AKASH KUMAR BANIK

24MCA0242

TASK - 1

Task 1: Import the diabetes dataset to your platform from the link given below and perform the following:

1. Create a data frame and display the number of samples and features with the datatype
2. Count the number of diabetes patient who never smoke
3. Display all the statistical measure about the data frame
4. Find the number of samples having missing values
5. Print the first 50 samples from the dataset

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
df=pd.read_csv('/content/drive/MyDrive/ML_Datasets/diabetes_prediction_dataset.
csv')
```

```
print("Number of row:",df.shape[0])  # Size of the sample
```

```
Number of row: 100000
```

```
print("Number of columns:",df.shape[1])   # Number of features in dataset
```

```
Number of columns: 9
```

```
# Printing the datatypes of the features

df.dtypes
```

	0
gender	object
age	float64
hypertension	int64
heart_disease	int64
smoking_history	object
bmi	float64
HbA1c_level	float64
blood_glucose_level	int64
diabetes	int64

dtype: object

```
# 2. Count the number of diabetes patient who never smoke

never_smoked_diabetes = df[(df['smoking_history'] == 'never') & (df['diabetes'] == 1)]
num_never_smoked_diabetes = len(never_smoked_diabetes)
print("No. of patients who never smoked:",num_never_smoked_diabetes)
```

```
No. of patients who never smoked: 3346
```

```
# 3. Display all the statistical measure about the data frame

print(df.describe())
```

```
                 age  hypertension  heart_disease            bmi  \
count  100000.000000  100000.00000  100000.000000  100000.000000
mean       41.885856       0.07485       0.039420      27.320767
std        22.516840       0.26315       0.194593       6.636783
min         0.080000       0.00000       0.000000      10.010000
25%        24.000000       0.00000       0.000000      23.630000
50%        43.000000       0.00000       0.000000      27.320000
75%        60.000000       0.00000       0.000000      29.580000
max        80.000000       1.00000       1.000000      95.690000

         HbA1c_level  blood_glucose_level       diabetes
count  100000.000000        100000.000000  100000.000000
mean        5.527507           138.058060       0.085000
std         1.070672            40.708136       0.278883
min         3.500000            80.000000       0.000000
25%         4.800000           100.000000       0.000000
50%         5.800000           140.000000       0.000000
75%         6.200000           159.000000       0.000000
max         9.000000           300.000000       1.000000
```

```
# 4. Find the number of samples having missing values

missing_values = df.isnull().sum()
print(missing_values)
```

```
gender                 0
age                    0
hypertension           0
heart_disease          0
smoking_history        0
bmi                    0
HbA1c_level            0
blood_glucose_level    0
diabetes               0
dtype: int64
```

