Double-click (or enter) to edit

EDA

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
plt.style.use('ggplot')
pd.set_option('max_columns',200)
```

```
df = pd.read_csv("music_genre.csv")
```

```
# top 5 row in the dataset
df.head()
```

	instance_id	artist_name	track_name	popularity	acousticness	danceability	duratio
0	32894.0	Röyksopp	Röyksopp's Night Out	27.0	0.00468	0.652	
1	46652.0	Thievery Corporation	The Shining Path	31.0	0.01270	0.622	218
2	30097.0	Dillon Francis	Hurricane	28.0	0.00306	0.620	215
3	62177.0	Dubloadz	Nitro	34.0	0.02540	0.774	166
4	24907.0	What So Not	Divide & Conquer	32.0	0.00465	0.638	222

```
#how many number of columns and  row
df.shape
```

```
(50005, 18)
```

```
#All columns name
df.columns
```

```
Index(['instance_id', 'artist_name', 'track_name', 'popularity',
       'acousticness', 'danceability', 'duration_ms', 'energy',
       'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
       'speechiness', 'tempo', 'obtained_date', 'valence', 'music_genre'],
      dtype='object')
```

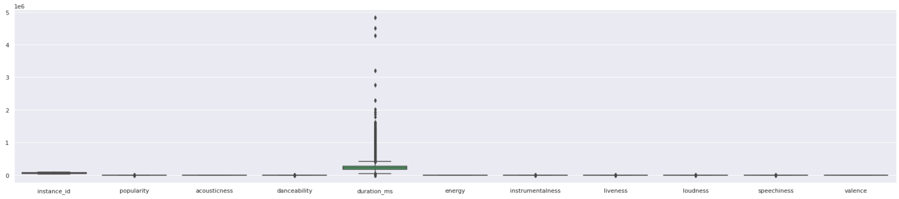
```
#All columns name
df.columns
```

```
Index(['instance_id', 'artist_name', 'track_name', 'popularity',
       'acousticness', 'danceability', 'duration_ms', 'energy',
       'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
       'speechiness', 'tempo', 'obtained_date', 'valence', 'music_genre'],
      dtype='object')
```

```
df.describe()
```

	instance_id	popularity	acousticness	danceability	duration_ms	energy
count	50000.000000	50000.000000	50000.000000	50000.000000	5.000000e+04	50000.00000
mean	55888.396360	44.220420	0.306383	0.558241	2.212526e+05	0.59975
std	20725.256253	15.542008	0.341340	0.178632	1.286720e+05	0.26455

```
sns.set(rc={'figure.figsize':(30,6)})

# Create box plot
sns.boxplot(data=df)

# Show plot
plt.show()
```



```
#Check for missing value
df.isna().sum()
```

```
     instance_id         5
     artist_name         5
     track_name          5
     popularity          5
     acousticness        5
     danceability        5
     duration_ms         5
     energy              5
     instrumentalness    5
     key                 5
     liveness            5
     loudness            5
     mode                5
     speechiness         5
     tempo               5
     obtained_date       5
     valence             5
     music_genre         5
     dtype: int64
```

```
df= df.dropna()
```

```
# from sklearn.preprocessing import LabelEncoder

# le = LabelEncoder()
# df['GENDER'] = le.fit_transform(df['GENDER'])
# df['LUNG_CANCER'] = le.fit_transform(df['LUNG_CANCER'])
```

```
df.dtypes
```

```
     instance_id         float64
     artist_name          object
     track_name           object
     popularity          float64
     acousticness        float64
     danceability        float64
     duration_ms         float64
     energy              float64
```

```
instrumentalness    float64
key                  object
liveness            float64
loudness            float64
mode                 object
speechiness         float64
tempo                object
obtained_date        object
valence             float64
music_genre          object
dtype: object
```

```
df.head()
```

	instance_id	artist_name	track_name	popularity	acousticness	danceability	duratio
0	32894.0	Röyksopp	Röyksopp's Night Out	27.0	0.00468	0.652	
1	46652.0	Thievery Corporation	The Shining Path	31.0	0.01270	0.622	2182
2	30097.0	Dillon Francis	Hurricane	28.0	0.00306	0.620	2156
3	62177.0	Dubloadz	Nitro	34.0	0.02540	0.774	1668
4	24907.0	What So Not	Divide & Conquer	32.0	0.00465	0.638	2223

