Data Science Internship — EdiGlobe

Minor Project Report

EdiGlobe

Name: Varun Kumar H C

Predictive Modeling for Advertisement Click Prediction Using Logistic Regression

➢ Abstract:

This project aims to predict whether a user will click on an online advertisement using Logistic Regression. By analyzing demographic and behavioral factors such as Age, Area Income, Daily Internet Usage, and Time Spent on Site, the model helps optimize ad targeting. The goal is to improve click-through rate (CTR) and reduce advertising costs by identifying potential customers likely to engage.

➢ Problem Statement, Goal, Objective:

Problem Statement : *Many companies waste advertising budgets showing ads to uninterested users. This project addresses that by identifying which users are most likely to click, improving marketing efficiency.*

Project Goal : *Develop a Logistic Regression model to classify users into "likely to click" or "not likely to click" based on user behavior and demographics.*

Business Objective : *Integrate the model into digital ad platforms to target the right audience, improving ROI and CTR.*

➢ Dataset Description:

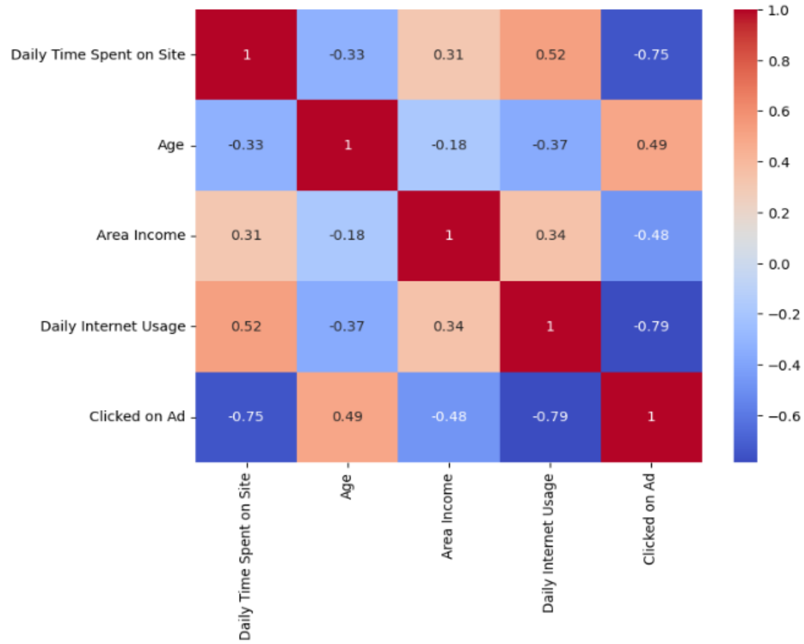
Dataset: r"location\ of \ file\advertising.csv"

Features:

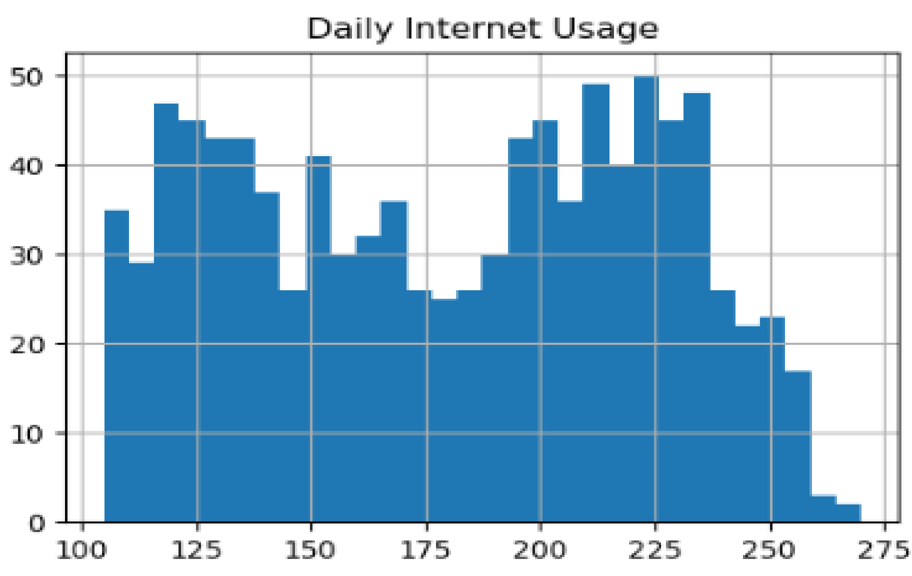
Feature	Description
Daily Time Spent on Site	Average minutes user spends daily
Age	User's age
Area Income	Average income in user's area
Daily Internet Usage	Total daily internet usage
Clicked on Ad	Target variable (1 = Clicked, 0 = Not Clicked)