```
# 5. Print the first 50 samples from the dataset

df.head(50)
```

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
0	Female	80.0	0	1	never	25.19	6.6	140	0
1	Female	54.0	0	0	No Info	27.32	6.6	80	0
2	Male	28.0	0	0	never	27.32	5.7	158	0
3	Female	36.0	0	0	current	23.45	5.0	155	0
4	Male	76.0	1	1	current	20.14	4.8	155	0
5	Female	20.0	0	0	never	27.32	6.6	85	0
6	Female	44.0	0	0	never	19.31	6.5	200	1
7	Female	79.0	0	0	No Info	23.86	5.7	85	0
8	Male	42.0	0	0	never	33.64	4.8	145	0
9	Female	32.0	0	0	never	27.32	5.0	100	0
10	Female	53.0	0	0	never	27.32	6.1	85	0
11	Female	54.0	0	0	former	54.70	6.0	100	0
12	Female	78.0	0	0	former	36.05	5.0	130	0
13	Female	67.0	0	0	never	25.69	5.8	200	0
14	Female	76.0	0	0	No Info	27.32	5.0	160	0
15	Male	78.0	0	0	No Info	27.32	6.6	126	0
16	Male	15.0	0	0	never	30.36	6.1	200	0
17	Female	42.0	0	0	never	24.48	5.7	158	0
18	Female	42.0	0	0	No Info	27.32	5.7	80	0
19	Male	37.0	0	0	ever	25.72	3.5	159	0
20	Male	40.0	0	0	current	36.38	6.0	90	0
21	Male	5.0	0	0	No Info	18.80	6.2	85	0
22	Female	69.0	0	0	never	21.24	4.8	85	0
23	Female	72.0	0	1	former	27.94	6.5	130	0
24	Female	4.0	0	0	No Info	13.99	4.0	140	0
25	Male	30.0	0	0	never	33.76	6.1	126	0
26	Male	67.0	0	1	not current	27.32	6.5	200	1
27	Male	40.0	0	0	former	27.85	5.8	80	0
28	Male	45.0	1	0	never	26.47	4.0	158	0
29	Male	43.0	0	0	never	26.08	6.1	155	0
30	Female	53.0	0	0	No Info	31.75	4.0	200	0
31	Male	50.0	0	0	No Info	25.15	4.0	145	0
32	Female	41.0	0	0	current	22.01	6.2	126	0
33	Female	20.0	0	0	never	22.19	3.5	100	0
34	Female	76.0	0	0	never	23.55	5.0	85	0
35	Male	5.0	0	0	No Info	15.10	5.8	85	0
36	Female	15.0	0	0	No Info	21.76	4.5	130	0
37	Female	26.0	0	0	never	21.22	6.6	200	0
38	Male	50.0	1	0	current	27.32	5.7	260	1
39	Female	34.0	0	0	never	56.43	6.2	200	0
40	Male	73.0	0	0	former	25.91	9.0	160	1
41	Male	5.0	0	0	No Info	27.32	6.6	130	0
42	Female	77.0	1	1	never	32.02	5.0	159	0
43	Female	66.0	0	0	No Info	29.30	4.8	159	0
44	Female	67.0	0	0	No Info	27.32	3.5	160	0
45	Female	44.0	0	0	never	24.93	6.1	100	0
46	Female	29.0	0	0	never	19.95	5.0	90	0
47	Female	60.0	0	0	never	18.03	4.0	159	0
48	Female	38.0	0	0	never	28.27	6.2	155	0

49	Female	3.0	0	0	No Info	19.27	6.5	100	0

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

TASK-2 PREPOCESSING

Task 2:

1. Find the number of missing samples for each feature and if there is any missing value found, then fill the values based on the type of data(continuous/discrete).
2. Remove the duplicated sample from the dataset.
3. Normalize the input feature “blood_glucose_level” to the range of 0 to 1(Use the appropriate normalization type based on the requirement).
4. Map all the categorical data to ordinal data.
5. Identify the outlier range (lower bound and upper bound) for the above dataset and list the outliers.

```
# 1. Find the number of missing samples for each feature and if there is any missing value found, then fill the values based on the type

#Check for missing values
print("Number of missing values:")
print(df.isnull().sum())
```

```
Number of missing values:
gender                 0
age                    0
hypertension           0
heart_disease          0
smoking_history        0
bmi                    0
HbA1c_level            0
blood_glucose_level    0
diabetes               0
dtype: int64
```

```
# 1. Fill Missing Values (continuous -> mean, categorical -> mode)

for col in df.columns:
  if df[col].dtype == 'object':
    df[col].fillna(df[col].mode()[0])
  else:
    df[col].fillna(df[col].mean())
```

```
# 2. Remove the duplicated sample from the dataset

print("Number of duplicates:")
print(df.duplicated().sum())  #Identifying duplicates

#Removing duplicates
print("Number of duplicates after removing duplicates:")
df = df.drop_duplicates()
print(df.duplicated().sum())
```

```
Number of duplicates:
3854
Number of duplicates after removing duplicates:
0
```

```
# 3. Normalize the input feature “blood_glucose_level” to the range of 0 to 1.
# (Use the appropriate normalization type based on the requirement).

