RICE IMAGE CLASSIFICATION

Utilizing CNN & TensorFlow

PRESENTED BY

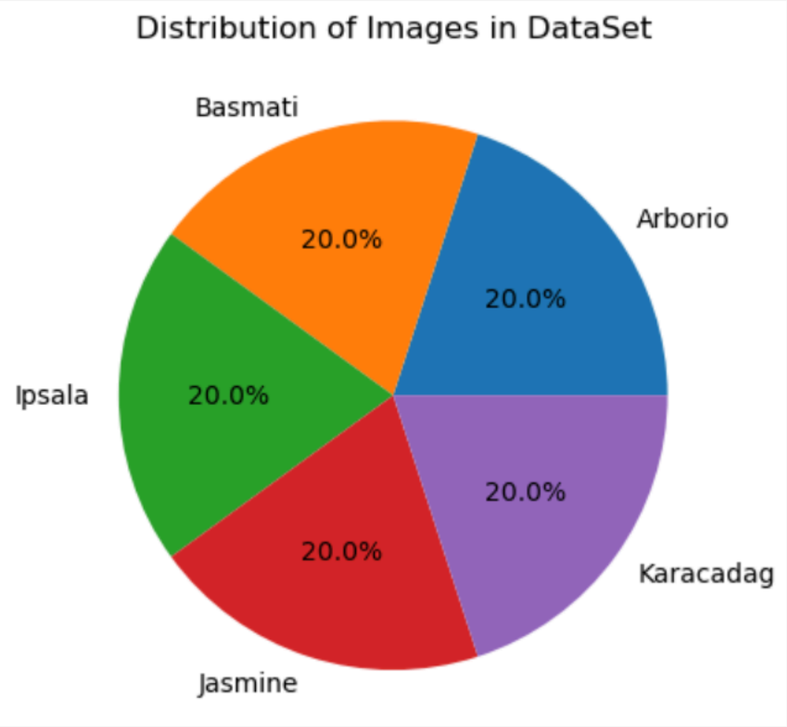
Anwer
Deji
Ameera
Manroop
Sukhwinder
Ellen

RICE IMAGE
CLASSIFICATION

TABLE OF CONTENTS

INTRODUCTION

- Data source: https://www.kaggle.com/code/nmaleknasr/rice-image-classification-cnn-tensorflow/notebook
- What is the goal?
- What is CNN (not the cable news channel)
- Python Libraries
- Data set - 75,000 images




```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import numpy as np
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
import os
import random
from matplotlib.patches import Rectangle
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

CN MODEL EXPLANATION

Step 2: Define the CNN Model

```
# Using the simplest CNN architecture designed for quick testing
test1_cnn = Sequential()

# 1: Convolution Layer
# The first convolutional layer with minimal filters and ReLU activation
test1_cnn.add(Conv2D(filters=8, kernel_size=3, activation='relu', input_shape=(32, 32, 3)))

# 2: Pooling Layer
# Max-pooling reduces the spatial dimensions for computational efficiency
test1_cnn.add(MaxPooling2D(pool_size=2, strides=2))

# 3: Flattening Layer
# Converting into a 1D vector for dense layers
test1_cnn.add(Flatten())

# 4: Full connection is a dense layer with Sigmoid activation for non-linear transformation
test1_cnn.add(Dense(units=8, activation='sigmoid'))

# 5: Output layer is the final dense layer with Sigmoid activation for multi-class output
test1_cnn.add(Dense(units=5, activation='sigmoid'))  # Number of classes

# Display Summary
test1_cnn.summary()
```

CNN models were built with the fundamental layers commonly used in image classification tasks.

- Convolution Layer: Extracts features like edges and textures.
- Pooling Layer: Reduces spatial dimensions for computational efficiency.
- Flattening Layer: Prepares data for dense layers by converting feature maps into a vector.
- Fully Connected Layer: Connects features to output predictions, using Sigmoid activation for non-linear transformation.
- Output Layer: Final dense layer with Sigmoid activation for multi-class output.

RESULTS OF TEST MODELS

```
# Train the model with only 4 epochs for faster testing
test1_fit = test1_cnn.fit(
    train_generator,
    epochs=4,
    validation_data=validation_generator
)
```

```
c:\Users\manro\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121:
  self._warn_if_super_not_called()
Epoch 1/4
1172/1172 ━━━━━━━━━━━━━━━━━━━━ 680s 580ms/step - accuracy: 0.5754 - loss: 1.1849 - val_accuracy: 0.8725 - val_loss: 0.6985
Epoch 2/4
1172/1172 ━━━━━━━━━━━━━━━━━━━━ 33s 28ms/step - accuracy: 0.9484 - loss: 0.4463 - val_accuracy: 0.8791 - val_loss: 0.5617
Epoch 3/4
1172/1172 ━━━━━━━━━━━━━━━━━━━━ 34s 29ms/step - accuracy: 0.9649 - loss: 0.2623 - val_accuracy: 0.8854 - val_loss: 0.5114
Epoch 4/4
1172/1172 ━━━━━━━━━━━━━━━━━━━━ 33s 28ms/step - accuracy: 0.9683 - loss: 0.1737 - val_accuracy: 0.8922 - val_loss: 0.5148
```

TEST MODEL 1

1-Data Split: A 50-50% split between training and validation was chosen to evenly test the model's performance.
2-Activation Functions: ReLU was used in the convolutional layers for its efficiency in training, while Sigmoid was used in the dense layers for simplicity.
3-Epochs- Limited to 4 for quick testing without overfitting.
4-Input Size- Reduced to (32, 32) for faster computation during training.

RESULTS OF TEST MODELS