➢ Data Visualization:

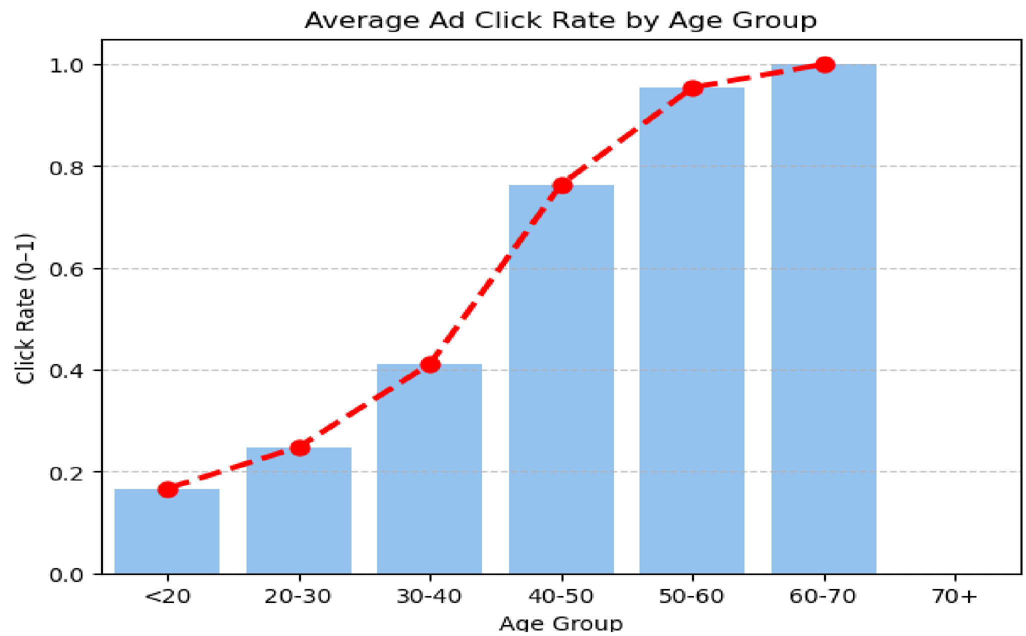
- **Correlation Heatmap**: strong negative correlation between Time Spent on Site and Clicked on Ad