```
#convert categorical columns to numbers
# df = port
from sklearn.preprocessing import LabelEncoder

# Create a label encoder object
le = LabelEncoder()

# Iterate over all the values of each column and extract their dtypes
for col in df:
    if df[col].dtype=='object':
        # Use the label encoder object to fit_transform
        df[col]=le.fit_transform(df[col])
```

```
df.head()
```

	instance_id	artist_name	track_name	popularity	acousticness	danceability	duratio
0	32894.0	5029	28371	27.0	0.00468	0.652	
1	46652.0	6117	34817	31.0	0.01270	0.622	2182
2	30097.0	1591	15024	28.0	0.00306	0.620	2156
3	62177.0	1707	23372	34.0	0.02540	0.774	1668
4	24907.0	6519	8649	32.0	0.00465	0.638	2223

```
df.dtypes
```

```
instance_id         float64
artist_name           int64
track_name            int64
popularity          float64
acousticness        float64
danceability        float64
duration_ms         float64
energy              float64
instrumentalness    float64
key                   int64
liveness            float64
loudness            float64
mode                  int64
speechiness         float64
tempo                 int64
obtained_date         int64
valence             float64
```

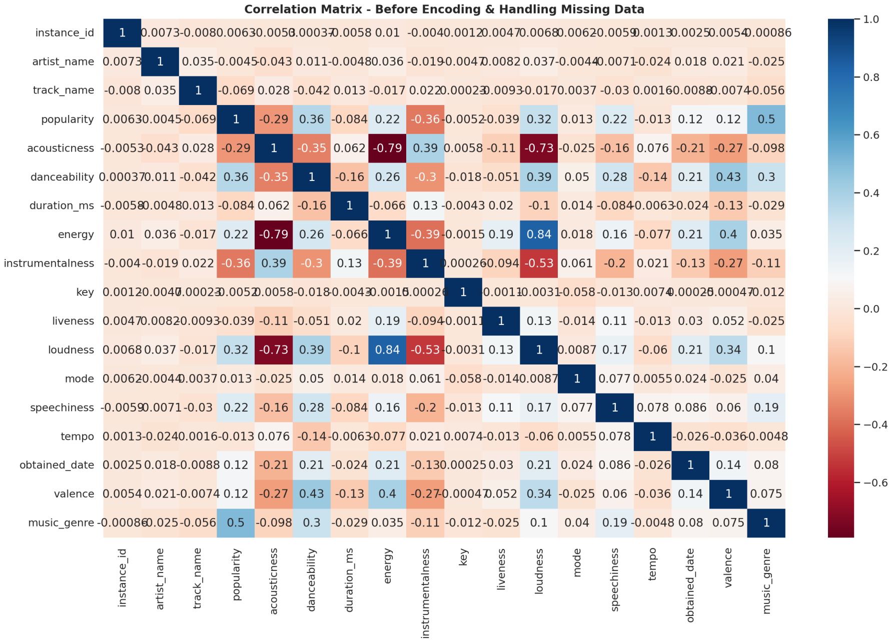
```
    music_genre          int64
    dtype: object
```

```
df = df.dropna()
```

```
df_corr = df[['instance_id', 'artist_name', 'track_name', 'popularity',
       'acousticness', 'danceability', 'duration_ms', 'energy',
       'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
       'speechiness', 'tempo', 'obtained_date', 'valence', 'music_genre']].corr()
df_corr
```

	instance_id	artist_name	track_name	popularity	acousticness	danceab
instance_id	1.000000	0.007326	-0.008042	0.006317	-0.005268	0.0
artist_name	0.007326	1.000000	0.034748	-0.004463	-0.043094	0.0
track_name	-0.008042	0.034748	1.000000	-0.069139	0.027580	-0.0
popularity	0.006317	-0.004463	-0.069139	1.000000	-0.290453	0.3
acousticness	-0.005268	-0.043094	0.027580	-0.290453	1.000000	-0.3
danceability	0.000372	0.010550	-0.041728	0.356420	-0.347681	1.0
duration_ms	-0.005848	-0.004754	0.013115	-0.083809	0.061862	-0.1
energy	0.009952	0.036248	-0.016984	0.216345	-0.791250	0.2
instrumentalness	-0.004015	-0.019471	0.021728	-0.364960	0.387970	-0.3
key	0.001189	-0.004723	0.000232	-0.005212	0.005796	-0.0
liveness	0.004737	0.008216	-0.009292	-0.039468	-0.109220	-0.0
loudness	0.006847	0.037436	-0.017243	0.317941	-0.730401	0.3
mode	0.006189	-0.004415	0.003666	0.013427	-0.025161	0.0
speechiness	-0.005908	-0.007084	-0.030364	0.224309	-0.163377	0.2
tempo	0.001303	-0.023968	0.001585	-0.012906	0.076434	-0.1
obtained_date	0.002523	0.018221	-0.008815	0.120444	-0.205964	0.2
valence	0.005385	0.021287	-0.007354	0.124913	-0.270238	0.4
music_genre	-0.000861	-0.025123	-0.056281	0.502133	-0.097969	0.3