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

# Convert to float64 before scaling
df['blood_glucose_level'] = df['blood_glucose_level'].astype(float)

df.loc[:, 'blood_glucose_level'] = scaler.fit_transform(df[['blood_glucose_level']]).round(3)

df = df.round(3)  # Round all columns to 3 decimal places
print(df['blood_glucose_level'].head())
```

```
0    0.273
1    0.000
2    0.355
3    0.341
4    0.341
Name: blood_glucose_level, dtype: float64
```

```
# 4. Map all the categorical data to ordinal data.

from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
categorical_cols = ['gender', 'smoking_history']
for col in categorical_cols:
  df[col] = encoder.fit_transform(df[col])

print(df[['gender','smoking_history']].head())
```

```
   gender  smoking_history
0       0                4
1       0                0
2       1                4
3       0                1
4       1                1
```

```
# 5. Identify the outlier range (lower bound and upper bound)
# for the above dataset and list the outliers.

Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

lower_bound = (Q1 - 1.5 * IQR).round(3)
upper_bound = (Q3 + 1.5 * IQR).round(3)

# Print lower and upper bounds for each column
print("Lower and Upper Bounds for each columns:")
for column in df.columns:
    print(f"{column:25} lower bound: {lower_bound[column]:<10} upper bound: {upper_bound[column]:<10}")

# Identify and list outliers (this part remains the same)
outliers = df[((df < lower_bound) | (df > upper_bound)).any(axis=1)]
print("\nOutliers detected:\n",outliers.sum())
print("\nOutliers:")
print(outliers)
```

```
Lower and Upper Bounds for each columns:
gender                    lower bound: -1.5       upper bound: 2.5
age                       lower bound: -28.5      upper bound: 111.5
hypertension              lower bound: 0.0        upper bound: 0.0
heart_disease             lower bound: 0.0        upper bound: 0.0
smoking_history           lower bound: -6.0       upper bound: 10.0
bmi                       lower bound: 13.71      upper bound: 39.55
HbA1c_level               lower bound: 2.7        upper bound: 8.3
blood_glucose_level       lower bound: -0.311     upper bound: 0.761
diabetes                  lower bound: 0.0        upper bound: 0.0

Outliers detected:
 gender                    8677.000
age                    1121708.080
hypertension              7461.000
heart_disease             3923.000
smoking_history          51448.000
bmi                     639438.530
HbA1c_level             118250.800
blood_glucose_level       7047.033
diabetes                  8482.000
dtype: float64

Outliers:
       gender   age  hypertension  heart_disease  smoking_history    bmi  \
0           0  80.0             0              1                4  25.19
4           1  76.0             1              1                1  20.14
6           0  44.0             0              0                4  19.31
11          0  54.0             0              0                3  54.70
23          0  72.0             0              1                3  27.94
...       ...   ...           ...            ...              ...    ...
99962       0  58.0             1              0                4  38.31
99963       0  51.0             1              0                0  28.67
99979       0  61.0             0              0                1  30.11
99984       1  80.0             1              0                0  20.96
99993       0  40.0             0              0                4  40.69

       HbA1c_level  blood_glucose_level  diabetes
0              6.6                0.273         0
```

```
4           4.8           0.341      0
6           6.5           0.545      1
11          6.0           0.091      0
23          6.5           0.227      0
...         ...             ...    ...
99962       7.0           0.545      1
99963       6.1           0.295      0
99979       6.2           0.727      1
99984       6.6           0.023      0
99993       3.5           0.341      0

