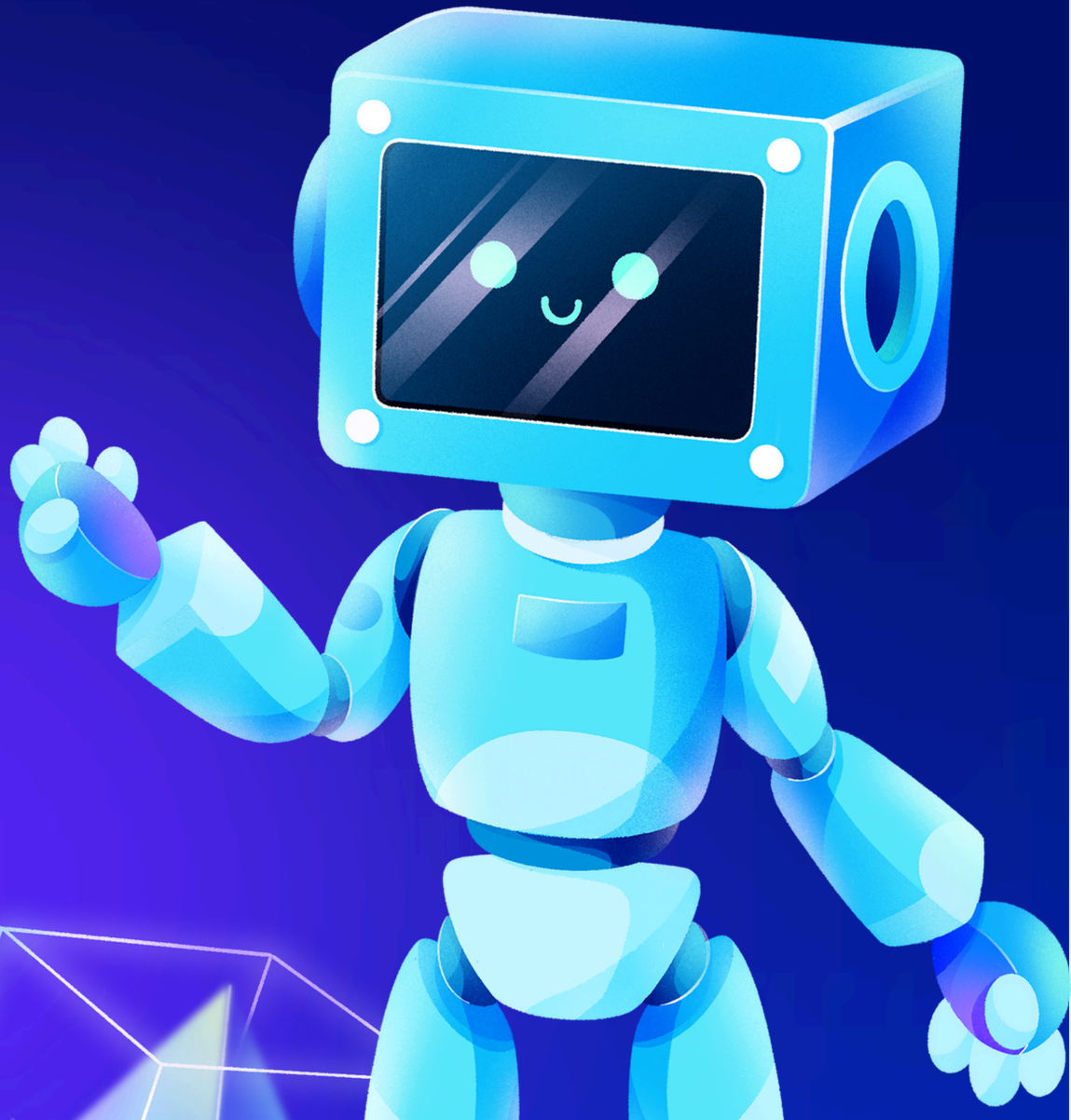
DATA-SCIENCE-TOOLS PROJECT

By

Ali Mohamed Sayed	20221449583
Ziad Ashraf Ibrahem	20221369225
Safy Fathy Abdelsadek	20221450558
Mohamed Ahmed Hamdy	20221444348
Yassin Ahmed Saad-Allah	20221310260
Mohamed Hassan Gaber	20221444401
Abdelwahab Mohamed Abdelaal	20221369240

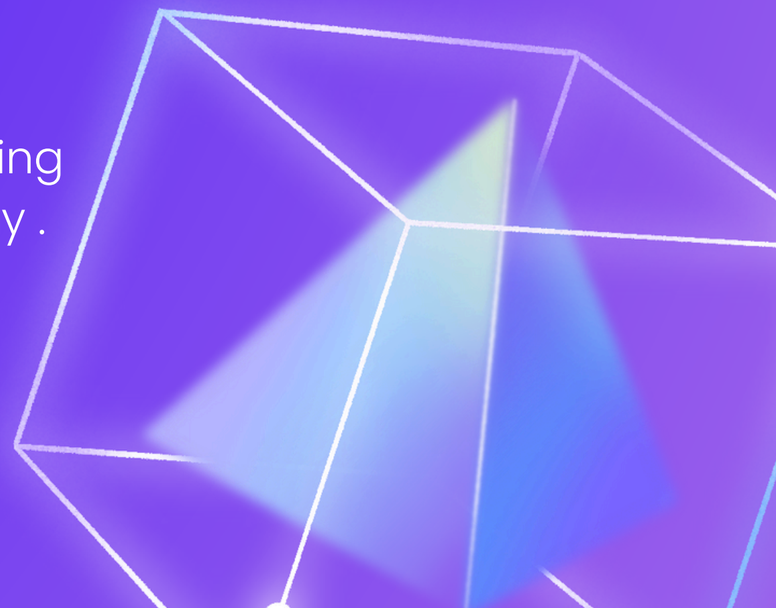


INTRODUCTION

In this report we'll be focusing on multiple datasets. we will use them to demonstrate our projects mainly our models. , these datasets might differ in their usage.