```
plt.figure(figsize = (16,10), dpi=200)
ax = plt.axes()
sns.heatmap(df.corr(), annot = True, cmap='RdBu', ax=ax)
ax.set_title('Correlation Matrix - Before Encoding & Handling Missing Data', weight='bold')
plt.show()
```



```
columns = ['key','mode','tempo']
for column in columns:
    print(df[column].unique())
```

```
 [ 1  5 11  4  9  2 10  8  0  3  7  6]
 [1 0]
 [  274  4193  8138 ...  3631 16869 29157]
```

```
def preprocess_inputs(df):
    df = df.copy()

    df = df.drop(['instance_id','artist_name','track_name','obtained_date'],axis=1)

    df['mode'] = df['mode'].replace({'Minor' : 0,
                                     'Major' : 1})

    embarked_dummies = pd.get_dummies(df.key)
    df = pd.concat([df, embarked_dummies], axis=1)
    df = df.drop('key',axis=1)

    df['tempo'] = df['tempo'].replace('?',np.nan)
    df["tempo"] = df["tempo"].astype("float")
    df['tempo'] = df['tempo'].fillna(df['tempo'].mean())

    df['music_genre'] = df['music_genre'].replace({'Electronic':0, 'Anime':1, 'Jazz':2, 'Alternative':3, 'Country':4, 'Rap':5,
                                                    'Blues':5, 'Rock':6, 'Classical':7, 'Hip-Hop':8})

    y = df['music_genre']
    X = df.drop('music_genre',axis=1)

    X_train, X_test, y_train, y_test = train_test_split(X,y, train_size = 0.7, shuffle=True, random_state=43)

    scaler = StandardScaler()
    scaler.fit(X_train)

    X_train = pd.DataFrame(scaler.transform(X_train), index = X_train.index, columns = X_train.columns)
    X_test = pd.DataFrame(scaler.transform(X_test), index = X_test.index, columns = X_test.columns)

    return X_train, X_test, y_train, y_test
```

DT

```
import pandas as pd
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

# Split the data into features and labels
X = df.drop('music_genre', axis=1)
y = df['music_genre']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
# train, test, labels_train, labels_test = sklearn.model_selection.train_test_split(iris.data, iris.target, train_size=0.80)


# Define the Decision Tree model
dt = DecisionTreeClassifier()

# Train the Decision Tree model
dt.fit(X_train, y_train)

# Make predictions on the test set
y_pred_dt = dt.predict(X_test)

# Evaluation using accuracy score
acc_dt = accuracy_score(y_test, y_pred_dt)
print("Decision Tree Accuracy:", acc_dt)

# Evaluation using confusion matrix
confusion_matrix_dt = confusion_matrix(y_test, y_pred_dt)
print("Decision Tree Confusion Matrix:")
print(confusion_matrix_dt)

# Evaluation using classification report
print("Decision Tree classification report:")
print(classification_report(y_test, y_pred_dt))
```

```
    Decision Tree Accuracy: 0.4617
    Decision Tree Confusion Matrix:
    [[303  22  54  23 132  84  80  80  78 183]
     [ 29 737  87  62  22  67   0  28   1   6]
     [ 58  72 435  35  72 107  16 147   2  34]
     [ 17  58  37 786  10  28   0  86   0   7]
     [128  16  84   9 426  47  29  78  31 136]
     [ 67  64 105  20  40 483  23 142  22  30]
     [ 89   2   4   1  23  28 333  17 441  47]
     [ 74  35 147  67  61 145  20 393  17  24]
     [ 73   1   4   0  31  20 400  12 333 100]
     [180   5  46   6 139  20  62  35 112 388]]
    Decision Tree classification report:
                  precision    recall  f1-score   support