[19505 rows x 9 columns]
```

TASK-3.1 LINEAR REGRESSION (MANUAL)

TASK - 3.1

From the above dataset, consider only two attribute in the input dataset namely “blood_glucose_level” and “diabetes” for the Task3.1 and Task 3.2. Based on the blood glucose level, the person is classified under diabetic as “1” or non-diabetic as “0”. Perform the following,

1. Split the given dataset into training and testing dataset.
2. Assign the weights and bias using any of the approach (formula or taking random values).
3. Write your own code for building the Linear Regression model using the training dataset and predict the target class.
4. Calculate the Error deviation using test dataset by applying any measures and also find the Accuracy of the model.
5. Once the model is built, predict whether a person is diabetic or not for the given blood_glucose_level =155.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
```

```
# 1. Split the given dataset into training and testing dataset.
x = df[['blood_glucose_level']]
y = df['diabetes']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
# Assign the weights and bias using any of the approach (formula or taking random values).

model= LinearRegression()
X=np.array(x_train).reshape(-1,1)
Y=np.array(y_train).reshape(-1,1)
model.fit(X,Y)                   #Fitting data to model
w= float(model.coef_[0].item())    #Weight(Slope)
b= float(model.intercept_.item())  #Bias(Intercept)
print("Weight: ", w, "Bias: ", b)
```

```
Weight:  0.6446693668199555 Bias:  -0.08240817870212248
```

```
# 3. Write your own code for building the Linear Regression model using the training dataset and predict the target class.

def linearReg(w,b,x):
    m= x.shape[0]
    y= np.zeros(m)
    for i in range(m):
        y[i]= w*x[i]+b
    return y
```

```
y_pred= linearReg(w,b,np.array(x_test).reshape(-1,1))
y_pred_class= [1 if i >0.5 else 0 for i in y_pred]
print(y_pred_class)
# Sample Output: [0, 0, 0, 0, 0, 0,......, 0, 0, 0, 0, 0] output too long
```

Show hidden output

```
# 4. Calculate the Error deviation using test dataset by applying any measures and also find the Accuracy of the model.

mae= mean_absolute_error(y_test,y_pred_class)
mse = mean_squared_error(y_test, y_pred_class)
r2 = r2_score(y_test, y_pred_class)
print(mae)
print(mse)
print(r2)
```

```
0.07566302652106084
0.07566302652106084
0.07146118163053783
```

```
df.head(10)
```

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
0	0	80.0	0	1	4	25.19	6.6	0.273	0
1	0	54.0	0	0	0	27.32	6.6	0.000	0
2	1	28.0	0	0	4	27.32	5.7	0.355	0
3	0	36.0	0	0	1	23.45	5.0	0.341	0
4	1	76.0	1	1	1	20.14	4.8	0.341	0
5	0	20.0	0	0	4	27.32	6.6	0.023	0
6	0	44.0	0	0	4	19.31	6.5	0.545	1
7	0	79.0	0	0	0	23.86	5.7	0.023	0
8	1	42.0	0	0	4	33.64	4.8	0.295	0
9	0	32.0	0	0	4	27.32	5.0	0.091	0

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

```
# 5. Once the model is built, predict whether a person is diabetic or not for the given blood_glucose_level =155

# calculating threshold value to determine whether diabetic or not
med=np.median(df['blood_glucose_level'])
print("Median Value:",med)

# predicting whether a person is diabetic or not for the given blood_glucose_level =155
t=w*155+b
if(t>0.273):
    print(1)
else:
    print(0)
```

```
Median Value: 0.273
1
```

TASK - 3.2 LINEAR REGRESSION (USING IN-BUILT FUNCTIONS)

TASK - 3.2

Modify the code executed in task-3.1. Step 2, 3 & 4 in the above task can be replaced with inbuilt functions. Compare the error and accuracy in both the task. If there is any deviation, write your inference and observations in the last line of code as comment line

```
y_pred2=model.predict(np.array(x_test).reshape(-1,1))
y_pred_class2= [1 if i >0.5 else 0 for i in y_pred2]
```

```
mae2 = mean_absolute_error(y_test,y_pred_class2)
mse2 = mean_squared_error(y_test, y_pred_class2)
r22 = r2_score(y_test, y_pred_class2)
print(mae2)
print(mse2)
print(r22)
```

```
0.07566302652106084
0.07566302652106084
0.07146118163053783
```

```
# Since the Weight and Bias are calculated using inbuilt functions both methods have same error and accuracy.
```

TASK - 4 MULTI-LINEAR REGRESSION

TASK - 4:

Using the diabetes dataset, build the multilinear regression model and perform the following