The first project: This project focuses on predicting bankruptcy in companies using financial data from the Emerging Markets Information Service(Poland's Economy .

The second project focuses on classifying Arabic character images using Convolutional Neural Networks (CNN) and Support Vector Machines (SVM).



First Data

In this project, we'll work with financial data from the Emerging Markets Information Service. We'll look at financial indicators from Poland and Taiwan, and build a model that predicts bankruptcy.

The following figures Indicates what does each attribute refer to.

X1	net profit / total assets
X2	total liabilities / total assets
X3	working capital / total assets
X4	current assets / short-term liabilities
X5	[(cash + short-term securities + receivables - short-term liabilities) / (operating expenses - depreciation)] * 365
X6	retained earnings / total assets
X7	EBIT / total assets
X8	book value of equity / total liabilities
X9	sales / total assets
X10	equity / total assets
X11	(gross profit + extraordinary items + financial expenses) / total assets
X12	gross profit / short-term liabilities
X13	(gross profit + depreciation) / sales
X14	(gross profit + interest) / total assets
X15	(total liabilities * 365) / (gross profit + depreciation)
X16	(gross profit + depreciation) / total liabilities
X17	total assets / total liabilities
X18	gross profit / total assets
X19	gross profit / sales
X20	(inventory * 365) / sales
X21	sales (n) / sales (n-1)
X22	profit on operating activities / total assets
X23	net profit / sales
X24	gross profit (in 3 years) / total assets
X25	(equity - share capital) / total assets
X26	(net profit + depreciation) / total liabilities
X27	profit on operating activities / financial expenses
X28	working capital / fixed assets
X29	logarithm of total assets
X30	(total liabilities - cash) / sales
X31	(gross profit + interest) / sales
X32	(current liabilities * 365) / cost of products sold
X33	operating expenses / short-term liabilities
X34	operating expenses / total liabilities
X35	profit on sales / total assets
X36	total sales / total assets
X37	(current assets - inventories) / long-term liabilities
X38	constant capital / total assets
X39	profit on sales / sales
X40	(current assets - inventory - receivables) / short-term liabilities
X41	total liabilities / ((profit on operating activities + depreciation) * (12/365))
X42	profit on operating activities / sales
X43	rotation receivables + inventory turnover in days
X44	(receivables * 365) / sales
X45	net profit / inventory
X46	(current assets - inventory) / short-term liabilities
X47	(inventory * 365) / cost of products sold
X48	EBITDA (profit on operating activities - depreciation) / total assets
X49	EBITDA (profit on operating activities - depreciation) / sales
X50	current assets / total liabilities
X51	short-term liabilities / total assets
X52	(short-term liabilities * 365) / cost of products sold)
X53	equity / fixed assets
X54	constant capital / fixed assets
X55	working capital
X56	(sales - cost of products sold) / sales
X57	(current assets - inventory - short-term liabilities) / (sales - gross profit - depreciation)
X58	total costs /total sales
X59	long-term liabilities / equity
X60	sales / inventory
X61	sales / receivables
X62	(short-term liabilities *365) / sales
X63	sales / short-term liabilities
X64	sales / fixed assets

Our first step is to Import the necessary libraries for data manipulation, machine learning, imputation, visualization, and evaluation.

```python
import json
import sklearn
import numpy
import pandas as pd
from sklearn.impute import SimpleImputer ,KNNImputer
from sklearn.model_selection import train_test_split,cross_val_score,GridSearchCV
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from sklearn.pipeline import make_pipeline
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from scipy.io import arff
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
from sklearn.metrics import classification_report
```

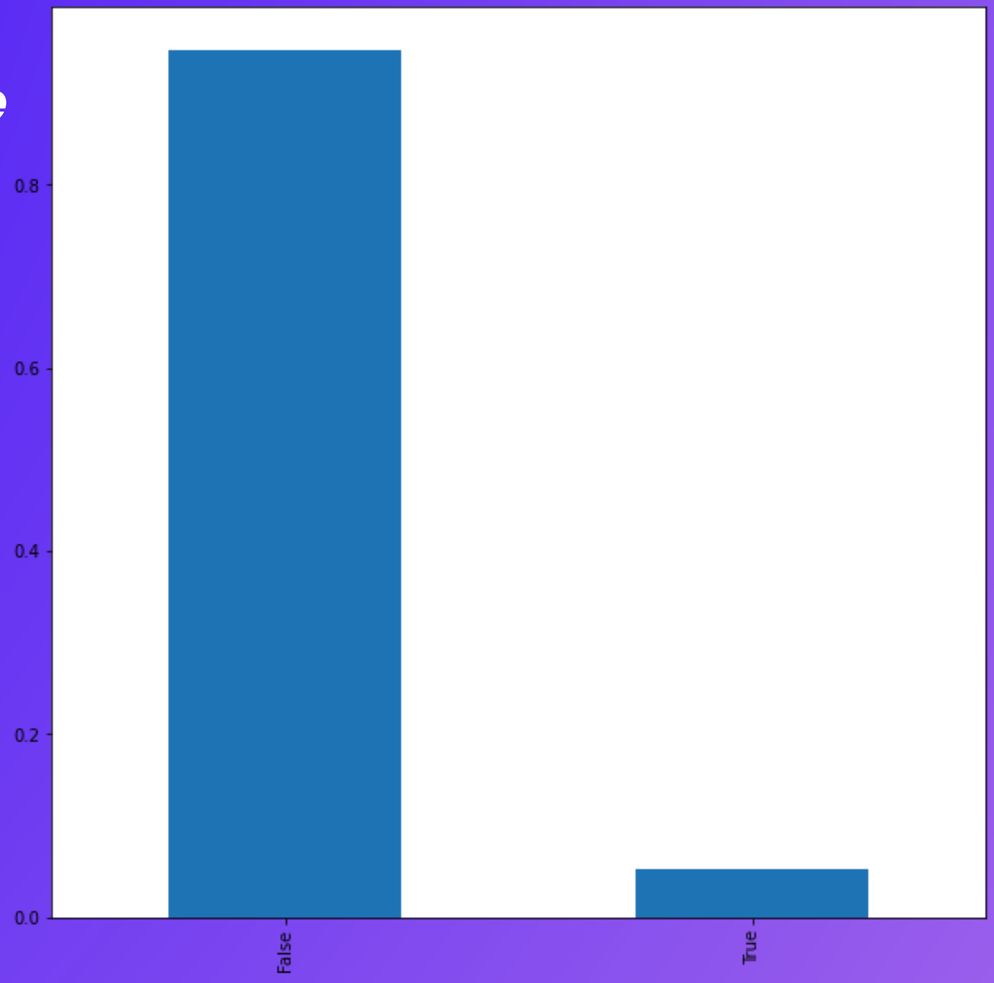
```python
wrangle(path):
data, meta = arff.loadarff(path)
df = pd.DataFrame(data)
bool_mapping = {df['class'].unique()[0]: False, df['class'].unique()[1]: True}
df.replace(bool_mapping, inplace=True)
return df
```

```python
wrangle(r"D:\4year.arff")
```

