
 9   province                  1000 non-null   object
```

```
 10  category                 1000 non-null   object
dtypes: float64(3), int64(2), object(6)
memory usage: 86.1+ KB
```

```
from google.colab import files
import io
import pandas as pd

uploaded = files.upload()
df = pd.read_csv(io.BytesIO(uploaded[list(uploaded.keys())[0]]))
df.head(5)
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving charlotte_data.csv to charlotte_data (1).csv
```

	Unnamed: 0	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Male	Timestamp	Clicked on Ad	city	province	category
0	0	93.67	62	69844.53	101.87	Male	2024-04-06	No	Charlotte	North Carolina	Finance
1	1	191.62	60	70121.18	35.95	Female	2024-10-23	No	Charlotte	North Carolina	Health
2	2	154.44	51	95206.52	187.48	Female	2020-05-07	No	Charlotte	North Carolina	Tech
3	3	131.77	26	31423.90	257.40	Male	2024-08-17	No	Charlotte	North Carolina	Finance
4	4	56.52	43	60698.10	178.22	Male	2021-09-15	No	Charlotte	North Carolina	Tech

Data Exploration

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 11 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Unnamed: 0               1000 non-null   int64
 1   Daily Time Spent on Site  978 non-null    float64
 2   Age                      1000 non-null   int64
 3   Area Income              983 non-null    float64
 4   Daily Internet Usage     979 non-null    float64
 5   Male                     975 non-null    object
 6   Timestamp                1000 non-null   object
 7   Clicked on Ad            1000 non-null   object
 8   city                     1000 non-null   object
 9   province                 1000 non-null   object
 10  category                 1000 non-null   object
dtypes: float64(3), int64(2), object(6)
memory usage: 86.1+ KB
```

```
# Checking shape of dataframe
print(f'Number of rows: {df.shape[0]}')
print(f'Number of columns {df.shape[1]}')
```

```
Number of rows: 1000
Number of columns 11
```

```
# Dataset overview
result = []
for col in df.columns:
    result.append([col, df[col].dtype, df[col].isna().sum(), 100*df[col].isna().sum()/len(df[col]), df[col].nunique(), df[col].unique()[:5]]

output = pd.DataFrame(data=result, columns = 'column data_type no._null percent_null no._unique unique_sample'.split())
output
```

	column	data_type	no._null	percent_null	no._unique	unique_sample
0	Unnamed: 0	int64	0	0.0	1000	[0, 1, 2, 3, 4]
1	Daily Time Spent on Site	float64	22	2.2	954	[93.67, 191.62, 154.44, 131.77, 56.52]
2	Age	int64	0	0.0	52	[62, 60, 51, 26, 43]
3	Area Income	float64	17	1.7	983	[69844.53, 70121.18, 95206.52, 31423.9, 60698.1]
4	Daily Internet Usage	float64	21	2.1	963	[101.87, 35.95, 187.48, 257.4, 178.22]
5	Male	object	25	2.5	2	[Male, Female, nan]
6	Timestamp	object	0	0.0	764	[2024-04-06, 2024-10-23, 2020-05-07, 2024-08-1...
7	Clicked on Ad	object	0	0.0	2	[No, Yes]
8	city	object	0	0.0	1	[Charlotte]
9	province	object	0	0.0	1	[North Carolina]
10	category	object	0	0.0	5	[Finance, Health, Tech, Entertainment, Sports]

Start coding or generate with AI.

About The Dataset

Overview:

- Dataset contains 1000 rows, 10 features and 1 **Unnamed: 0** column which is the ID column.
- Dataset consists of 3 data types; float64, int64 and object.
- **Timestamp** feature could be changed into datetime data type.
- Dataset contains null values in various columns.

Description:

- **Unnamed: 0** = ID of Customers
- **Daily Time Spent on Site** = Time spent by the user on a site in minutes
- **Age** = Customer's age in terms of years
- **Area Income** = Average income of geographical area of consumer
- **Daily Internet Usage** = Average minutes in a day consumer is on the internet
- **Male** = Gender of the customer
- **Timestamp** = Time at which user clicked on an Ad or the closed window
- **Clicked on Ad** = Whether or not the customer clicked on an Ad (Target Variable)
- **city** = City of the consumer
- **province** = Province of the consumer
- **category** = Category of the advertisement

Exploratory Data Analysis

Feature engineering for EDA

Timestamp

```
df_eda = df.copy()
```

```
# Converting Timestamp column into datetime

df_eda['Timestamp'] = pd.to_datetime(df_eda['Timestamp'])
df_eda['Timestamp'].dtypes
```

```
dtype('<M8[ns]')
```

```
df_eda['Timestamp'].dt.year.unique()
```

```
array([2024, 2020, 2021, 2023, 2022, 2025], dtype=int32)
```

Data is from 2016 only.

```
df_eda['Timestamp'].dt.month.unique()
```

```
array([ 4, 10,  5,  8,  9,  6, 12,  2,  1,  7, 11,  3], dtype=int32)
```

Unnamed: 0

```
# Renaming "Unnamed: 0 column into ID column"
df_eda.rename(columns={'Unnamed: 0':'ID'}, inplace=True)
```

Male

```
# Renaming "Male" column into Gender column
df_eda.rename(columns={'Male':'Gender'}, inplace=True)
```

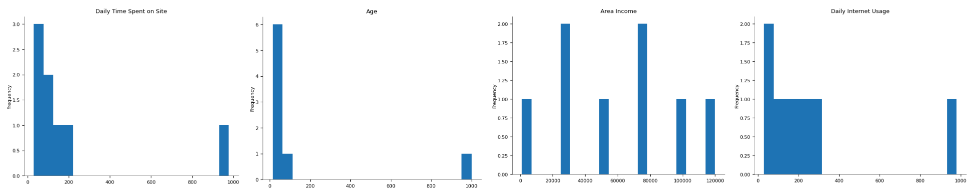
Descriptive Statistics

```
# Creating list of numerical and categorical columns
nums = [col for col in df_eda.columns if (df_eda[col].dtype == 'int64' or df_eda[col].dtype == 'float64') and col != 'ID' ]
cats = [col for col in df_eda.columns if df_eda[col].dtype == 'object']
```

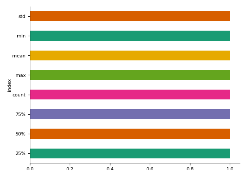
```
# Descriptive stats for numerical features
df_eda[nums].describe()
```

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage
count	978.000000	1000.000000	983.000000	979.000000
mean	113.297270	43.391000	74266.300916	169.966834
std	49.748253	14.813425	26147.315786	76.895112
min	30.790000	18.000000	30016.960000	30.120000
25%	69.330000	30.000000	50943.390000	105.180000
50%	114.625000	43.000000	73955.470000	173.610000
75%	156.657500	56.000000	97017.770000	236.105000
max	199.950000	69.000000	119974.240000	299.880000

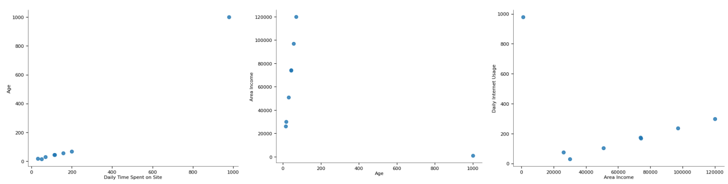
Distributions



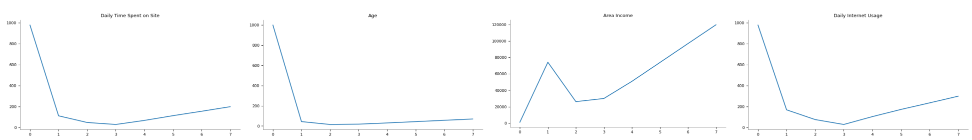
Categorical distributions



2-d distributions

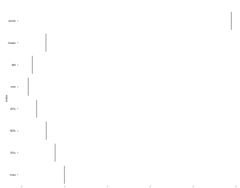


Values



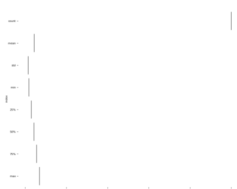
Faceted distributions

```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend
```



```
<string>:5: FutureWarning:
```

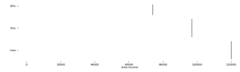
```
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend
```



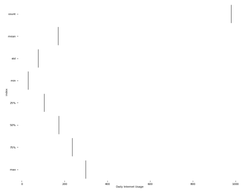
```
<string>:5: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend
```

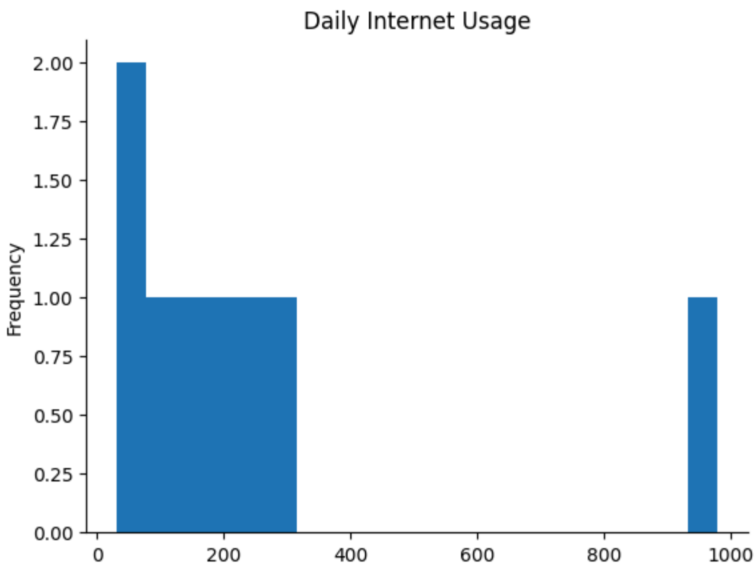
```
<string>:5: FutureWarning:
```



```
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend
```



```
from matplotlib import pyplot as plt
_df_4['Daily Internet Usage'].plot(kind='hist', bins=20, title='Daily Internet Usage')
plt.gca().spines[['top', 'right',]].set_visible(False)
```




```
# Descriptive stats for categorical features
df_eda[cats].describe()
```

	Gender	Clicked on Ad	city	province	category
count	997	1000	1000	1000	1000
unique	2	2	30	16	10
top	Perempuan	No	Surabaya	Daerah Khusus Ibukota Jakarta	Otomotif
freq	518	500	64	253	112

```
result = []
for col in df_eda.columns:
    result.append([col, df_eda[col].dtype, df_eda[col].isna().sum(), 100*df_eda[col].isna().sum()/len(df_eda[col]), df_eda[col].nunique(), d

output = pd.DataFrame(data=result, columns = 'column data_type no._null percent_null no._unique unique_sample'.split())
output
```

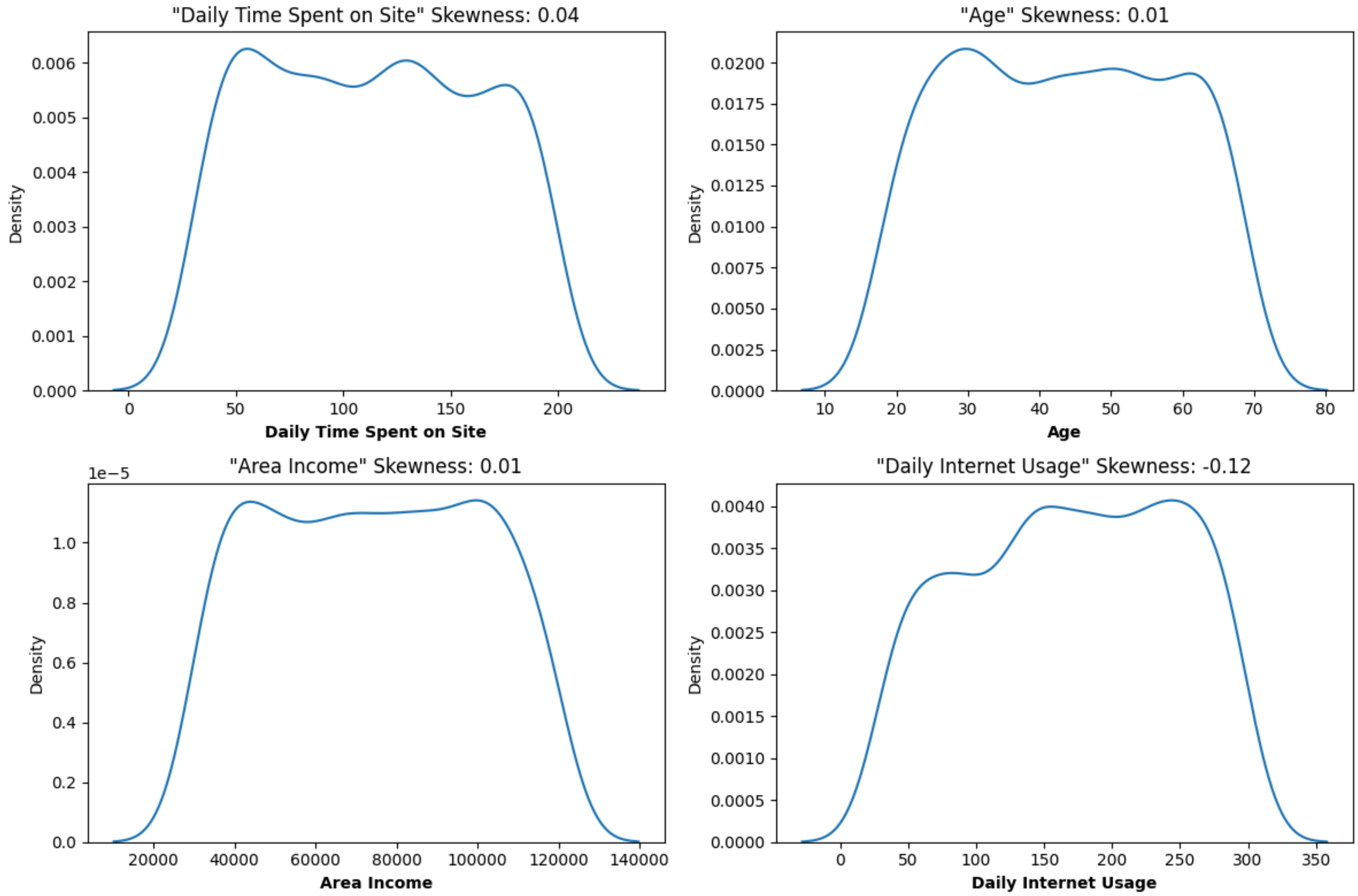
	column	data_type	no._null	percent_null	no._unique	unique_sample
0	ID	int64	0	0.0	1000	[0, 1, 2, 3, 4]
1	Daily Time Spent on Site	float64	13	1.3	890	[68.95, 80.23, 69.47, 74.15, 68.37]
2	Age	int64	0	0.0	43	[35, 31, 26, 29, 23]
3	Area Income	float64	13	1.3	987	[432837300.0, 479092950.00000006, 418501580.0,...
4	Daily Internet Usage	float64	11	1.1	955	[256.09, 193.77, 236.5, 245.89, 225.58]
5	Gender	object	3	0.3	2	[Perempuan, Laki-Laki, nan]
6	Timestamp	datetime64[ns]	0	0.0	997	[2016-03-27T00:53:00.000000000, 2016-04-04T01:...
7	Clicked on Ad	object	0	0.0	2	[No, Yes]
8	city	object	0	0.0	30	[Jakarta Timur, Denpasar, Surabaya, Batam, Medan]
9	province	object	0	0.0	16	[Daerah Khusus Ibukota Jakarta, Bali, Jawa Tim...
10	category	object	0	0.0	10	[Furniture, Food, Electronic, House, Finance]

Univariate analysis

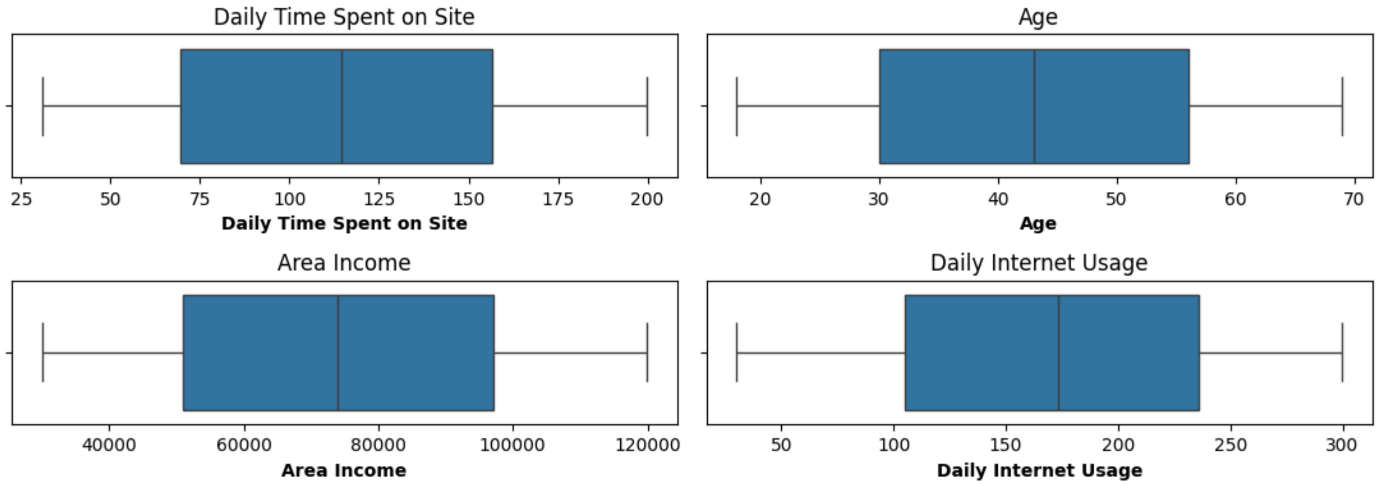
Numerical features

```
skewness = df_eda[nums].skew()

plt.figure(figsize = (12,8))
for i in range(len(nums)):
    plt.subplot(2, 2, i+1)
    sns.kdeplot(df_eda[nums[i]])
    plt.xlabel(nums[i], fontsize=10, fontweight = 'bold')
    plt.title(f'"{nums[i]}"'+f' Skewness: {round(skewness[i], 2)}')
    plt.tight_layout()
```

```
<ipython-input-24-cc177698837f>:8: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, inte
  plt.title(f'"{nums[i]}"'+f' Skewness: {round(skewness[i], 2)}')
<ipython-input-24-cc177698837f>:8: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, inte
  plt.title(f'"{nums[i]}"'+f' Skewness: {round(skewness[i], 2)}')
<ipython-input-24-cc177698837f>:8: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, inte
  plt.title(f'"{nums[i]}"'+f' Skewness: {round(skewness[i], 2)}')
<ipython-input-24-cc177698837f>:8: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, inte
  plt.title(f'"{nums[i]}"'+f' Skewness: {round(skewness[i], 2)}')
```




```
plt.figure(figsize=(11, 4))
for i in range(len(nums)):
    plt.subplot(2, 2, i+1)
    sns.boxplot(x = df_eda[nums[i]])
    plt.xlabel(nums[i], fontsize=10, fontweight = 'bold')
    plt.title(f'{nums[i]}')
    plt.tight_layout()
```

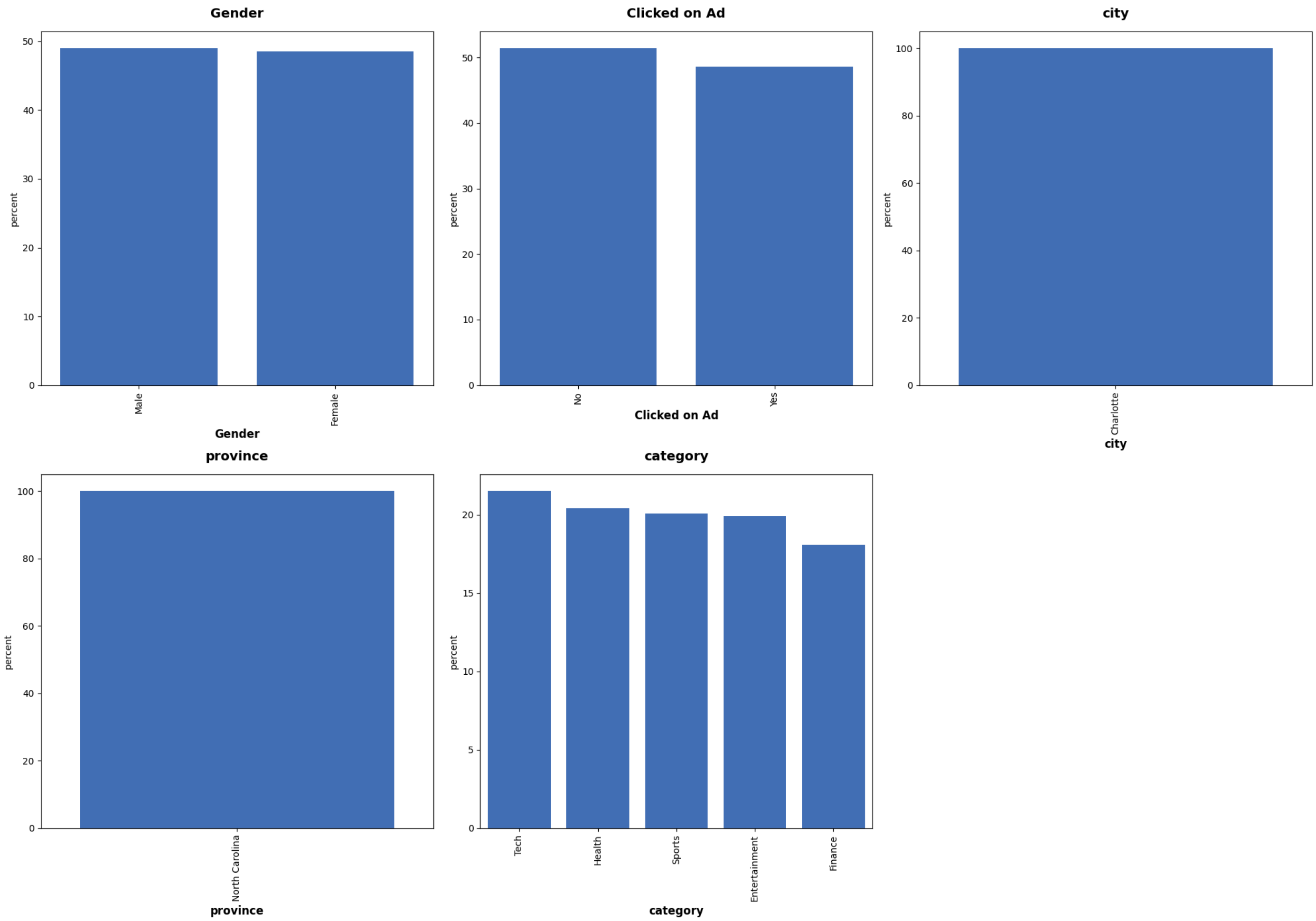


Analysis:

- **Area Income** is the only feature with a slight skew (left-skewed).
- **Daily Internet Usage** is nearly uniformly distributed.
- While **Age** and **Daily Time Spent on Site** is nearly normally distributed. Conclusion: with this analysis Adgreen has the potential to tailor advertisements distributed based off of daily internet usage, age and area income and reduces unwanted ad spend and adservers allocated.

Categorical features

```
plt.figure(figsize=(20,14))
for i in range(len(cats)):
    order = df_eda[cats[i]].value_counts().index
    plt.subplot(2, 3, i+1)
    if len(df_eda[cats[i]].unique()) > 3:
      sns.countplot(x = df_eda[cats[i]], data = df_eda, order=order, color = '#326cc9', stat='percent')
    else:
      sns.countplot(x = df_eda[cats[i]], data = df_eda, order=order, color = '#326cc9', stat='percent')
    plt.xticks(rotation=90)
    plt.xlabel(cats[i], fontsize=12, fontweight = 'bold')
    plt.title(f'{cats[i]}', fontsize=14, fontweight='bold', pad=15)
    plt.tight_layout()
```



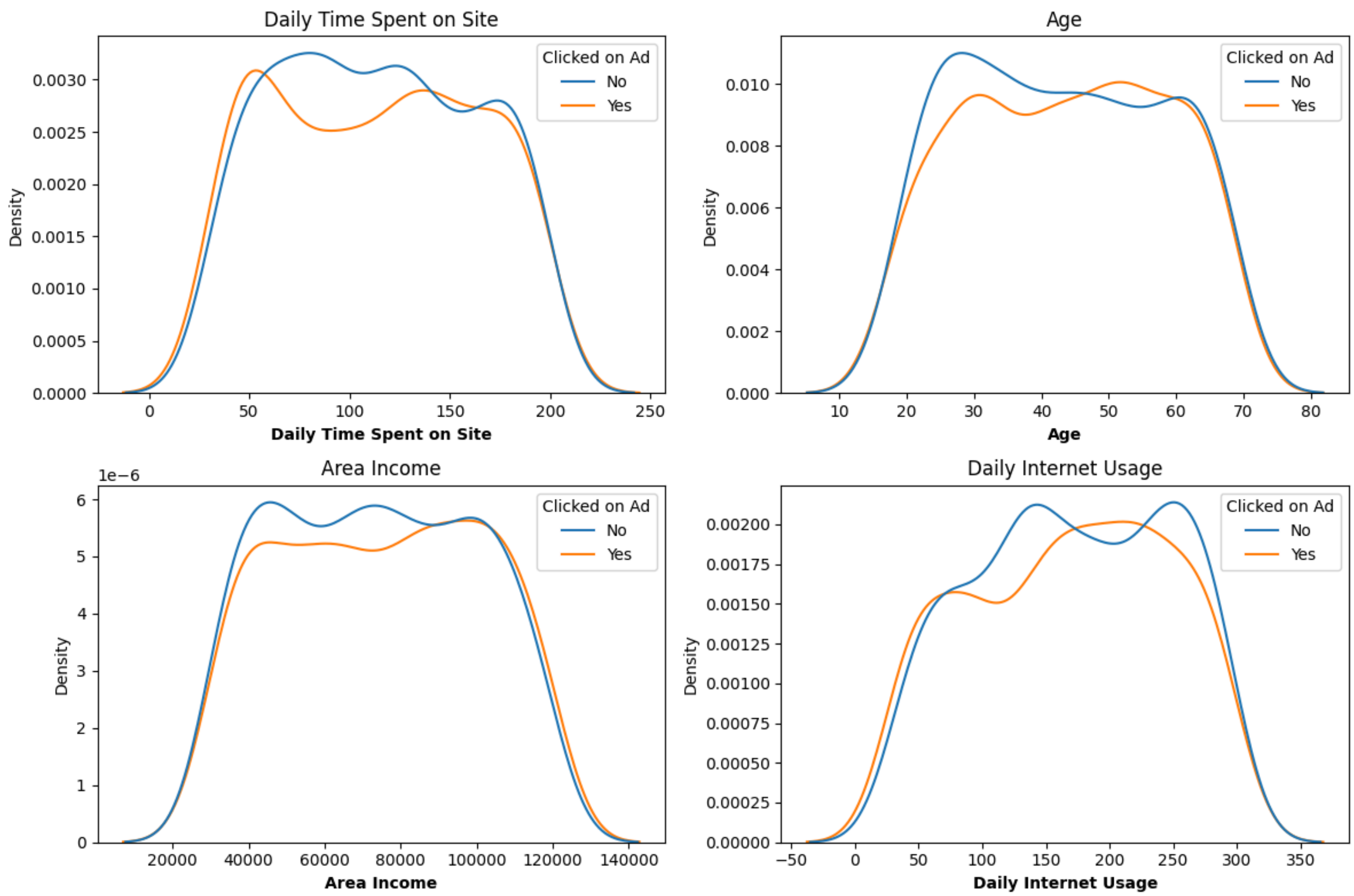
Analysis:

- **Gender** has an almost equal distribution of male and female.
- **Clicked on Ad** has an equal distribution of No and Yes.
- **Province and City** has 2 somewhat dominant values
- **category** is almost equally distributed among the all the values in the two sample locations.