1. Split the dataset for training and testing
2. Display the intercepts/constant values calculated
3. Calculate the accuracy of the model
4. Draw the comparison graph with y and predicted y
5. Predict the person is diabetic or not for the new input feature “Female,36,0,0,current,32.27,6.2,220”

```
# 1. Split the dataset for training and testing

most_frequent_gender = df['gender'].mode()[0]
df['gender'].fillna(most_frequent_gender, inplace=True)
most_frequent = df['smoking_history'].mode()[0]
df['smoking_history'].fillna(most_frequent, inplace=True)
print(df.isna().sum())
X=df.drop('diabetes',axis=1) #Independent Classes
Y=df['diabetes']  #Target Class
#Training and testing data split
x_train, x_test, y_train, y_test= train_test_split(X, Y, test_size=0.25, random_state=42)
```

```
gender                 0
age                    0
hypertension           0
heart_disease          0
smoking_history        0
bmi                    0
HbA1c_level            0
blood_glucose_level    0
diabetes               0
dtype: int64
<ipython-input-171-6eb0c9d8225e>:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained as
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[co]


  df['gender'].fillna(most_frequent_gender, inplace=True)
<ipython-input-171-6eb0c9d8225e>:6: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained as
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[co]


  df['smoking_history'].fillna(most_frequent, inplace=True)
```

```
#Bulilding model
mlr= LinearRegression()
mlr.fit(x_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
# 3. Display the intercepts/constant values calculated

print("Intercept: ", mlr.intercept_)
print("Coefficients:")
list(zip(X, mlr.coef_))
```

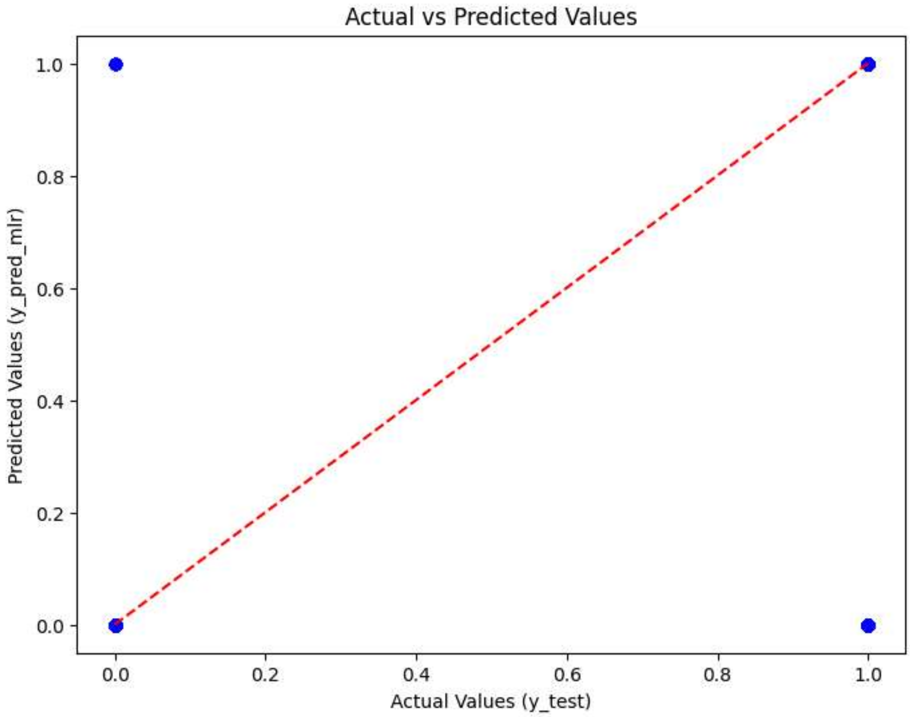
```
Intercept:  -0.6999401542304063
Coefficients:
[('gender', 0.013977001479357933),
 ('age', 0.0014636029402034694),
 ('hypertension', 0.09139938900720733),
 ('heart_disease', 0.11934209213589558),
 ('smoking_history', 0.0012872660602438997),
 ('bmi', 0.004153848086306815),
 ('HbA1c_level', 0.0828751756495833),
 ('blood_glucose_level', 0.5069342514180285)]
```

```
# Predicting testing data
y_pred_mlr= mlr.predict(x_test)
rest=[1 if i >0.5 else 0 for i in y_pred_mlr]
```

```
# 3. Calculate the accuracy of the model

mlr_mae= mean_absolute_error(y_test,rest)
mlr_mse = mean_squared_error(y_test, rest)
mlr_r2 = r2_score(y_test, rest)
print(mlr_mae)
print(mlr_mse)
print(mlr_r2)
```

```
0.06298622956275741
0.06298622956275741
0.23211100370967497
```

```
# 4. Draw the comparison graph with y and predicted y

import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))
plt.scatter(y_test, rest,color='blue')
plt.title('Actual vs Predicted Values')
plt.xlabel('Actual Values (y_test)')
plt.ylabel('Predicted Values (y_pred_mlr)')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--')
plt.show()
```




```
# 5. Predict the person is diabetic or not for the new input feature
# “Female,36,0,0,current,32.27,6.2,220”

sample= [[0,36,0,0,2,32.27,6.2,.220]]
sam=pd.DataFrame(sample, columns=['gender','age','hypertension','heart_disease','smoking_history','bmi','HbA1c_level','blood_glucose_lev
mlr_rest= mlr.predict(sam)
if mlr_rest<0.5:
    print(1)
else:
    print(0)
```

```
1
```