               0       0.30      0.29      0.29      1039
               1       0.73      0.71      0.72      1039
               2       0.43      0.44      0.44       978
               3       0.78      0.76      0.77      1029
               4       0.45      0.43      0.44       984
               5       0.47      0.48      0.48       996
               6       0.35      0.34      0.34       985
               7       0.39      0.40      0.39       983
               8       0.32      0.34      0.33       974
               9       0.41      0.39      0.40       993

        accuracy                           0.46     10000
       macro avg       0.46      0.46      0.46     10000
    weighted avg       0.46      0.46      0.46     10000
```

```
# input_data = np.array([1,69,1,2,2,1,1,2,1,2,2,2,2,2,2]).reshape(1,-1)
# prediction = dt.predict(input_data)
```

```
# prediction
```

```
!pip install lime
import lime
import lime.lime_tabular
from __future__ import print_function
np.random.seed(1)
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: lime in /usr/local/lib/python3.8/dist-packages (0.2.0.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from lime) (1.21.6)
Requirement already satisfied: scikit-image>=0.12 in /usr/local/lib/python3.8/dist-packages (from lime) (0.18.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.8/dist-packages (from lime) (3.2.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.8/dist-packages (from lime) (4.64.1)
Requirement already satisfied: scipy in /usr/local/lib/python3.8/dist-packages (from lime) (1.7.3)
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.8/dist-packages (from lime) (1.0.2)
Requirement already satisfied: networkx>=2.0 in /usr/local/lib/python3.8/dist-packages (from scikit-image>=0.12->lime) (3.0)
Requirement already satisfied: imageio>=2.3.0 in /usr/local/lib/python3.8/dist-packages (from scikit-image>=0.12->lime) (2.9.0)
Requirement already satisfied: pillow!=7.1.0,!=7.1.1,>=4.3.0 in /usr/local/lib/python3.8/dist-packages (from scikit-image>=0.12->lime) (
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.8/dist-packages (from scikit-image>=0.12->lime) (2022.10.1
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.8/dist-packages (from scikit-image>=0.12->lime) (1.4.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib->lime) (1.4.4)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib->lime
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-packages (from matplotlib->lime) (0.11.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib->lime) (2.8.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from scikit-learn>=0.18->lime) (3.1.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.8/dist-packages (from scikit-learn>=0.18->lime) (1.2.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from python-dateutil>=2.1->matplotlib->lime) (1.15.0)
```

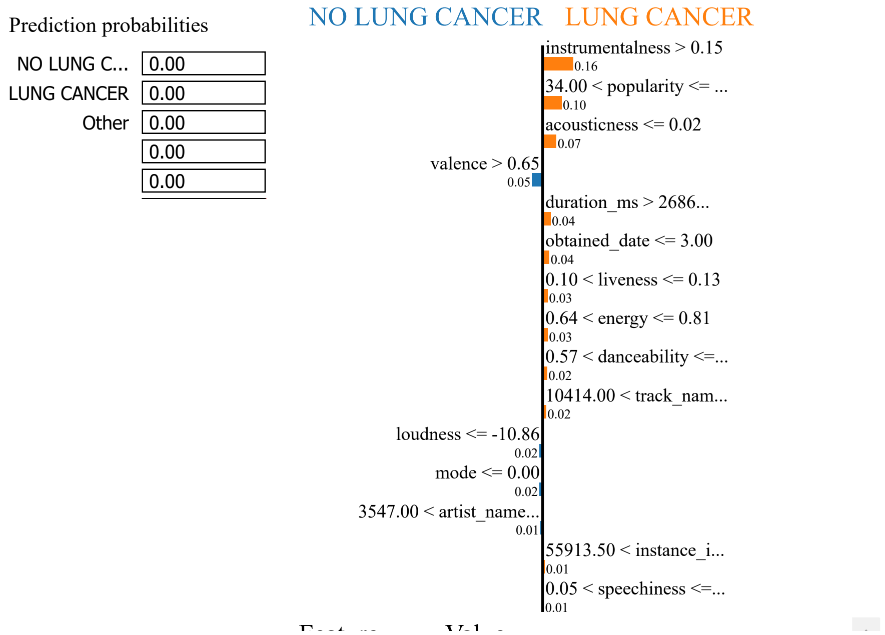
```
df.columns
X=df[['instance_id', 'artist_name', 'track_name', 'popularity',
       'acousticness', 'danceability', 'duration_ms', 'energy',
       'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
       'speechiness', 'tempo', 'obtained_date', 'valence']]
```