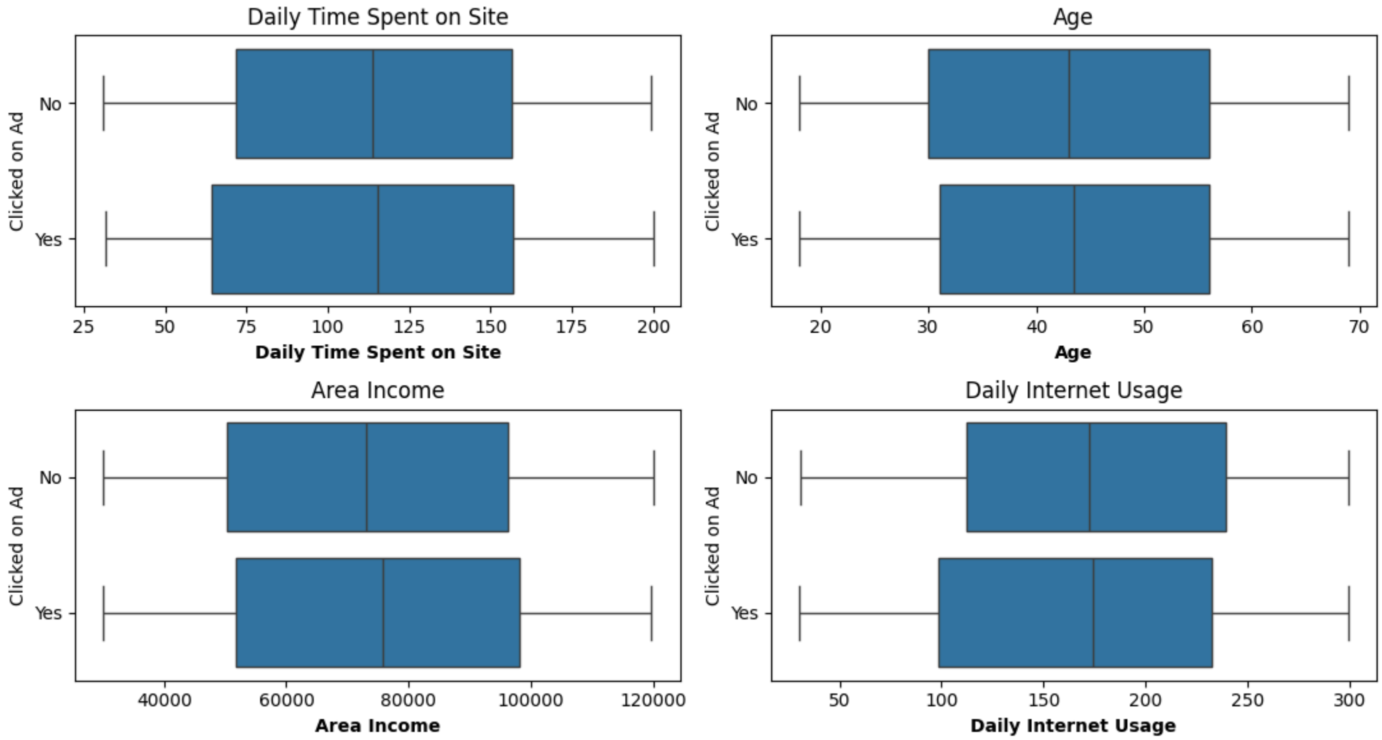
Bivariate analysis

Numerical features

```
plt.figure(figsize = (12,8))
for i in range(len(nums)):
    plt.subplot(2, 2, i+1)
    sns.kdeplot(x=nums[i], hue='Clicked on Ad', data=df_eda)
    plt.xlabel(nums[i], fontsize=10, fontweight = 'bold')
    plt.title(f'{nums[i]}')
    plt.tight_layout()
```




```
plt.figure(figsize=(11, 6))
for i in range(len(nums)):
    plt.subplot(2, 2, i+1)
    sns.boxplot(x = nums[i], y='Clicked on Ad', data=df_eda)
    plt.xlabel(nums[i], fontsize=10, fontweight = 'bold')
    plt.title(f'{nums[i]}')
    plt.tight_layout()
```

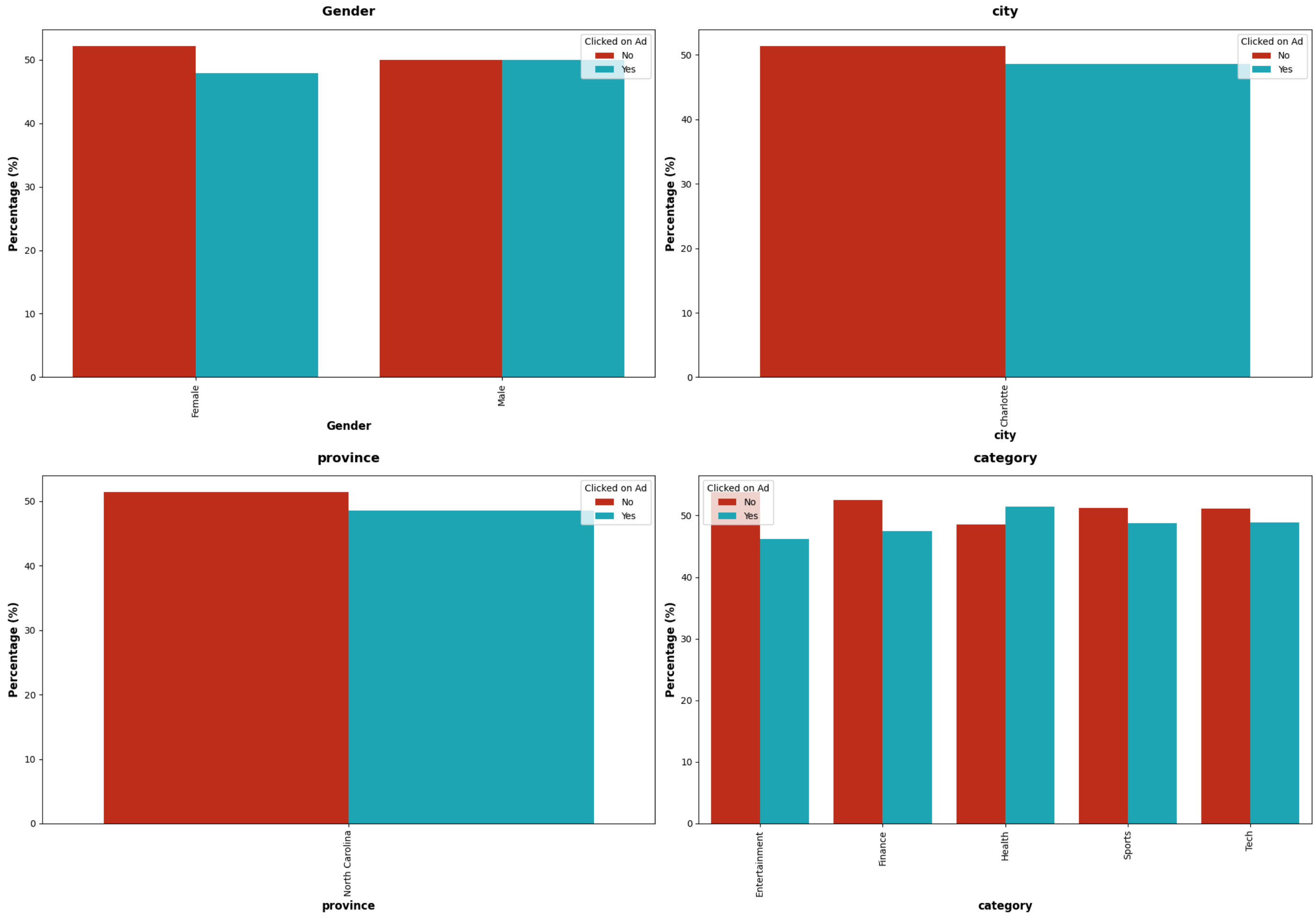


Analysis:

- The more time is spent on site by the customer the less likely they will click on an ad.
- The average age of customers that clicked on an ad is 40, while the average for those that didn't is 31.
- The average area income of customers that clicked on an ad is slightly higher than those that didn't.
- Similar to time spent, the more the daily internet usage is, the less likely the customer will click on an ad.

Categorical features

```
cats1=cats.copy()
cats1.remove('Clicked on Ad')
df_temp = df_eda.copy()
plt.figure(figsize=(20,14))
for i in range(len(cats1)):
    df_total = df_temp.groupby(cats1[i])['ID'].count().reset_index().rename(columns={'ID':'total'})
    df_subtotal = df_temp.groupby([cats1[i], 'Clicked on Ad'])['ID'].count().reset_index().rename(columns={'ID':'subtotal'})
    dfm = df_subtotal.merge(df_total, on=cats1[i])
    dfm['Percentage'] = round(dfm['subtotal']/dfm['total']*100, 2)

    plt.subplot(2, 2, i+1)
    sns.barplot(x = cats1[i], y='Percentage', data = dfm, palette = ['#de1a00', '#06bdd1'], hue='Clicked on Ad')
    plt.xticks(rotation=90)
    plt.xlabel(cats1[i], fontsize=12, fontweight = 'bold')
    plt.ylabel('Percentage (%)', fontsize=12, fontweight = 'bold')
    plt.title(f'{cats1[i]}', fontsize=14, fontweight='bold', pad=15)
    plt.tight_layout()
```

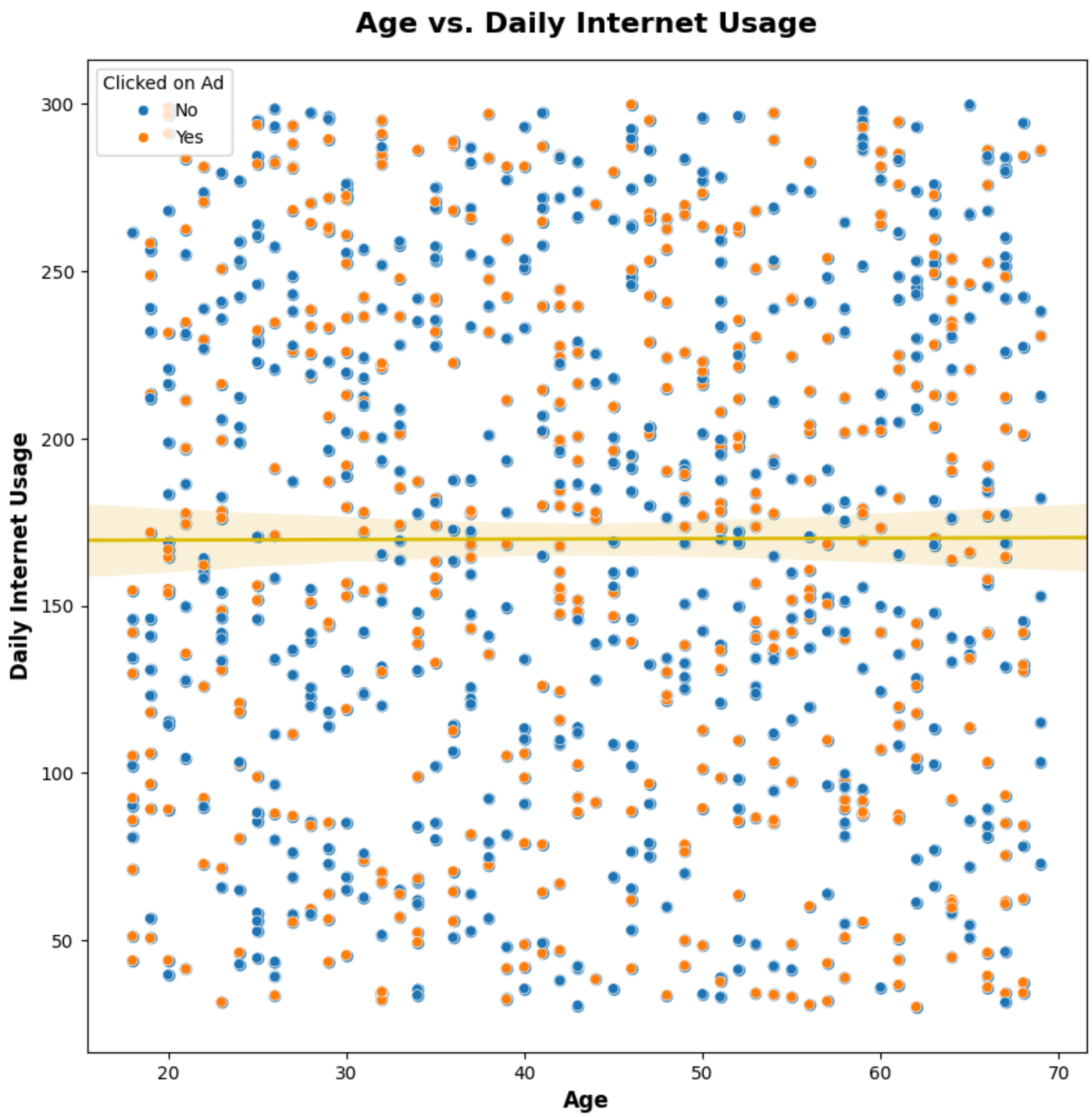


Analysis:

- Females clicked on an ad slightly more than males overall.
- The city with the highest click rate is Serang with 81%, while the city with the lowest is Jakarta Pusat with 26%.
- The top 3 provinces with the highest click rates are Kalimantan Selatan, Banten, Sumatra Barat.
- Ad categories' click rates are pretty equal with none below 40% and none above 60%.

Age vs. Daily Internet Usage

```
plt.figure(figsize=(10,10))
sns.regplot(data=df_eda, x="Age", y="Daily Internet Usage", truncate=False, line_kws={"linewidth": 2, 'color': '#deba04'})
sns.scatterplot(x = 'Age', y = 'Daily Internet Usage', data = df_eda, hue='Clicked on Ad')
plt.ylabel('Daily Internet Usage', fontsize=12, fontweight = 'bold')
plt.xlabel('Age', fontsize=12, fontweight = 'bold')
plt.title('Age vs. Daily Internet Usage', fontsize=16, fontweight = 'bold', pad = 15)

plt.show()
```

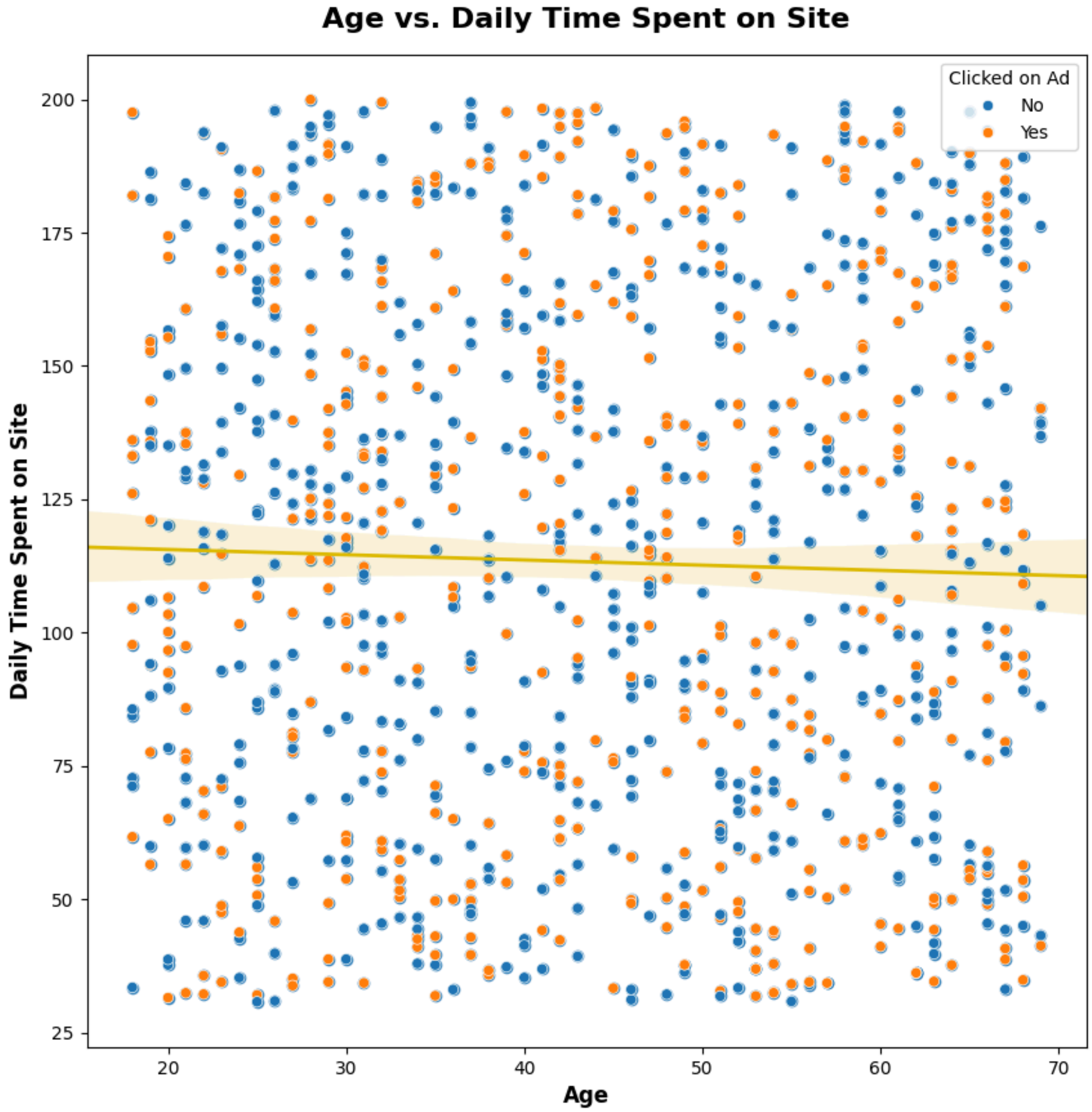


Analysis:

Age is slightly negatively correlated with Daily Internet Usage. Older customers spend less time on the internet on average compared to younger customers.

Age vs. Daily Time Spent on Site

```
plt.figure(figsize=(10,10))
sns.regplot(data=df_eda, x="Age", y="Daily Time Spent on Site", truncate=False, line_kws={"linewidth": 2, 'color': '#deba04'})
sns.scatterplot(x = 'Age', y = 'Daily Time Spent on Site', data = df_eda, hue='Clicked on Ad')
plt.ylabel('Daily Time Spent on Site', fontsize=12, fontweight = 'bold')
plt.xlabel('Age', fontsize=12, fontweight = 'bold')
plt.title('Age vs. Daily Time Spent on Site', fontsize=16, fontweight = 'bold', pad = 15)

plt.show()
```

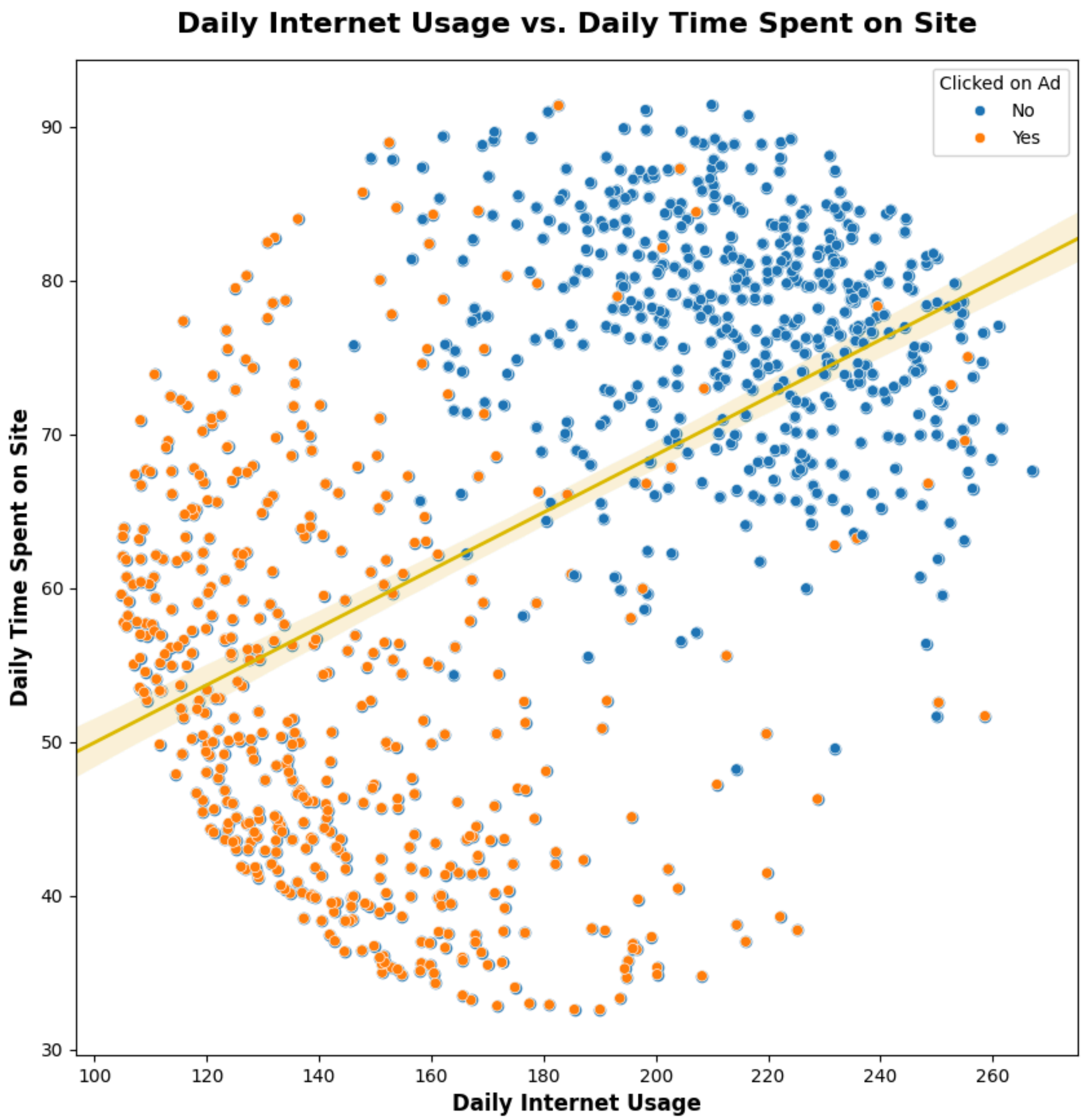


Analysis: CHARLOTTE DATASET:

Same as with Daily Internet Usage, Age is slightly negatively correlated with Daily Time Spent on Site.

Daily Internet Usage vs. Daily Time Spent on Site

```
plt.figure(figsize=(10,10))
sns.regplot(data=df_eda, x="Daily Internet Usage", y="Daily Time Spent on Site", truncate=False, line_kws={"linewidth": 2, 'color': '#deba04
sns.scatterplot(x = 'Daily Internet Usage', y = 'Daily Time Spent on Site', data = df_eda, hue='Clicked on Ad')
plt.ylabel('Daily Time Spent on Site', fontsize=12, fontweight = 'bold')
plt.xlabel('Daily Internet Usage', fontsize=12, fontweight = 'bold')
plt.title('Daily Internet Usage vs. Daily Time Spent on Site', fontsize=16, fontweight = 'bold', pad = 15)

plt.show()
```



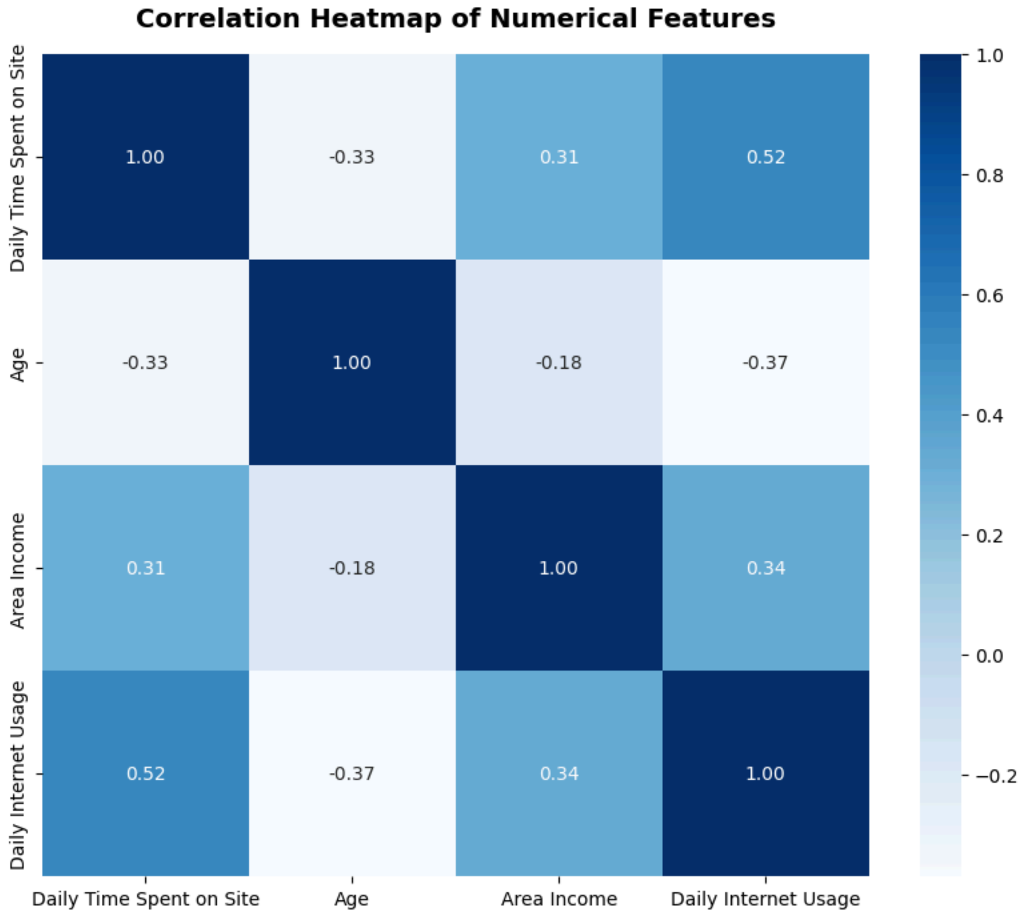
Analysis:

Internet usage is positively correlated with time spent on site. As can be seen from the above chart, there is a quite clear separation between two clusters of data. One cluster is less active and the other more so. Less active customers have a higher tendency to click on an ad compared to more active customers.