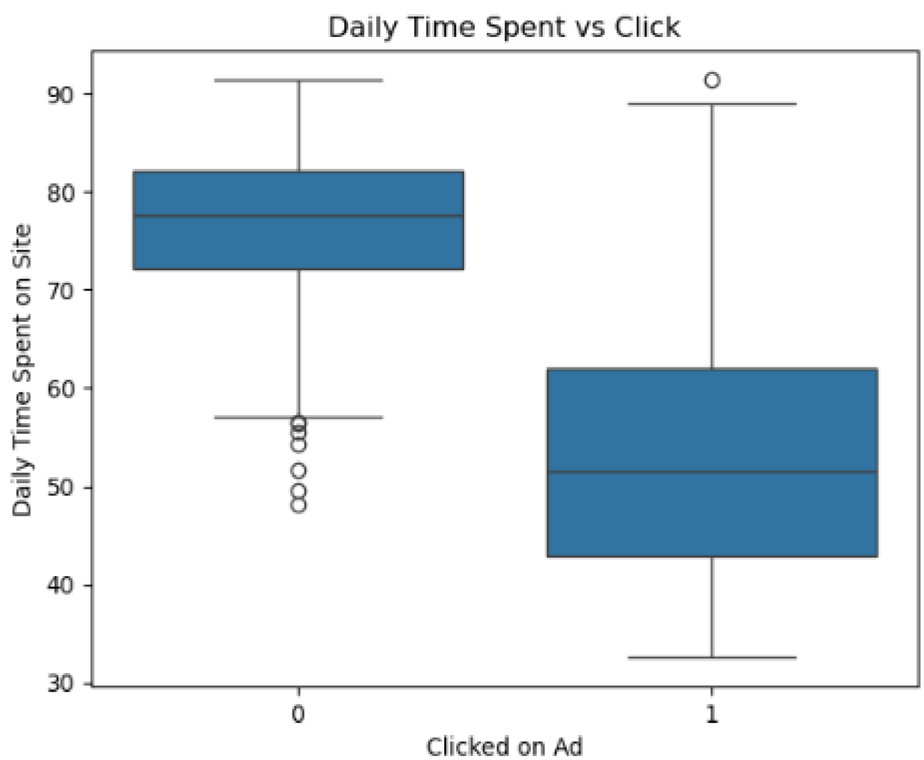
- **Histogram (Daily Internet Usage):** Users with lower internet usage tend to click ads more often.



- **Histogram (Age):** Most users fall between 25–40 years, showing diverse age distribution.



- **Boxplot (Time Spent vs Clicked):** Users spending less time on the site are more likely to click ads.



➢ Model Development:

Algorithm: Logistic Regression

Libraries Used: pandas, numpy, seaborn, matplotlib, scikit-learn

Train–Test Split: 70:30(**(70%) data is** used to train the model , **(30%)** are used to test the model)

➢ Model Evaluation:

```
Accuracy of Model: 97.60%

Confusion Matrix

Predicted 0      Predicted 1
Actual 0         123              2
Actual 1         4                121

Classification Report

              precision    recall  f1-score   support

           0       0.97      0.98      0.98       125
           1       0.98      0.97      0.98       125

    accuracy                           0.98       250
   macro avg       0.98      0.98      0.98       250
weighted avg       0.98      0.98      0.98       250
```

Accuracy of Model: 97.60%”, confusion matrix values, and classification report.

Key Insights

- Users who spend less time on site tend to click more.
- Older users show higher ad-click probability.
- Area Income has a mild positive influence.
- The Logistic Regression model is reliable for future ad-click predictions.

Conclusion

The project successfully built a predictive model capable of identifying potential ad clickers with high accuracy. By deploying this model, businesses can focus on promising users, increase CTR, and optimize digital marketing budgets.

References

1. Scikit-learn Documentation (https://scikit-learn.org)
2. Matplotlib, Seaborn Official Docs
3. EdiGlobe Training Materials

Git : https://github.com/varun5812/Advertisement-Click-Prediction-Using-Logistic-Regression/tree/main jupyter minior_project Last Checkpoint: 35 minutes ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python [conda env:base] * Anaconda Toolbox

#Importing Libraries we needed

[19]:
```
import os
import sys
import logging
from datetime import datetime
import json
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# sklearn
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, roc_auc_score,
    confusion_matrix, classification_report, roc_curve
)
```

```
2025-11-08 21:59:13 INFO Environment setup complete
2025-11-08 21:59:13 INFO Environment setup complete
```

 # give path of dataset

[9]:
```
DATA_PATH = (r"D:\python\automatic\csv files\advertising.csv"  )
df = pd.read_csv(DATA_PATH)

df.head()
```

```
2025-11-08 21:50:51 INFO Loaded data from D:\python\automatic\csv files\advertising.csv, shape: (1000, 10)
```