TASK - 5 LOGISTIC REGRESSION

TASK - 5:

Using the diabetes dataset build the logistic regression model and perform the following

1. Split the dataset for training and testing
2. Build and Test the model
3. Evaluate the model using confusion matrix and other measures. Also, Calculate the accuracy of the model
4. Draw the comparison graph with y and predicted y
5. Predict the person is diabetic or not for the new input feature “Male,80,0,0,never,22.06,9,155”

```
# 1. Split the dataset for training and testing
# 2. Build and Test the model

from sklearn.linear_model import LogisticRegression

# instantiate the model (using the default parameters)
X = df.drop('diabetes', axis = 1)
Y = df['diabetes']
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, random_state=42)
logreg = LogisticRegression(random_state=16)
logreg.fit(x_train, y_train)
y_pred = logreg.predict(x_test)

from sklearn.metrics import classification_report
target_names = ['0', '1']
print(classification_report(y_test, y_pred,target_names=target_names))
```

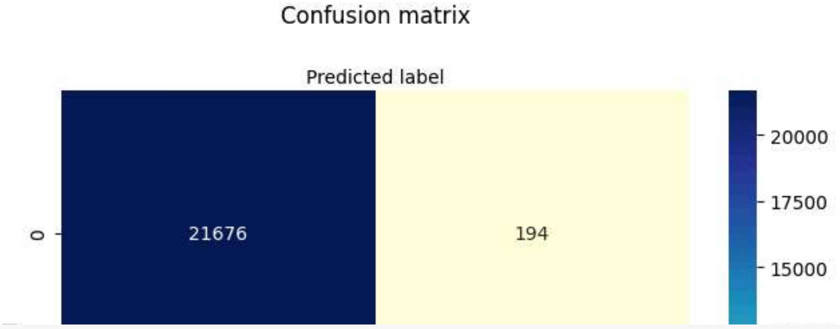
```
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     21870
           1       0.87      0.61      0.72      2167

    accuracy                           0.96     24037
   macro avg       0.92      0.80      0.85     24037
weighted avg       0.95      0.96      0.95     24037

/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
# 3. Evaluate the model using confusion matrix and other measures. Also, Calculate the accuracy of the model

from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
print(cnf_matrix)
```

```
[[21676   194]
 [  840  1327]]
```

```
# Heatmap Plot

ax= sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu",fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 427.9555555555555, 'Predicted label')
```




```
# 4. Draw the comparison graph with y and predicted y

plt.figure(figsize=(8,6))
plt.scatter(y_test, rest,color='blue')
plt.title('Actual vs Predicted Values')
plt.xlabel('Actual Values (y_test)')
plt.ylabel('Predicted Values (y_pred_mlr)')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--')
plt.show()
```