Multivariate analysis

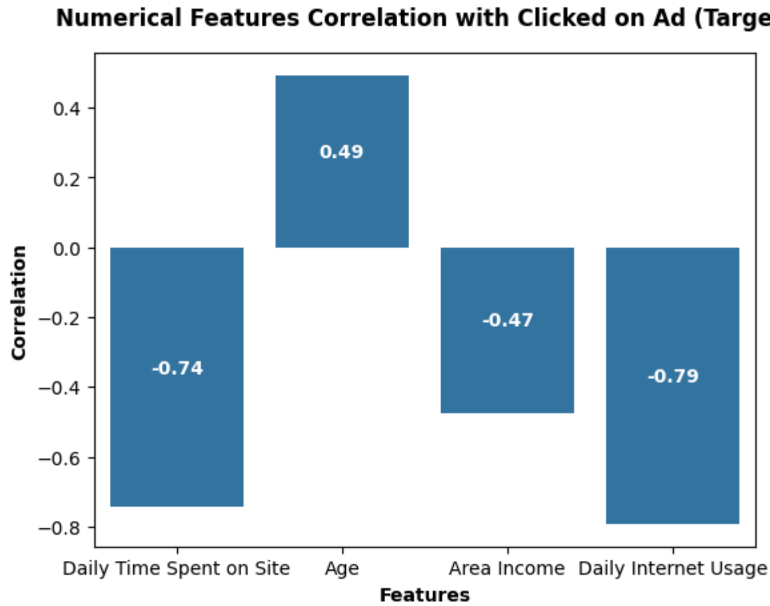
Correlation heatmap of numerical features

```
plt.figure(figsize=(10, 8))
sns.heatmap(df_eda[nums].corr(), cmap='Blues', annot=True, fmt='.2f')
plt.title('Correlation Heatmap of Numerical Features', fontsize=14, fontweight='bold', pad=15)
plt.show()
```



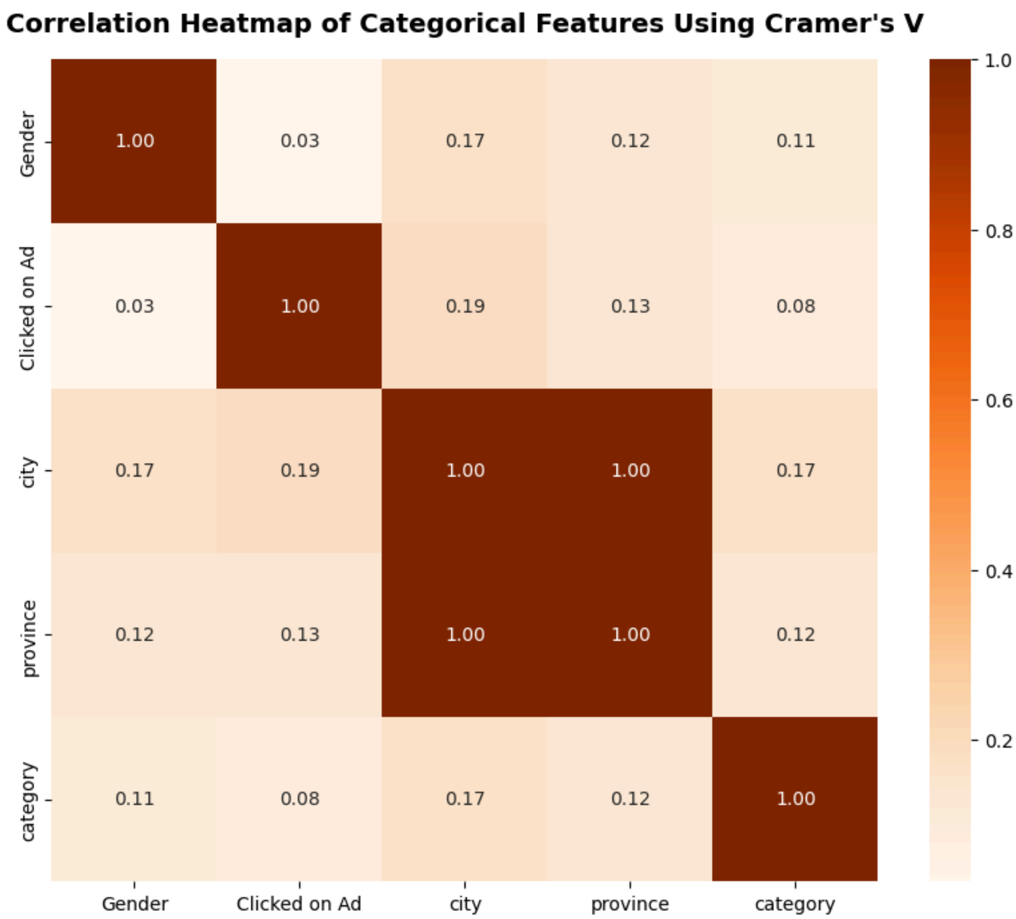
Numerical features' correlation with target

```
correlation = []
df_pb = df_eda.copy()
df_pb['Clicked on Ad'] = np.where(df_pb['Clicked on Ad'] == 'Yes', 1, 0)
df_pb.dropna(inplace=True)
for i in range(len(nums)):
  corr, p = stats.pointbiserialr(df_pb['Clicked on Ad'], df_pb[nums[i]])
  vals = [nums[i], corr]
  correlation.append(vals)

df_corr = pd.DataFrame(data = correlation, columns=['Features', 'Correlation'])
bar = sns.barplot(x='Features', y='Correlation', data=df_corr)
plt.ylabel('Correlation', fontsize=10, fontweight = 'bold')
plt.xlabel('Features', fontsize=10, fontweight = 'bold')
plt.title('Numerical Features Correlation with Clicked on Ad (Target)', fontsize=12, fontweight = 'bold', pad = 15)

for i in bar.patches:
  height = i.get_height()
  width = i.get_width()
  x = i.get_x()
  y = i.get_y()
  bar.annotate(f'{round(height, 2)}', (x + width/2, y + (height/2)), ha='center', va='bottom', color = 'white', fontweight = 'bold')

plt.show()
```



Categorical features' correlation (Cramer's V)

```
def cramers_v(var1, var2):
    data = pd.crosstab(var1, var2).values
    chi_2 = stats.chi2_contingency(data)[0]
    n = data.sum()
    phi_2 = chi_2 / n
    r, k = data.shape
    return np.sqrt(phi_2 / min((k-1), (r-1)))


crv=[]
for i in range(len(cats)):
  row=[]
  for j in range(len(cats)):
    val = cramers_v(df_eda[cats[i]], df_eda[cats[j]])
    row.append(val)
  crv.append(row)
df_crv = pd.DataFrame(data=crv, columns=cats, index=cats)
df_crv

plt.figure(figsize=(10, 8))

sns.heatmap(df_crv, cmap='Oranges', annot=True, fmt='.2f')
plt.title("Correlation Heatmap of Categorical Features Using Cramer's V", fontsize=14, fontweight='bold', pad=15)
plt.show()
```



Analysis

Numerical Correlations:

- **Daily Time Spent on Site and Daily Internet Usage** (0.52):
  These two variables have a relatively strong positive correlation. This means that as people use the internet more, they tend to spend more time on the site on a daily basis. This could indicate that the website is engaging and users spend more time using it while online.
- **Daily Time Spent on Site and Age** (-0.33):
  There is a moderate negative correlation between age and the time spent on the site. Older users tend to spend less time on the site. This could suggest that younger individuals may be more active on the site.
- **Daily Time Spent on Site and Area Income** (0.31):
  There is a moderate positive correlation between daily time spent on the site and area income. This could mean that people with higher area income levels spend more time on the site, although the correlation is not very strong.
- **Age and Daily Internet Usage** (-0.37):
  There is a moderate negative correlation between age and daily internet usage. Older individuals tend to use the internet less on a daily basis. This might be because younger individuals are more likely to be digitally active.
- **Age and Area Income** (-0.18):
  There is a weak negative correlation between age and area income. This suggests that older individuals tend to have slightly lower area income, but the correlation is not strong at all.
- **Area Income and Daily Internet Usage** (0.34):
  There is a moderate positive correlation between area income and daily internet usage. People with higher area incomes tend to use the internet more on a daily basis. This could be because they have better access to technology and higher-speed internet connections.

Categorical Correlations (Cramer's V):

- **Gender and City** (0.17):
  There is a moderate positive association between gender and city. This suggests that there may be some relationship between a user's gender and the city in which they are located. However, the association is not particularly strong.

- **Gender and Province** (0.12):
  There is a moderate positive association between gender and province. This implies that a user's gender might be somewhat related to their province of residence, but again, the association is not very strong.
- **Gender and Category** (0.11):
  There is a moderate positive association between gender and category. This indicates that a user's gender might have some influence on the category they are interested in, but the association is not very strong.
- **City and Province** (1.0):
  The perfect correlation coefficient of 1 indicates that city and province are perfectly associated. This is likely due to the dataset structure and may not provide meaningful information about the relationship between these variables.
- **City and Category** (0.17):
  There is a moderate positive association between city and category. This suggests that a user's city may be related to the category they are interested in, but the association is not very strong.
- **Province and Category** (0.12):
  There is a moderate positive association between province and category. This implies that the province of a user may have some influence on the category they are interested in, although the association is not very strong.

Correlations with Target Variable:

- **Gender and Clicked on Ad** (0.03):
  There is a very weak positive association between gender and whether a user clicked on the ad.
- **Clicked on Ad and City** (0.19):
  There is a moderate positive association between whether a user clicked on the ad and their city. This suggests that the city of the user may be somewhat related to their likelihood of clicking on the ad.
- **Clicked on Ad and Province** (0.13):
  There is a moderate positive association between whether a user clicked on the ad and their province. This implies that a user's province might have some influence on their likelihood of clicking on the ad.
- **Clicked on Ad and Category** (0.08):
  There is a very weak positive association between whether a user clicked on the ad and the category.
- **Daily Time Spent on Site** (-0.74):
  The negative correlation coefficient of -0.74 indicates a strong negative relationship between "Daily Time Spent on Site" and the likelihood of a user clicking on the ad. This suggests that as users spend more time on the site, they are less likely to click on the ad. This could mean that users who spend a lot of time on the site might be more engaged with the content and less likely to click on ads.
- **Age** (0.49):
  The positive correlation coefficient of 0.49 suggests a moderate positive relationship between a user's age and the likelihood of clicking on the ad. In other words, older individuals are more likely to click on the ad.
- **Area Income** (-0.47):
  The negative correlation coefficient of -0.47 indicates a moderate negative relationship between "Area Income" and the likelihood of clicking on the ad. Users in areas with lower income levels are more likely to click on the ad.
- **Daily Internet Usage** (-0.79):
  The negative correlation coefficient of -0.79 suggests a strong negative relationship between "Daily Internet Usage" and the likelihood of clicking on the ad. Users who spend more time on the internet are less likely to click on the ad. This could imply that users who are more active internet users might be less responsive to online advertisements.

Data Cleaning & Preprocessing

```
df1 = df.copy()
```

```
# Creating list of numerical and categorical columns
nums = [col for col in df1.columns if (df1[col].dtype == 'int64' or df1[col].dtype == 'float64') and col != 'Unnamed: 0' ]
cats = [col for col in df1.columns if df1[col].dtype == 'object' and col != 'Timestamp']
```

Handling missing values

```
result = []
for col in df1.columns:
    result.append([col, df1[col].dtype, df1[col].isna().sum(), 100*df1[col].isna().sum()/len(df1[col])])

output = pd.DataFrame(data=result, columns = 'column data_type no._null percent_null'.split())
output
```

	column	data_type	no._null	percent_null
0	Unnamed: 0	int64	0	0.0
1	Daily Time Spent on Site	float64	13	1.3
2	Age	int64	0	0.0
3	Area Income	float64	13	1.3
4	Daily Internet Usage	float64	11	1.1
5	Male	object	3	0.3
6	Timestamp	object	0	0.0
7	Clicked on Ad	object	0	0.0
8	city	object	0	0.0
9	province	object	0	0.0
10	category	object	0	0.0

There are 4 features with null values; `Daily Time Spent on Site`, `Area Income`, `Daily Internet Usage`, `Male`.

Numerical Features

```
df1[nums].describe()
```

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage
count	987.000000	1000.000000	9.870000e+02	989.000000
mean	64.929524	36.009000	3.848647e+08	179.863620
std	15.844699	8.785562	9.407999e+07	43.870142
min	32.600000	19.000000	9.797550e+07	104.780000
25%	51.270000	29.000000	3.286330e+08	138.710000
50%	68.110000	35.000000	3.990683e+08	182.650000
75%	78.460000	42.000000	4.583554e+08	218.790000
max	91.430000	61.000000	5.563936e+08	267.010000

- **By looking at the Univariate Analysis in the previous section, the feature with missing values and a skewed distribution is `Area Income`. This feature's null values will therefore be imputed using the `median`.**
- **The rest of the numerical features with null values will be imputed using the `mean`.**

Categorical Features

```
df1[cats].describe()
```

	Male	Clicked on Ad	city	province	category
count	997	1000	1000	1000	1000
unique	2	2	30	16	10
top	Perempuan	No	Surabaya	Daerah Khusus Ibukota Jakarta	Otomotif
freq	518	500	64	253	112

The null values in the `Male` feature will be imputed using the `mode`.

Imputing null values

```
# Imputing numerical features
df1['Area Income'].fillna(df1['Area Income'].median(), inplace=True)
df1['Daily Time Spent on Site'].fillna(df1['Daily Time Spent on Site'].mean(), inplace=True)
df1['Daily Internet Usage'].fillna(df1['Daily Internet Usage'].mean(), inplace=True)

# Imputing categorical features
df1['Male'].fillna(df1['Male'].mode()[0], inplace=True)

# Checking result
print(f'Total null values in dataset: {df1.isna().sum().sum()}')
```

```
Total null values in dataset: 0
```

Handling duplicated data

```
print(f'Dataset contains duplicated values: {df1.duplicated().any()}')
print(f'Number of duplicates present in dataset: {df1.duplicated().sum()}')
```

```
Dataset contains duplicated values: False
Number of duplicates present in dataset: 0
```

The Dataset does not have `duplicates`.

Feature Engineering

Feature Extraction

```
# Extracting Year, Month, Week and Day from Timestamp feature

# Changing data type of Timestamp feature into datetime
df1['Timestamp'] = pd.to_datetime(df1['Timestamp'])

# Extracting Year
df1['Year'] = df1.Timestamp.dt.year

# Extracting Month
df1['Month'] = df1.Timestamp.dt.month

# Extracting Week
df1['Week'] = df1.Timestamp.dt.isocalendar().week

# Extracting Day
df1['Day'] = df1.Timestamp.dt.dayofweek

df1.sample(5)
```

	Unnamed: 0	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Male	Timestamp	Clicked on Ad	city	province	category	Year	Month	Week	Day
866	866	86.58	32	421062390.0	195.93	Laki-Laki	2016-02-26 23:44:00	No	Jakarta Barat	Daerah Khusus Ibukota Jakarta	Electronic	2016	2	8	4
53	53	50.33	50	438602710.0	133.20	Laki-Laki	2016-03-02 04:57:00	Yes	Banjarmasin	Kalimantan Selatan	House	2016	3	9	2
444	444	32.84	40	288630230.0	171.72	Perempuan	2016-03-10 01:36:00	Yes	Palembang	Sumatra Selatan	Bank	2016	3	10	3
587	587	43.83	45	249793740.0	129.01	Perempuan	2016-01-29 05:39:00	Yes	Tasikmalaya	Jawa Barat	Finance	2016	1	4	4
							2016-02			Daerah					

```
df1[df1['Week']==53].head(5)
```

	Unnamed: 0	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Male	Timestamp	Clicked on Ad	city	province	category	Year	Month	Week	Day
132	132	51.24	36	534578170.0	176.73	Perempuan	2016-01-03 16:01:00	Yes	Malang	Jawa Timur	Health	2016	1	53	6
180	180	39.85	38	219403730.0	145.96	Perempuan	2016-01-03 03:22:00	Yes	Semarang	Jawa Tengah	Fashion	2016	1	53	6
190	190	50.08	30	291409020.0	123.91	Perempuan	2016-01-03 05:34:00	Yes	Jakarta Timur	Daerah Khusus Ibukota Jakarta	Furniture	2016	1	53	6
337	337	75.32	28	419989500.0	233.60	Laki-Laki	2016-01-01 21:58:00	No	Bekasi	Jawa Barat	Fashion	2016	1	53	4

As can be seen from the above, `Week` has the value 53 in it, even though the dataset only has data up until `Month` 7. This is a consequence of how **ISO** week numbering works. Therefore `Week` 53 will be converted to `Week` 0, as to preserve the order of the `Week` feature.

Note: `Day` is Monday to Sunday, with 0 being Monday and 6 Sunday.

```
df1['Week'] = np.where(df1['Week'] == 53, 0, df1['Week'])
df1['Week'] = df1['Week'].astype(int)
df1[df1['Week']==0].head(3)
```

	Unnamed: 0	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Male	Timestamp	Clicked on Ad	city	province	category	Year	Month	Week	Day
132	132	51.24	36	534578170.0	176.73	Perempuan	2016-01-03	Yes	Malang	Jawa Timur	Health	2016	1	0	6
180	180	39.85	38	219403730.0	145.96	Perempuan	03 03:22:00	Yes	Semarang	Jawa Tengah	Fashion	2016	1	0	6
190	190	50.08	30	291409020.0	123.91	Perempuan	2016-01-03	Yes	Jakarta	Daerah Khusus	Furniture	2016	1	0	6

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 15 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Unnamed: 0                1000 non-null   int64
 1   Daily Time Spent on Site  1000 non-null   float64
 2   Age                       1000 non-null   int64
 3   Area Income               1000 non-null   float64
 4   Daily Internet Usage      1000 non-null   float64
 5   Male                      1000 non-null   object
 6   Timestamp                 1000 non-null   datetime64[ns]
```

```
 7   Clicked on Ad            1000 non-null   object
 8   city                     1000 non-null   object
 9   province                 1000 non-null   object
 10  category                 1000 non-null   object
 11  Year                     1000 non-null   int64
 12  Month                    1000 non-null   int64
 13  Week                     1000 non-null   int64
 14  Day                      1000 non-null   int64
dtypes: datetime64[ns](1), float64(3), int64(6), object(5)
memory usage: 117.3+ KB
```

Changing inappropriate column names

```
df1.rename(columns = {'Unnamed: 0':'ID', 'Male':'Gender'}, inplace = True)
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 15 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   ID                       1000 non-null   int64
 1   Daily Time Spent on Site 1000 non-null   float64
 2   Age                      1000 non-null   int64
 3   Area Income              1000 non-null   float64
 4   Daily Internet Usage     1000 non-null   float64
 5   Gender                   1000 non-null   object
 6   Timestamp                1000 non-null   datetime64[ns]
 7   Clicked on Ad            1000 non-null   object
 8   city                     1000 non-null   object
 9   province                 1000 non-null   object
 10  category                 1000 non-null   object
 11  Year                     1000 non-null   int64
 12  Month                    1000 non-null   int64
 13  Week                     1000 non-null   int64
 14  Day                      1000 non-null   int64
dtypes: datetime64[ns](1), float64(3), int64(6), object(5)
memory usage: 117.3+ KB
```

```
# Creating list of numerical and categorical columns
nums = [col for col in df1.columns if (df1[col].dtype == 'int64' or df1[col].dtype == 'float64') and col != 'ID' and col != 'Year' and col !
cats = [col for col in df1.columns if df1[col].dtype == 'object']
```

Handling Outliers

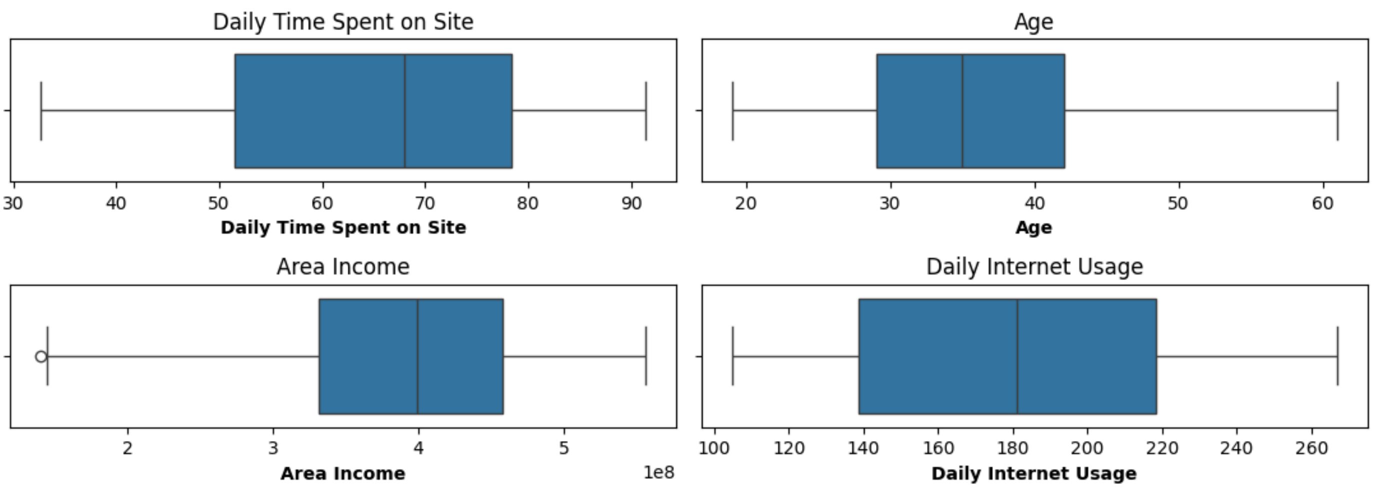
From the Univariate Analysis it can be seen that the only numerical feature with outliers is `Area Income`. Therefore only said feature will have its outliers handled (using the IQR Method).

```
# Trimming outliers using the IQR method
print(f'Number of rows prior to filtering: {len(df1)}')

q1 = df1['Area Income'].quantile(0.25)
q3 = df1['Area Income'].quantile(0.75)
iqr = q3 - q1
low = q1 - (1.5 * iqr)
upper = q3 + (1.5 * iqr)
df1 = df1[(df1['Area Income']>=low)&(df1['Area Income']<=upper)]

print(f'Number of rows after filtering: {len(df1)}')
```

```
Number of rows prior to filtering: 1000
Number of rows after filtering: 991
```

Checking boxplot

```
plt.figure(figsize=(11, 4))
for i in range(len(nums)):
    plt.subplot(2, 2, i+1)
    sns.boxplot(x = df1[nums[i]])
    plt.xlabel(nums[i], fontsize=10, fontweight = 'bold')
    plt.title(f'{nums[i]}')
    plt.tight_layout()
```



As can be seen above, `Area Income` has no glaring outliers remaining.

Feature Encoding

```
df_enc = df1.copy()
df_enc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 991 entries, 0 to 999
Data columns (total 15 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   ID                        991 non-null    int64
 1   Daily Time Spent on Site  991 non-null    float64
 2   Age                       991 non-null    int64
 3   Area Income               991 non-null    float64
 4   Daily Internet Usage      991 non-null    float64
 5   Gender                    991 non-null    object
 6   Timestamp                 991 non-null    datetime64[ns]
 7   Clicked on Ad             991 non-null    object
 8   city                      991 non-null    object
 9   province                  991 non-null    object
 10  category                  991 non-null    object
 11  Year                      991 non-null    int64
 12  Month                     991 non-null    int64
 13  Week                      991 non-null    int64
 14  Day                       991 non-null    int64
dtypes: datetime64[ns](1), float64(3), int64(6), object(5)
memory usage: 123.9+ KB
```

The `Year`, `Month`, `Week` and `Day` features have already been encoded as integers. Therefore only the features with `object` data type will be encoded.

Label encoding

Only `Gender` and `Clicked on Ad` features will be label encoded.

```
df_enc['Gender'].value_counts()
```

```
Perempuan    518
Laki-Laki    473
Name: Gender, dtype: int64
```

```
# Label encoding Gender feature
df_enc['Gender'] = np.where(df_enc['Gender'] == 'Laki-Laki', 1, 0)
df_enc['Gender'].value_counts()
```

```
0    518
1    473
Name: Gender, dtype: int64
```

```
df_enc['Clicked on Ad'].value_counts()
```

```
No     500
Yes    491
Name: Clicked on Ad, dtype: int64
```

```
# Label encoding Target feature
df_enc['Clicked on Ad'] = np.where(df_enc['Clicked on Ad'] == 'Yes', 1, 0)
df_enc['Clicked on Ad'].value_counts()
```

```
0    500
1    491
Name: Clicked on Ad, dtype: int64
```

One-Hot Encoding

```
print(f"Unique values of category: {df_enc['category'].nunique()}")
print(f"Unique values of category: {df_enc['city'].nunique()}")
print(f"Unique values of category: {df_enc['province'].nunique()}")
```

```
Unique values of category: 10
Unique values of category: 30
Unique values of category: 16
```

To avoid dimensionality problems the only feature that will be One-Hot encoded is the `category` feature. The rest will be discarded later to downscale our project implementation.