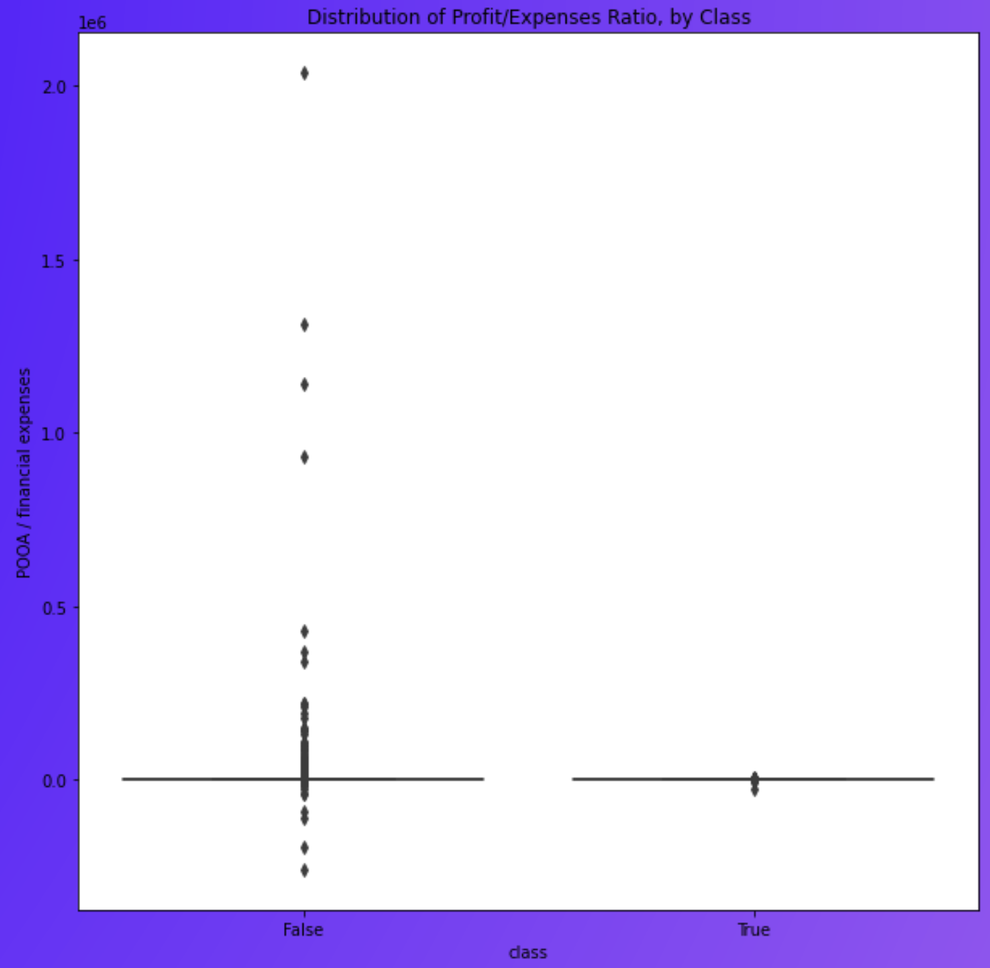
Then we load our Dataset using a wrangle function and whilst we are at it we decided to manipulate the “class” column so it’s values either be False or True.

It looks like most of the companies in our dataset are doing all right for themselves, let's drill down a little further. However, it also shows us that we have an imbalanced dataset, where our majority class is far bigger than our minority class.

```
plt.figure(figsize=(10,10))
df['class'].value_counts(normalize=True).plot(kind='bar')
```



Right here we are testing a random attribute to see whether the dataset is skewed or not whilst also checking for outliers and anomalies.