```
explainer = lime.lime_tabular.LimeTabularExplainer(X.values, feature_names=df.columns.values.tolist(), class_names=['NO LUNG CANCER','LUNG CA
```

```
j = 22
exp = explainer.explain_instance(X.values[j], dt.predict_proba, num_features=15, top_labels=1)
exp
```

```
Intercept 0.014863179653321151
Prediction_local [0.42821651]
Right: 1.0
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but DecisionTreeClassifier
  warnings.warn(
<lime.explanation.Explanation at 0x7f68b3eff190>
```

```
exp.show_in_notebook(show_table=True)
```



Hybrid Model

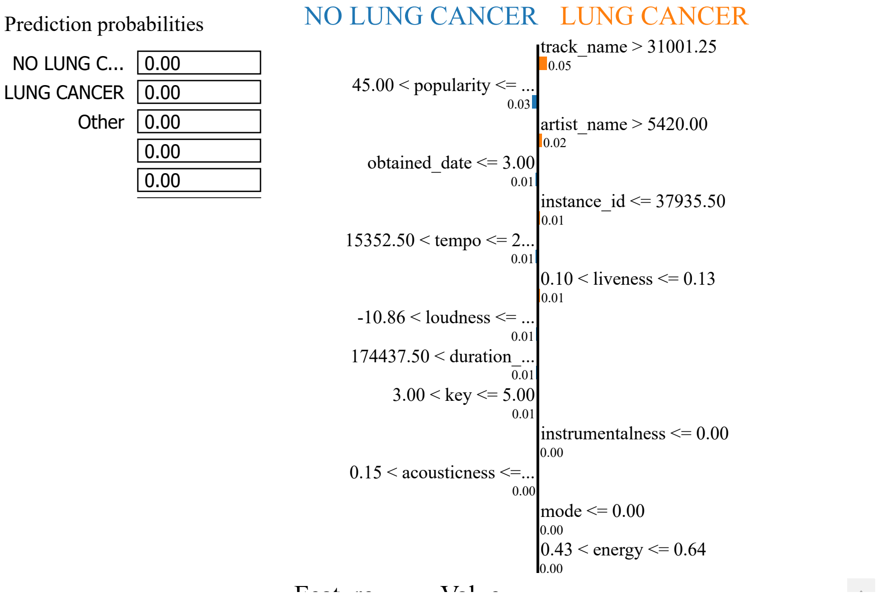
```
# Use the decision tree to select the most important features from the training data
important_features = dt.feature_importances_
X_train_important = X_train.iloc[:, important_features > 0]
X_test_important = X_test.iloc[:, important_features > 0]

# Train a neural network on the selected features
nn = MLPClassifier()
nn.fit(X_train_important, y_train)

# Make predictions on the test set using the neural network
y_pred_nn = nn.predict(X_test_important)

# Calculate the accuracy of the hybrid model
accuracy = accuracy_score(y_test, y_pred_nn)
print("Accuracy:", accuracy)

# Evaluation using accuracy score
acc_nn = accuracy_score(y_test, y_pred_nn)
print("Hybrid Model Accuracy:", acc_nn)

# Evaluation using confusion matrix
confusion_matrix_nn = confusion_matrix(y_test, y_pred_nn)
print("Hybrid Model Confusion Matrix:")
print(confusion_matrix_nn)

# Evaluation using classification report
print("Hybrid Model report:")
print(classification_report(y_test, y_pred_nn))
```