```
# Trainning the CNN Model for 6 epochs
test2_fit = test2_cnn.fit(
    train_generator,
    epochs=6,
    validation_data=validation_generator
)
```

```
Epoch 1/6
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 36s 25ms/step - accuracy: 0.6477 - loss: 1.0583 - val_accuracy: 0.8406 - val_loss: 0.6989
Epoch 2/6
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 40s 28ms/step - accuracy: 0.9407 - loss: 0.2256 - val_accuracy: 0.8563 - val_loss: 0.6955
Epoch 3/6
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 39s 28ms/step - accuracy: 0.9569 - loss: 0.1442 - val_accuracy: 0.8644 - val_loss: 0.7589
Epoch 4/6
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 40s 28ms/step - accuracy: 0.9652 - loss: 0.1121 - val_accuracy: 0.8679 - val_loss: 0.9115
Epoch 5/6
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 40s 29ms/step - accuracy: 0.9700 - loss: 0.0959 - val_accuracy: 0.8674 - val_loss: 0.7344
Epoch 6/6
1407/1407 ━━━━━━━━━━━━━━━━━━━━ 53s 38ms/step - accuracy: 0.9694 - loss: 0.0943 - val_accuracy: 0.8672 - val_loss: 0.7762
```

TEST MODEL 2

1-Data Split- A larger portion (60%) of the dataset is used for training and 40% validation

2-Activation Functions: Sigmoid in the convolutional layer to explores an alternative to the commonly used ReLU for feature extraction.

3-Epochs- Training for 6 epochs ensures the model has sufficient iterations to learn patterns in the data.

4-Input Size- This remains same as previous model.

THE OPTIMIZED CNN MODEL

Creating the Convolutional Neural Network (CNN)

```
# Define the model
cnn = tf.keras.models.Sequential()

# 1: Convolution
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu', input_shape=(250, 250, 3)))

# 2: Pooling
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

# 3: Adding a second convolutional layer
cnn.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

# 4: Flattening
cnn.add(tf.keras.layers.Flatten())

# 5: Full connection
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))

# 6: Output layer
cnn.add(tf.keras.layers.Dense(units=5, activation='softmax'))  # # Changing the number of outputs as the number of classes

#  Check the structure of the Sequential model
cnn.summary()
```

The model is created using TensorFlow's Sequential class, which allows layers to be added in a linear stack.

For this model the data was split into 80% for the data training and 20% for data testing.

- A Convolutional Layer with 32 filters and a 3x3 kernel.
- A Pooling Layer (MaxPooling) after the first convolutional layer.
- A Second Convolutional Layer with 64 filters and a 3x3 kernel.
- A Flattening Layer to prepare data for the fully connected layers.
- A Dense Layer with 128 neurons.
- An Output Layer with 5 neurons (for the number of classes).

OPTIMIZED CNN MODEL

```
fit_model = cnn.fit(
    train_generator,
    epochs=10,
    validation_data=test_generator,
    callbacks=[early_stopping, checkpoint]
)
```