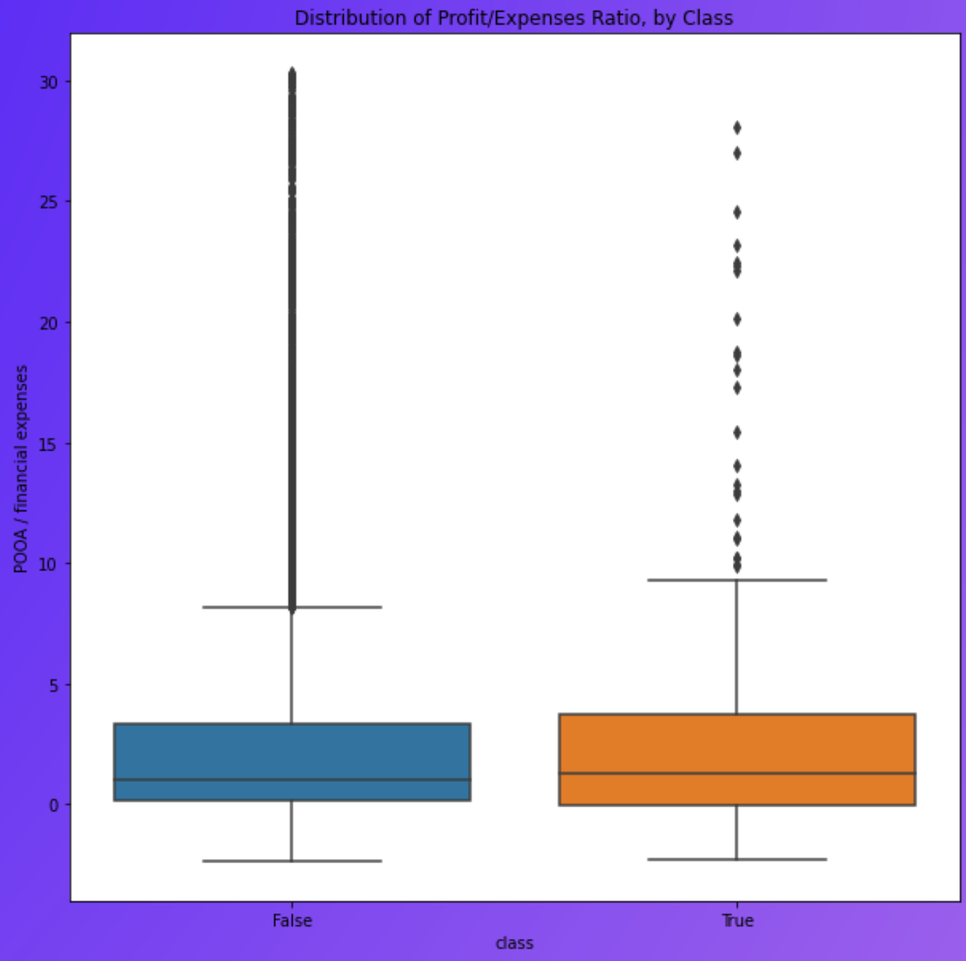

```
df['Attr27'].describe().apply("{0:,.0f}".format)
```

```
count          9,075
mean           1,123
std           31,570
min         -259,010
25%                0
50%                1
75%                5
max        2,037,300
Name: Attr27, dtype: object
```

We discovered that the median is around 1, but the mean is over 1000. That suggests that this feature is skewed to the right.

More context on "feat_27": Profit on operating activities is profit that a company makes through its "normal" operations. For instance, a car company profits from the sale of its cars. However, a company may have other forms of profit, such as financial investments. So a company's total profit may be positive even when its profit on operating activities is negative.

We decided to deal with the outliers by restricting the quantile range that we're working with, so we add a quantile between (0.1 and 0.9).



After detecting and dealing with the outliers in attribute 27, we deduced that the other attributes might contain outliers too. So we created a function to delete the outliers from all data

```
# Function to detect outliers
def outlier_detecting(dataframe, variable):
    quartile1 = dataframe[variable].quantile(0.25)
    quartile3 = dataframe[variable].quantile(0.75)
    interquantile_range = quartile3 - quartile1
    up_limit = quartile3 + 1.5 * interquantile_range
    low_limit = quartile1 - 1.5 * interquantile_range
    return low_limit, up_limit


# function to remove outliers
def outlier_removing(dataframe, numeric_columns):
    for variable in numeric_columns:
        low_limit, up_limit = outlier_detecting(dataframe, variable)
        dataframe.loc[(dataframe[variable] < low_limit), variable] = dataframe[variable].mean()
        dataframe.loc[(dataframe[variable] > up_limit), variable] = dataframe[variable].mean()
```

Spliting the data

in this case, we Divide data (X and y) into training and test sets using a randomized train-test split. and validation set should be 40% of your total data. And make a random_state for reproducibility.

```
target = "class"
X = df.drop(target,axis=1)
y = df[target]

print("X shape:", X.shape)
print("y shape:", y.shape)
```

```
X shape: (9710, 64)
y shape: (9710,)
```

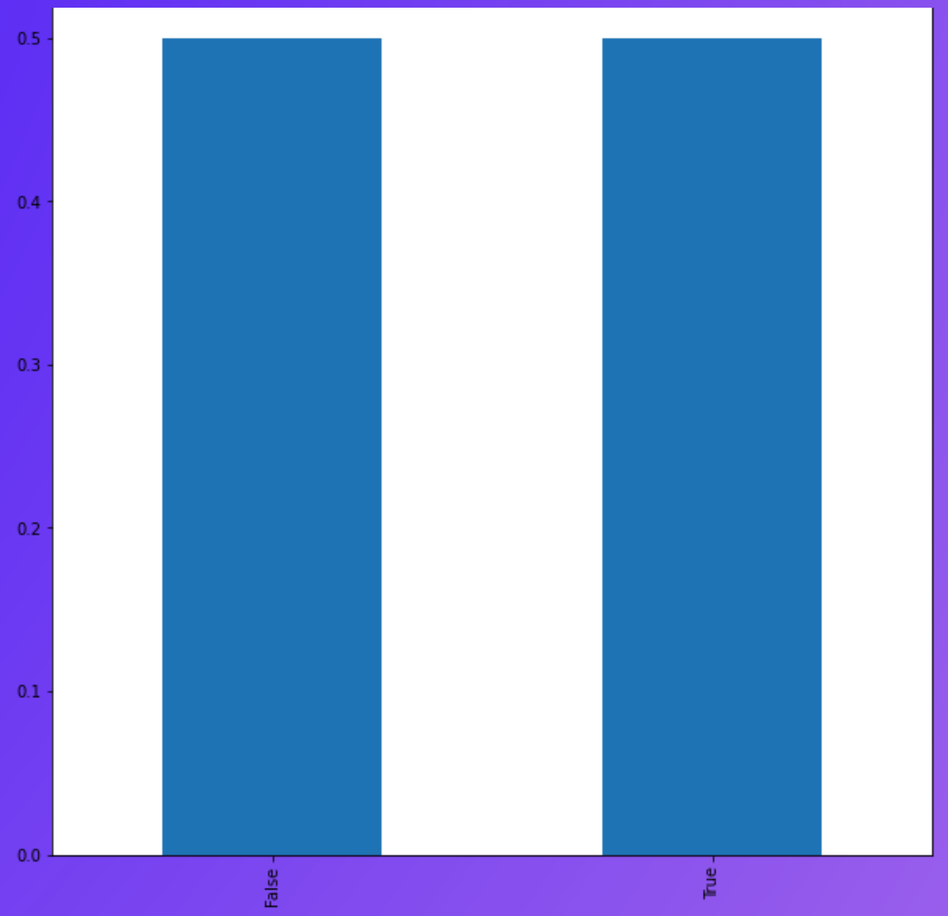
```
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=42,test_size=0.4)

print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (7282, 64)
y_train shape: (7282,)
X_test shape: (2428, 64)
y_test shape: (2428,)
```

