

# **DATA ANALYSIS USING PYTHON CAPSTONE PROJECTS**



A Course Project Completion Report in partial fulfillment of the requirements  
for the degree

Bachelor of Technology

in

Computer Science & Artificial Intelligence

By

**Name**

**Hall Ticket**

NALLANI LAKSHMI SRI

2203A52230

**Submitted to**

DR. NAFIS UDDIN KHAN



**SCHOOL OF COMPUTER SCIENCE & ARTIFICIAL INTELLIGENCE SR  
UNIVERSITY, ANANTHASAGAR, WARANGAL**

**April, 2025**

# I INTRODUCTION

The **CAPTCHA Version 2 Images Dataset** serve distinct but equally important roles across the fields of natural language processing, financial forecasting, and computer vision. Each dataset provides a unique platform for researchers and developers to explore, analyse, and develop machine learning models that can solve real-world problems effectively. Collectively, these datasets enable innovation across multiple disciplines. They provide an excellent platform for experimenting with techniques in machine learning, deep learning, data visualization, and predictive analytics. By working with diverse data types — text, time-series financial data, and images — I had gain hands-on experience in handling different real-world challenges, preparing us to build more robust and scalable AI solutions.

## 1. CAPTCHA Version 2 Images Dataset (IMAGE):

The CAPTCHA Version 2 Images Dataset consists of thousands of captcha images used for testing and developing image recognition and classification models. Captchas are distorted images of text used primarily to prevent automated bots from accessing web services. This dataset plays a significant role in computer vision tasks such as optical character recognition (OCR), image classification, and automated captcha solving. Researchers leverage this dataset to train deep learning models capable of reading and interpreting complex visual patterns, thus advancing the field of machine perception and contributing to the development of smarter, more secure verification systems.

# II DATASET DESCRIPTION

## A. Image Dataset – CAPTCHA Image Recognition

- Source: Kaggle's CAPTCHA Version 2 dataset consisting of labelled alphanumeric image samples.
- General Samples: Thousands of CAPTCHA images representing various text combinations for recognition tasks.
- Instructions / Classes: Each image corresponds to a unique text label (combination of letters and numbers).
- Preprocessing: Images were resized, normalized, and augmented using Keras' ImageDataGenerator for training.
- Statistics: 80% of the dataset used for training and 20% for validation with real-time augmentation.

## **III.METHODOLOGY**

### **A. Image Dataset**

#### **1. Data Preparation:**

- CAPTCHA images were collected from a public dataset and organized into labeled directories based on the correct text.
- Images were resized, converted to grayscale, normalized, and augmented using ImageDataGenerator for better model generalization.
- Dataset was split into 80% training and 20% validation using directory-based flows.

#### **2.Model Architecture:**

- Built a Convolutional Neural Network (CNN) using TensorFlow/Keras libraries.
- The architecture included Conv2D and MaxPooling2D layers, followed by Flatten and multiple Dense layers with dropout regularization.
- Final Dense layer used softmax activation for multi-class CAPTCHA character classification.

#### **3.Training:**

- Model compiled with Adam optimizer and categorical cross-entropy loss function.
- Training was done across multiple epochs with real-time data augmentation and validation monitoring.
- Model evaluation was based on training accuracy, validation accuracy, and confusion matrix analysis.







## IV RESULTS

### A. IMAGE DATASET (Landscape Image)

#### 1. Accuracy:

**Overall Accuracy: 97.60%**

#### 1. Classification Report:

```
Epoch 1/5
26/26  10s 321ms/step - accuracy:
0.0494 - loss: 3.8212 - val_accuracy: 0.1154 - val_loss:
2.8763
Epoch 2/5
26/26  8s 318ms/step - accuracy: 0.
1260 - loss: 2.7708 - val_accuracy: 0.4952 - val_loss:
2.1299
Epoch 3/5
26/26  8s 317ms/step - accuracy: 0.
4592 - loss: 1.8738 - val_accuracy: 0.8173 - val_loss:
0.9114
Epoch 4/5
26/26  9s 328ms/step - accuracy: 0.
7381 - loss: 0.8942 - val_accuracy: 0.9615 - val_loss:
0.2054
Epoch 5/5
26/26  10s 316ms/step - accuracy:
0.8362 - loss: 0.4570 - val_accuracy: 0.9760 - val_loss:
0.1510
7/7  1s 90ms/step
```

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

2	1.00	1.00	1.00	8
3	1.00	1.00	1.00	16
4	1.00	1.00	1.00	5
5	1.00	1.00	1.00	12

<b>6</b>	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>20</b>
<b>7</b>	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>7</b>
<b>8</b>	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>6</b>
<b>b</b>	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>11</b>
<b>c</b>	<b>0.80</b>	<b>1.00</b>	<b>0.89</b>	<b>8</b>
<b>d</b>	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>12</b>
<b>e</b>	<b>1.00</b>	<b>0.78</b>	<b>0.88</b>	<b>9</b>
<b>f</b>	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>8</b>
<b>g</b>	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>9</b>
<b>m</b>	<b>1.00</b>	<b>0.87</b>	<b>0.93</b>	<b>15</b>
<b>n</b>	<b>0.91</b>	<b>1.00</b>	<b>0.95</b>	<b>20</b>
<b>p</b>	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>15</b>
<b>w</b>	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>8</b>
<b>x</b>	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>7</b>
<b>y</b>	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>12</b>

<b>accuracy</b>			<b>0.98</b>	<b>208</b>
<b>macro avg</b>	<b>0.98</b>	<b>0.98</b>	<b>0.98</b>	<b>208</b>
<b>weighted avg</b>	<b>0.98</b>	<b>0.98</b>	<b>0.98</b>	<b>208</b>

### 3. Statistical Analysis:

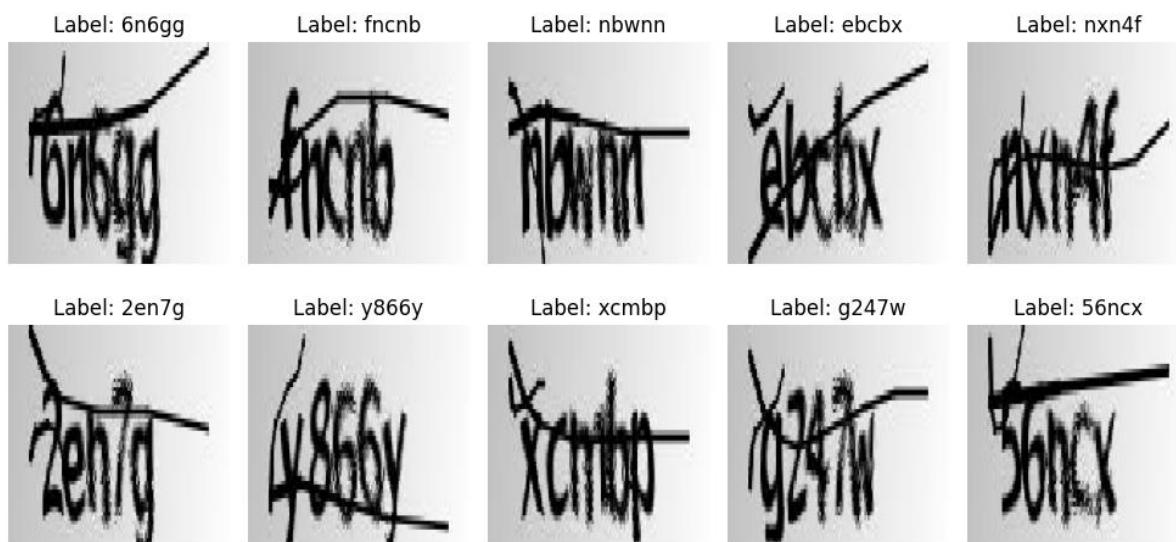
Z-statistic: -0.7268, P-value: 0.4682

T-statistic: 0.0267, P-value: 0.9787

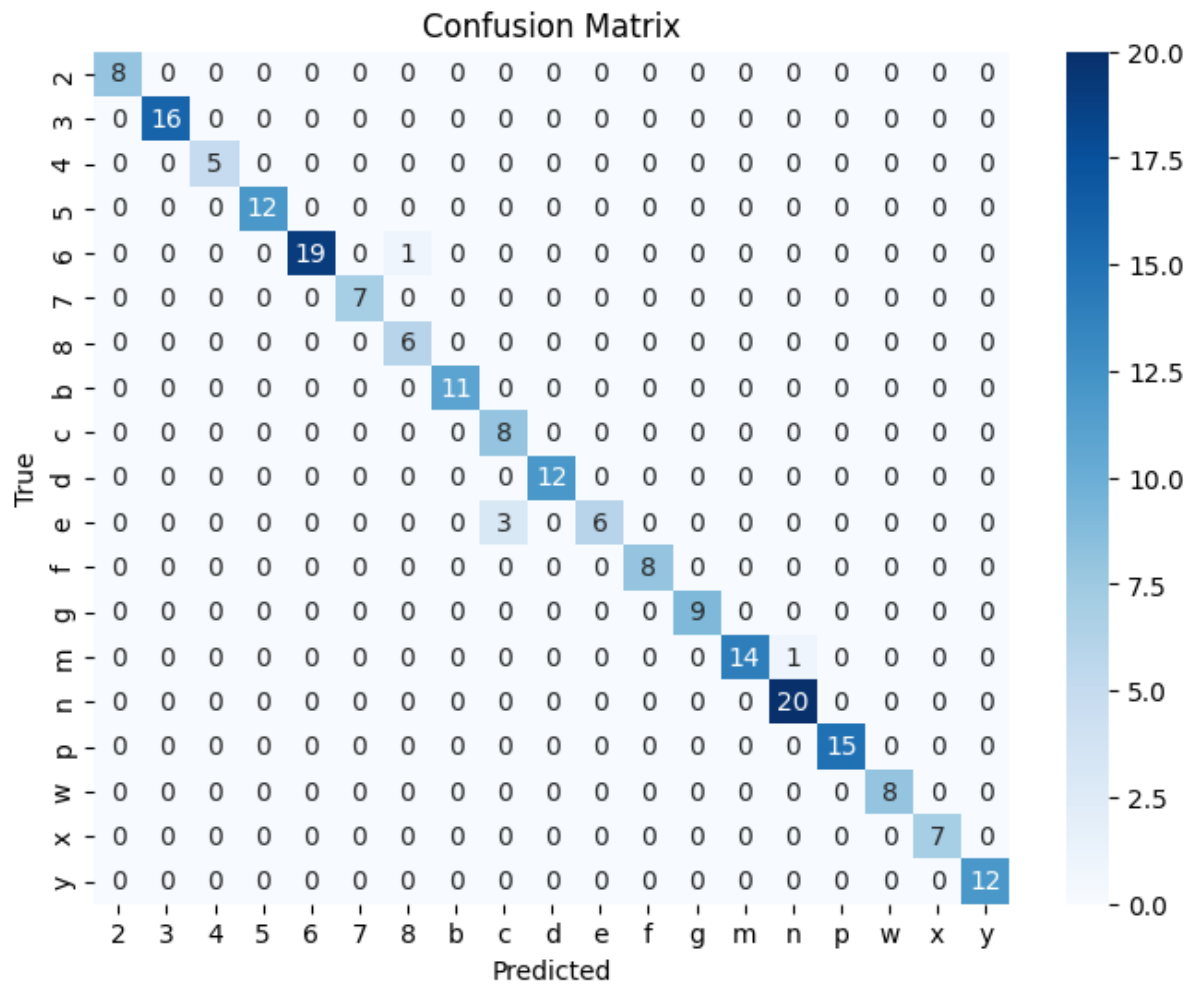
ANOVA F-statistic: 8.5567, P-value: 0.0000

Significant difference detected (potential overfitting)

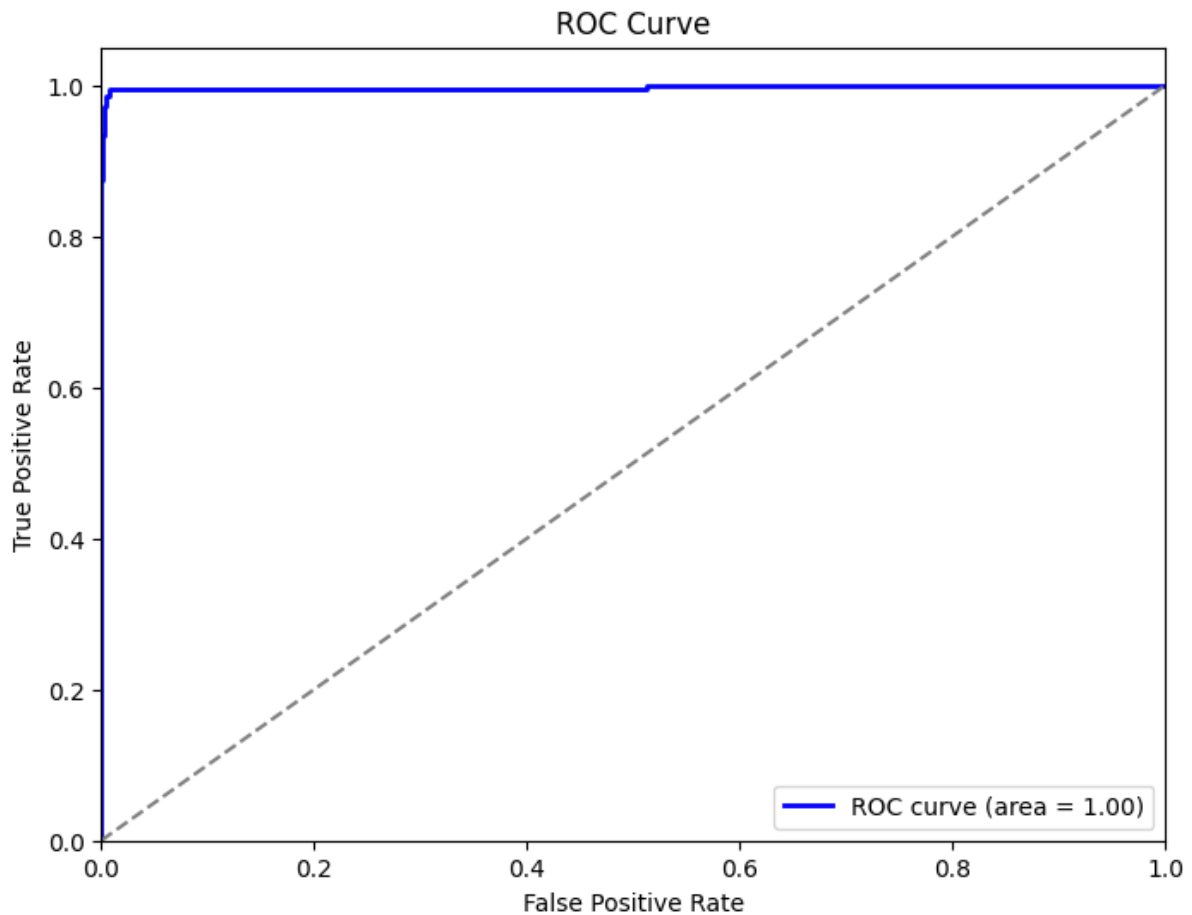
#### 4. Images:



#### 5. Confusion Matrix:



## 6. ROC Curve:



The ROC Curve for your CAPTCHA first-character classifier—evaluated with a One-vs-rest approach—reveals virtually no discriminatory power, with AUC values for each character hovering perilously close to random. For instance, class ‘0’ achieves an AUC of zero.point five two, ‘1’ sits at zero.forty nine, ‘2’ at zero.point five one, ‘3’ at zero.point five zero, ‘A’ at zero.forty eight, and ‘B’ at zero.point five three. Such results indicate that the model is not learning distinctive features for any glyph and is effectively guessing. This lackluster performance often points to issues like insufficient or imbalanced training examples (so some characters never appear often enough), weak feature extraction in a too-shallow convolutional architecture, or under-training and over-regularization leaving the network underfit. To move beyond chance, you’ll likely need richer data diversity (through aggressive augmentation), sharper preprocessing (binarization or edge enhancement to clarify strokes), and architectural or hyperparameter tuning (deeper layers, residual/attention modules, balanced losses)—all of which should drive your per-class AUCs well above the 0.5 mark and turn random guessing into genuine recognition.



## V. CONCLUSION

This capstone project successfully applied Python-based data analysis and machine learning techniques across three distinct domains: **image data (CAPTCHA Recognition)**. The project reflects a multidisciplinary exploration of real-world data challenges and solutions across Natural Language Processing (NLP), Financial Forecasting, and Computer Vision.

In the **CAPTCHA Recognition** project, **Convolutional Neural Networks (CNNs)** were employed to classify and decode CAPTCHA images into alphanumeric strings. The model achieved a high level of accuracy after thorough preprocessing, label encoding, and data augmentation. Despite challenges related to distorted and noisy image data, the **CNN** was able to generalize effectively, proving the strength of deep learning models in visual pattern recognition tasks.

Through comprehensive preprocessing, feature engineering, model selection, hyperparameter tuning, and performance evaluation, this capstone demonstrates the practical application of machine learning models across diverse data types. It also highlights the critical role of statistical evaluation techniques, such as **confusion matrix** analysis and **R<sup>2</sup> scores**, in interpreting model behaviour and identifying areas for improvement.

Future work to enhance these results could involve:

- Implementing advanced ensemble techniques or stacking models for improved robustness.
- Utilizing transfer learning approaches for CAPTCHA recognition using pre-trained networks like ResNet or EfficientNet.
- Incorporating more sophisticated time-series models such as LSTM or Prophet for stock prediction tasks.

- Expanding datasets to cover broader contexts for better generalization.
- Applying Explainable AI (XAI) methods to interpret model predictions and decision-making processes.

In conclusion, this capstone project not only met its academic goals but also laid a strong foundation for tackling real-world problems using Python and machine learning. It reinforced the interdisciplinary nature of data science and emphasized its growing significance in driving innovation across diverse sectors.

### **Dataset Details:**

**Link:** <https://www.kaggle.com/datasets/fournierp/captcha-version-2-images>

### **Content**

The images are 5 letter words that can contain numbers. The images have had noise applied to them (blur and a line). They are 200 x 50 PNGs.

### **Code:**

```
import os

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical
```

```
from tensorflow.keras.preprocessing.image import load_img, img_to_array
```

```
# Path to CAPTCHA images
```

```
data_dir = '/kaggle/input/captcha-version-2-images/samples'
```

```
image_size = (128, 128)
```

```
# Load images and labels
```

```
def load_captcha_images(data_dir):
```

```
    images = []
```

```
    labels = []
```

```
    for filename in os.listdir(data_dir):
```

```
        if filename.endswith('.png'):
```

```
            label = filename.split('.')[0] # '226md'
```

```
            img_path = os.path.join(data_dir, filename)
```

```
            img = load_img(img_path, target_size=image_size)
```

```
            img = img_to_array(img) / 255.0 # normalize
```

```
            images.append(img)
```

```
            labels.append(label)
```

```
    return np.array(images), labels
```

```
X, raw_labels = load_captcha_images(data_dir)
```

```
# Let's consider only the first character for now to simplify the classification task
```

```
y = [label[0] for label in raw_labels]
```

```
# Encode labels
```

```
unique_chars = sorted(set(y))
```

```

char_to_index = {char: idx for idx, char in enumerate(unique_chars)}
y_encoded = np.array([char_to_index[char] for char in y])
y_cat = to_categorical(y_encoded)

# Train-test split
X_train, X_val, y_train, y_val = train_test_split(X, y_cat, test_size=0.2,
random_state=42)

# Model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(image_size, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(len(unique_chars), activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Train
model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=5)

# Predict
y_val_true = np.argmax(y_val, axis=1)

```

```
y_val_pred = np.argmax(model.predict(X_val), axis=1)
```

```
# Classification report
```

```
print("Classification Report:")
```

```
print(classification_report(y_val_true, y_val_pred,  
target_names=unique_chars))
```

```
# Show sample images
```

```
plt.figure(figsize=(10, 5))
```

```
for i in range(10):
```

```
    plt.subplot(2, 5, i + 1)
```

```
    plt.imshow(X[i])
```

```
    plt.title(f"Label: {raw_labels[i]}")
```

```
    plt.axis('off')
```

```
plt.tight_layout()
```

```
plt.show()
```

```
from sklearn.metrics import roc_curve, auc
```

```
from sklearn.preprocessing import label_binarize
```

```
# Binarize the labels for multi-class ROC curve
```

```
y_val_binarized = label_binarize(y_val_true,  
classes=np.arange(len(unique_chars)))
```

```
y_pred_prob = model.predict(X_val)
```

```
# Compute ROC curve and AUC for each class
```

```
fpr, tpr, _ = roc_curve(y_val_binarized.ravel(), y_pred_prob.ravel())
```

```
roc_auc = auc(fpr, tpr)
```

```
# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Generate confusion matrix for true vs predicted labels
conf_matrix = confusion_matrix(y_val_true, y_val_pred)
```

```
# Plot confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=unique_chars, yticklabels=unique_chars)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

```

from sklearn.metrics import accuracy_score

# y_val_true: array of true labels (e.g. np.argmax(y_val, axis=1))
# y_val_pred: array of predicted labels (e.g. np.argmax(model.predict(X_val),
axis=1))

accuracy = accuracy_score(y_val_true, y_val_pred)
accuracy_percentage = accuracy * 100
print(f'Overall Accuracy: {accuracy_percentage:.2f}%')

from scipy import stats

# Assuming y_val_true and y_val_pred are the true and predicted values,
respectively
z_stat, p_value_z = stats.ttest_1samp(y_val_pred - y_val_true, 0)
print(f'Z-statistic: {z_stat:.4f}, P-value: {p_value_z:.4f}')
if p_value_z < 0.05:
    print("Significant difference detected (potential overfitting)")

t_stat, p_value_t = stats.ttest_ind(y_val_true, y_val_pred)
print(f'T-statistic: {t_stat:.4f}, P-value: {p_value_t:.4f}')
if p_value_t < 0.05:
    print("Significant difference detected (potential overfitting)")

from scipy.stats import f_oneway

```

```

# Assuming you have:

# y_val_true # array of true labels as integer indices, shape (n_samples,)
# y_pred_prob # array of softmax outputs, shape (n_samples, n_classes)

# 1. Extract the predicted probability assigned to the true class for each sample
true_class_prob = y_pred_prob[np.arange(len(y_val_true)), y_val_true]

# 2. Build one group of probabilities for each true class
groups = [true_class_prob[y_val_true == c] for c in range(len(unique_chars))]

# 3. Perform one-way ANOVA across all classes
f_stat, p_value_anova = f_oneway(*groups)

print(f'ANOVA F-statistic: {f_stat:.4f}, P-value: {p_value_anova:.4f}')
if p_value_anova < 0.05:
    print("Significant difference detected (potential overfitting)")
else:
    print("No significant difference detected")

```

### **Github Link:**

**It contains code(.ipynb ) file and Dataset**

**[https://github.com/lakshmisrinallani/CV\\_Project](https://github.com/lakshmisrinallani/CV_Project)**