```
Accuracy: 0.2148
Hybrid Model Accuracy: 0.2148
Hybrid Model Confusion Matrix:
[[  6   4   0   2   6   0 389 568   0  64]
 [  6 161   4  79   7   3  39 735   0   5]
 [  2  15   0  10   5   0  59 868   0  19]
 [  3  26   1 429   5   0  11 552   0   2]
 [  4  12   0   1  12   0 261 624   0  70]
 [  9  13   0   3   6   0 140 807   1  17]
 [  8   0   2   0   3   0 582 269   0 121]
 [  5  20   1  16  10   1  66 851   1  12]
 [ 11   1   0   1   3   0 611 229   2 116]
 [ 12   1   0   0   5   0 506 364   0 105]]
Hybrid Model report:
              precision    recall  f1-score   support

           0       0.09      0.01      0.01      1039
           1       0.64      0.15      0.25      1039
           2       0.00      0.00      0.00       978
           3       0.79      0.42      0.55      1029
           4       0.19      0.01      0.02       984
           5       0.00      0.00      0.00       996
           6       0.22      0.59      0.32       985
           7       0.15      0.87      0.25       983
           8       0.50      0.00      0.00       974
           9       0.20      0.11      0.14       993

    accuracy                           0.21     10000
   macro avg       0.28      0.22      0.15     10000
weighted avg       0.28      0.21      0.16     10000
```

```
explainer = lime.lime_tabular.LimeTabularExplainer(X_train_important.values, feature_names=df.columns.values.tolist(), class_names=['NO LUNG
# Select a random sample from the test set to explain
i = 21
exp = explainer.explain_instance(X_train_important.iloc[i, :], nn.predict_proba, num_features=14)
```

```
Intercept 0.03591413916701211
Prediction_local [0.05568912]
Right: 2.9774491742121863e-83
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but MLPClassifier was fitt
  warnings.warn(
```

```
# Show the explanation
exp.show_in_notebook(show_table=True, show_all=False)
```



```
import xgboost as xgb
from sklearn.metrics import accuracy_score

# Split data into training and test sets
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create XGBoost data matrices
dtrain = xgb.DMatrix(data=X_train, label=y_train)
dtest = xgb.DMatrix(data=X_test, label=y_test)

# Set XGBoost parameters
params = {'objective': 'binary:logistic', 'eval_metric': 'error'}

# Train the model
model = xgb.train(params=params, dtrain=dtrain, num_boost_round=100)

# Predict on the test set
y_pred = model.predict(dtest)

# Convert predictions to binary labels
y_pred = [round(pred) for pred in y_pred]

# Calculate accuracy
acc = accuracy_score(y_test, y_pred)

# Print accuracy
print("Accuracy: ", acc)
```

```
---------------------------------------------------------------------------
XGBoostError                              Traceback (most recent call last)
<ipython-input-118-3cd21813e6b7> in <module>
     13 
     14 # Train the model
---> 15 model = xgb.train(params=params, dtrain=dtrain, num_boost_round=100)
     16 
     17 # Predict on the test set

                              3 frames
/usr/local/lib/python3.8/dist-packages/xgboost/core.py in _check_call(ret)
    174     """
    175     if ret != 0:
--> 176         raise XGBoostError(py_str(_LIB.XGBGetLastError()))
    177 
    178 

XGBoostError: [13:37:22] /workspace/src/objective/regression_obj.cu:101: label must be
in [0,1] for logistic regression
Stack trace:
  [bt] (0) /usr/local/lib/python3.8/dist-
packages/xgboost/./lib/libxgboost.so(dmlc::LogMessageFatal::~LogMessageFatal()+0x24)
[0x7f68aab35cb4]
  [bt] (1) /usr/local/lib/python3.8/dist-
packages/xgboost/./lib/libxgboost.so(xgboost::obj::RegLossObj<xgboost::obj::LogisticClas
const&, xgboost::MetaInfo const&, int,
xgboost::HostDeviceVector<xgboost::detail::GradientPairInternal<float> >*)+0x805)
[0x7f68aad3f9d5]
  [bt] (2) /usr/local/lib/python3.8/dist-
packages/xgboost/./lib/libxgboost.so(xgboost::LearnerImpl::UpdateOneIter(int,
xgboost::DMatrix*)+0x345) [0x7f68aabcf505]
  [bt] (3) /usr/local/lib/python3.8/dist-
packages/xgboost/./lib/libxgboost.so(XGBoosterUpdateOneIter+0x35) [0x7f68aab32aa5]
```