```
pd.set_option('display.max_columns', None)
```

```
# One hot encoding
df_enc = pd.get_dummies(df_enc, columns=['category'])
df_enc.head(3)
```

	ID	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Gender	Timestamp	Clicked on Ad	city	province	Year	Month	Week	Day	category_Bank	category_
0	0	68.95	35	432837300.0	256.09	0	2016-03-27 00:53:00	0	Jakarta Timur	Daerah Khusus Ibukota Jakarta	2016	3	12	6	0	
1	1	80.23	31	479092950.0	193.77	1	2016-04-04 01:39:00	0	Denpasar	Bali	2016	4	14	0	0	
2	2	69.47	26	418501580.0	236.50	0	2016-03-13 20:35:00	0	Surabaya	Jawa Timur	2016	3	10	6	0	

```
df_enc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 991 entries, 0 to 999
Data columns (total 24 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   ID                        991 non-null    int64
 1   Daily Time Spent on Site  991 non-null    float64
 2   Age                       991 non-null    int64
 3   Area Income               991 non-null    float64
 4   Daily Internet Usage      991 non-null    float64
```

```
 5   Gender                  991 non-null    int64
 6   Timestamp               991 non-null    datetime64[ns]
 7   Clicked on Ad           991 non-null    int64
 8   city                    991 non-null    object
 9   province                991 non-null    object
 10  Year                    991 non-null    int64
 11  Month                   991 non-null    int64
 12  Week                    991 non-null    int64
 13  Day                     991 non-null    int64
 14  category_Bank           991 non-null    uint8
 15  category_Electronic     991 non-null    uint8
 16  category_Fashion        991 non-null    uint8
 17  category_Finance        991 non-null    uint8
 18  category_Food           991 non-null    uint8
 19  category_Furniture      991 non-null    uint8
 20  category_Health         991 non-null    uint8
 21  category_House          991 non-null    uint8
 22  category_Otomotif       991 non-null    uint8
 23  category_Travel         991 non-null    uint8
dtypes: datetime64[ns](1), float64(3), int64(8), object(2), uint8(10)
memory usage: 125.8+ KB
```

Feature selection

- **`ID` will be discarded because it is an index column.**
- **`Timestamp` will be discarded because its values have already been extracted.**
- **`city` will be discarded because of high number of unique values.**
- **`province` will similarly be discarded because of high number of unique values.**
- **`Year` will be discarded because its value is constant (2016).**

```
df_clean = df_enc.select_dtypes(['float64', 'int64','uint8'])
df_clean = df_clean.drop(columns=['ID', 'Year'])
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 991 entries, 0 to 999
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Daily Time Spent on Site  991 non-null    float64
 1   Age                     991 non-null    int64
 2   Area Income             991 non-null    float64
 3   Daily Internet Usage    991 non-null    float64
 4   Gender                  991 non-null    int64
 5   Clicked on Ad           991 non-null    int64
 6   Month                   991 non-null    int64
 7   Week                    991 non-null    int64
 8   Day                     991 non-null    int64
 9   category_Bank           991 non-null    uint8
 10  category_Electronic     991 non-null    uint8
 11  category_Fashion        991 non-null    uint8
 12  category_Finance        991 non-null    uint8
 13  category_Food           991 non-null    uint8
 14  category_Furniture      991 non-null    uint8
 15  category_Health         991 non-null    uint8
 16  category_House          991 non-null    uint8
 17  category_Otomotif       991 non-null    uint8
 18  category_Travel         991 non-null    uint8
dtypes: float64(3), int64(6), uint8(10)
memory usage: 87.1 KB
```

Splitting dataset

An experiment will be conducted in the modeling section where each ML model will be implemented using 2 different data, one normalized data and the other non-normalized data. Therefore the dataset will be cloned into 2 identical copies where one will be normalized and the other won't be.

```
X = df_clean.drop(columns='Clicked on Ad')
y = df_clean['Clicked on Ad'].values
X1 = X.copy()
y1 = y.copy()
```

```
X2 = X.copy()
y2 = y.copy()
```

NOTE :

- **`X1` and `y1` : Data without normalization/standardization**
- **`X2` and `y2` : Data that will be normalized/standardized**

Data will be split in 75:25 ratio, 75% train set and 25% test set

```
from sklearn.model_selection import train_test_split

X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.25, random_state=42)
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.25, random_state=42)

print(f'Train set size: {X1_train.shape[0]}')
print(f'Test set size: {X1_test.shape[0]}')
```

```
Train set size: 743
Test set size: 248
```

Feature scaling

Feature scaling will only be applied to `X2_train` and `X2_test`

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
X2_train_scaled = X2_train.copy()

for n in nums:
  scaler = ss.fit(X2_train_scaled[[n]])
  X2_train_scaled[n] = scaler.transform(X2_train_scaled[[n]])
  X2_test[n] = scaler.transform(X2_test[[n]])
```

```
X2_train_scaled[nums].describe().T
```

	count	mean	std	min	25%	50%	75%	max
Daily Time Spent on Site	743.0	-6.395363e-16	1.000674	-2.033329	-0.851087	0.196149	0.851934	1.678052
Age	743.0	-3.335152e-16	1.000674	-1.908301	-0.797032	-0.130271	0.758744	2.647900
Area Income	743.0	9.563159e-18	1.000674	-2.636284	-0.640776	0.149032	0.780451	1.902248
Daily Internet Usage	743.0	6.072606e-16	1.000674	-1.710447	-0.901793	0.026712	0.887430	1.860936

As can be seen above, all of the numerical features have 0 `mean` and 1 `std`. This means that the standardization was succesful, however it was successful with a demo dataset of common sensible values.

Modeling

Before normalization/standardization

Helper functions

```
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_validate
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix
from sklearn.pipeline import Pipeline
import time
def eval_classification1(model):
    y_pred = model.predict(X1_test)
    y_pred_train = model.predict(X1_train)
```

```
    y_pred_proba = model.predict_proba(X1_test)
    y_pred_proba_train = model.predict_proba(X1_train)

    print("Accuracy (Test Set): %.2f" % accuracy_score(y1_test, y_pred))
    print("Accuracy (Train Set): %.2f" % accuracy_score(y1_train, y_pred_train))
    print("Precision (Test Set): %.2f" % precision_score(y1_test, y_pred, zero_division=0))
    print("Recall (Test Set): %.2f" % recall_score(y1_test, y_pred))
    print("Recall (Train Set): %.2f" % recall_score(y1_train, y_pred_train))
    print("F1-Score (Test Set): %.2f" % f1_score(y1_test, y_pred))

    print("roc_auc (test-proba): %.2f" % roc_auc_score(y1_test, y_pred_proba[:, 1]))
    print("roc_auc (train-proba): %.2f" % roc_auc_score(y1_train, y_pred_proba_train[:, 1]))

    cv = RepeatedStratifiedKFold(random_state=42, n_repeats = 3)
    score = cross_validate(model, X=X1_train, y=y1_train, cv=cv, scoring='accuracy', return_train_score=True)
    print('Accuracy (crossval train): '+ str(score['train_score'].mean()))
    print('Accuracy (crossval test): '+ str(score['test_score'].mean()))


def grid_pipe1(pipedict, hyperdict, scoring='accuracy', display=True):
    fitted_models1={}
    fit_time1 = []
    for name, pipeline in pipedict.items():
    # Construct grid search
        cv = RepeatedStratifiedKFold(random_state=42, n_repeats = 3)
        model = GridSearchCV(estimator=pipeline,
                             param_grid=hyperdict[name],
                             scoring=scoring,
                             cv=cv, verbose=2, n_jobs=-1, return_train_score = True, error_score='raise')

        # Fit using grid search
        start = time.time()
        model.fit(X1_train, y1_train)
        end = time.time()
        fit_time1.append(round(end-start, 2))
        #Append model
        fitted_models1[name]=model
        if display:
            #Print when the model has been fitted
            print(f'The {name} model has been fitted.')
            # print fit time
            print('Total Fit Time: %.3fs' % (end-start))
            # Best accuracy
            print('Best accuracy: %.3f' % model.best_score_)
            # Best params
            print('Best params:\n', model.best_params_,'\n')

    return fitted_models1, fit_time1

def confusion1(model):
    y_pred_proba = model.predict_proba(X1_test)
    y_predict = model.predict(X1_test)
    print('Accuracy: %.2f%%' % (accuracy_score(y1_test, y_predict) * 100 ))
    print('Precision: %.2f%%' % (precision_score(y1_test, y_predict, zero_division=0) * 100))
    print('Recall: %.2f%%' % (recall_score(y1_test, y_predict) * 100))
    print('F1_Score: %.2f%%' % (f1_score(y1_test, y_predict) * 100))
    print('ROC_AUC: %.2f%%' % (roc_auc_score(y1_test, y_pred_proba[:,1]) * 100))
    confusion_matrix_model = confusion_matrix(y1_test, y_predict)
    plt.figure(figsize=(12,8))
    ax = plt.subplot()
    sns.heatmap(confusion_matrix_model, annot=True, fmt='g', ax = ax)
    ax.set_xlabel('Predicted Label')
    ax.set_ylabel('Actual Label')
    ax.set_title(f'Confusion Matrix - {model}')
    ax.xaxis.set_ticklabels(['0','1'])
    ax.yaxis.set_ticklabels(['0','1'])
```

Vanilla Models

Logistic Regression

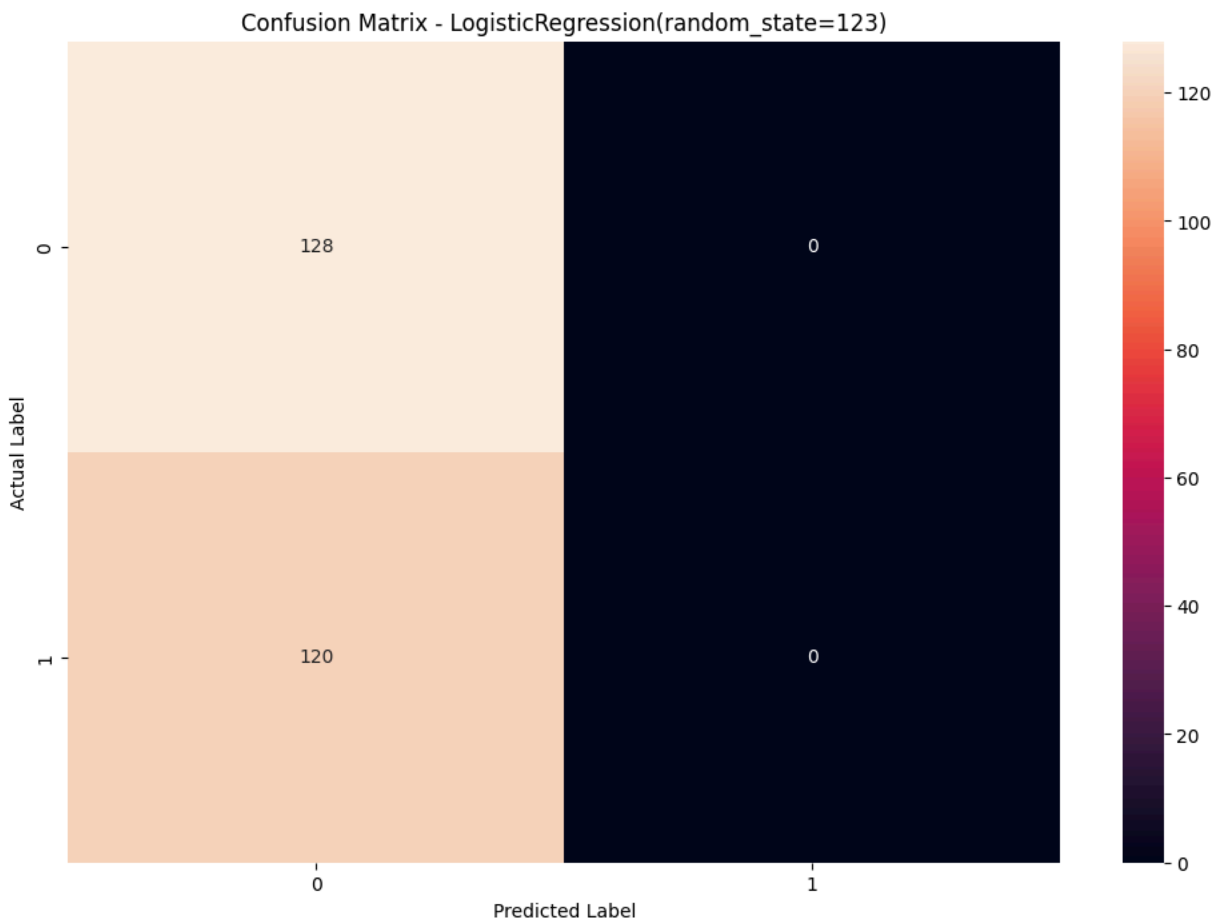
```
from sklearn.linear_model import LogisticRegression
```

```
logreg1 = LogisticRegression(random_state=123)
logreg1.fit(X1_train, y1_train)

eval_classification1(logreg1)
```

```
Accuracy (Test Set): 0.52
Accuracy (Train Set): 0.50
Precision (Test Set): 0.00
Recall (Test Set): 0.00
Recall (Train Set): 0.00
F1-Score (Test Set): 0.00
roc_auc (test-proba): 0.70
roc_auc (train-proba): 0.79
Accuracy (crossval train): 0.5006728347904819
Accuracy (crossval test): 0.5006711409395973
```

```
confusion1(logreg1)
```

```
Accuracy: 51.61%
Precision: 0.00%
Recall: 0.00%
F1_Score: 0.00%
ROC_AUC: 69.82%
```



Decision Tree

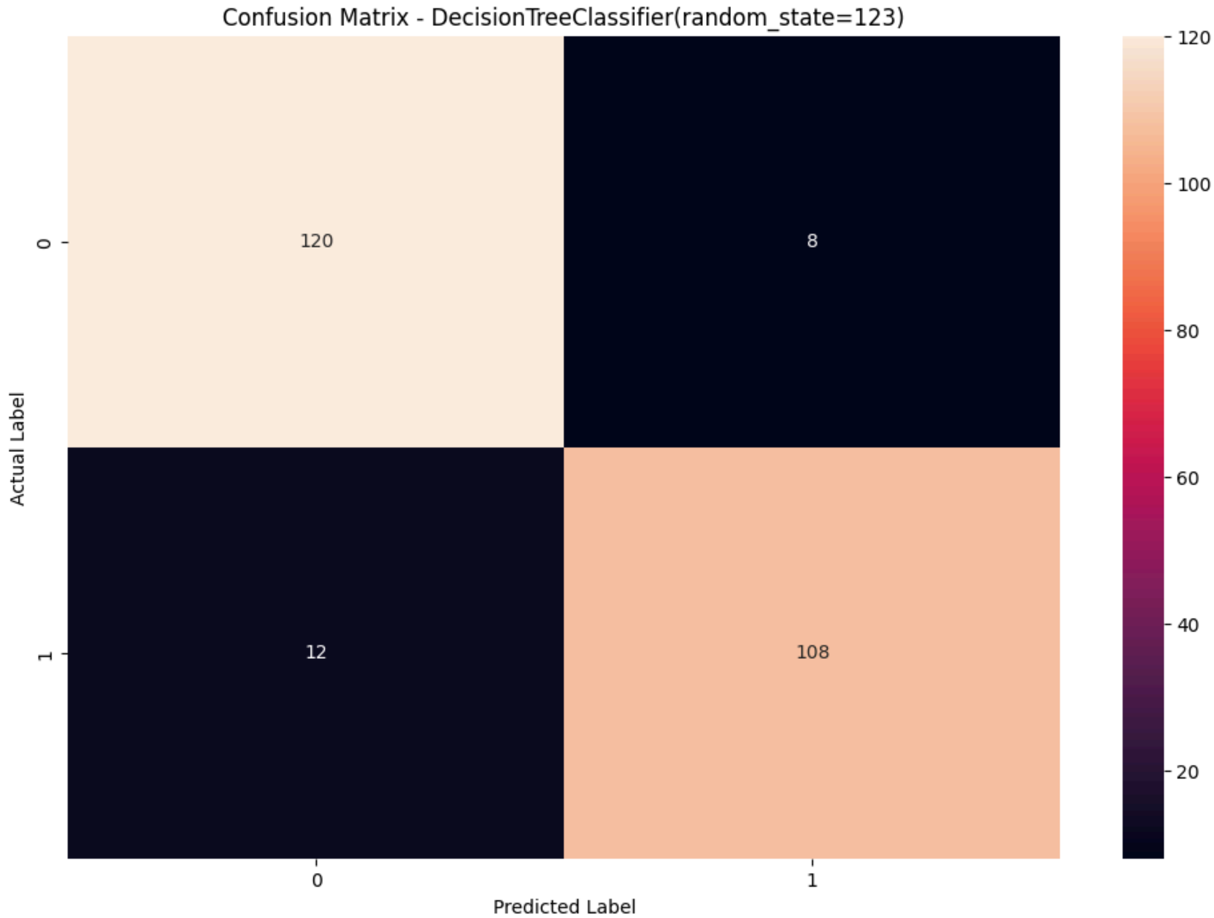
```
from sklearn.tree import DecisionTreeClassifier

dt1 = DecisionTreeClassifier(random_state=123)
dt1.fit(X1_train, y1_train)

eval_classification1(dt1)
```

```
Accuracy (Test Set): 0.92
Accuracy (Train Set): 1.00
Precision (Test Set): 0.93
Recall (Test Set): 0.90
Recall (Train Set): 1.00
F1-Score (Test Set): 0.92
roc_auc (test-proba): 0.92
```

```
roc_auc (train-proba): 1.00
Accuracy (crossval train): 1.0
Accuracy (crossval test): 0.9390017534312836
```

```
confusion1(dt1)
```

```
Accuracy: 91.94%
Precision: 93.10%
Recall: 90.00%
F1_Score: 91.53%
ROC_AUC: 91.88%
```



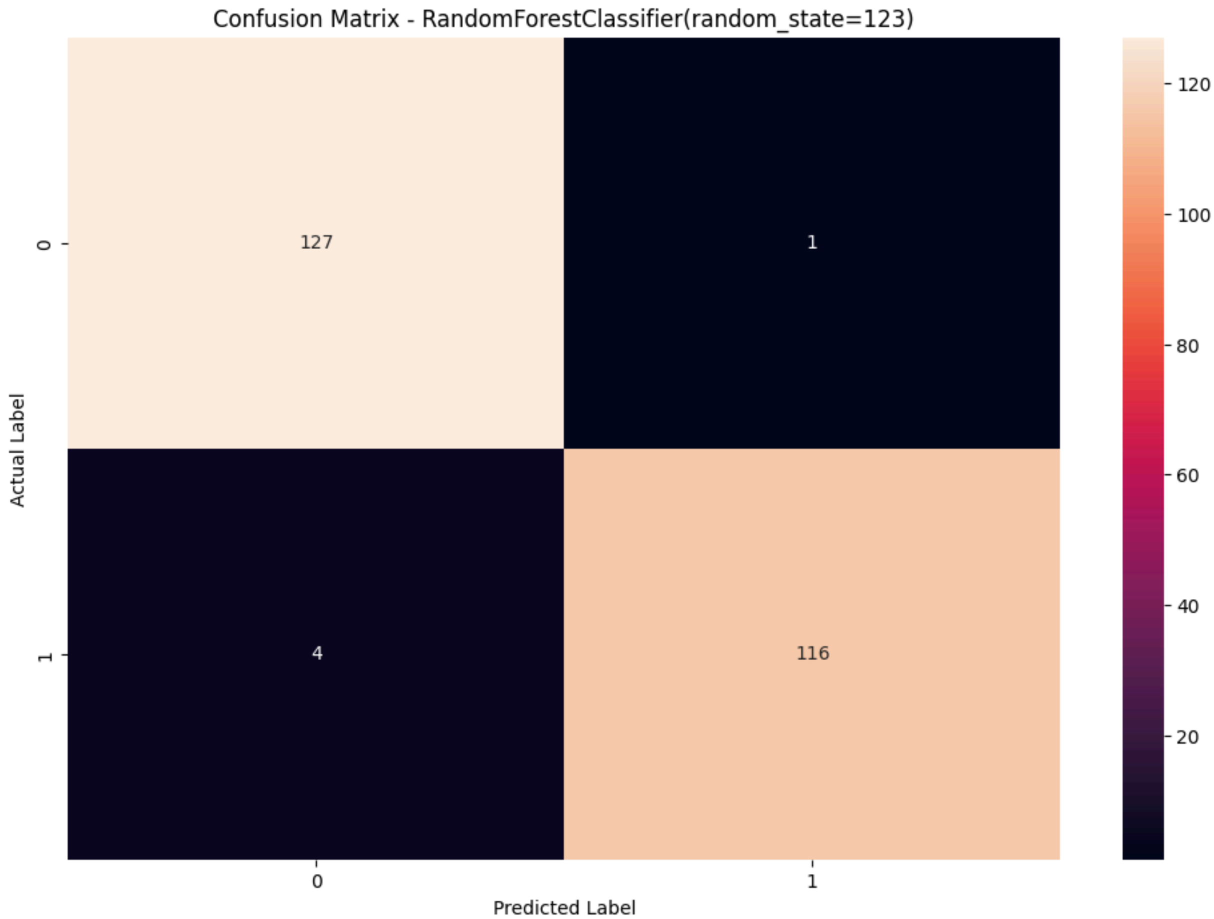
Random Forest

```
from sklearn.ensemble import RandomForestClassifier

rf1 = RandomForestClassifier(random_state=123)
rf1.fit(X1_train, y1_train)

eval_classification1(rf1)
```

```
Accuracy (Test Set): 0.98
Accuracy (Train Set): 1.00
Precision (Test Set): 0.99
Recall (Test Set): 0.97
Recall (Train Set): 1.00
F1-Score (Test Set): 0.98
roc_auc (test-proba): 0.99
roc_auc (train-proba): 1.00
Accuracy (crossval train): 1.0
Accuracy (crossval test): 0.9600852530382732
```

```
confusion1(rf1)
```

```
Accuracy: 97.98%
Precision: 99.15%
Recall: 96.67%
F1_Score: 97.89%
ROC_AUC: 98.95%
```



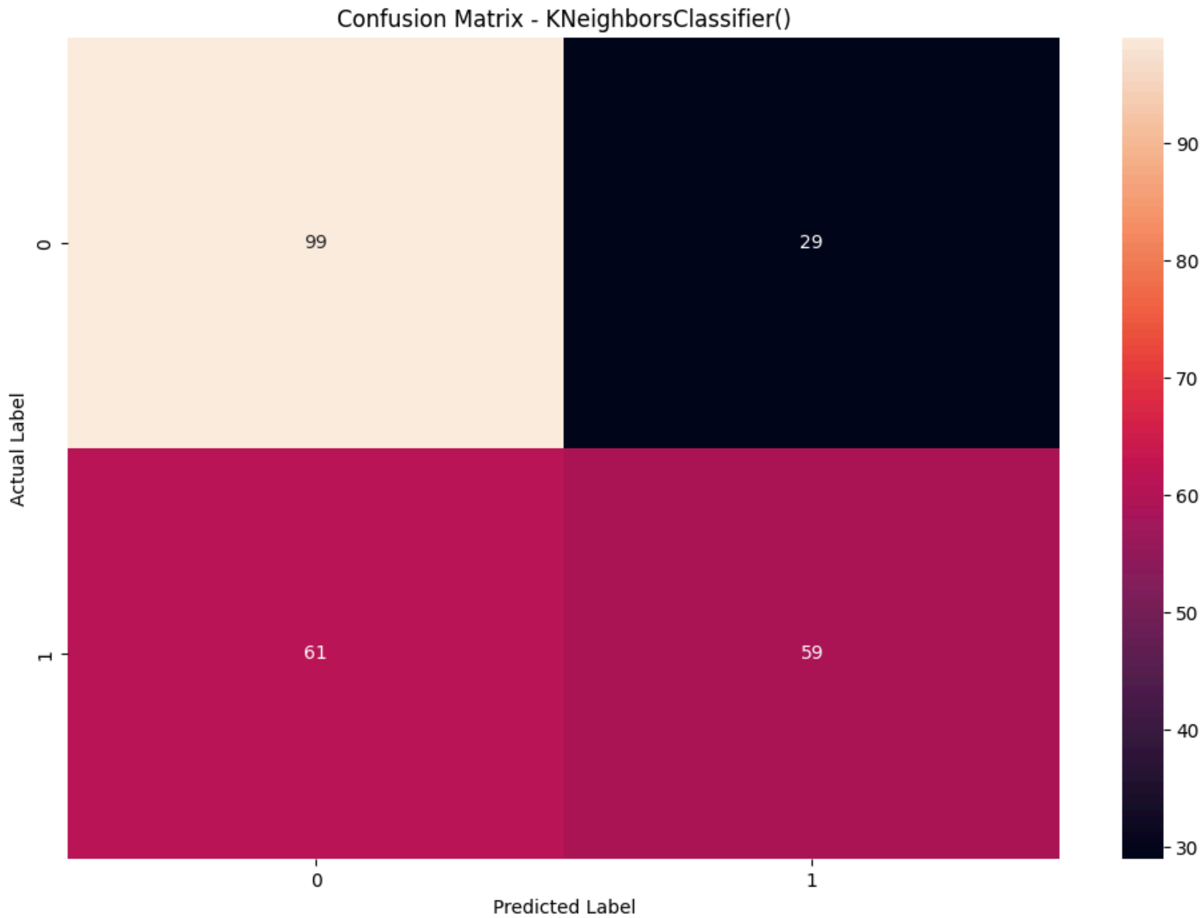
K-Nearest Neighbours

```
from sklearn.neighbors import KNeighborsClassifier

knn1 = KNeighborsClassifier()
knn1.fit(X1_train, y1_train)

eval_classification1(knn1)
```

```
Accuracy (Test Set): 0.64
Accuracy (Train Set): 0.77
Precision (Test Set): 0.67
Recall (Test Set): 0.49
Recall (Train Set): 0.73
F1-Score (Test Set): 0.57
roc_auc (test-proba): 0.64
roc_auc (train-proba): 0.86
Accuracy (crossval train): 0.7752305501325111
Accuracy (crossval test): 0.6828375355220992
```

```
confusion1(knn1)
```

```
Accuracy: 63.71%
Precision: 67.05%
Recall: 49.17%
F1_Score: 56.73%
ROC_AUC: 64.43%
```