in Random over sampling, we increase the number of observations in the minority class by randomly making copies of the existing observations in order to maintain the balance between them so the data will be consistent.

```
over_sampler = RandomOverSampler(random_state=42)
X_train_over, y_train_over = over_sampler.fit_resample(X_train,y_train)
print(X_train_over.shape)
X_train_over.head()
```

```
(13806, 64)
```



The baseline accuracy is very high because our classes are imbalanced(catering to the false class), We should keep this in mind because, even if our trained model gets a high validation accuracy score, that doesn't mean it's good.

```
acc_baseline = y_train.value_counts(normalize=True).max()
print("Baseline Accuracy:", round(acc_baseline, 4))
```

```
Baseline Accuracy: 0.948
```

We will be building our first model using an algorithm called RandomForest, but first we have to deal with the missing data by using a simple imputer to fill in the missing data with ("mean") and select the best feature with ("kbest")

```
clf1 = make_pipeline(
        SimpleImputer(strategy='mean'),
        SelectKBest(score_func=f_classif, k=60),
        RandomForestClassifier(random_state=42))
print(clf1)
```

```
Pipeline(steps=[('simpleimputer', SimpleImputer()),
                ('selectkbest', SelectKBest(k=60)),
                ('randomforestclassifier',
                 RandomForestClassifier(random_state=42))])
```

```
clf1.fit(X_train_over,y_train_over)
print("the score of training over data : ",clf1.score(X_train,y_train))
print("the score of testing data : ",clf1.score(X_test,y_test))
```

```
the score of training over data :  1.0
the score of testing data :  0.9456342668863262
```

Create a dictionary with the range of hyperparameters that we want to evaluate for our classifier.

To be able to get the best performance from our model, we need to tune its hyperparameter so we would get the best set of paramters to fit the model perfectly. But we dont have a validation set, So we'll be using the cross-validation technique.

in this case, we will cross-validate the model accuracy five times in order to improve the accuracy.

```
params = {
    "simpleimputer__strategy":["mean","median"]
    ,'selectkbest__k':range(45,65,5)
    ,"randomforestclassifier__n_estimators":range(25,100,25)
    ,"randomforestclassifier__max_depth":range(50,100,10)
}
model = GridSearchCV(
    clf1,
    param_grid=params,cv=5,n_jobs=-1,verbose=1
)
model.fit(X_train_over,y_train_over)
```

```
Fitting 5 folds for each of 120 candidates, totalling 600 fits

GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('simpleimputer', SimpleImputer()),
                                       ('selectkbest', SelectKBest(k=60)),
                                       ('randomforestclassifier',
                                        RandomForestClassifier(random_state=42))]),
             n_jobs=-1,
             param_grid={'randomforestclassifier__max_depth': range(50, 100, 10),
                         'randomforestclassifier__n_estimators': range(25, 100, 25),
                         'selectkbest__k': range(45, 65, 5),
                         'simpleimputer__strategy': ['mean', 'median']},
             verbose=1)
```

```
cv_acc_scores = cross_val_score(clf1,X_test,y_test,cv=5,n_jobs=-1)
cv_acc_scores
```

```
array([0.94444444, 0.9382716 , 0.94032922, 0.94845361, 0.94020619])
```

We discovered that the accuracy of our model is in the acceptable range of the baseline accuracy so it’s a success

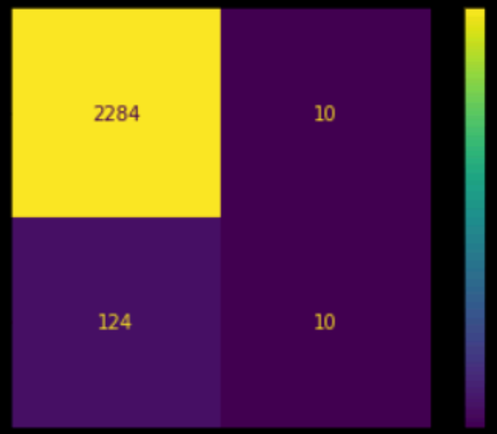
We're going to use a confusion matrix to see how our model performs and how accurate its predictions.

```
print("score of training data : ",model.score(X_train,y_train))
print("score of testing data : ",model.score(X_test,y_test))
```

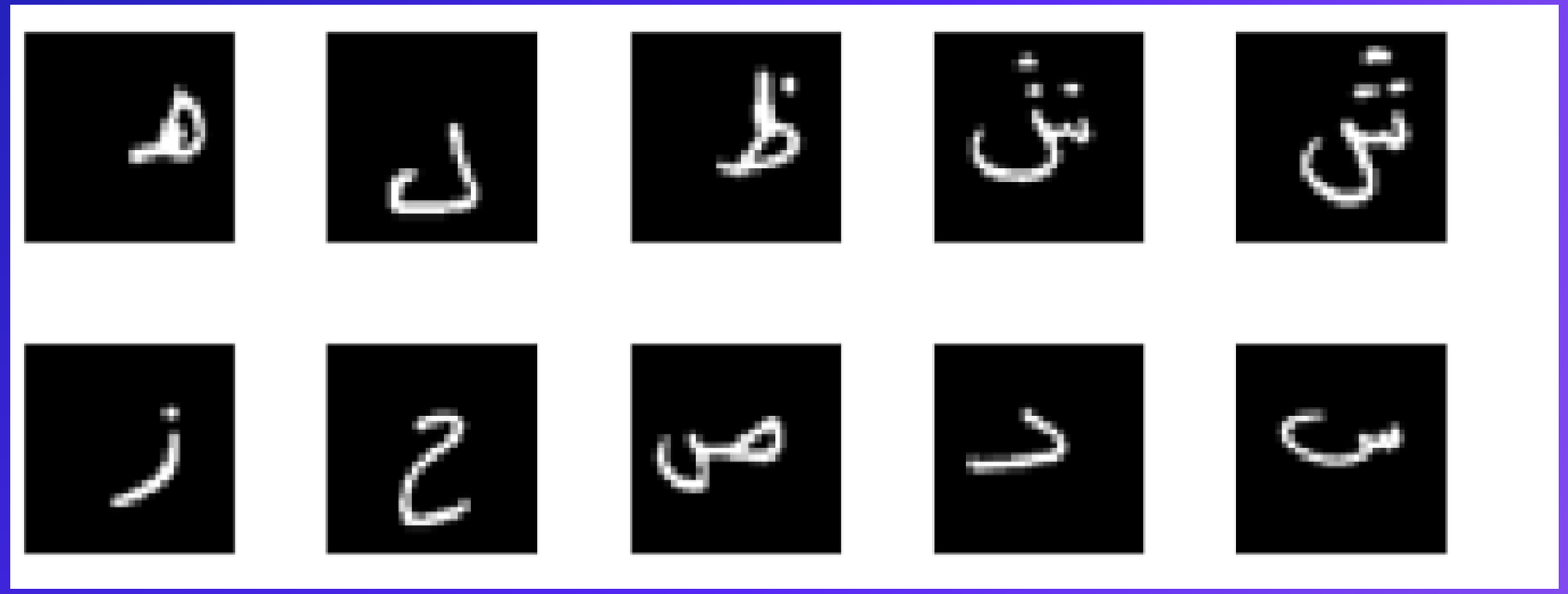
```
score of training data :  1.0
score of testing data :  0.9448105436573312
```

```
plt.figure(figsize=(50,50))
ConfusionMatrixDisplay.from_estimator(model,X_test,y_test);
```

```
<Figure size 3600x3600 with 0 Axes>
```



Second Data



Sample of the Arabic characters to be recognized by computer

Using the wrangle function, we start by loading the dataset then we decided to reshape it to a 4 dimensional arrays with the intent to make it easier for the model to deal with.

```
# make a wrangle fun to set data
def wrangle(filepath1,filepath2,filepath3,filepath4):

    #define my data set
    X_train=pd.read_csv(filepath1).to_numpy() # for image train

    y_train=pd.read_csv(filepath2).to_numpy()-1 # for label train

    x_test=pd.read_csv(filepath3).to_numpy()  # for image test

    y_test=pd.read_csv(filepath4).to_numpy()-1 # for label test

    # Reshape flattened image data (1D) to 4D format (32x32 grayscale) for convolutional neural networks
    X_train = X_train.reshape(-1,32,32,1)
    x_test = x_test.reshape(-1,32,32,1)

    # the target variable (y train and test) often represents different classes we must convert to categr
    y_train = to_categorical(y_train)
    y_test = to_categorical(y_test)

    return X_train,y_train,x_test,y_test
```

we do not need to split that data , it is already split into train and test

```
split this data to train and test
test,y_test=wrangle("csvTrainImages 13440x1024.csv","csvTrainLabel 13440x1.csv","csvTestImages 3360x1024.csv","csvTes
```

The shapes are crucial for understanding the structure of the data so we use method shape for that

```
# shape of X train and test afetr reshaping
print('X_train shape is ',X_train.shape)
print('X_test shape is ',x_test.shape)
```

```
X_train shape is  (13439, 32, 32, 1)
X_test shape is  (3359, 32, 32, 1)
```

```
# shape of y train and test afetr reshaping
print('y_train shape is ',y_train.shape)
print('y_test shape is ',y_test.shape)
```

```
y_train shape is  (13439, 28)
y_test shape is  (3359, 28)
```

Data augmentation is a technique to increase the diversity of the training dataset, reducing overfitting and helping the model generalize better to unseen data we used various different techniques such as; Zooming, rescaling, and flipping are common augmentation techniques.
Using the "ImageDataGenerator " to perform data augmentation on the training data.

```
datagen = ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    rotation_range=10,
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=False,
    vertical_flip=False
)
train_gen = datagen.flow(X_train, y_train, batch_size=64)
test_gen  = datagen.flow(x_test , y_test , batch_size=64)
```

This code trains a neural network model on the provided data using a generator, monitors the validation accuracy, and saves the best model to a file named "best.hdf5.". The training history is stored in the history variable for later analysis and visualization.

```
# Train the model using the data generator
model_checkpoint_callback = ModelCheckpoint(
    filepath="best.hdf5",
    monitor='val_accuracy',
    verbose=1,
    save_best_only=True,
    mode='max'
)

history = model.fit(
    train_gen,
    epochs=30,
    verbose=1,
    steps_per_epoch=X_train.shape[0] // 64,
    validation_data=test_gen,
    validation_steps=x_test.shape[0] // 64,
    callbacks=[model_checkpoint_callback]
)
```

The model is designed for image classification tasks, specifically for recognizing Arabic characters.

- MaxPooling is used for downsampling, and dropout layers provide regularization.
- The softmax activation in the output layer enables multi-class classification.
- padding refers to the additional pixels added around the input image before applying convolutional operations.

```
model = Sequential()

model.add(Conv2D(32, (5, 5), padding="same", activation="relu", input_shape=(32, 32, 1)))
model.add(Conv2D(32, (5, 5), activation="relu"))
model.add(Conv2D(32, (5, 5), activation="relu"))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.1))
model.add(BatchNormalization())

model.add(Conv2D(64, (5, 5), padding="same", activation="relu"))
model.add(Conv2D(64, (5, 5), activation="relu"))
model.add(Conv2D(64, (5, 5), activation="relu"))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.1))
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(128, activation="relu"))
model.add(Dense(128, activation="relu"))
model.add(Dropout(0.4))

model.add(Dense(28, activation="softmax"))
```

The SVM model is now trained on the provided training data. The trained model is stored in the svm_model variable.

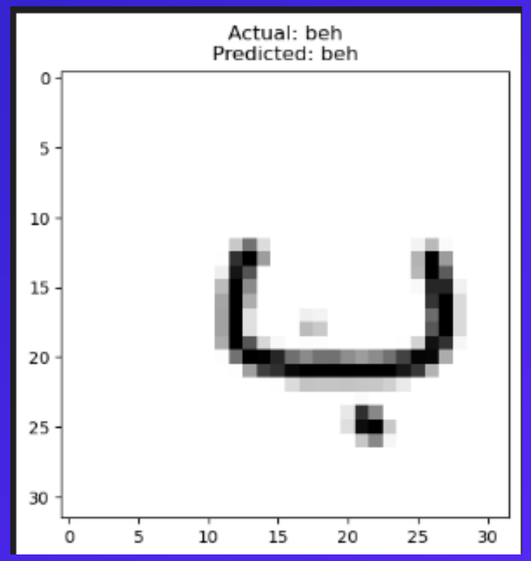
```
# Initialize and train the SVM model
svm_model = SVC()
svm_model.fit(X_traifn_svm, np.argmax(y_train, axis=1))
```

The provided code evaluates the trained SVM model on the test set and reports the accuracy.

```
# Evaluate the SVM model on the test set
svm_predictions = svm_model.predict(x_test_svm)
svm_accuracy = accuracy_score(np.argmax(y_test, axis=1), svm_predictions)
print(f'SVM Accuracy: {svm_accuracy}')
```

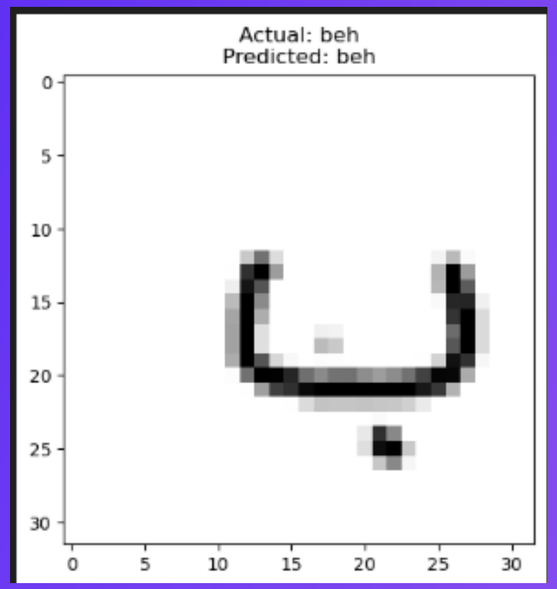
CNN prediction

The accuracy: 0.9699



SVM prediction

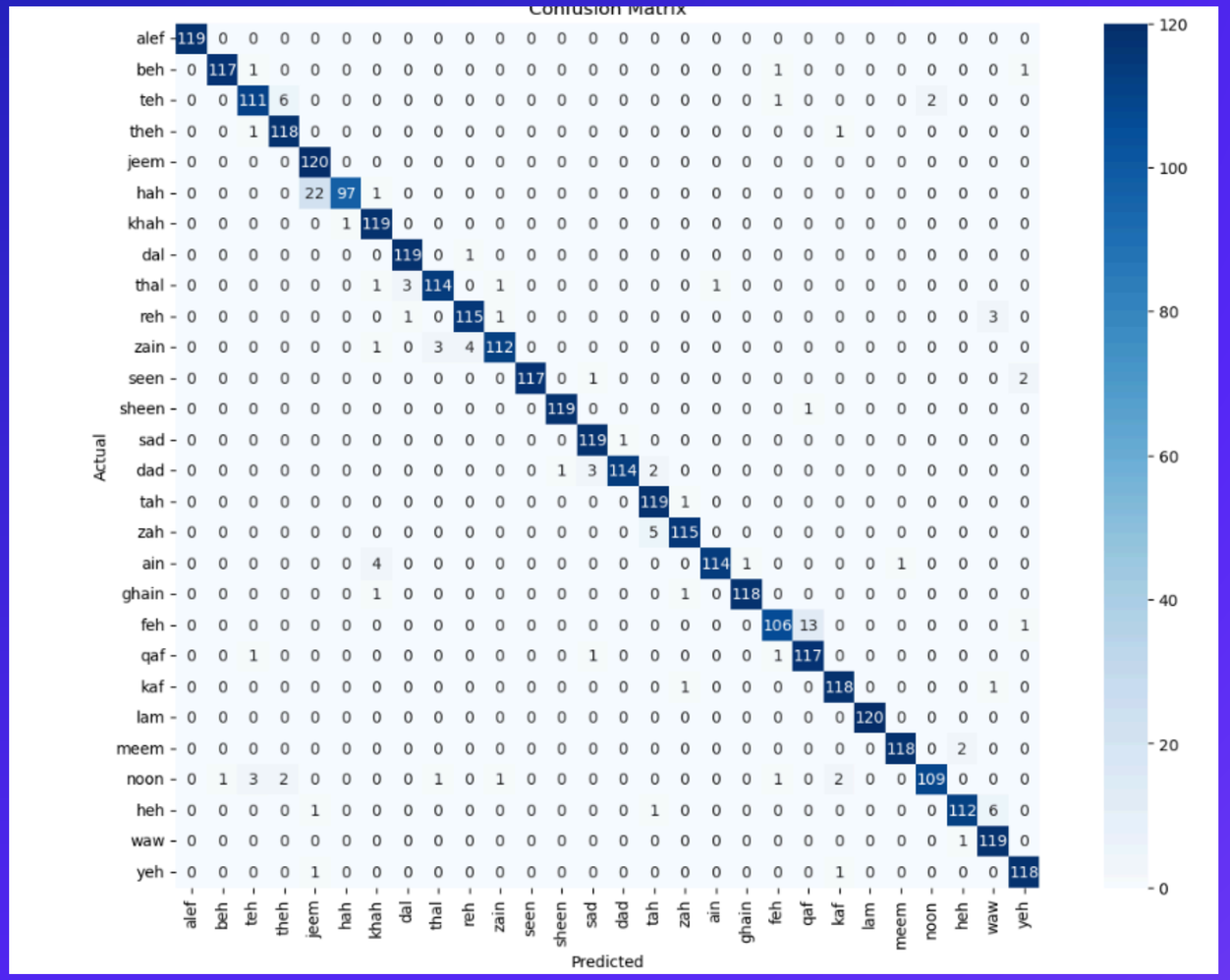
SVM Accuracy: 0.65



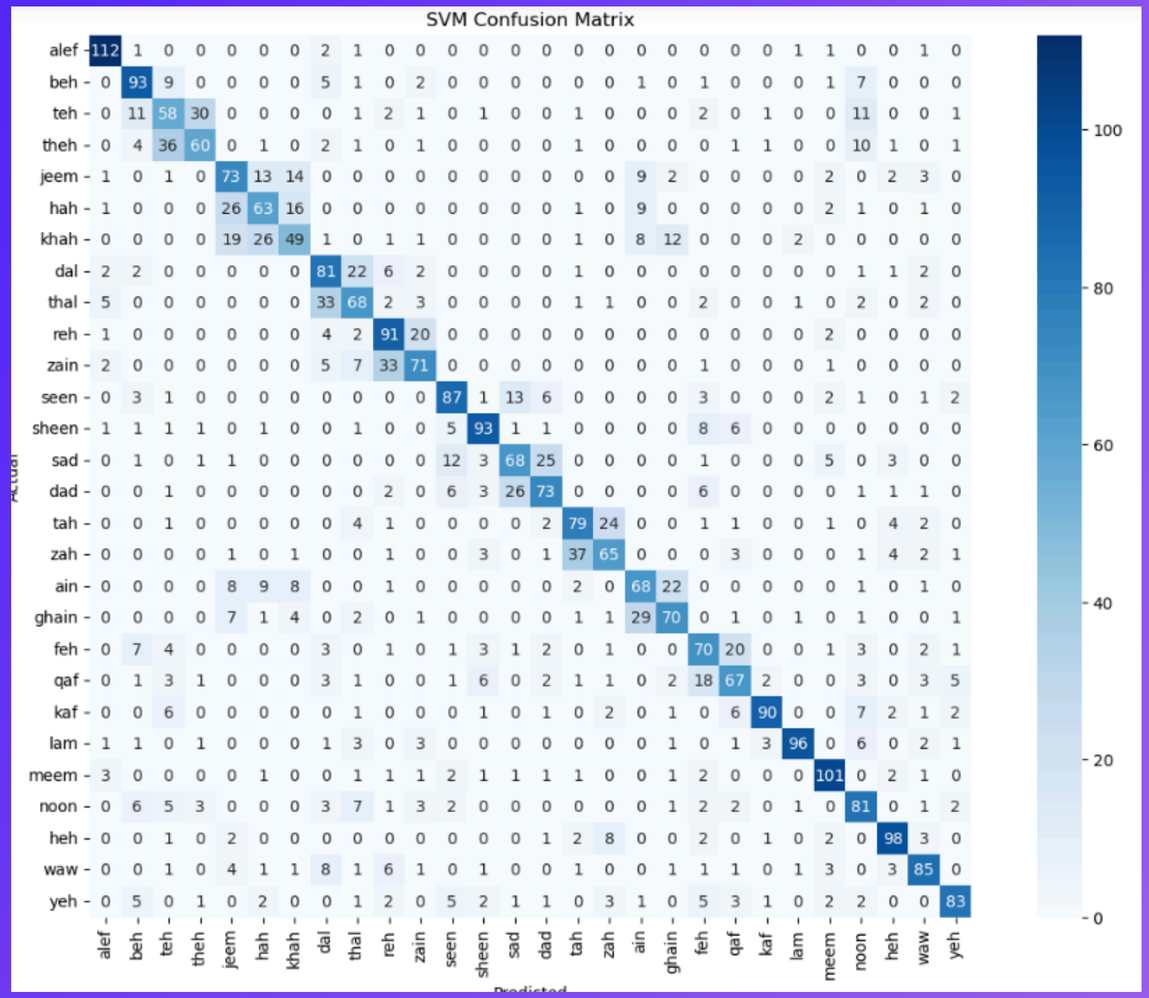
in this case, we found a CNN model is better than SVM

We're going to use a confusion matrix to see how our model performs and how accurate its predictions.(SVM and CNN)

CNN Confusion matrix



SVM confusion matrix



CONCLUSION

We Made 2 different Projects
The first project about Bankrupt
contains 1 model: RandomForest
with accuracy 94%

for Second Project about Handwritten
characters it contains 2 models (CNN,SVM)
CNN accuracy 97%
SVM accuracy 65%