[9]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp	Clicked on Ad
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11	0
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02	0
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:35:42	0
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19	0
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18	0

#Load data & quick cheaking of data set

[10]:
```
print("Shape:", df.shape)
print("Columns:", df.columns.tolist())
print("Missing values:\n", df.isna().sum())
print("Duplicates:", df.duplicated().sum())
```

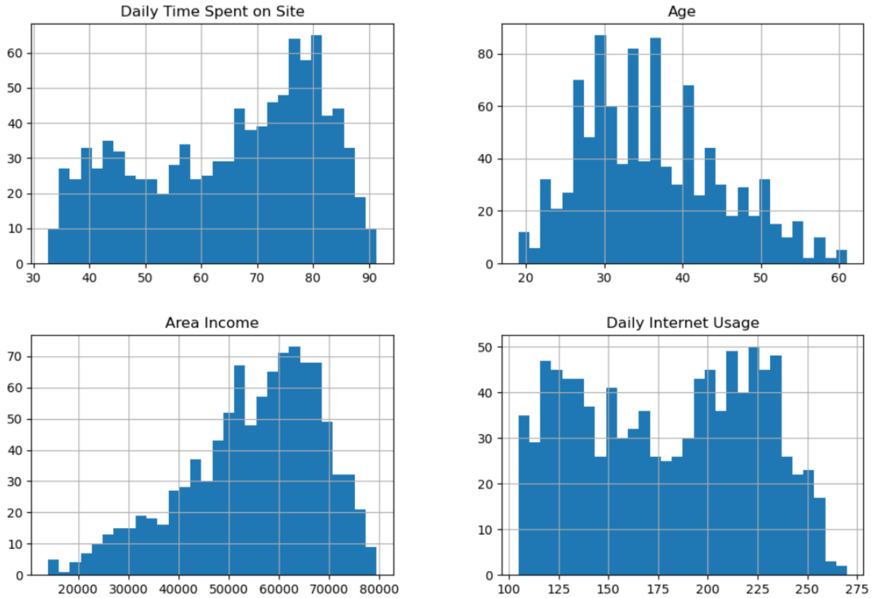
```
Shape: (1000, 10)
Columns: ['Daily Time Spent on Site', 'Age', 'Area Income', 'Daily Internet Usage', 'Ad Topic Line', 'City', 'Male', 'Country', 'Timestamp', 'Clicked on A
d']
Missing values:
 Daily Time Spent on Site    0
Age                         0
Area Income                 0
Daily Internet Usage        0
Ad Topic Line               0
City                        0
Male                        0
Country                     0
Timestamp                   0
Clicked on Ad               0
dtype: int64
Duplicates: 0
```

#Exploratory Data Analysis and visualization

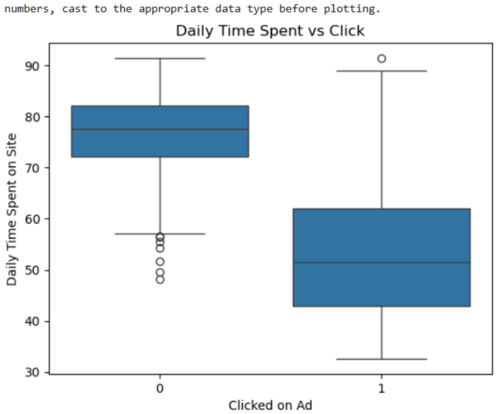
[11]:
```
import matplotlib.pyplot as plt
%matplotlib inline


target_col = "Clicked on Ad"
print(df[target_col].value_counts(normalize=True))
sns.countplot(x=target_col, data=df)
plt.title("Target distribution")
plt.show()


num_cols = ["Daily Time Spent on Site", "Age", "Area Income", "Daily Internet Usage"]
df[num_cols].hist(bins=30, figsize=(12,8))
plt.show()


sns.boxplot(x=target_col, y="Daily Time Spent on Site", data=df)
plt.title("Daily Time Spent vs Click")
plt.show()


plt.figure(figsize=(8,6))
sns.heatmap(df[num_cols + [target_col]].corr(), annot=True, cmap="coolwarm")
plt.show()
```