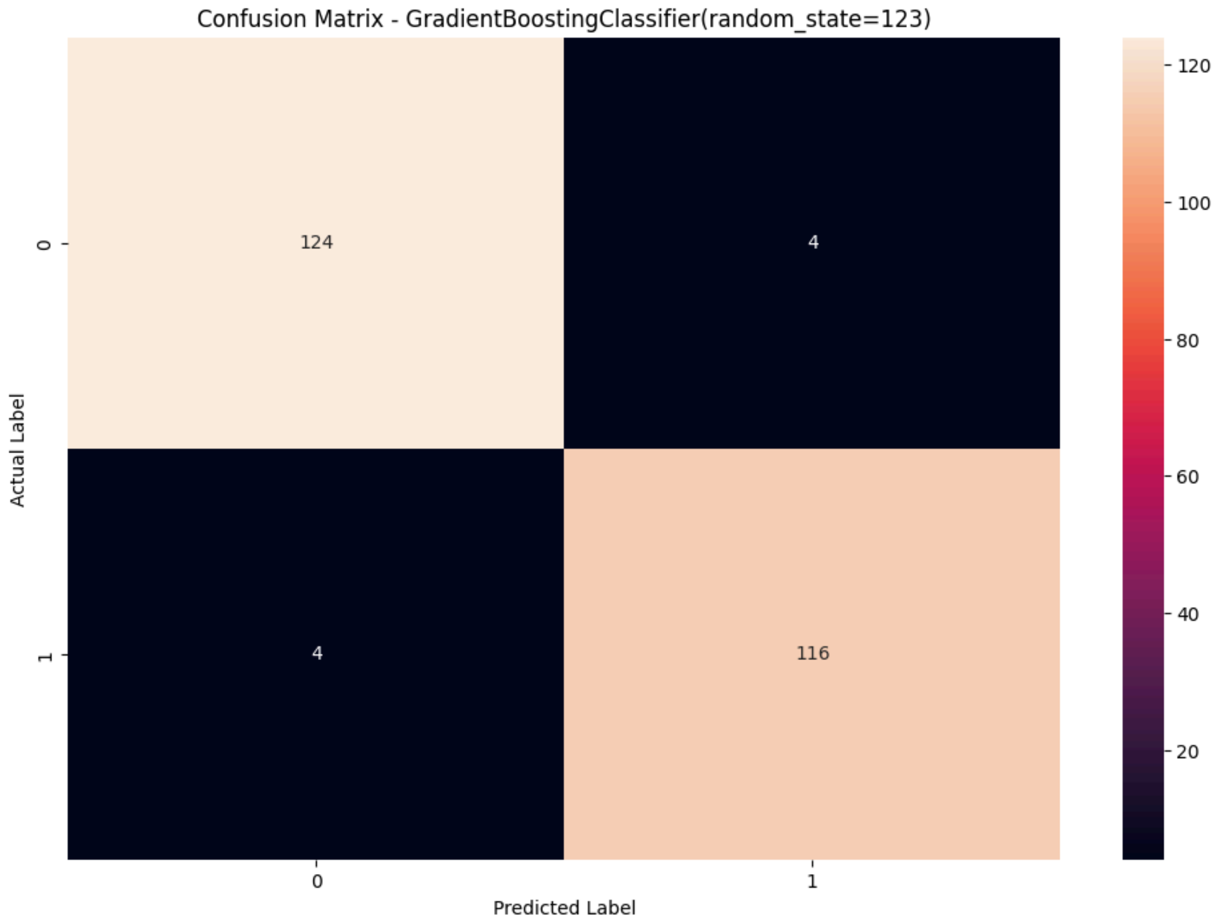
Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier

gb1 = GradientBoostingClassifier(random_state=123)
gb1.fit(X1_train, y1_train)

eval_classification1(gb1)
```

```
Accuracy (Test Set): 0.97
Accuracy (Train Set): 1.00
Precision (Test Set): 0.97
Recall (Test Set): 0.97
Recall (Train Set): 1.00
F1-Score (Test Set): 0.97
roc_auc (test-proba): 0.99
roc_auc (train-proba): 1.00
Accuracy (crossval train): 0.9998879551820729
Accuracy (crossval test): 0.9529082774049217
```

```
confusion1(gb1)
```

```
Accuracy: 96.77%
Precision: 96.67%
Recall: 96.67%
F1_Score: 96.67%
ROC_AUC: 98.63%
```



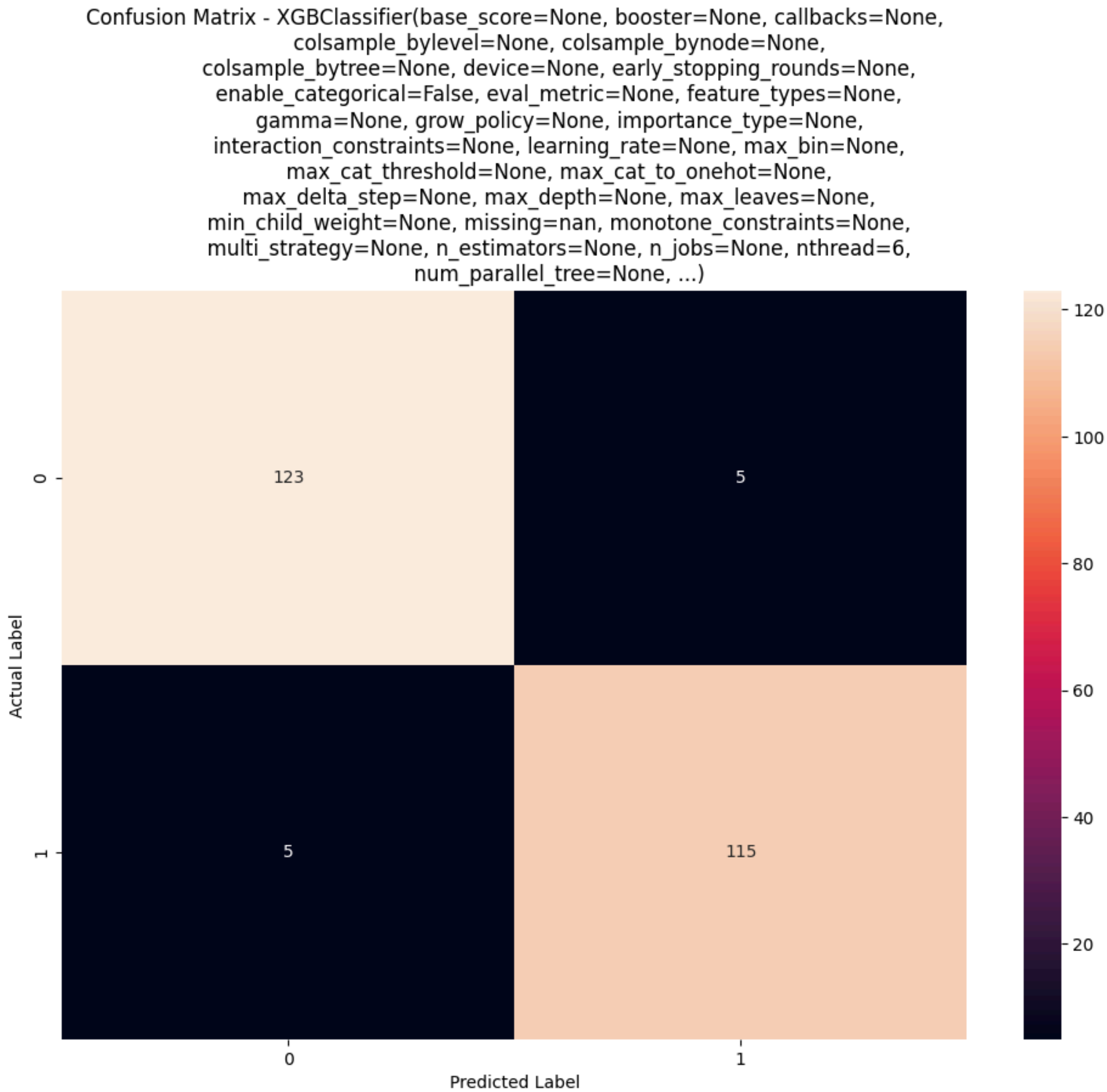
XGBoost

```
from xgboost import XGBClassifier
xgb1 = XGBClassifier(nthread=6, tree_method='hist', random_state=123)

xgb1.fit(X1_train, y1_train)

eval_classification1(xgb1)
```

```
Accuracy (Test Set): 0.96
Accuracy (Train Set): 1.00
Precision (Test Set): 0.96
Recall (Test Set): 0.96
Recall (Train Set): 1.00
F1-Score (Test Set): 0.96
roc_auc (test-proba): 0.99
roc_auc (train-proba): 1.00
Accuracy (crossval train): 1.0
Accuracy (crossval test): 0.9502176673317613
```

```
confusion1(xgb1)
```

```
Accuracy: 95.97%
Precision: 95.83%
Recall: 95.83%
F1_Score: 95.83%
ROC_AUC: 98.98%
```



As can be seen in the above section, overfitting exists in a few of the models tested. Hyperparameter tuning will be done to sort this issue.

Hyperparameter Tuning

```
#Create pipeline for each of the classifiers.
pipelines = {'logisticregression1': Pipeline([('clf', LogisticRegression(random_state=123))]),
           'decisiontree1':Pipeline([('clf', DecisionTreeClassifier(random_state = 123))]),
           'randomforest1':Pipeline([('clf', RandomForestClassifier(random_state = 123))]),
           'knn1':Pipeline([('clf', KNeighborsClassifier())]),
           'gb1':Pipeline([('clf', GradientBoostingClassifier(random_state = 123))]),
           'xgboost1': Pipeline([('clf', XGBClassifier(nthread=6, tree_method='hist', random_state=123))])}

#Define Hyperparameters for each pipeline
hyperparameters_lr1 ={'clf__C': [float(x) for x in np.linspace(0.002, 1, 100)],
                    'clf__penalty' : ['l2'],
                    'clf__solver' : ['newton-cg', 'lbfgs', 'newton-cholesky', 'liblinear'],
                    'clf__max_iter':[10000]}

hyperparameters_dt1 ={'clf__criterion' : ['entropy', 'gini'],
                    'clf__max_depth' : [int(x) for x in np.linspace(1, 20, 20)],
                    'clf__min_samples_split' : [int(x) for x in np.linspace(start = 2, stop = 50, num = 5)],
```

```
                     'clf__min_samples_leaf' : [int(x) for x in np.linspace(start = 2, stop = 50, num = 5)],
                     'clf__max_features' : ['sqrt'],
                     'clf__splitter' : ['best']}

hyperparameters_rf1 ={'clf__n_estimators': [50,60,75, 100, 120],
                     'clf__criterion': ['entropy', 'gini'],
                     'clf__max_features':['sqrt' , None],
                     'clf__min_samples_leaf':[0.05, 0.1, 0.2]}

hyperparameters_knn1 ={'clf__n_neighbors' : list(range(1,30)),
                      'clf__weights' : ['uniform'],
                      'clf__p' : [1, 2],
                      'clf__algorithm' : ['auto', 'ball_tree', 'kd_tree', 'brute']}

hyperparameters_gb1 ={'clf__n_estimators' : [int(x) for x in np.linspace(10, 50, num = 5)],
                     'clf__criterion' : ['friedman_mse', 'squared_error'],
                     'clf__max_depth' : [1, 2, 3],
                     'clf__min_samples_split' : [2, 3, 5],
                     'clf__min_samples_leaf' : [2, 3, 5],
                     'clf__max_features' : ['sqrt'],
                     'clf__loss' : ['exponential']}

hyperparameters_xgb1 ={'clf__eta': [float(x) for x in np.linspace(0.1, 0.7, 20)],
                     'clf__max_depth': [1,3,5]}

#Instantiate hyperparapeter dictionary
hyperparameters = {'logisticregression1':hyperparameters_lr1,
                   'decisiontree1':hyperparameters_dt1,
                   'randomforest1':hyperparameters_rf1,
                   'knn1':hyperparameters_knn1,
                   'gb1':hyperparameters_gb1,
                   'xgboost1': hyperparameters_xgb1}


fitted_models1, fit_time1 = grid_pipe1(pipelines,hyperparameters,scoring='accuracy')
```

```
Fitting 15 folds for each of 400 candidates, totalling 6000 fits
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:425: LineSearchWarning: Rounding errors prevent the line search fr
  warn(msg, LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:203: UserWarning: Line Search failed
  warnings.warn("Line Search failed")
The logisticregression1 model has been fitted.
Total Fit Time: 216.717s
Best accuracy: 0.933
Best params:
 {'clf__C': 0.002, 'clf__max_iter': 10000, 'clf__penalty': 'l2', 'clf__solver': 'newton-cg'}

Fitting 15 folds for each of 1000 candidates, totalling 15000 fits
The decisiontree1 model has been fitted.
Total Fit Time: 105.064s
Best accuracy: 0.907
Best params:
 {'clf__criterion': 'gini', 'clf__max_depth': 8, 'clf__max_features': 'sqrt', 'clf__min_samples_leaf': 2, 'clf__min_samples_split': 26,

Fitting 15 folds for each of 60 candidates, totalling 900 fits
The randomforest1 model has been fitted.
Total Fit Time: 137.173s
Best accuracy: 0.948
Best params:
 {'clf__criterion': 'entropy', 'clf__max_features': 'sqrt', 'clf__min_samples_leaf': 0.05, 'clf__n_estimators': 75}

Fitting 15 folds for each of 232 candidates, totalling 3480 fits
The knn1 model has been fitted.
Total Fit Time: 182.964s
Best accuracy: 0.721
Best params:
 {'clf__algorithm': 'auto', 'clf__n_neighbors': 23, 'clf__p': 1, 'clf__weights': 'uniform'}

Fitting 15 folds for each of 270 candidates, totalling 4050 fits
The gb1 model has been fitted.
Total Fit Time: 138.110s
Best accuracy: 0.960
Best params:
 {'clf__criterion': 'friedman_mse', 'clf__loss': 'exponential', 'clf__max_depth': 3, 'clf__max_features': 'sqrt', 'clf__min_samples_leaf

Fitting 15 folds for each of 60 candidates, totalling 900 fits
The xgboost1 model has been fitted.
```

```
Total Fit Time: 92.668s
Best accuracy: 0.960
Best params:
 {'clf__eta': 0.22631578947368422, 'clf__max_depth': 1}
```

After Hyperparameter Tuning

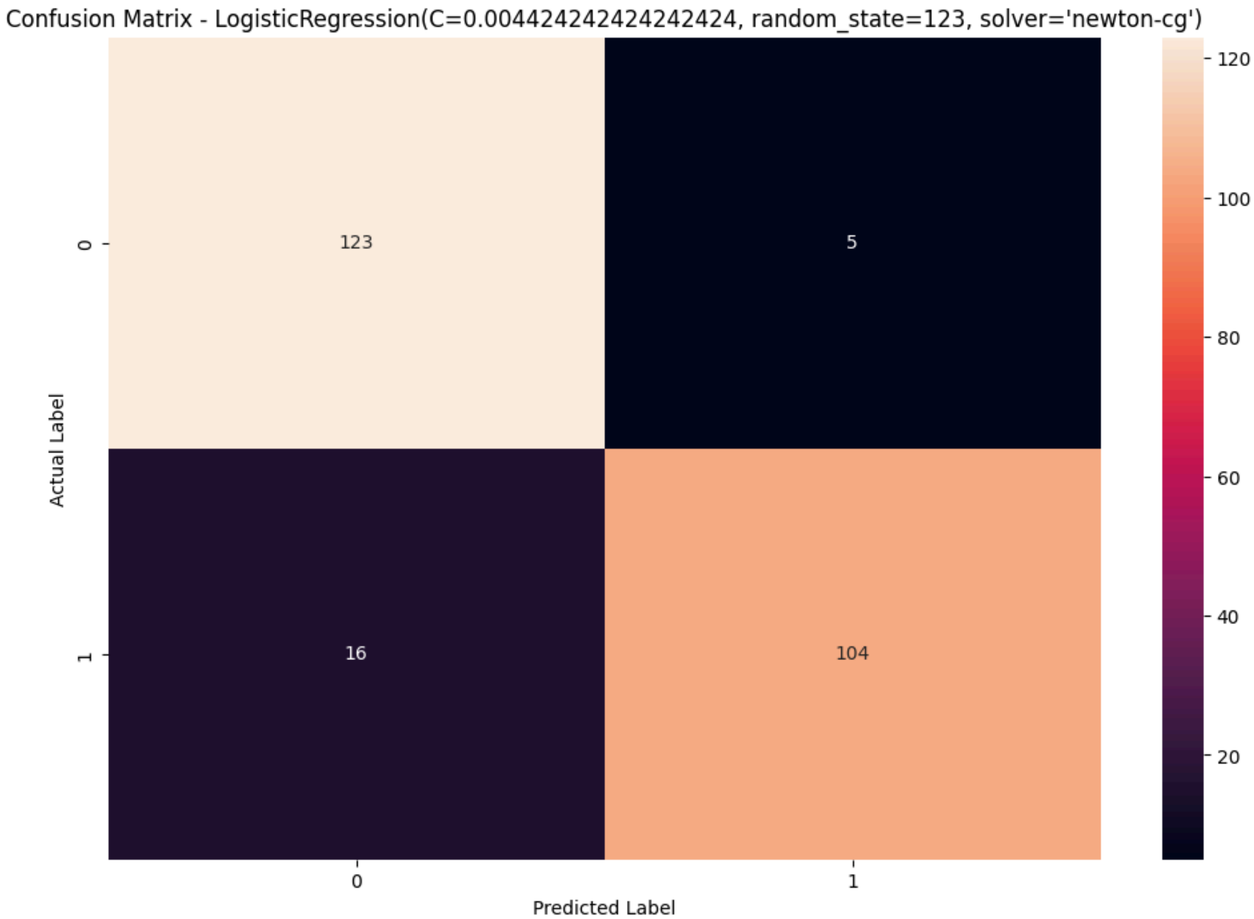
Logistic Regression

```
logreg1_tuned = LogisticRegression(random_state=123, C = 0.004424242424242424, penalty = 'l2', solver = 'newton-cg')
logreg1_tuned.fit(X1_train, y1_train)

eval_classification1(logreg1_tuned)
```

```
Accuracy (Test Set): 0.92
Accuracy (Train Set): 0.90
Precision (Test Set): 0.95
Recall (Test Set): 0.87
Recall (Train Set): 0.87
F1-Score (Test Set): 0.91
roc_auc (test-proba): 0.96
roc_auc (train-proba): 0.96
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not conve
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not conve
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:425: LineSearchWarning: Rounding errors prevent the line search
  warn(msg, LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:203: UserWarning: Line Search failed
  warnings.warn("Line Search failed")
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not conve
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not conve
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:210: ConvergenceWarning: newton-cg failed to converge. Increase the
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not conve
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not conve
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:425: LineSearchWarning: Rounding errors prevent the line search
  warn(msg, LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:203: UserWarning: Line Search failed
  warnings.warn("Line Search failed")
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not conve
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not conve
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:210: ConvergenceWarning: newton-cg failed to converge. Increase the
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not conve
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not conve
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:210: ConvergenceWarning: newton-cg failed to converge. Increase the
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not conve
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not conve
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:210: ConvergenceWarning: newton-cg failed to converge. Increase the
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not conve
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not conve
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:425: LineSearchWarning: Rounding errors prevent the line search
  warn(msg, LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:203: UserWarning: Line Search failed
  warnings.warn("Line Search failed")
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not conve
```

```
confusion1(logreg1_tuned)
```

```
Accuracy: 91.53%
Precision: 95.41%
Recall: 86.67%
F1_Score: 90.83%
ROC_AUC: 96.45%
```



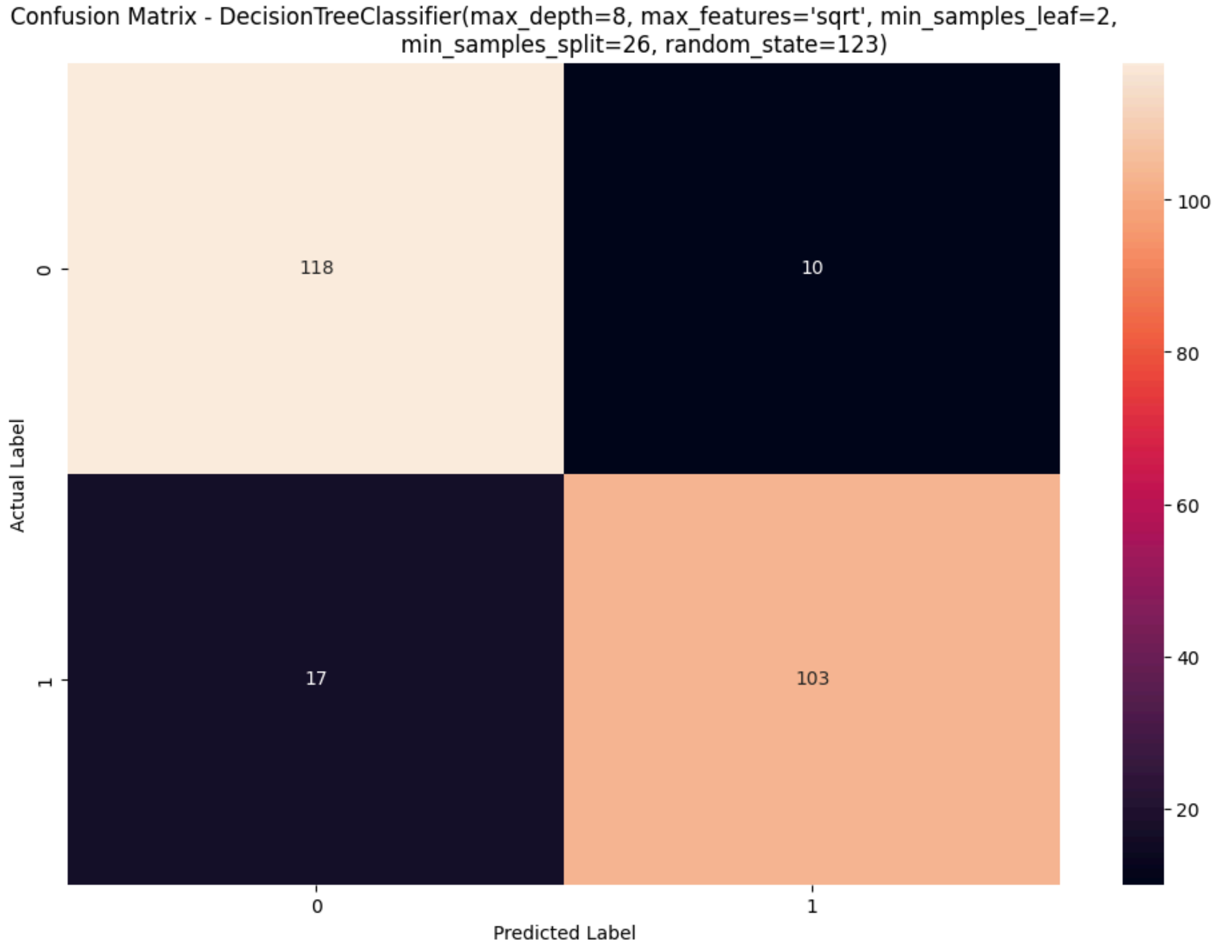
Decision Tree

```
dt1_tuned = DecisionTreeClassifier(random_state=123, criterion = 'gini', max_depth = 8, max_features = 'sqrt', min_samples_leaf = 2, min_sam
dt1_tuned.fit(X1_train, y1_train)

eval_classification1(dt1_tuned)
```

```
Accuracy (Test Set): 0.89
Accuracy (Train Set): 0.95
Precision (Test Set): 0.91
Recall (Test Set): 0.86
Recall (Train Set): 0.93
F1-Score (Test Set): 0.88
roc_auc (test-proba): 0.94
roc_auc (train-proba): 0.99
Accuracy (crossval train): 0.9324848862103766
Accuracy (crossval test): 0.9071618598464235
```

```
confusion1(dt1_tuned)
```

```
Accuracy: 89.11%
Precision: 91.15%
Recall: 85.83%
F1_Score: 88.41%
ROC_AUC: 94.15%
```



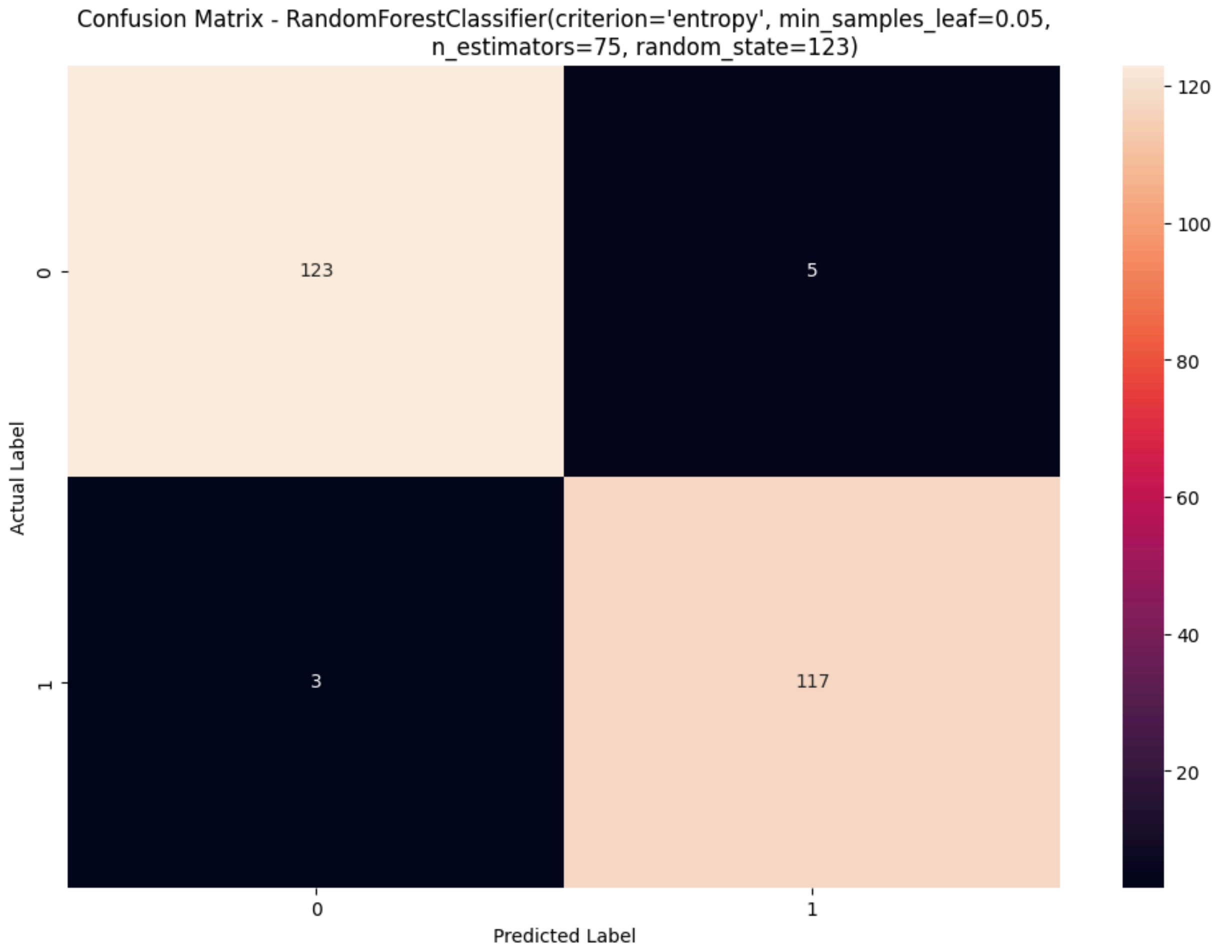
Random Forest

```
rf1_tuned = RandomForestClassifier(random_state=123, criterion = 'entropy', max_features = 'sqrt', min_samples_leaf = 0.05, n_estimators = 7
rf1_tuned.fit(X1_train, y1_train)

eval_classification1(rf1_tuned)
```

```
Accuracy (Test Set): 0.97
Accuracy (Train Set): 0.96
Precision (Test Set): 0.96
Recall (Test Set): 0.97
Recall (Train Set): 0.96
F1-Score (Test Set): 0.97
roc_auc (test-proba): 0.99
roc_auc (train-proba): 0.99
Accuracy (crossval train): 0.959511643041055
Accuracy (crossval test): 0.9484158655299594
```

```
confusion1(rf1_tuned)
```

```
Accuracy: 96.77%
Precision: 95.90%
Recall: 97.50%
F1_Score: 96.69%
ROC_AUC: 98.86%
```



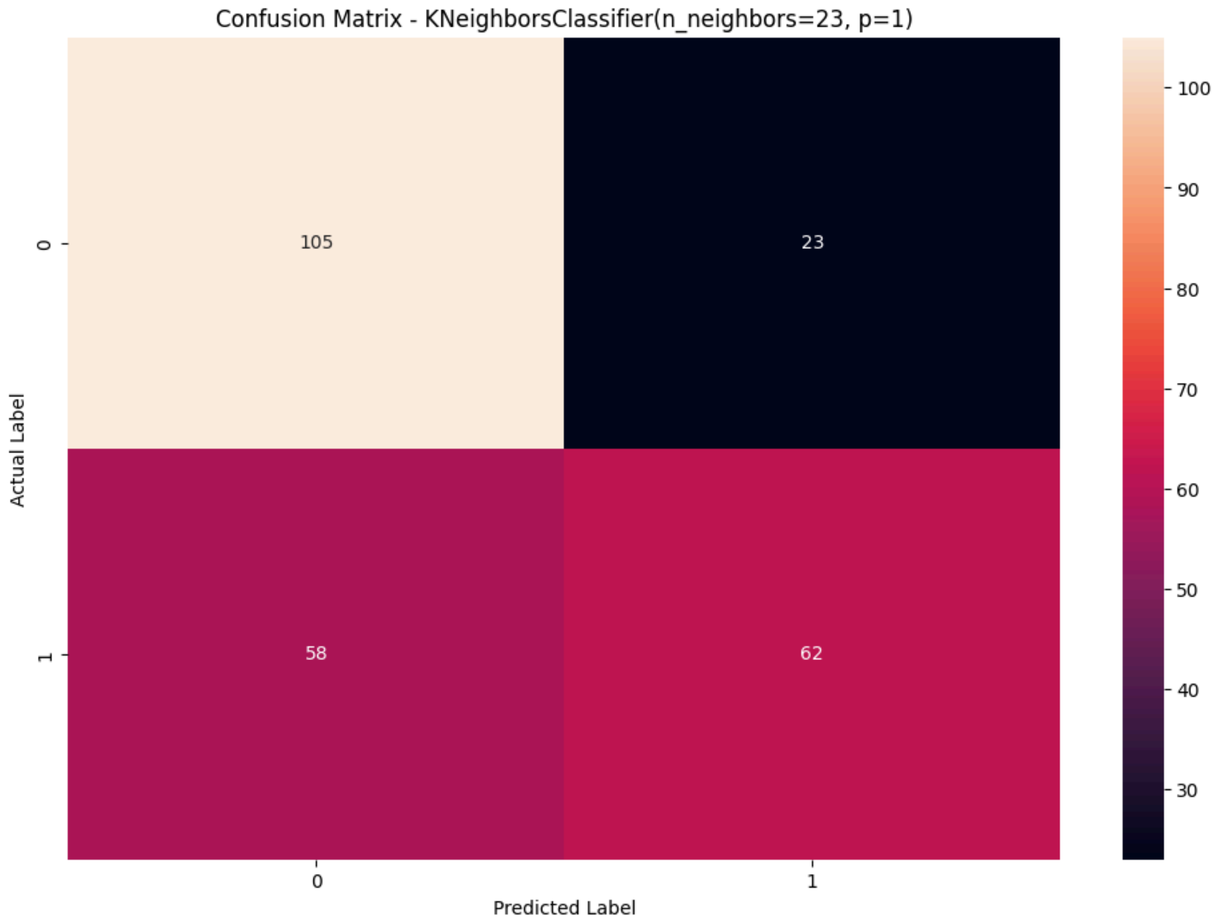
K-Nearest Neighbours

```
knn1_tuned = KNeighborsClassifier(algorithm = 'auto', n_neighbors = 23, p = 1, weights = 'uniform')
knn1_tuned.fit(X1_train, y1_train)

eval_classification1(knn1_tuned)
```

```
Accuracy (Test Set): 0.67
Accuracy (Train Set): 0.74
Precision (Test Set): 0.73
Recall (Test Set): 0.52
Recall (Train Set): 0.66
F1-Score (Test Set): 0.60
roc_auc (test-proba): 0.70
roc_auc (train-proba): 0.80
Accuracy (crossval train): 0.7350819115524998
Accuracy (crossval test): 0.7213767458733903
```

```
confusion1(knn1_tuned)
```

```
Accuracy: 67.34%
Precision: 72.94%
Recall: 51.67%
F1_Score: 60.49%
ROC_AUC: 69.66%
```



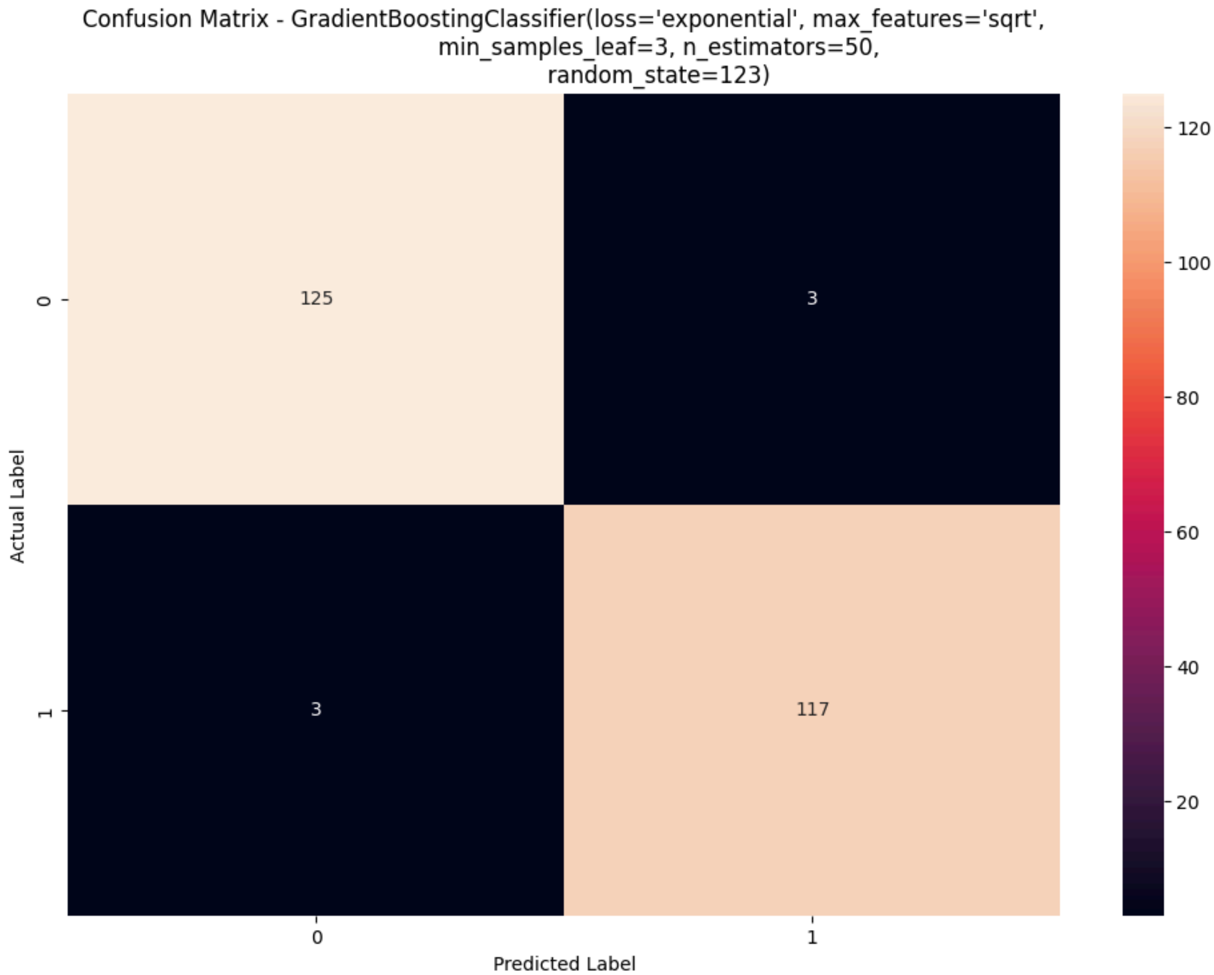
Gradient Boosting

```
gb1_tuned = GradientBoostingClassifier(random_state=123, criterion = 'friedman_mse', loss = 'exponential',
                                       max_depth = 3, max_features = 'sqrt', min_samples_leaf = 3, min_samples_split = 2, n_estimators = 50)
gb1_tuned.fit(X1_train, y1_train)

eval_classification1(gb1_tuned)
```

```
Accuracy (Test Set): 0.98
Accuracy (Train Set): 0.98
Precision (Test Set): 0.97
Recall (Test Set): 0.97
Recall (Train Set): 0.97
F1-Score (Test Set): 0.97
roc_auc (test-proba): 0.99
roc_auc (train-proba): 1.00
Accuracy (crossval train): 0.9860921068764205
Accuracy (crossval test): 0.9596378257452083
```

```
confusion1(gb1_tuned)
```

```
Accuracy: 97.58%
Precision: 97.50%
Recall: 97.50%
F1_Score: 97.50%
ROC_AUC: 99.18%
```



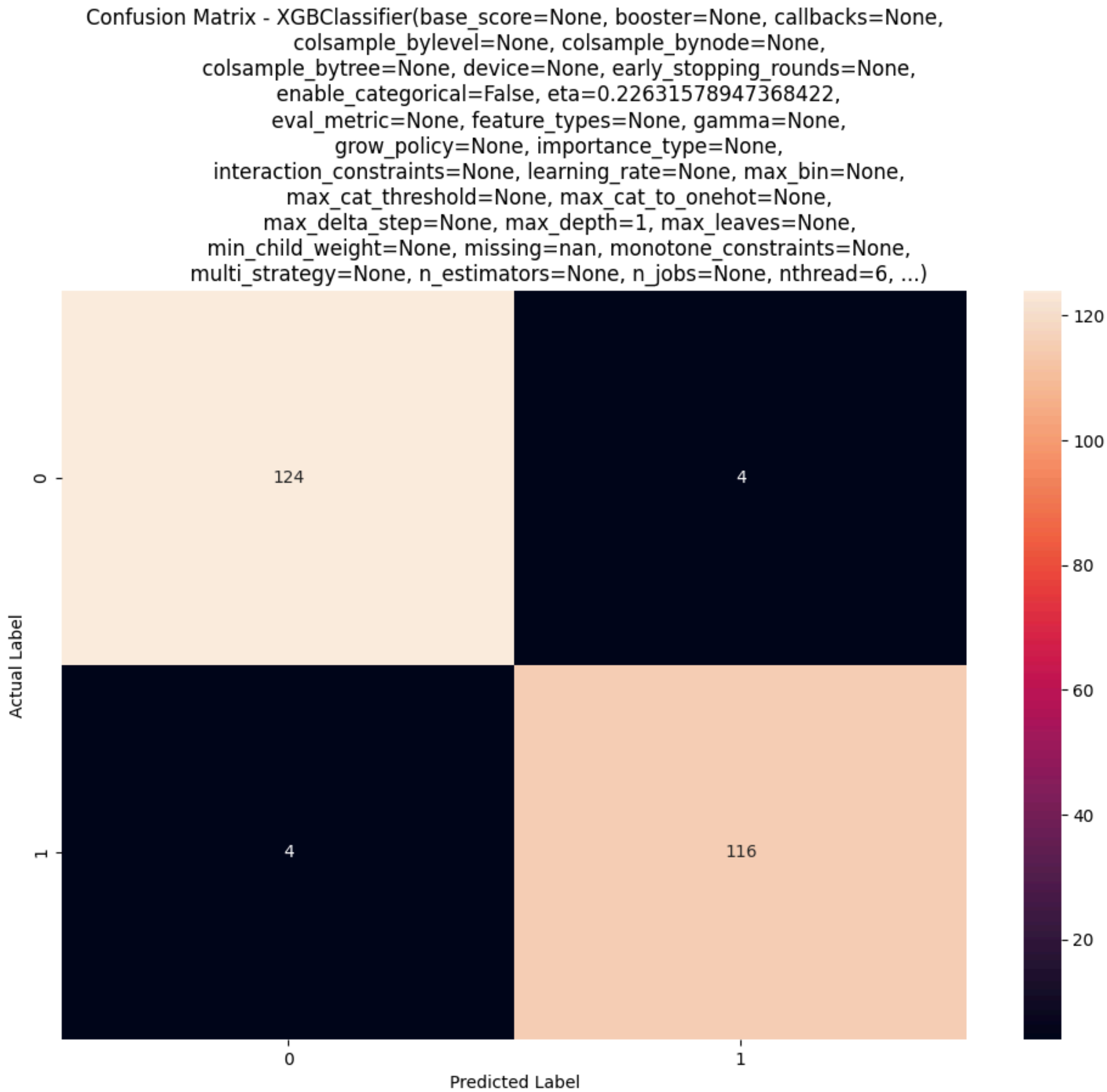
XGBoost

```
xgb1_tuned = XGBClassifier(nthread=6, tree_method='hist', random_state=123, eta = 0.22631578947368422, max_depth = 1)

xgb1_tuned.fit(X1_train, y1_train)

eval_classification1(xgb1_tuned)
```

```
Accuracy (Test Set): 0.97
Accuracy (Train Set): 0.98
Precision (Test Set): 0.97
Recall (Test Set): 0.97
Recall (Train Set): 0.96
F1-Score (Test Set): 0.97
roc_auc (test-proba): 0.99
roc_auc (train-proba): 1.00
Accuracy (crossval train): 0.9784664572899866
Accuracy (crossval test): 0.9596378257452084
```

```
confusion1(xgb1_tuned)
```

```
Accuracy: 96.77%
Precision: 96.67%
Recall: 96.67%
F1_Score: 96.67%
ROC_AUC: 98.72%
```



As can be seen in the above section, Overfitting has been significantly reduced.

After normalization/standardization

Helper Functions

```
from sklearn.pipeline import Pipeline

def eval_classification2(model):
    y_pred = model.predict(X2_test)
    y_pred_train = model.predict(X2_train_scaled)
    y_pred_proba = model.predict_proba(X2_test)
    y_pred_proba_train = model.predict_proba(X2_train_scaled)

    model1 = Pipeline([
    ('scaling', StandardScaler()),
    ('classification', model)
    ])
```

```
    print("Accuracy (Test Set): %.2f" % accuracy_score(y2_test, y_pred))
    print("Accuracy (Train Set): %.2f" % accuracy_score(y2_train, y_pred_train))
    print("Precision (Test Set): %.2f" % precision_score(y2_test, y_pred, zero_division=0))
    print("Recall (Test Set): %.2f" % recall_score(y2_test, y_pred))
    print("Recall (Train Set): %.2f" % recall_score(y2_train, y_pred_train))
    print("F1-Score (Test Set): %.2f" % f1_score(y2_test, y_pred))

    print("roc_auc (test-proba): %.2f" % roc_auc_score(y2_test, y_pred_proba[:, 1]))
    print("roc_auc (train-proba): %.2f" % roc_auc_score(y2_train, y_pred_proba_train[:, 1]))

    cv = RepeatedStratifiedKFold(random_state=42, n_repeats = 3)
    score = cross_validate(model1, X=X2_train, y=y2_train, cv=cv, scoring='accuracy', return_train_score=True)
    print('Accuracy (crossval train): '+ str(score['train_score'].mean()))
    print('Accuracy (crossval test): '+ str(score['test_score'].mean()))


def grid_pipe2(pipedict, hyperdict, scoring='accuracy', display=True):
    fitted_models2={}
    fit_time2 = []
    for name, pipeline in pipedict.items():
    # Construct grid search
        cv = RepeatedStratifiedKFold(random_state=42, n_repeats = 3)
        model = GridSearchCV(estimator=pipeline,
                             param_grid=hyperdict[name],
                             scoring=scoring,
                             cv=cv, verbose=2, n_jobs=-1, return_train_score = True)

        # Fit using grid search
        start = time.time()
        model.fit(X2_train, y2_train)
        end = time.time()
        fit_time2.append(round(end-start, 2))
        #Append model
        fitted_models2[name]=model
        if display:
            #Print when the model has been fitted
            print(f'The {name} model has been fitted.')
            # print fit time
            print('Total Fit Time: %.3fs' % (end-start))
            # Best accuracy
            print('Best accuracy: %.3f' % model.best_score_)
            # Best params
            print('Best params:\n', model.best_params_,'\n')

    return fitted_models2, fit_time2

def confusion2(model):
    y_pred_proba = model.predict_proba(X2_test)
    y_predict = model.predict(X2_test)
    print('Accuracy: %.2f%%' % (accuracy_score(y2_test, y_predict) * 100 ))
    print('Precision: %.2f%%' % (precision_score(y2_test, y_predict, zero_division=0) * 100))
    print('Recall: %.2f%%' % (recall_score(y2_test, y_predict) * 100))
    print('F1_Score: %.2f%%' % (f1_score(y2_test, y_predict) * 100))
    print('ROC_AUC: %.2f%%' % (roc_auc_score(y2_test, y_pred_proba[:,1]) * 100))
    confusion_matrix_model = confusion_matrix(y2_test, y_predict)
    plt.figure(figsize=(12,8))
    ax = plt.subplot()
    sns.heatmap(confusion_matrix_model, annot=True, fmt='g', ax = ax)
    ax.set_xlabel('Predicted Label')
    ax.set_ylabel('Actual Label')
    ax.set_title(f'Confusion Matrix - {model}')
    ax.xaxis.set_ticklabels(['0','1'])
    ax.yaxis.set_ticklabels(['0','1'])
```

Vanilla Models

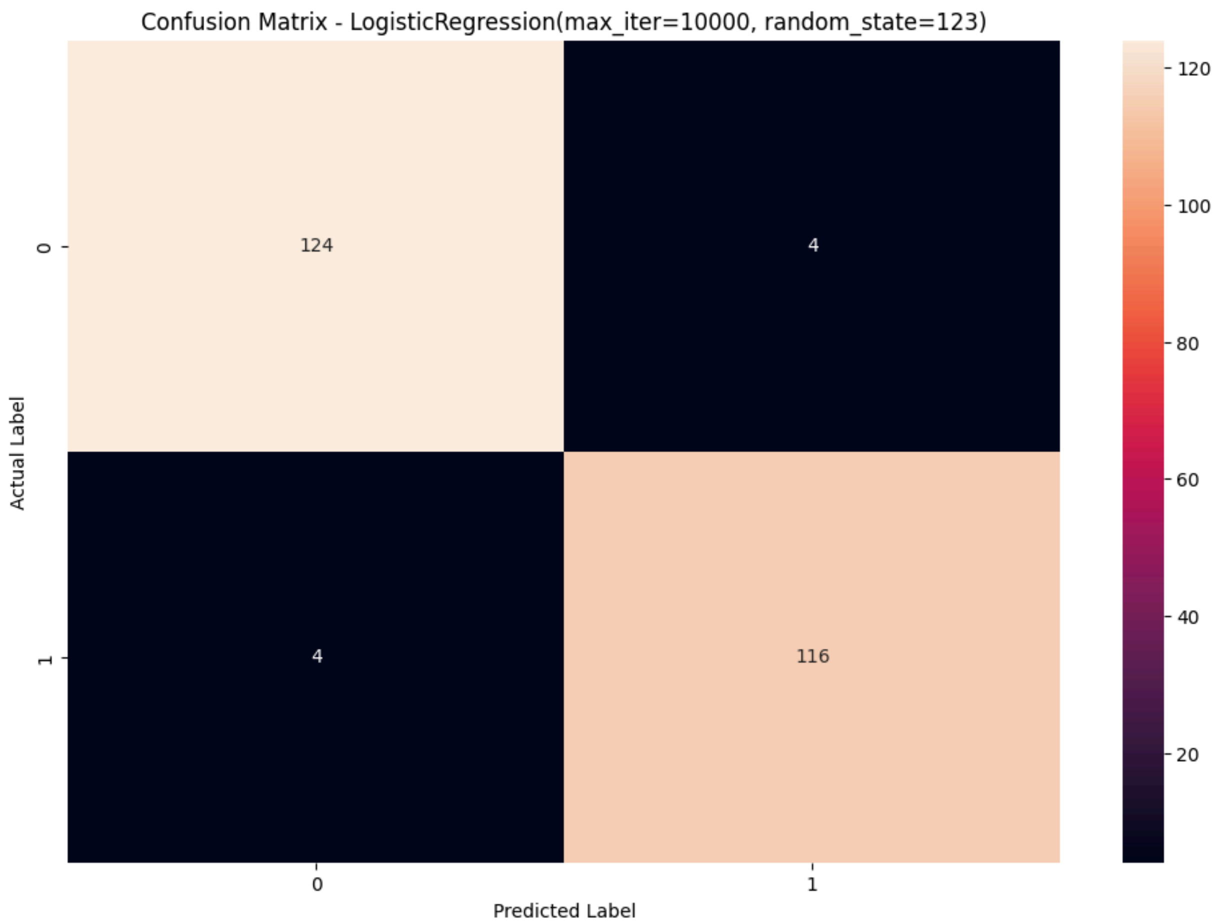
Logistic Regression

```
logreg2 = LogisticRegression(random_state=123, max_iter=10000)
logreg2.fit(X2_train_scaled, y2_train)

eval_classification2(logreg2)
```

```
Accuracy (Test Set): 0.97
Accuracy (Train Set): 0.97
Precision (Test Set): 0.97
Recall (Test Set): 0.97
Recall (Train Set): 0.96
F1-Score (Test Set): 0.97
roc_auc (test-proba): 0.99
roc_auc (train-proba): 0.99
Accuracy (crossval train): 0.9710639542012092
Accuracy (crossval test): 0.9654604268698229
```

```
confusion2(logreg2)
```

```
Accuracy: 96.77%
Precision: 96.67%
Recall: 96.67%
F1_Score: 96.67%
ROC_AUC: 98.79%
```



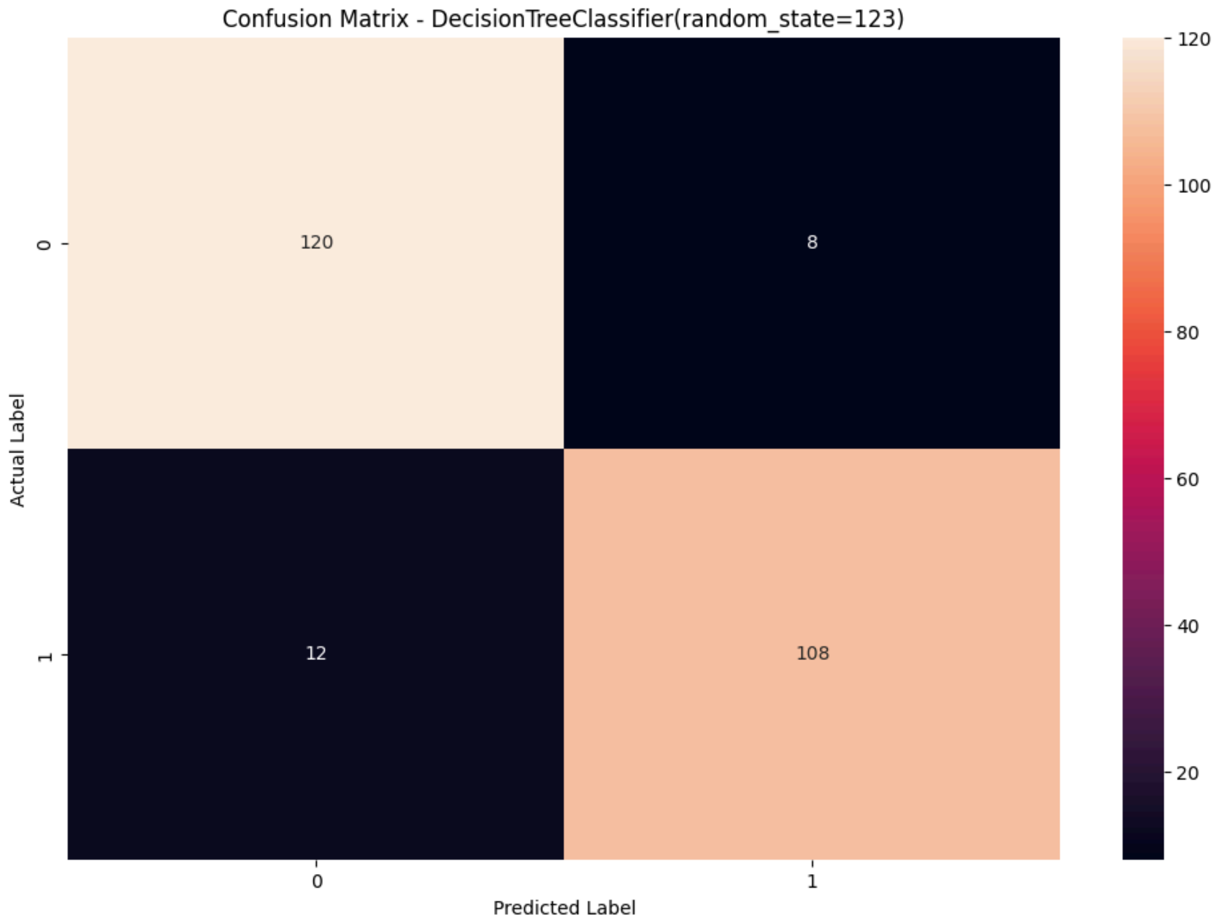
Decision Tree

```
dt2 = DecisionTreeClassifier(random_state=123)
dt2.fit(X2_train_scaled, y2_train)

eval_classification2(dt2)
```

```
Accuracy (Test Set): 0.92
Accuracy (Train Set): 1.00
Precision (Test Set): 0.93
Recall (Test Set): 0.90
Recall (Train Set): 1.00
F1-Score (Test Set): 0.92
roc_auc (test-proba): 0.92
roc_auc (train-proba): 1.00
Accuracy (crossval train): 1.0
Accuracy (crossval test): 0.9394491807243485
```

```
confusion2(dt2)
```

```
Accuracy: 91.94%
Precision: 93.10%
Recall: 90.00%
F1_Score: 91.53%
ROC_AUC: 91.88%
```



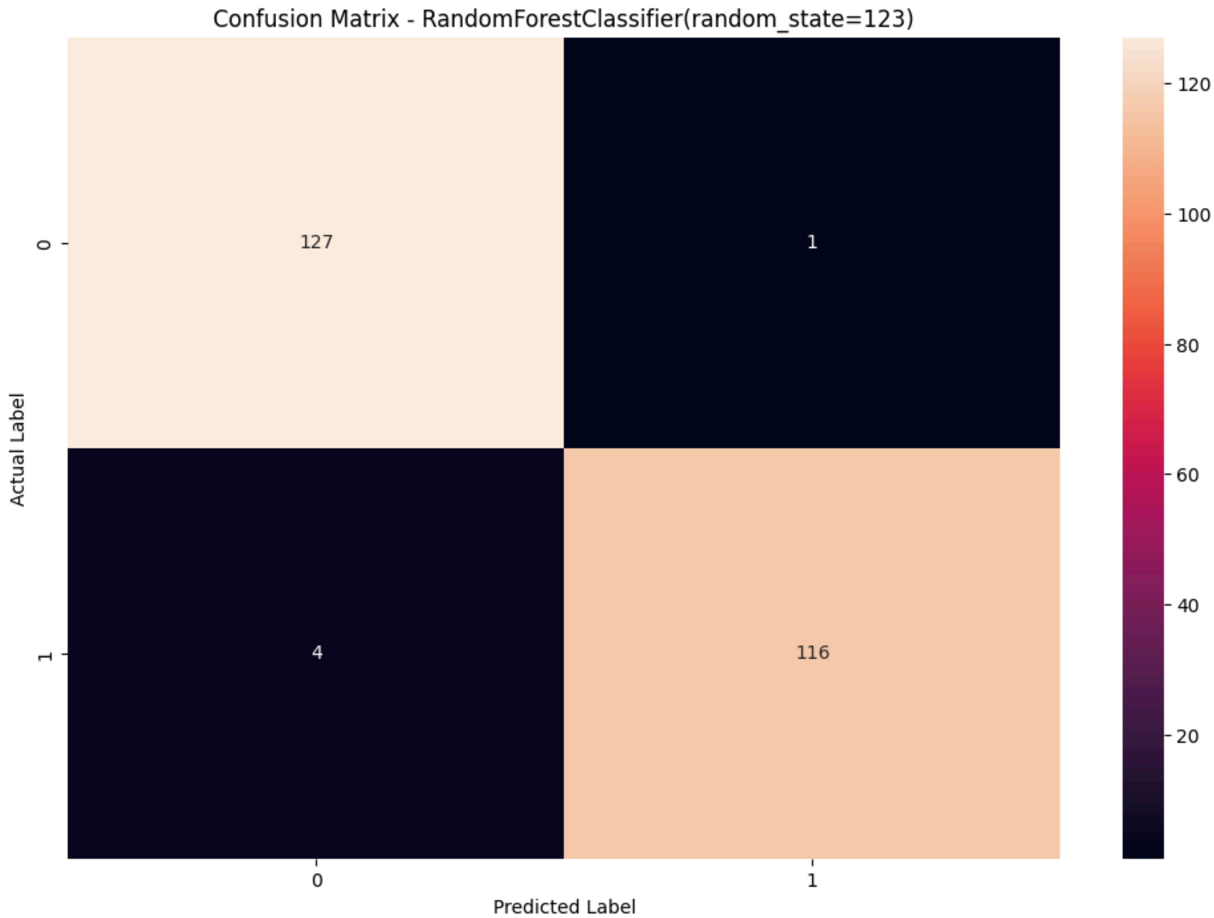
Random Forest

```
rf2 = RandomForestClassifier(random_state=123)
rf2.fit(X2_train_scaled, y2_train)

eval_classification2(rf2)
```

```
Accuracy (Test Set): 0.98
Accuracy (Train Set): 1.00
Precision (Test Set): 0.99
Recall (Test Set): 0.97
Recall (Train Set): 1.00
F1-Score (Test Set): 0.98
roc_auc (test-proba): 0.99
roc_auc (train-proba): 1.00
Accuracy (crossval train): 1.0
Accuracy (crossval test): 0.9596378257452083
```

```
confusion2(rf2)
```

```
Accuracy: 97.98%
Precision: 99.15%
Recall: 96.67%
F1_Score: 97.89%
ROC_AUC: 98.96%
```



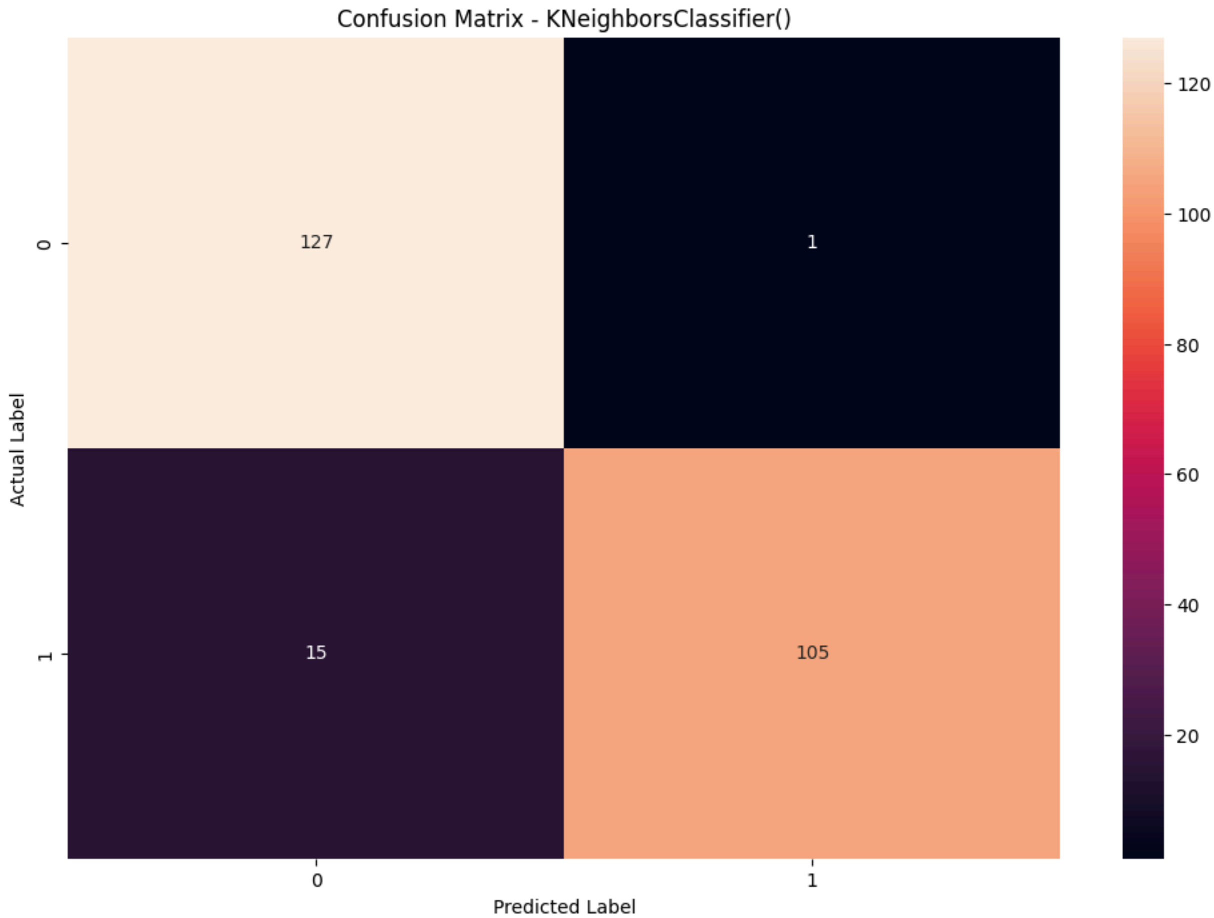
K-Nearest Neighbours

```
knn2 = KNeighborsClassifier()
knn2.fit(X2_train_scaled, y2_train)

eval_classification2(knn2)
```

```
Accuracy (Test Set): 0.94
Accuracy (Train Set): 0.95
Precision (Test Set): 0.99
Recall (Test Set): 0.88
Recall (Train Set): 0.90
F1-Score (Test Set): 0.93
roc_auc (test-proba): 0.98
roc_auc (train-proba): 0.99
Accuracy (crossval train): 0.9559228135698725
Accuracy (crossval test): 0.9340286595320154
```

```
confusion2(knn2)
```

```
Accuracy: 93.55%
Precision: 99.06%
Recall: 87.50%
F1_Score: 92.92%
ROC_AUC: 98.01%
```



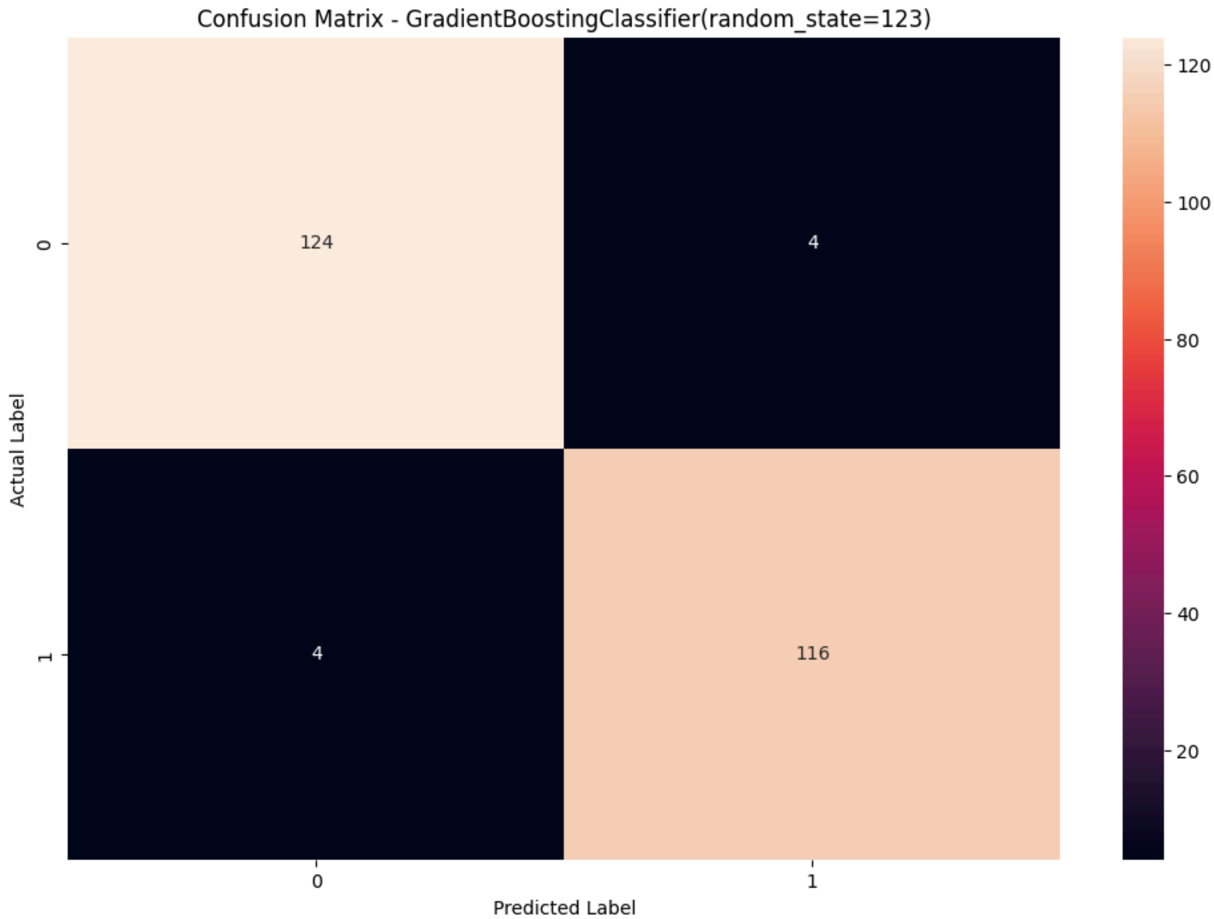
Gradient Boosting

```
gb2 = GradientBoostingClassifier(random_state=123)
gb2.fit(X2_train_scaled, y2_train)

eval_classification2(gb2)
```

```
Accuracy (Test Set): 0.97
Accuracy (Train Set): 1.00
Precision (Test Set): 0.97
Recall (Test Set): 0.97
Recall (Train Set): 1.00
F1-Score (Test Set): 0.97
roc_auc (test-proba): 0.99
roc_auc (train-proba): 1.00
Accuracy (crossval train): 0.9998879551820729
Accuracy (crossval test): 0.9529082774049217
```

```
confusion2(gb2)
```

```
Accuracy: 96.77%
Precision: 96.67%
Recall: 96.67%
F1_Score: 96.67%
ROC_AUC: 98.64%
```



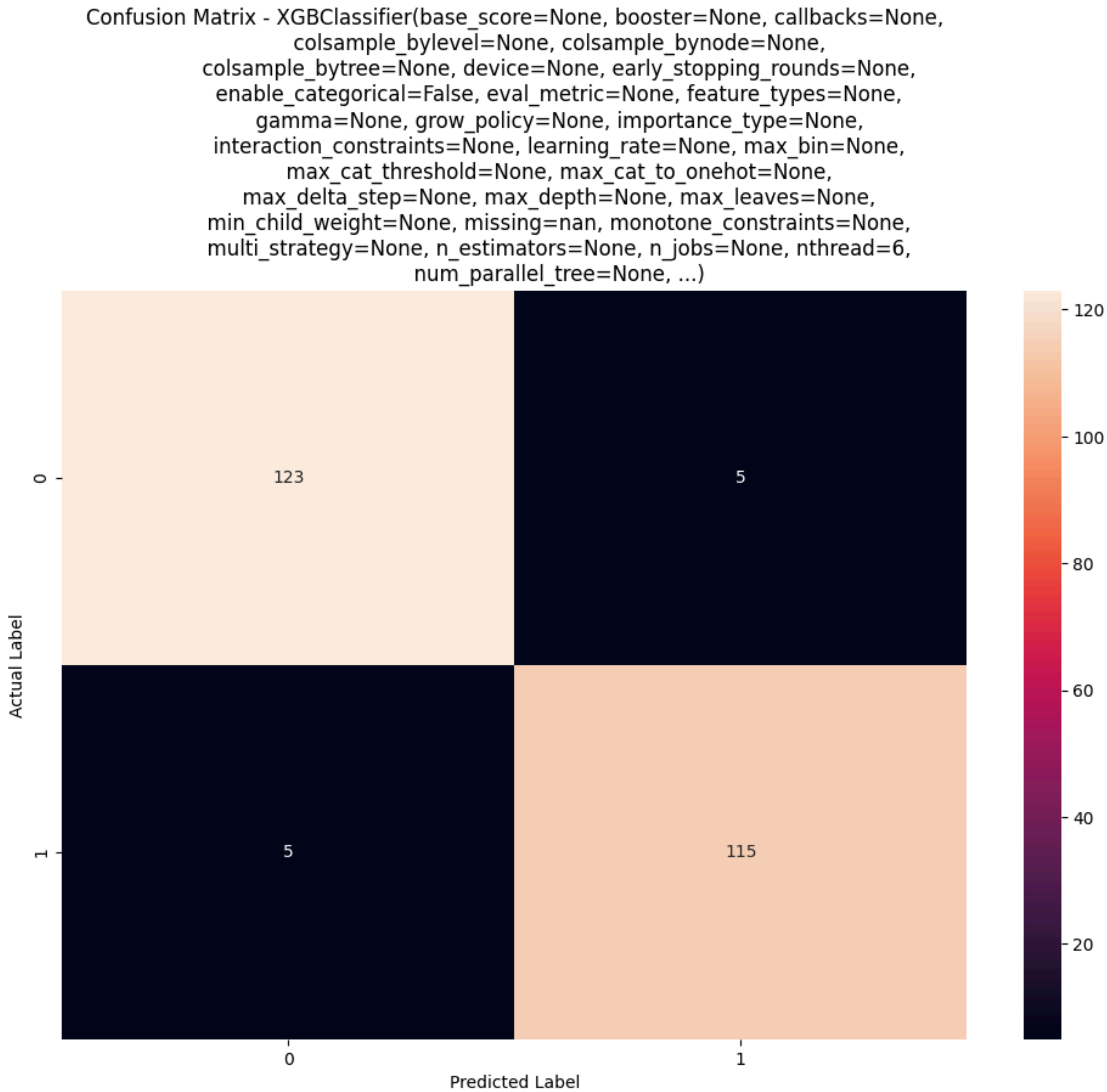
XGBoost

```
xgb2 = XGBClassifier(nthread=6, tree_method='hist', random_state=123)

xgb2.fit(X2_train_scaled, y2_train)

eval_classification2(xgb2)
```

```
Accuracy (Test Set): 0.96
Accuracy (Train Set): 1.00
Precision (Test Set): 0.96
Recall (Test Set): 0.96
Recall (Train Set): 1.00
F1-Score (Test Set): 0.96
roc_auc (test-proba): 0.99
roc_auc (train-proba): 1.00
Accuracy (crossval train): 1.0
Accuracy (crossval test): 0.9502176673317613
```

```
confusion2(xgb2)
```

```
Accuracy: 95.97%
Precision: 95.83%
Recall: 95.83%
F1_Score: 95.83%
ROC_AUC: 98.98%
```



As can be seen from the above section, overfitting still exists in a few of the models tested. Hyperparameter tuning will be done on each model to rectify this issue.

Hyperparameter Tuning

```
#Create pipeline for each of the classifiers.
pipelines = {'logisticregression2': Pipeline([('scaling', StandardScaler()), ('clf', LogisticRegression(random_state=123))]),
            'decisiontree2':Pipeline([('scaling', StandardScaler()), ('clf', DecisionTreeClassifier(random_state = 123))]),
            'randomforest2':Pipeline([('scaling', StandardScaler()), ('clf', RandomForestClassifier(random_state = 123))]),
            'knn2':Pipeline([('scaling', StandardScaler()), ('clf', KNeighborsClassifier())]),
            'gb2':Pipeline([('scaling', StandardScaler()), ('clf', GradientBoostingClassifier(random_state = 123))]),
            'xgboost2': Pipeline([('scaling', StandardScaler()), ('clf', XGBClassifier(nthread=6, tree_method='hist', random_state=123))])}

#Define Hyperparameters for each pipeline
hyperparameters_lr2 ={'clf__C': [float(x) for x in np.linspace(0.002, 1, 100)],
                      'clf__penalty' : ['l2'],
                      'clf__solver' : ['newton-cg', 'lbfgs', 'newton-cholesky', 'liblinear'],
                      'clf__max_iter':[10000]}

hyperparameters_dt2 ={'clf__criterion' : ['entropy', 'gini'],
```

```
                      'clf__max_depth' : [int(x) for x in np.linspace(1, 20, 20)],
                      'clf__min_samples_split' : [int(x) for x in np.linspace(start = 2, stop = 50, num = 5)],
                      'clf__min_samples_leaf' : [int(x) for x in np.linspace(start = 2, stop = 50, num = 5)],
                      'clf__max_features' : ['sqrt'],
                      'clf__splitter' : ['best']}

hyperparameters_rf2 ={'clf__n_estimators': [50,60,75, 100, 120],
                     'clf__criterion': ['entropy', 'gini'],
                     'clf__max_features':['sqrt' , None],
                     'clf__min_samples_leaf':[0.05, 0.1, 0.2]}

hyperparameters_knn2 ={'clf__n_neighbors' : list(range(1,30)),
                       'clf__weights' : ['uniform'],
                       'clf__p' : [1, 2],
                       'clf__algorithm' : ['auto', 'ball_tree', 'kd_tree', 'brute']}

hyperparameters_gb2 ={'clf__n_estimators' : [int(x) for x in np.linspace(10, 50, num = 5)],
                      'clf__criterion' : ['friedman_mse', 'squared_error'],
                      'clf__max_depth' : [1, 2, 3],
                      'clf__min_samples_split' : [2, 3, 5],
                      'clf__min_samples_leaf' : [2, 3, 5],
                      'clf__max_features' : ['sqrt'],
                      'clf__loss' : ['exponential']}

hyperparameters_xgb2 ={'clf__eta': [float(x) for x in np.linspace(0.1, 0.7, 20)],
                     'clf__max_depth': [1,3,5]}

#Instantiate hyperparapeter dictionary
hyperparameters = {'logisticregression2':hyperparameters_lr2,
                   'decisiontree2':hyperparameters_dt2,
                   'randomforest2':hyperparameters_rf2,
                   'knn2':hyperparameters_knn2,
                   'gb2':hyperparameters_gb2,
                   'xgboost2': hyperparameters_xgb2}


fitted_models2, fit_time2 = grid_pipe2(pipelines,hyperparameters,scoring='accuracy')
```

```
Fitting 15 folds for each of 400 candidates, totalling 6000 fits
The logisticregression2 model has been fitted.
Total Fit Time: 79.667s
Best accuracy: 0.965
Best params:
 {'clf__C': 0.7883030303030303, 'clf__max_iter': 10000, 'clf__penalty': 'l2', 'clf__solver': 'newton-cg'}

Fitting 15 folds for each of 1000 candidates, totalling 15000 fits
The decisiontree2 model has been fitted.
Total Fit Time: 145.681s
Best accuracy: 0.907
Best params:
 {'clf__criterion': 'gini', 'clf__max_depth': 8, 'clf__max_features': 'sqrt', 'clf__min_samples_leaf': 2, 'clf__min_samples_split': 26,

Fitting 15 folds for each of 60 candidates, totalling 900 fits
The randomforest2 model has been fitted.
Total Fit Time: 143.096s
Best accuracy: 0.948
Best params:
 {'clf__criterion': 'entropy', 'clf__max_features': 'sqrt', 'clf__min_samples_leaf': 0.05, 'clf__n_estimators': 75}

Fitting 15 folds for each of 232 candidates, totalling 3480 fits
The knn2 model has been fitted.
Total Fit Time: 216.646s
Best accuracy: 0.937
Best params:
 {'clf__algorithm': 'auto', 'clf__n_neighbors': 7, 'clf__p': 2, 'clf__weights': 'uniform'}

Fitting 15 folds for each of 270 candidates, totalling 4050 fits
The gb2 model has been fitted.
Total Fit Time: 137.075s
Best accuracy: 0.960
Best params:
 {'clf__criterion': 'friedman_mse', 'clf__loss': 'exponential', 'clf__max_depth': 3, 'clf__max_features': 'sqrt', 'clf__min_samples_leaf

Fitting 15 folds for each of 60 candidates, totalling 900 fits
The xgboost2 model has been fitted.
Total Fit Time: 89.809s
Best accuracy: 0.960
Best params:
 {'clf__eta': 0.22631578947368422, 'clf__max_depth': 1}
```

After Hyperparameter Tuning

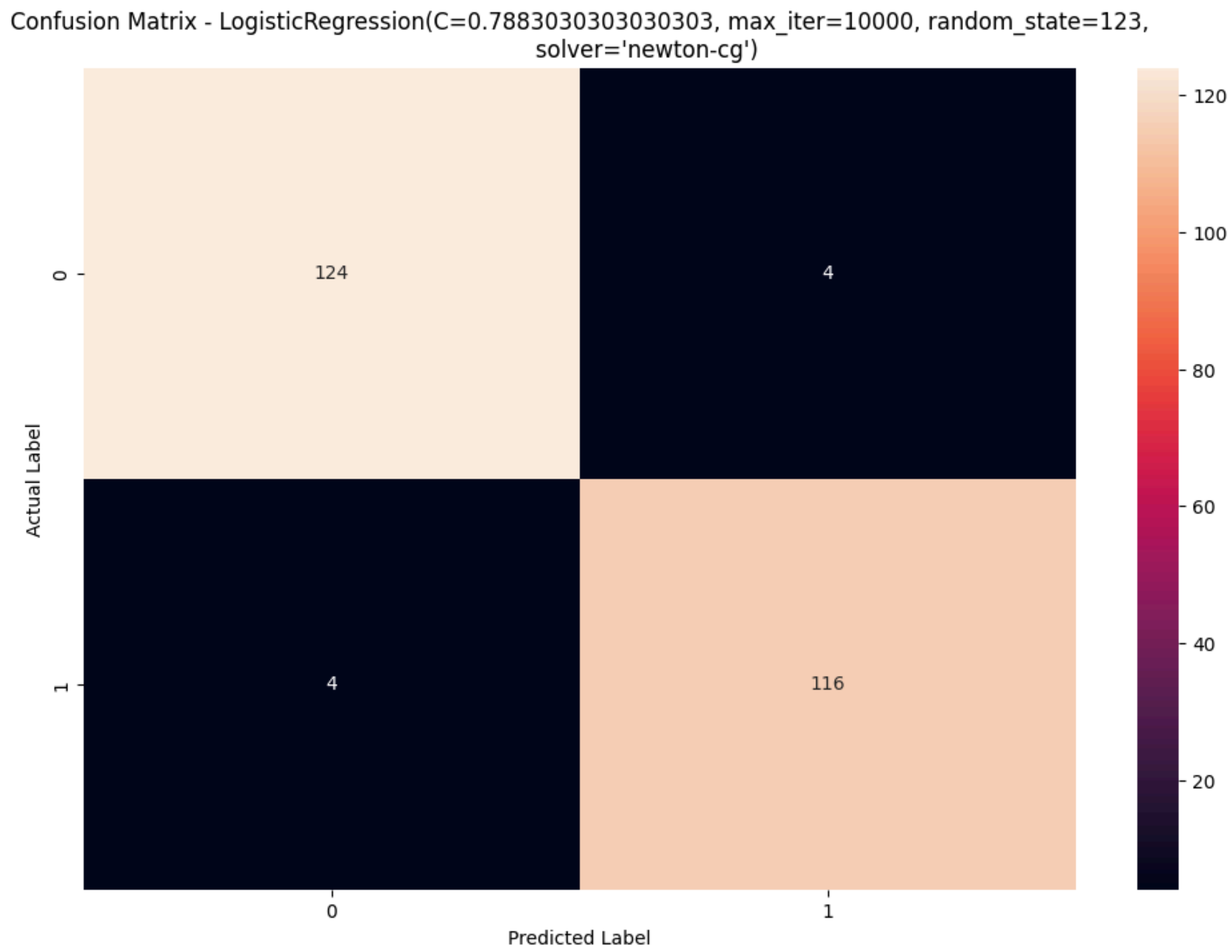
Logistic Regression

```
logreg2_tuned = LogisticRegression(random_state=123, C = 0.7883030303030303, max_iter = 10000, penalty = 'l2', solver = 'newton-cg')
logreg2_tuned.fit(X2_train_scaled, y2_train)

eval_classification2(logreg2_tuned)
```

```
Accuracy (Test Set): 0.97
Accuracy (Train Set): 0.97
Precision (Test Set): 0.97
Recall (Test Set): 0.97
Recall (Train Set): 0.96
F1-Score (Test Set): 0.97
roc_auc (test-proba): 0.99
roc_auc (train-proba): 0.99
Accuracy (crossval train): 0.9708394873100757
Accuracy (crossval test): 0.9654604268698229
```

```
confusion2(logreg2_tuned)
```

```
Accuracy: 96.77%
Precision: 96.67%
Recall: 96.67%
F1_Score: 96.67%
ROC_AUC: 98.82%
```



Decision Tree

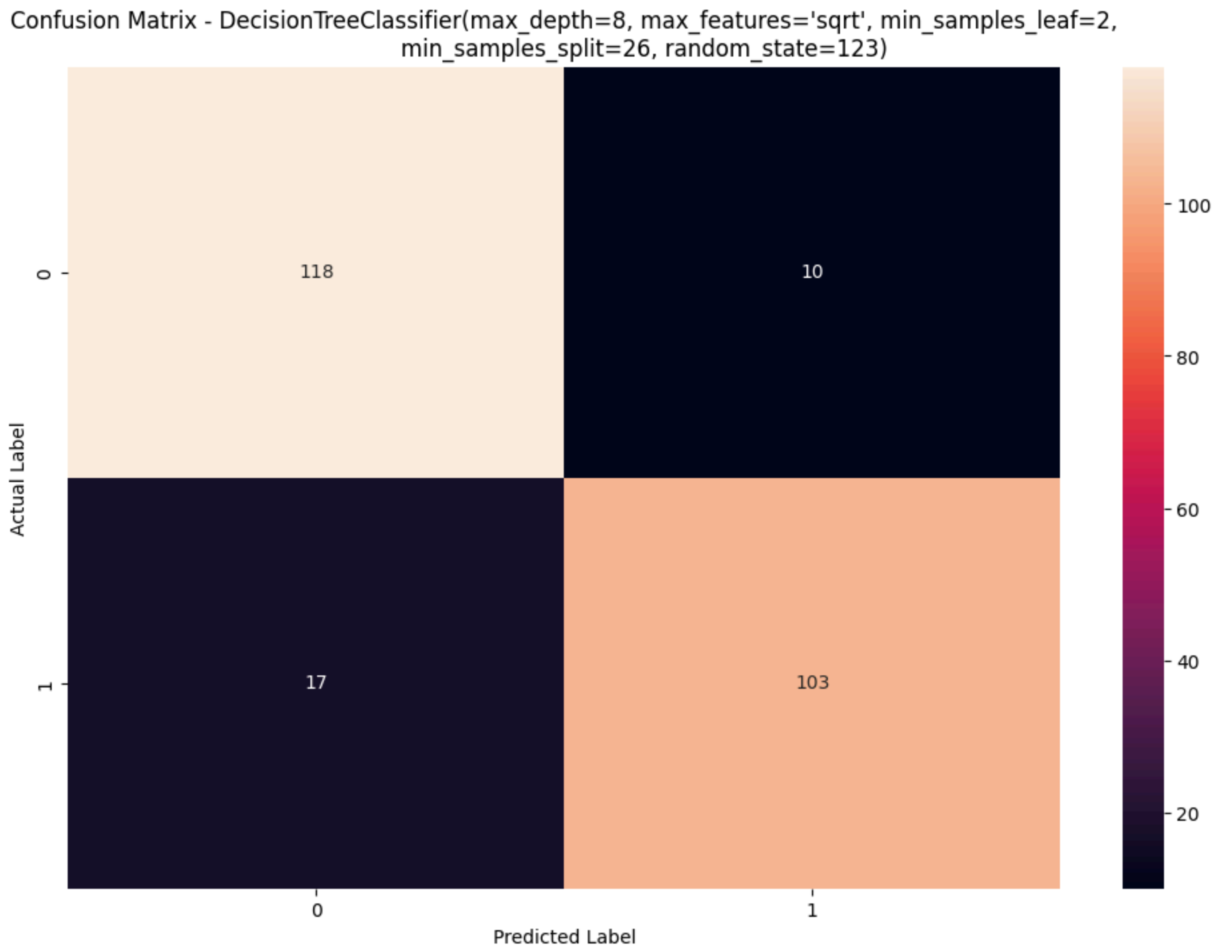
```
dt2_tuned = DecisionTreeClassifier(random_state=123, criterion = 'gini', max_depth = 8, max_features = 'sqrt', min_samples_leaf = 2, min_sam
dt2_tuned.fit(X2_train_scaled, y2_train)
```

```
eval_classification2(dt2_tuned)
```

```
Accuracy (Test Set): 0.89
Accuracy (Train Set): 0.95
Precision (Test Set): 0.91
Recall (Test Set): 0.86
Recall (Train Set): 0.93
F1-Score (Test Set): 0.88
roc_auc (test-proba): 0.94
roc_auc (train-proba): 0.99
Accuracy (crossval train): 0.9324848862103766
Accuracy (crossval test): 0.9071618598464235
```

```
confusion2(dt2_tuned)
```

```
Accuracy: 89.11%
Precision: 91.15%
Recall: 85.83%
F1_Score: 88.41%
ROC_AUC: 94.15%
```



Random Forest

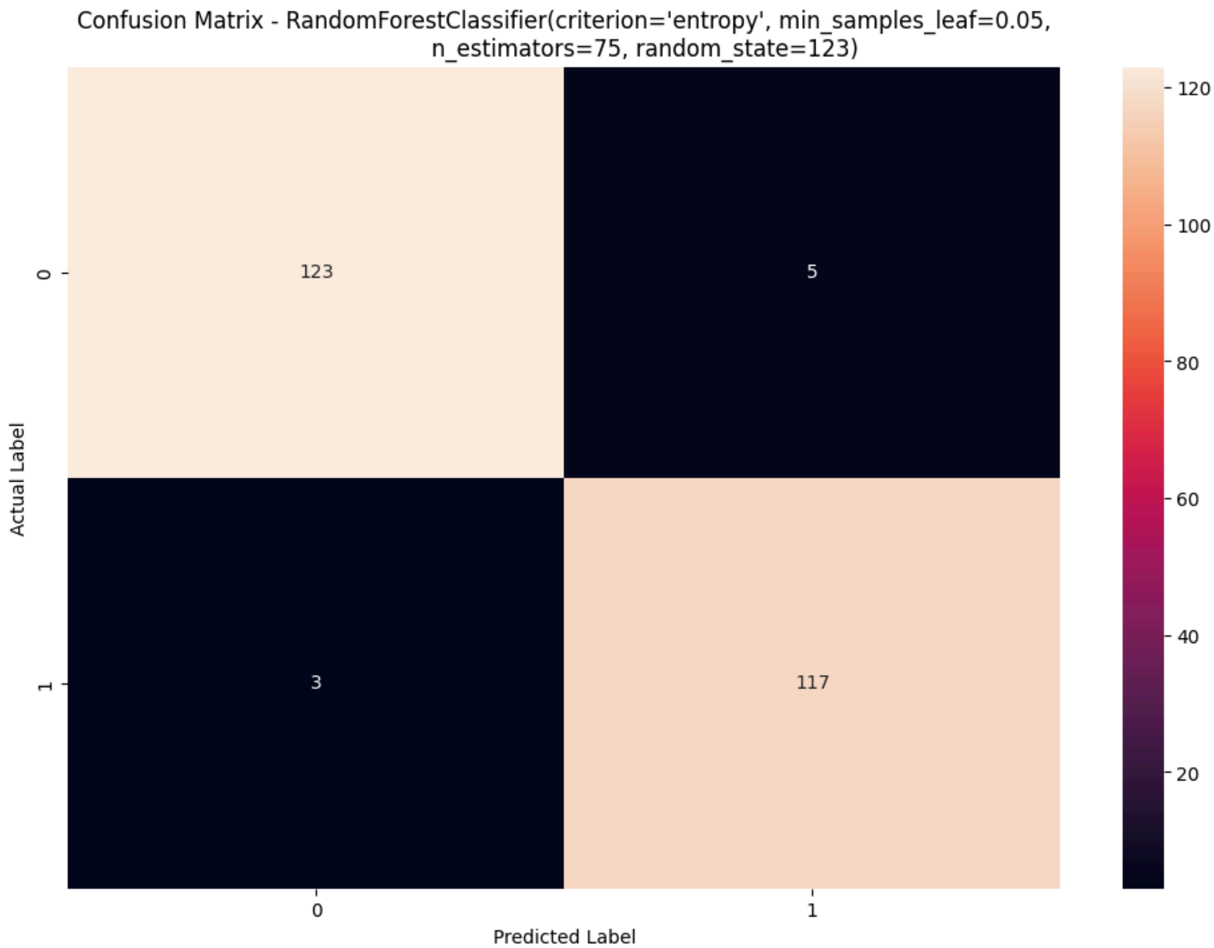
```
rf2_tuned = RandomForestClassifier(random_state=123, criterion = 'entropy', max_features = 'sqrt', min_samples_leaf = 0.05, n_estimators = 7
rf2_tuned.fit(X2_train_scaled, y2_train)
```

```
eval_classification2(rf2_tuned)
```

```
Accuracy (Test Set): 0.97
Accuracy (Train Set): 0.96
Precision (Test Set): 0.96
Recall (Test Set): 0.97
Recall (Train Set): 0.96
F1-Score (Test Set): 0.97
roc_auc (test-proba): 0.99
roc_auc (train-proba): 0.99
Accuracy (crossval train): 0.959511643041055
Accuracy (crossval test): 0.9484158655299594
```

```
confusion2(rf2_tuned)
```

```
Accuracy: 96.77%
Precision: 95.90%
Recall: 97.50%
F1_Score: 96.69%
ROC_AUC: 98.86%
```



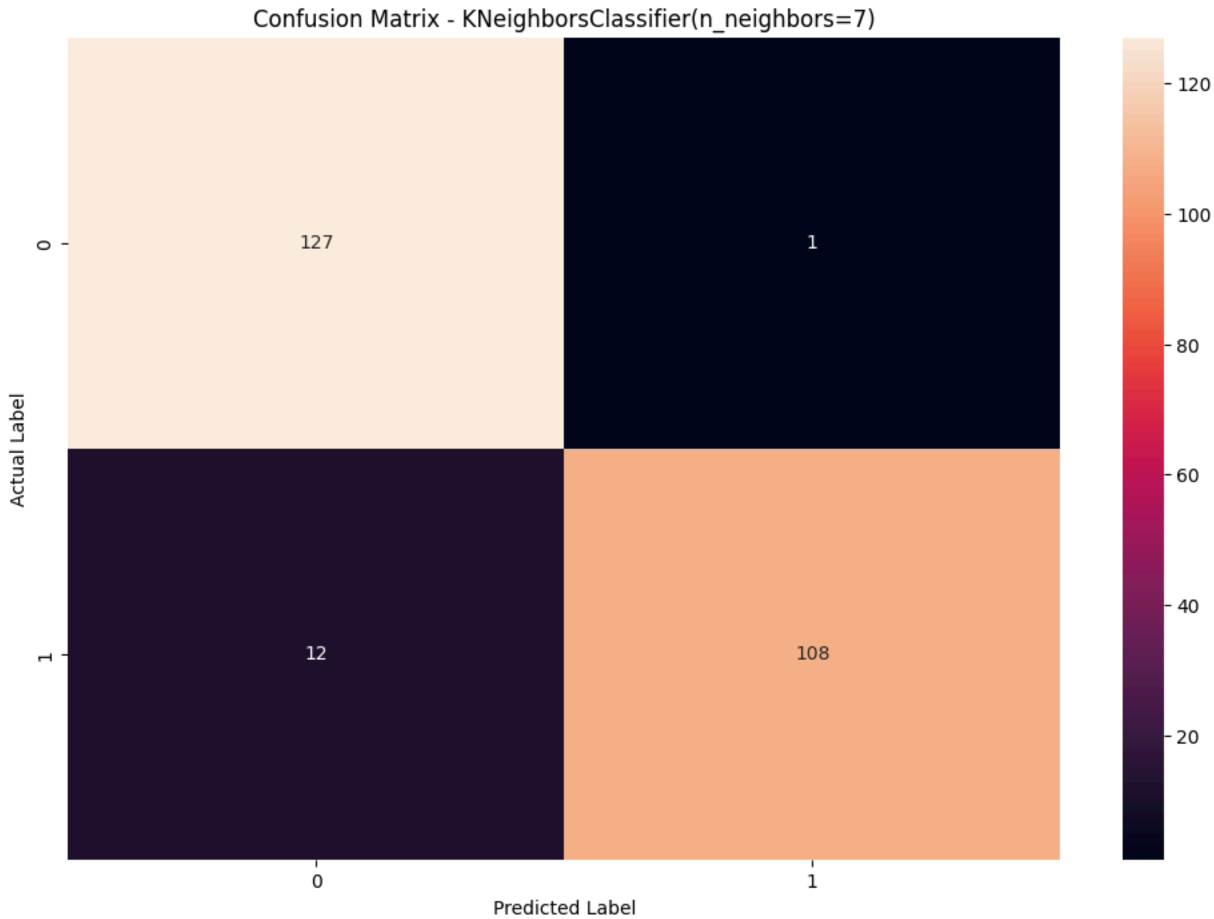
K-Nearest Neighbours

```
knn2_tuned = KNeighborsClassifier(algorithm = 'auto', n_neighbors = 7, p = 2, weights = 'uniform')
knn2_tuned.fit(X2_train_scaled, y2_train)

eval_classification2(knn2_tuned)
```

```
Accuracy (Test Set): 0.95
Accuracy (Train Set): 0.94
Precision (Test Set): 0.99
Recall (Test Set): 0.90
Recall (Train Set): 0.90
F1-Score (Test Set): 0.94
roc_auc (test-proba): 0.98
roc_auc (train-proba): 0.99
Accuracy (crossval train): 0.9525588282451029
Accuracy (crossval test): 0.9371848358425542
```

```
confusion2(knn2_tuned)
```

```
Accuracy: 94.76%
Precision: 99.08%
Recall: 90.00%
F1_Score: 94.32%
ROC_AUC: 97.90%
```



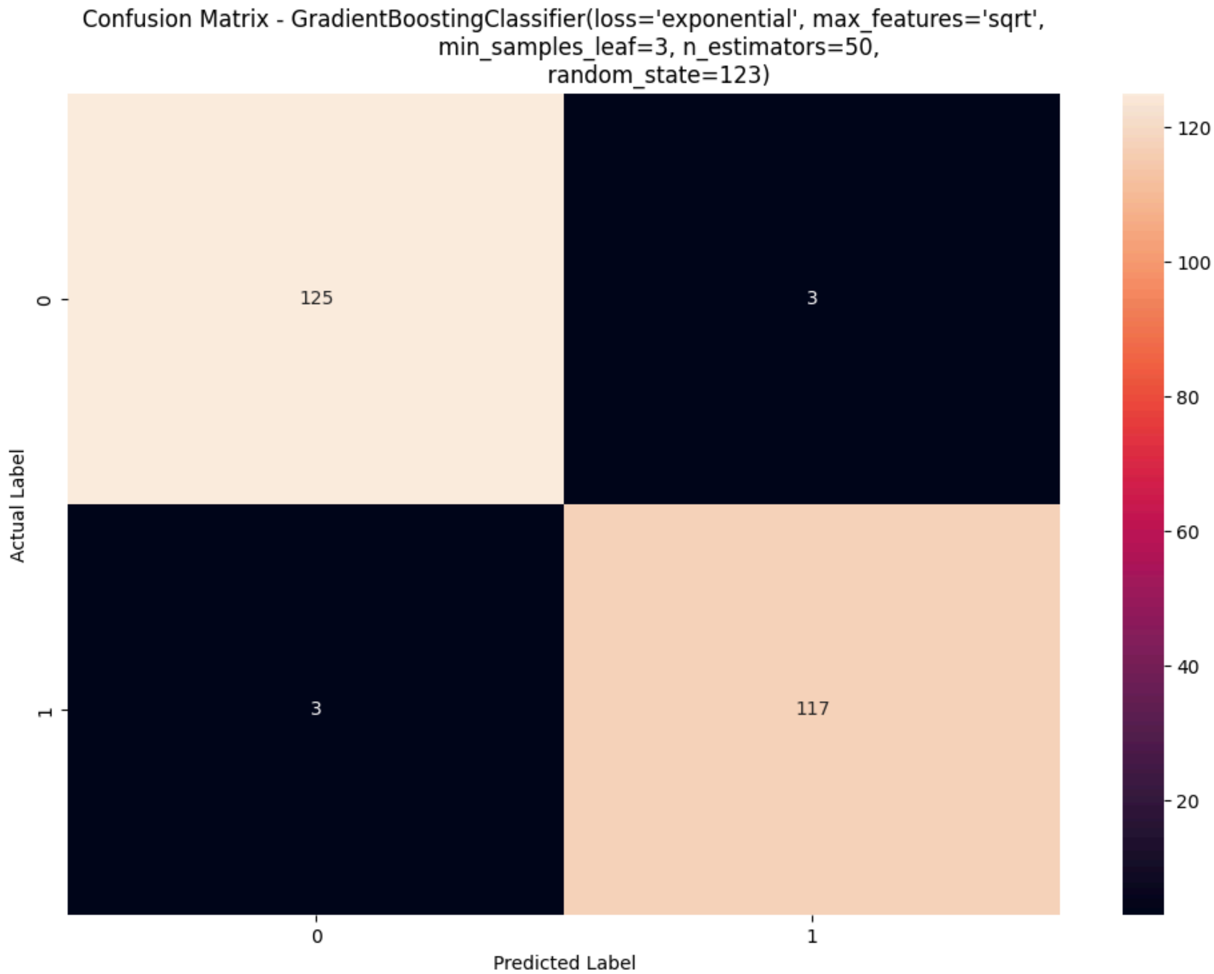
Gradient Boosting

```
gb2_tuned = GradientBoostingClassifier(random_state=123, criterion = 'friedman_mse', loss = 'exponential', max_depth = 3, max_features = 'sqr
gb2_tuned.fit(X2_train_scaled, y2_train)

eval_classification2(gb2_tuned)
```

```
Accuracy (Test Set): 0.98
Accuracy (Train Set): 0.98
Precision (Test Set): 0.97
Recall (Test Set): 0.97
Recall (Train Set): 0.97
F1-Score (Test Set): 0.97
roc_auc (test-proba): 0.99
roc_auc (train-proba): 1.00
Accuracy (crossval train): 0.9860921068764205
Accuracy (crossval test): 0.9596378257452083
```

```
confusion2(gb2_tuned)
```

```
Accuracy: 97.58%
Precision: 97.50%
Recall: 97.50%
F1_Score: 97.50%
ROC_AUC: 99.18%
```



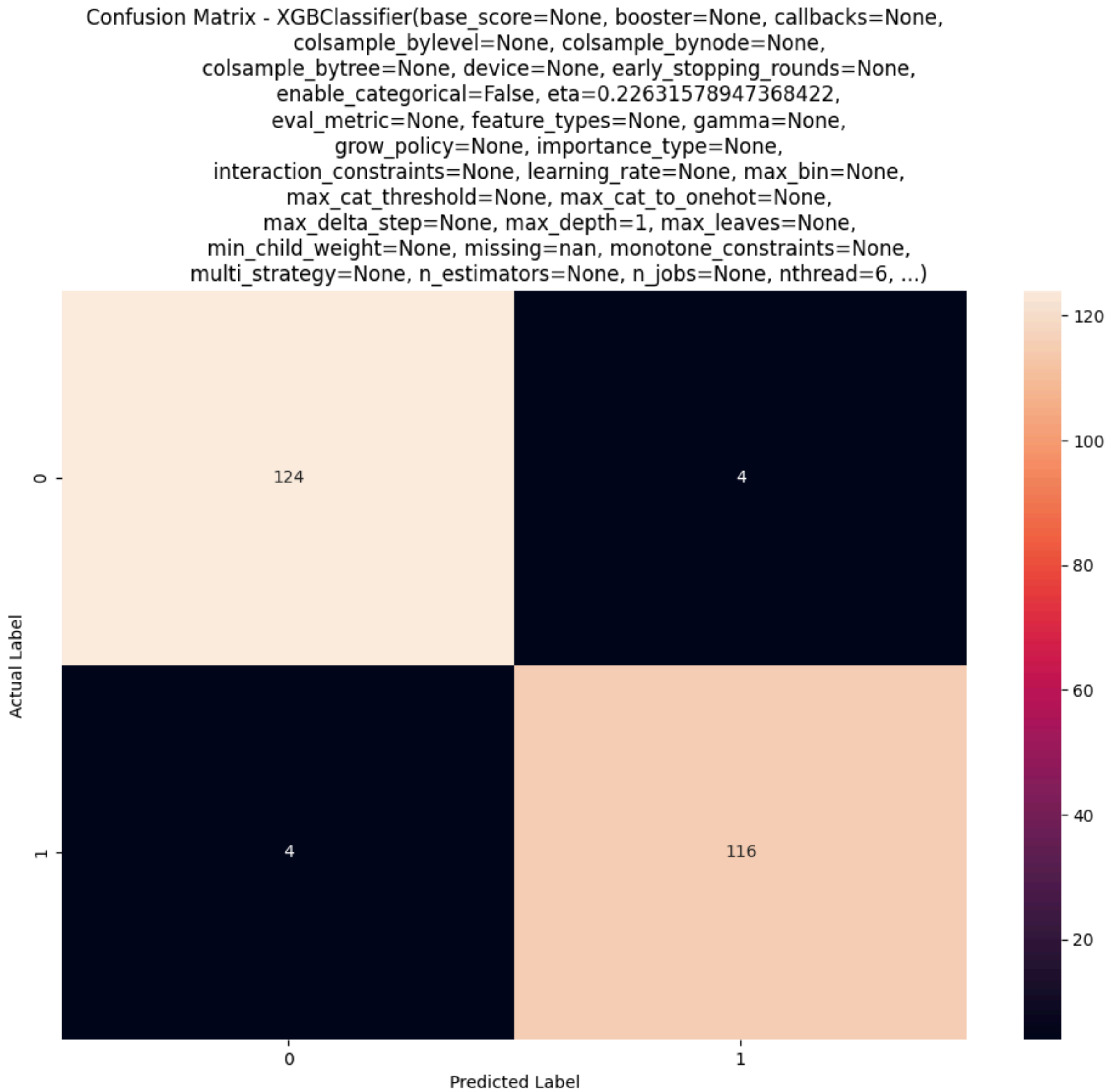
XGBoost

```
xgb2_tuned = XGBClassifier(nthread=6, tree_method='hist', random_state=123, eta = 0.22631578947368422, max_depth = 1)

xgb2_tuned.fit(X2_train_scaled, y2_train)

eval_classification2(xgb2_tuned)
```

```
Accuracy (Test Set): 0.97
Accuracy (Train Set): 0.98
Precision (Test Set): 0.97
Recall (Test Set): 0.97
Recall (Train Set): 0.96
F1-Score (Test Set): 0.97
roc_auc (test-proba): 0.99
roc_auc (train-proba): 1.00
Accuracy (crossval train): 0.9784664572899866
Accuracy (crossval test): 0.9596378257452084
```

```
confusion2(xgb2_tuned)
```

```
Accuracy: 96.77%
Precision: 96.67%
Recall: 96.67%
F1_Score: 96.67%
ROC_AUC: 98.72%
```



As can be seen from the above section, overfitting has been significantly reduced.

Model comparison

Because the target has perfect class balance the primary metric that will be used is `Accuracy`. `Recall` will be the secondary metric as to minimize false negatives.

Before normalization/standardization

```
# Creating models dict
models1_dict = {}
models1_dict['logreg1'] = logreg1_tuned
models1_dict['dt1'] = dt1_tuned
models1_dict['rf1'] = rf1_tuned
models1_dict['knn1'] = knn1_tuned
models1_dict['gb1'] = gb1_tuned
```

```
models1_dict['xgb1'] = xgb1_tuned



# Creating eval data frame
accuracy_test1 = []
accuracy_train1 = []
recall_test_list1 = []
recall_train_list1 = []
accuracy_train_cv1 = []
accuracy_test_cv1 = []
time_elapsed1 = []
for name, model in models1_dict.items():
  start = time.time()
  y_pred1 = model.predict(X1_test)
  y_pred_train1 = model.predict(X1_train)
  end = time.time()

  acc_test1 = accuracy_score(y1_test, y_pred1)
  acc_train1 = accuracy_score(y1_train, y_pred_train1)
  recall_test1 = recall_score(y1_test, y_pred1)
  recall_train1 = recall_score(y1_train, y_pred_train1)

  cv = RepeatedStratifiedKFold(random_state=42, n_repeats = 3)
  score = cross_validate(model, X=X1_train, y=y1_train, cv=cv, scoring='accuracy', return_train_score=True)
  acc_train_cv1 = score['train_score'].mean()
  acc_test_cv1 = score['test_score'].mean()

  accuracy_test1.append(acc_test1)
  accuracy_train1.append(acc_train1)
  recall_test_list1.append(recall_test1)
  recall_train_list1.append(recall_train1)
  accuracy_train_cv1.append(acc_train_cv1)
  accuracy_test_cv1.append(acc_test_cv1)
  time_elapsed1.append(end-start)

eval_dict1 = {'Model': models1_dict.keys(),
             'Accuracy_test': accuracy_test1,
             'Accuracy_train': accuracy_train1,
             'Recall_test': recall_test_list1,
             'Recall train': recall_train_list1,
             'Accuracy_test_crossval': accuracy_test_cv1,
             'Accuracy_train_crossval': accuracy_train_cv1,
             'Time_elapsed': time_elapsed1,
             'Fit_time': fit_time1}

eval_df1 = pd.DataFrame(data=eval_dict1)
eval_df1 = eval_df1.set_index('Model')

eval_df1.style.format(precision=3)
```

```
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:210: ConvergenceWarning: newton-cg failed to converge. Increase the nu
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:425: LineSearchWarning: Rounding errors prevent the line search fr
  warn(msg, LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:203: UserWarning: Line Search failed
  warnings.warn("Line Search failed")
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:210: ConvergenceWarning: newton-cg failed to converge. Increase the nu
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:210: ConvergenceWarning: newton-cg failed to converge. Increase the nu
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:210: ConvergenceWarning: newton-cg failed to converge. Increase the nu
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:425: LineSearchWarning: Rounding errors prevent the line search fr
  warn(msg, LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:203: UserWarning: Line Search failed
  warnings.warn("Line Search failed")
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:425: LineSearchWarning: Rounding errors prevent the line search fr
  warn(msg, LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:203: UserWarning: Line Search failed
  warnings.warn("Line Search failed")
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:425: LineSearchWarning: Rounding errors prevent the line search fr
  warn(msg, LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:203: UserWarning: Line Search failed
  warnings.warn("Line Search failed")
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:425: LineSearchWarning: Rounding errors prevent the line search fr
  warn(msg, LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:203: UserWarning: Line Search failed
  warnings.warn("Line Search failed")
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:210: ConvergenceWarning: newton-cg failed to converge. Increase the nu
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:425: LineSearchWarning: Rounding errors prevent the line search fr
  warn(msg, LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:203: UserWarning: Line Search failed
  warnings.warn("Line Search failed")
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/optimize.py:210: ConvergenceWarning: newton-cg failed to converge. Increase the nu
  warnings.warn(
```