```
:\Users\Anwer.AAMD\anaconda3\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning: Your `PyDatas
 self._warn_if_super_not_called()
poch 1/10
875/1875 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - accuracy: 0.7235 - loss: 0.6946
poch 1: val_loss improved from inf to 0.22877, saving model to best_model.keras
875/1875 ━━━━━━━━━━━━━━━━━━━━ 7417s 4s/step - accuracy: 0.7236 - loss: 0.6945 - val_accuracy: 0.9130 - val_loss: 0.2288
poch 2/10
875/1875 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.9202 - loss: 0.2132
poch 2: val_loss improved from 0.22877 to 0.13206, saving model to best_model.keras
875/1875 ━━━━━━━━━━━━━━━━━━━━ 3712s 2s/step - accuracy: 0.9202 - loss: 0.2132 - val_accuracy: 0.9533 - val_loss: 0.1321
poch 3/10
875/1875 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9496 - loss: 0.1396
poch 3: val_loss did not improve from 0.13206
875/1875 ━━━━━━━━━━━━━━━━━━━━ 2358s 1s/step - accuracy: 0.9496 - loss: 0.1396 - val_accuracy: 0.9246 - val_loss: 0.1897
poch 4/10
875/1875 ━━━━━━━━━━━━━━━━━━━━ 0s 16s/step - accuracy: 0.9600 - loss: 0.1122
poch 4: val_loss improved from 0.13206 to 0.06809, saving model to best_model.keras
875/1875 ━━━━━━━━━━━━━━━━━━━━ 29527s 16s/step - accuracy: 0.9600 - loss: 0.1122 - val_accuracy: 0.9772 - val_loss: 0.0681
poch 5/10
875/1875 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9683 - loss: 0.0925
poch 5: val_loss did not improve from 0.06809
875/1875 ━━━━━━━━━━━━━━━━━━━━ 2223s 1s/step - accuracy: 0.9683 - loss: 0.0925 - val_accuracy: 0.9724 - val_loss: 0.0808
poch 6/10
875/1875 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9717 - loss: 0.0824
poch 6: val_loss improved from 0.06809 to 0.06768, saving model to best_model.keras
875/1875 ━━━━━━━━━━━━━━━━━━━━ 2343s 1s/step - accuracy: 0.9717 - loss: 0.0824 - val_accuracy: 0.9763 - val_loss: 0.0677
poch 7/10
..
 Epoch 10/10
 1875/1875 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9795 - loss: 0.0615
 Epoch 10: val_loss did not improve from 0.04720
 1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2291s 1s/step - accuracy: 0.9795 - loss: 0.0615 - val_accuracy: 0.9807 - val_loss: 0.0576
```

1-Data Augmentation- Introduced augmentation techniques: rotation, shifting, shearing, zooming, and horizontal flipping for better generalization. The data split was changed to 80-20% for this model.

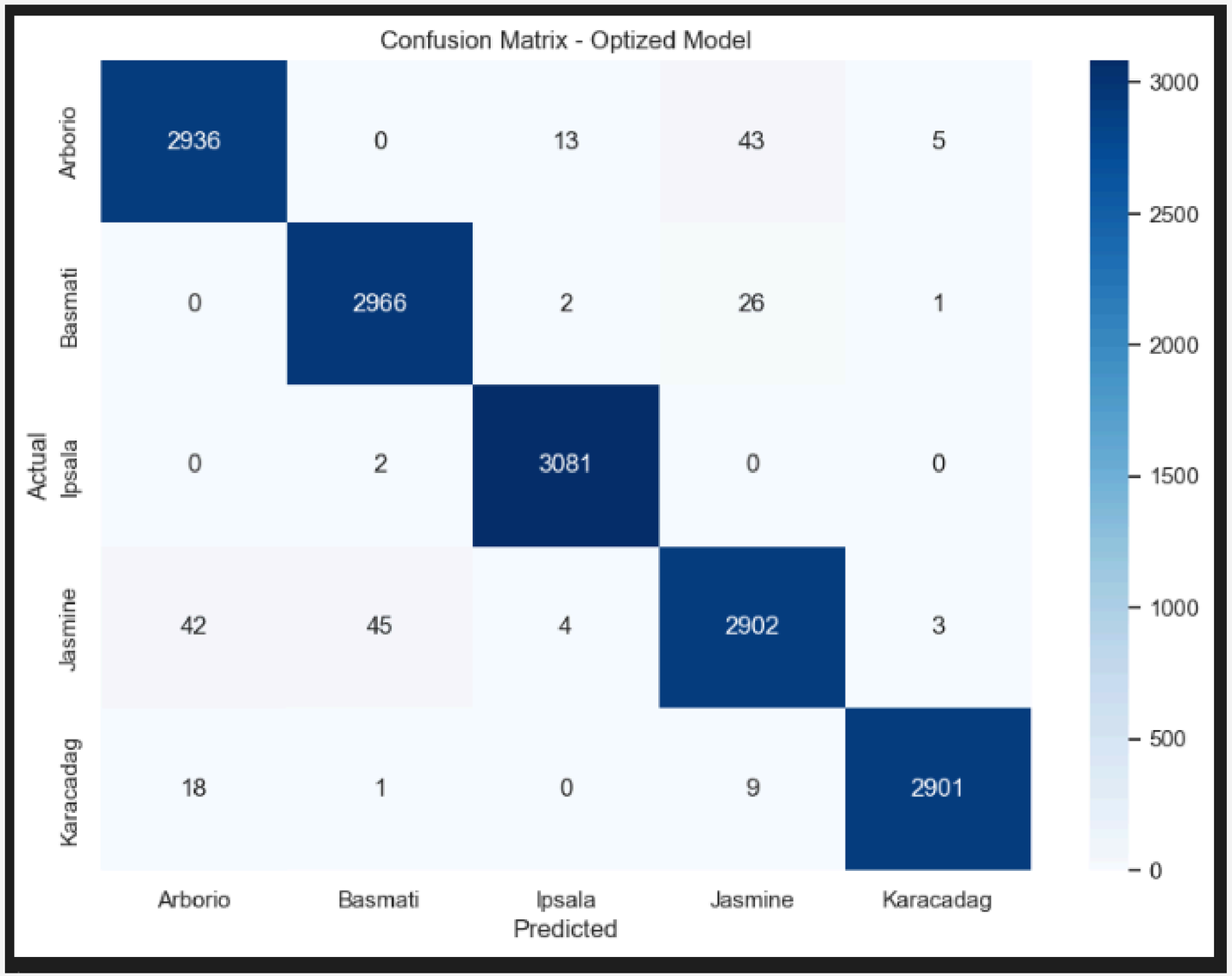
2-Input Image Size:Increased from 32 × 32 to 250 × 250 for detailed feature extraction.

3-Model Architecture- Added a second convolutional layer with 64 filters.
Increased the dense layer units from 32 to 128.
Activation Functions: Used ReLU in all layers except the output layer for better gradient flow and training efficiency. For the output layer softmax activation was used.

4-Training Parameters: Increased epochs from 6 to 10 to achieve more accuracy.

Performance Improvement:Accuracy improved from 97.01% to 98.52%.
Validation loss reduced from 0.0943 to 0.0499.

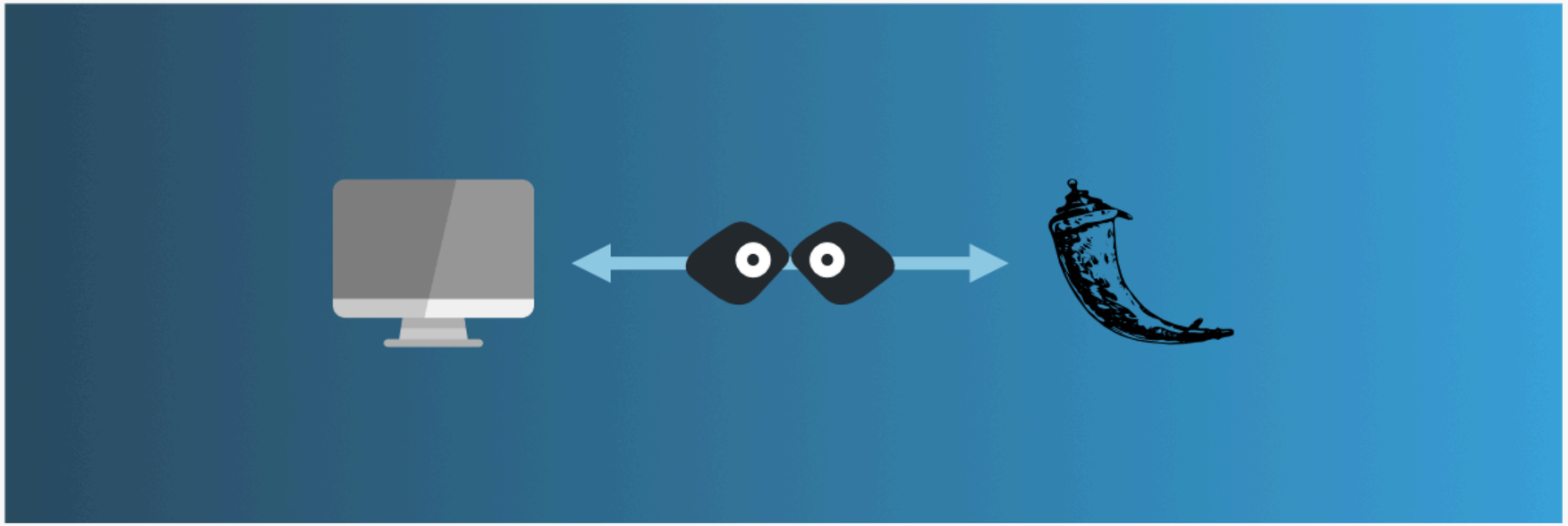
ANALYSIS OF CNN MODEL



This is a confusion matrix for the optimized CNN model, showing the performance of the classification across five types of rice: Arborio, Basmati, Ipsala, Jasmine, and Karacadag.
Key details:

- Diagonal values represent correct predictions (e.g., 2936 Arborio samples were correctly classified as Arborio).
- Off-diagonal values represent misclassifications (e.g., 45 Jasmine samples were misclassified as Arborio).
- The color gradient visually indicates the frequency, with darker shades representing higher values.
- The model performs well overall, as most predictions are along the diagonal, showing high accuracy in classification.

RICE IMAGE CLASSIFICATION API

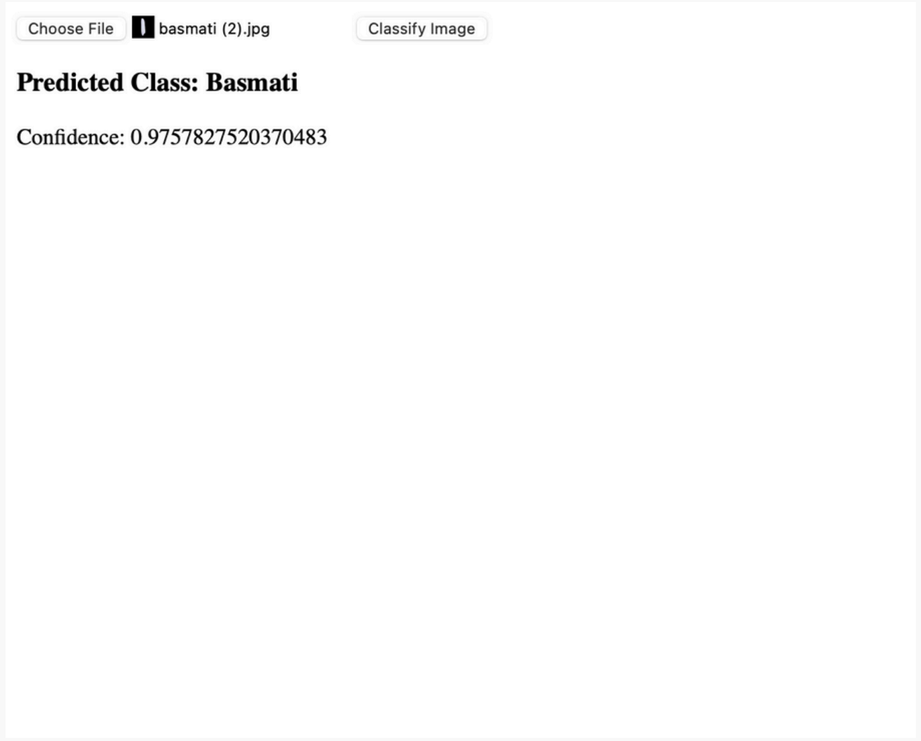


Powered by Flask and TensorFlow

OVERVIEW OF THE TASK

OBJECTIVE:

- Build a web application for classifying different types of rice using a trained Convolutional Neural Network (CNN).
- Users upload rice images, and the app returns the predicted class with confidence.



TECHNOLOGIES USED:

- Flask
- Tensor Flow/Keras
- Pillow (PIL)
- HTML

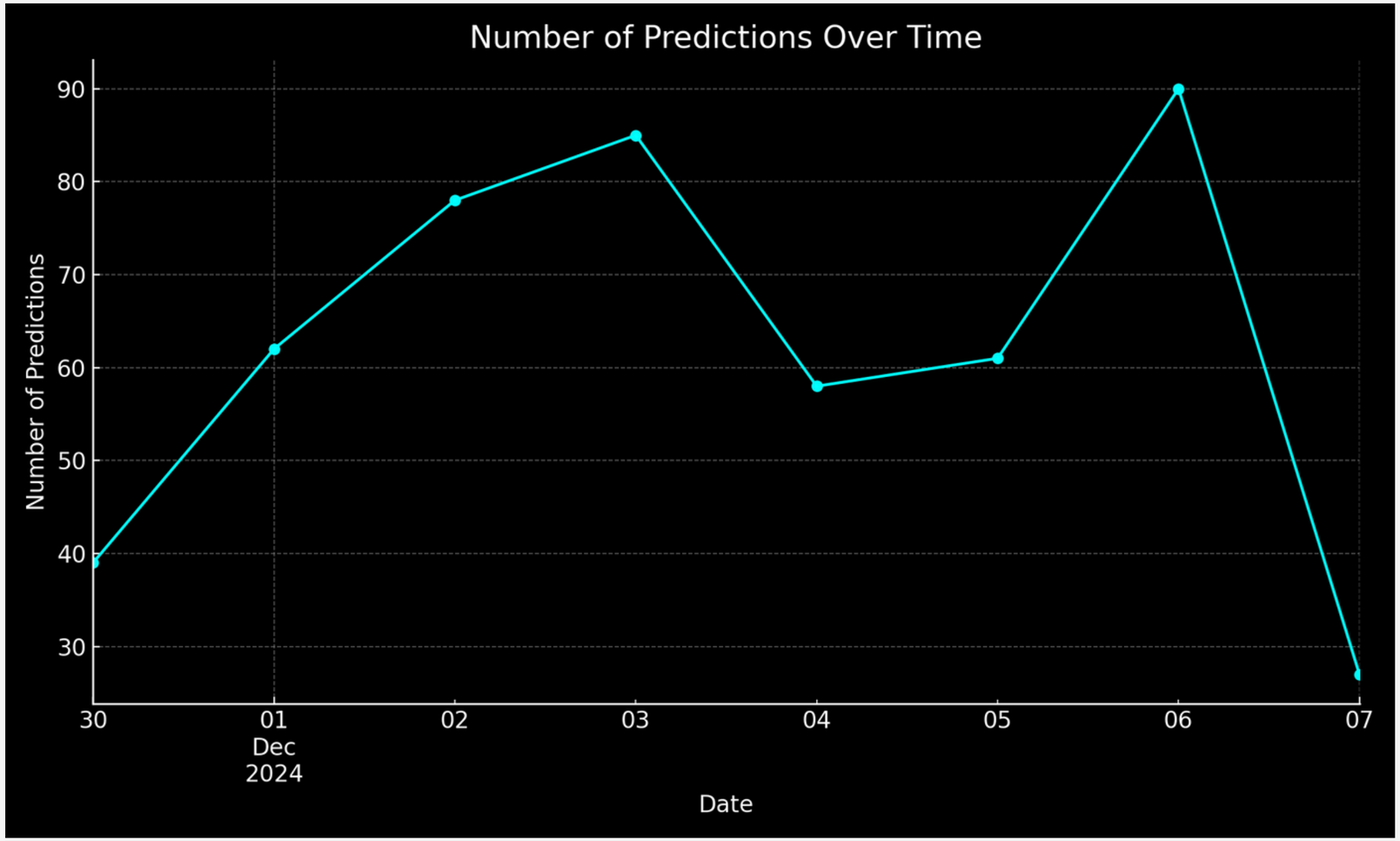
VISUALIZATIONS TABLEAU

PREDICTIONS OVER TIME - LINE CHART

This line chart tracks the number of predications made over time.

Analysis:

- Downward trend suggests a reduction in usage or delays in processing
- Consistent activity overtime reflects a steady model performance and usage
- Peaks in activity may reflect testing phases or specific events



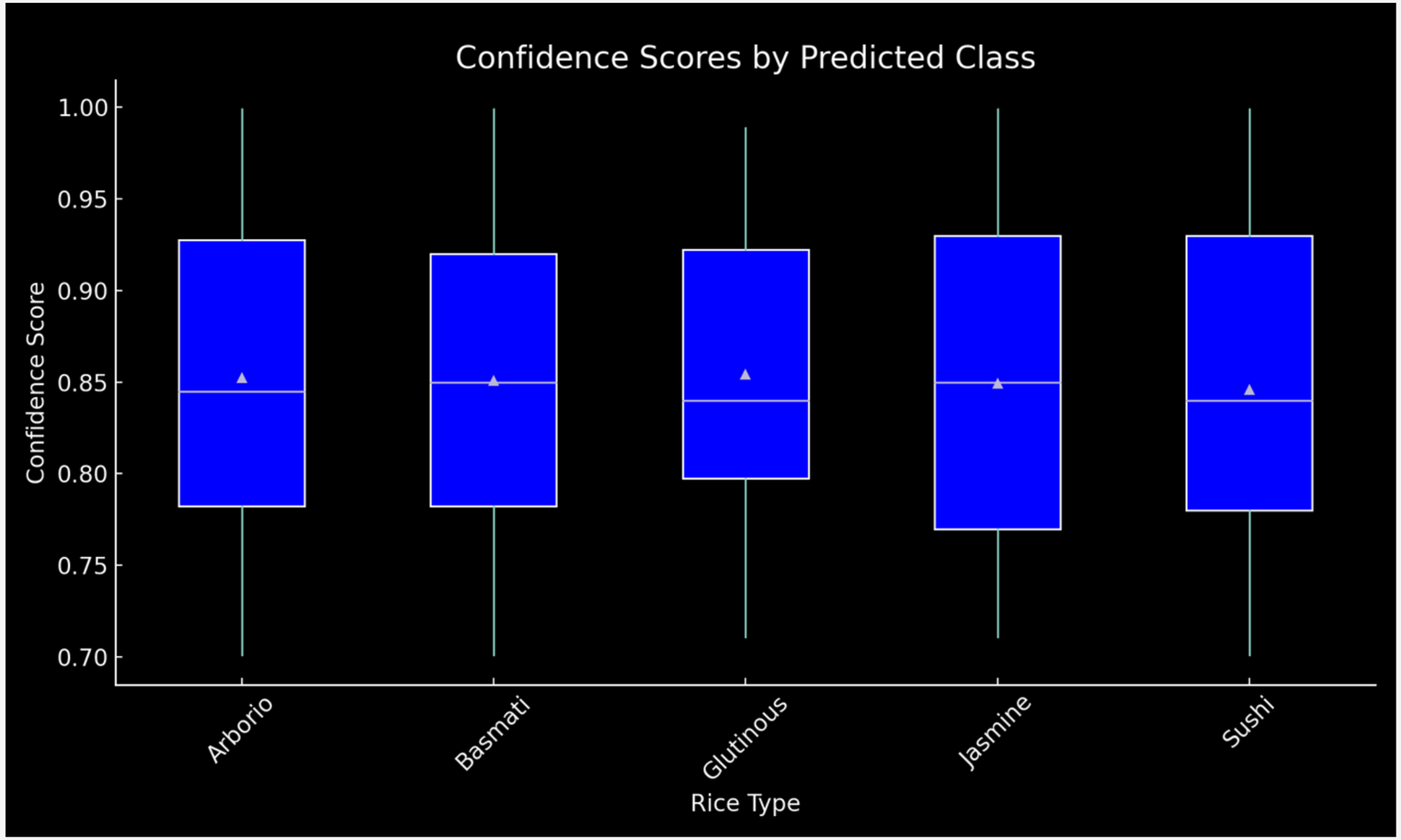
VISUALIZATIONS TABLEAU

CONFIDENCE SCORES - BOXPLOT

This box plot represents the range, median, and variability of confidence scores for each predicted rice class.

Analysis:

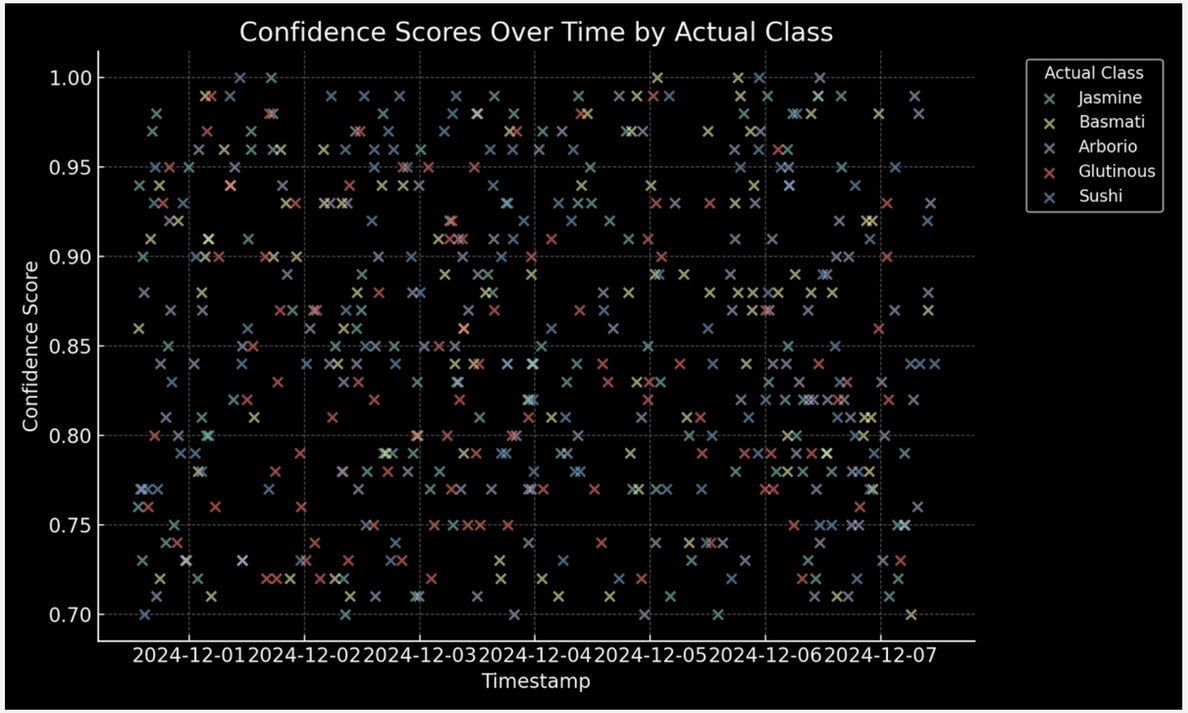
- Higher medians and narrower ranges suggest the model is confident in its predictions for specific classes
- Classes with wide ranges or lower medians might require additional model tuning or data augmentation
- Outliers in confidence scores can indicate predications that deviate significantly from the norm

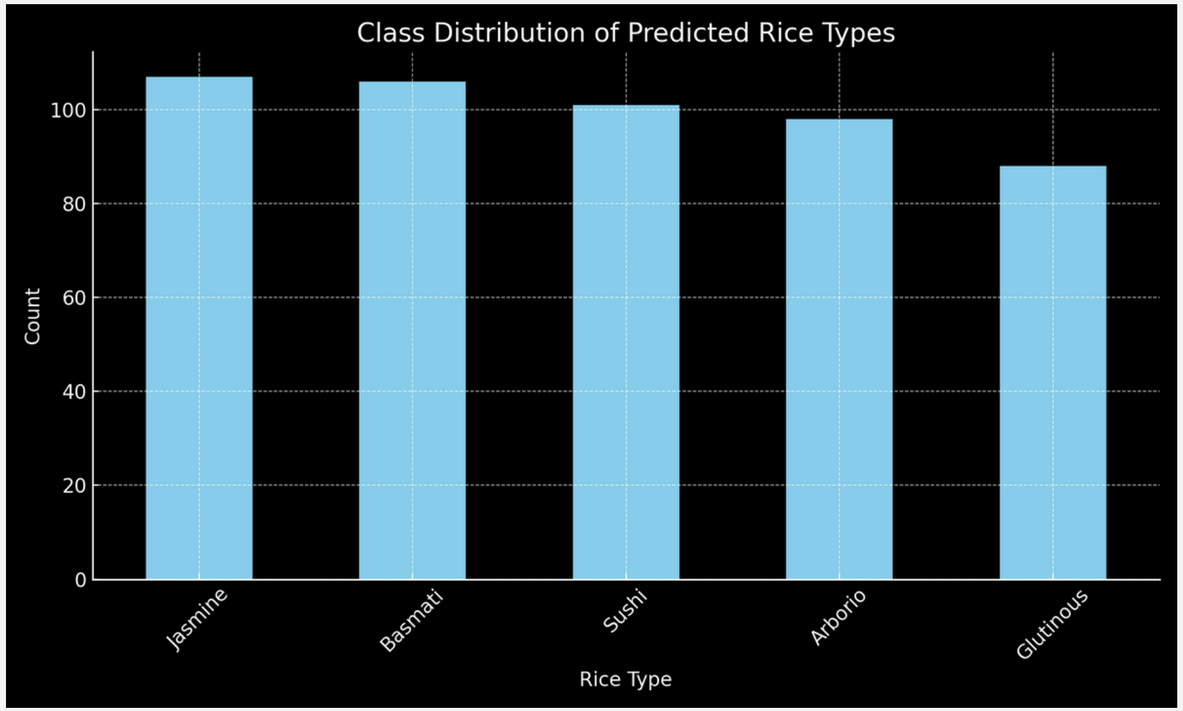


VISUALIZATIONS TABLEAU

CONFIDENCE OVER TIME - SCATTER PLOT

This scatter plot represents how confidence scores vary over time for each actual class of rice.





CLASS DISTRIBUTION - BAR CHART

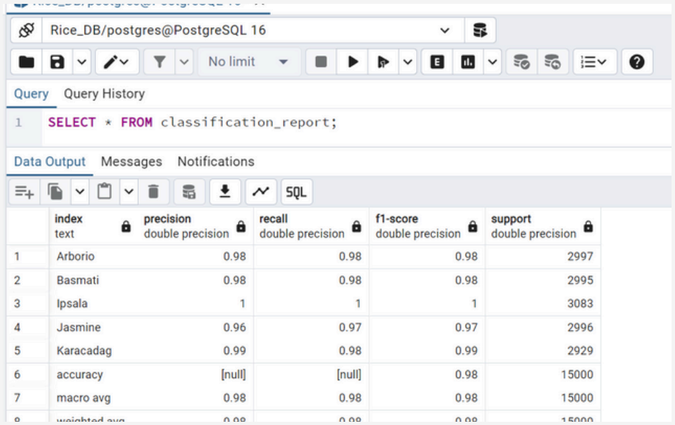
This bar chart represents the frequency of predictions for each rice type. The distribution highlights which rice classes were most commonly predicted by the model.

SQL STORAGE

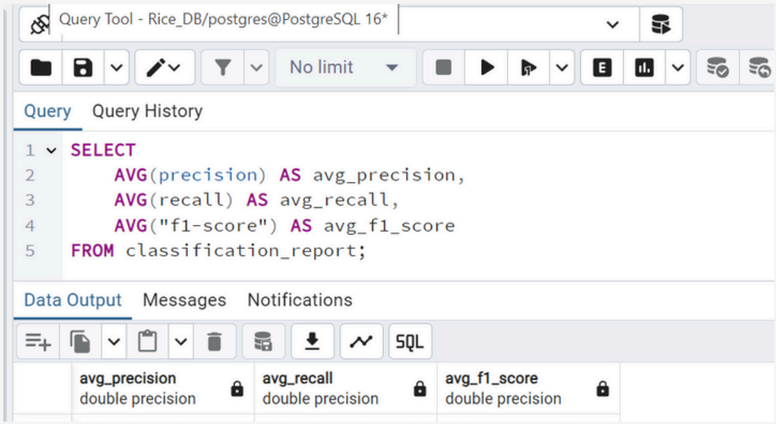
10

Using Python's SQLAlchemy library, the classification report was inserted into the database.

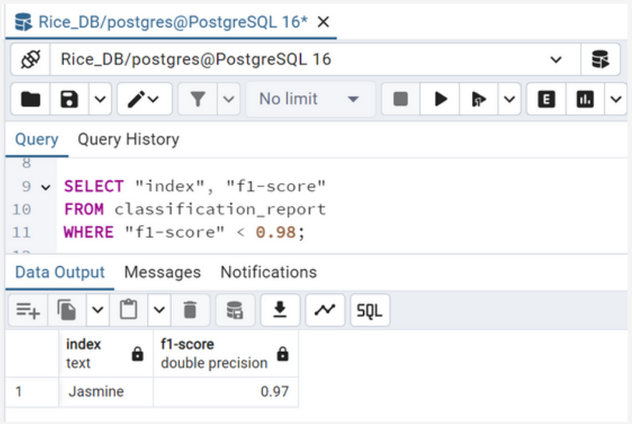
- The first query was used to retrieve all rows from the classification_report table.
- To understand the model's overall performance, average precision, recall, and F1-score were calculated.
- To identify under performing categories, classes with an F1-score below a threshold (0.98) were queried:



01



02

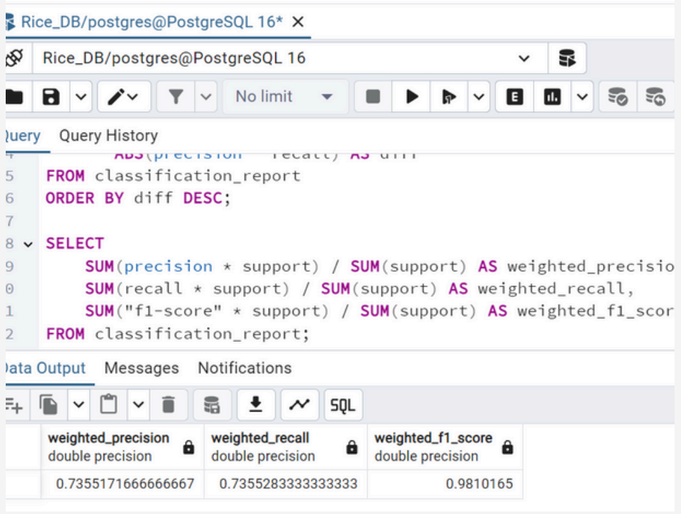


03

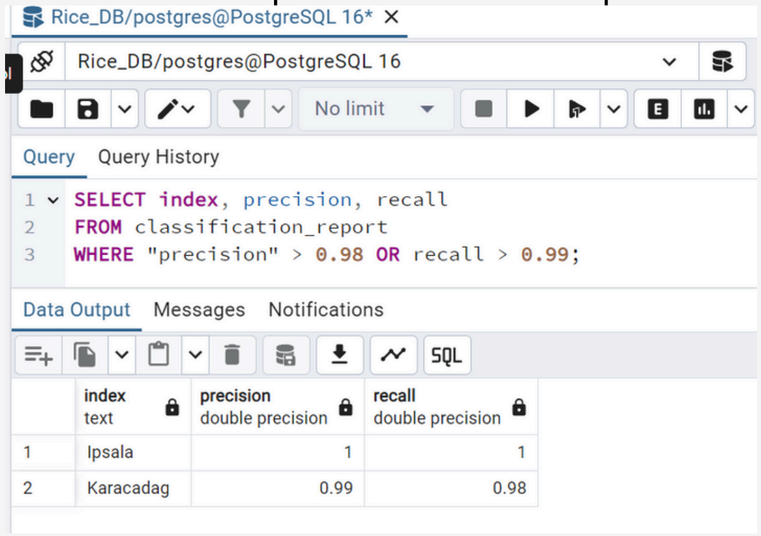
SQL QUERYING

10

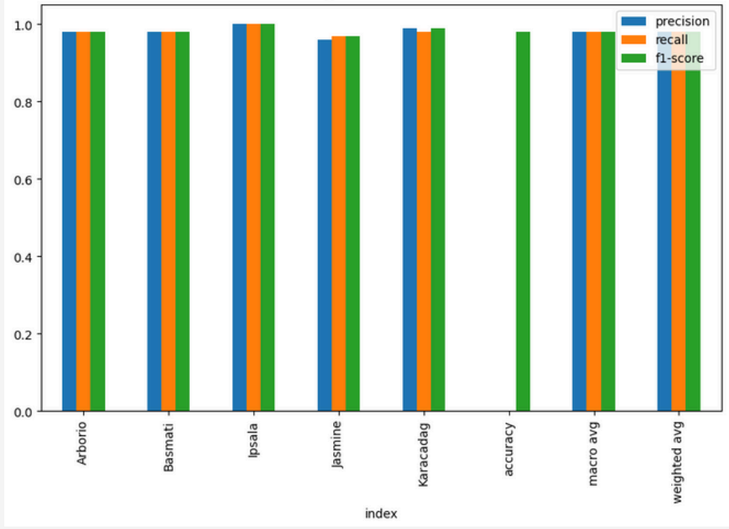
- To account for the varying support values, weighted averages of precision, recall, and F1-score were calculated
- To find classes where the model performed exceptionally well, categories with precision or recall above 99% were identified: Ipsala and Karacadag demonstrated near-perfect precision and recall.
- Although visualization was not directly performed in SQL, the retrieved data was imported into Python using pandas for plotting. This provided a clear visual comparison of model performance across rice categories.



01



02



03

THANK YOU!