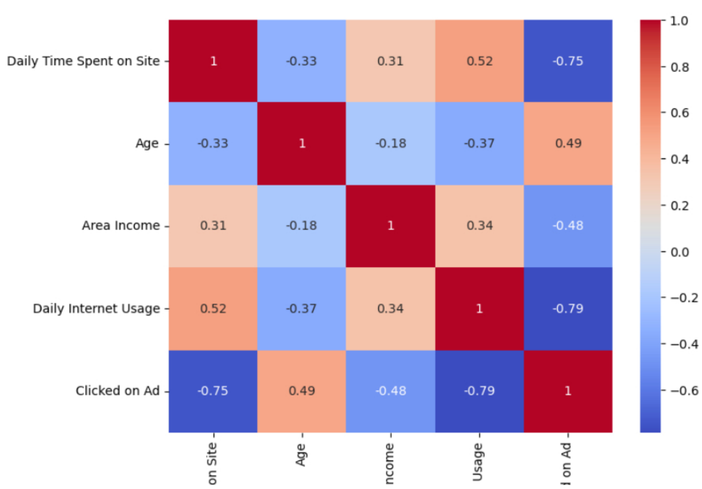
```
Clicked on Ad
0    0.5
1    0.5
Name: proportion, dtype: float64
2025-11-08 21:52:08 INFO Using categorical units to plot a list of strings that are all parsable as floats or dates. If these strings should be plotted as
```

```
numbers, cast to the appropriate data type before plotting.
2025-11-08 21:52:08 INFO Using categorical units to plot a list of strings that are all parsable as floats or dates. If these strings should be plotted as
numbers, cast to the appropriate data type before plotting.
```




```
2025-11-08 21:52:10 INFO Using categorical units to plot a list of strings that are all parsable as floats or dates. If these strings should be plotted as
numbers, cast to the appropriate data type before plotting.
2025-11-08 21:52:10 INFO Using categorical units to plot a list of strings that are all parsable as floats or dates. If these strings should be plotted as
numbers, cast to the appropriate data type before plotting.
```





Daily Time Spent

Area I

Daily Internet

Clickec

[12]:

```
print(df["Male"].value_counts() if "Male" in df.columns else "No Male column")
print(df["Country"].value_counts().head(10) if "Country" in df.columns else "No Country")
print("Sample Ad Topic Lines:")
print(df["Ad Topic Line"].sample(10).values)
```

```
Male
0    519
1    481
Name: count, dtype: int64
Country
Czech Republic    9
France            9
Senegal           8
Peru              8
Greece            8
Micronesia        8
Liberia           8
Turkey            8
Afghanistan       8
South Africa      8
Name: count, dtype: int64
Sample Ad Topic Lines:
['Customizable holistic archive'
 'Self-enabling zero administration neural-net'
 'Quality-focused maximized extranet' 'Organic logistical adapter'
 'Inverse high-level capability' 'Digitized interactive initiative'
 'Implemented disintermediate attitude'
 'Upgradable heuristic system engine' 'Managed national hardware'
 'Multi-layered user-facing paradigm']
```

[22]:

```
df = df.copy()


if "Timestamp" in df.columns:
    df["Timestamp"] = pd.to_datetime(df["Timestamp"])

    df["hour"] = df["Timestamp"].dt.hour
    df["dayofweek"] = df["Timestamp"].dt.dayofweek


if "Ad Topic Line" in df.columns:
    df["ad_topic_len"] = df["Ad Topic Line"].astype(str).apply(len)

    df["ad_topic_words"] = df["Ad Topic Line"].astype(str).apply(lambda x: len(x.split()))


if "Male" in df.columns:
    df["gender"] = df["Male"].map({1: "male", 0: "female"})
    df.drop(columns=["Male"], inplace=True)


cols_to_drop = []
for col in ["City", "Ad Topic Line", "Timestamp"]:
    if col in df.columns:
        cols_to_drop.append(col)
df.drop(columns=cols_to_drop, inplace=True)
logging.info(f"Dropped columns: {cols_to_drop}")

df.head()
```

```
2025-11-08 22:04:52 INFO Dropped columns: []
2025-11-08 22:04:52 INFO Dropped columns: []
```

[22]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Country	Clicked on Ad	hour	dayofweek	ad_topic_len	ad_topic_words	gender
0	68.95	35	61833.90	256.09	Tunisia	0	0	6	34	3	female
1	80.23	31	68441.85	193.77	Nauru	0	1	0	34	3	male
2	69.47	26	59785.94	236.50	San Marino	0	20	6	32	3	female
3	74.15	29	54806.18	245.89	Italy	0	2	6	37	3	male
4	68.37	35	73889.99	225.58	Iceland	0	3	4	29	3	female

[24]:

```
target = "Clicked on Ad"
all_features = [c for c in df.columns if c != target]
numeric_features = df.select_dtypes(include=[np.number]).columns.tolist()
numeric_features = [c for c in numeric_features if c != target]

categorical_features = [c for c in df.columns if c not in numeric_features + [target]]
logging.info(f"Numeric: {numeric_features}, Categorical: {categorical_features}")
```

```
2025-11-08 22:05:15 INFO Numeric: ['Daily Time Spent on Site', 'Age', 'Area Income', 'Daily Internet Usage', 'hour', 'dayofweek', 'ad_topic_len', 'ad_topi
c_words'], Categorical: ['Country', 'gender']
2025-11-08 22:05:15 INFO Numeric: ['Daily Time Spent on Site', 'Age', 'Area Income', 'Daily Internet Usage', 'hour', 'dayofweek', 'ad_topic_len', 'ad_topi
c_words'], Categorical: ['Country', 'gender']
```

[34]:

```
from sklearn.impute import SimpleImputer

numeric_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="constant", fill_value="missing")),
   ("onehot", OneHotEncoder(handle_unknown="ignore", sparse_output=False))

])

preprocessor = ColumnTransformer(transformers=[
    ("num", numeric_transformer, numeric_features),
    ("cat", categorical_transformer, categorical_features)
], remainder='drop')
```

[ ]:

[26]:

```
# cell 9
X = df[all_features]
y = df[target]
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=RANDOM_STATE, stratify=y
)
logging.info(f"Train shape: {X_train.shape}, Test shape: {X_test.shape}")
```

```
2025-11-08 22:06:01 INFO Train shape: (750, 10), Test shape: (250, 10)
```


```
2025-11-08 22:06:01 INFO Train shape: (750, 10), Test shape: (250, 10)
```


[ ]:

*[35]:

```
lr_pipeline = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("clf", LogisticRegression(random_state=RANDOM_STATE, max_iter=1000))
])

lr_pipeline.fit(X_train, y_train)
y_pred = lr_pipeline.predict(X_test)
y_proba = lr_pipeline.predict_proba(X_test)[:,1]


acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc = roc_auc_score(y_test, y_proba)
logging.info(f"Logistic Regression metrics - ACC: {acc:.4f}, PREC: {prec:.4f}, REC: {rec:.4f}, F1: {f1:.4f}, AUC: {roc:.4f}")

print(classification_report(y_test, y_pred))
```

```
2025-11-08 22:08:53 INFO Logistic Regression metrics - ACC: 0.9760, PREC: 0.9837, REC: 0.9680, F1: 0.9758, AUC: 0.9908
```


```
2025-11-08 22:08:53 INFO Logistic Regression metrics - ACC: 0.9760, PREC: 0.9837, REC: 0.9680, F1: 0.9758, AUC: 0.9908
```


```
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       125
           1       0.98      0.97      0.98       125

    accuracy                           0.98       250
   macro avg       0.98      0.98      0.98       250
weighted avg       0.98      0.98      0.98       250
```

[ ]: