Edge Intelligence

Krithana A

25MML0054

Github Link: https://github.com/Krithana/25MML0054_krithana_MACSE604

Code and Ouput :

Lab1:

```
In [2]:
import cv2
import matplotlib.pyplot as plt
import numpy as np

img_path = "/kaggle/input/intel-image-classification/seg_train/seg_train/forest/10007

img = cv2.imread(img_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

plt.imshow(img)
plt.title("Original RAW Image")
plt.axis("off")
```

Out[2]: (-0.5, 149.5, 149.5, -0.5)

Original RAW Image

```
In [3]:
img_resized = cv2.resize(img, (150, 150))

plt.imshow(img_resized)
plt.title("Resized Image (150x150)")
plt.axis("off")
```

Out[3]: (-0.5, 149.5, 149.5, -0.5)

Resized Image (150x150)

In [13]:
```
img_array = np.array(img_resized)
img_normalized = img_array / 255.0
```

In [14]:
```
img_reshaped = img_normalized.reshape(1, 150, 150, 3)
print(img_reshaped.shape)
```

```
(1, 150, 150, 3)
```

In [21]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2
)

train_data = datagen.flow_from_directory(
    "intel_image_dataset/seg_train/seg_train",
    target_size=(150,150),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

val_data = datagen.flow_from_directory(
    "intel_image_dataset/seg_train/seg_train",
    target_size=(150,150),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)
```

```
Found 11230 images belonging to 6 classes.
Found 2804 images belonging to 6 classes.
```

In [32]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(150,150,3)),
    MaxPooling2D(2,2),

    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),

    Flatten(),
    Dense(128, activation='relu'),
    Dense(6, activation='softmax')
])

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

model.summary()
```

```
L: CUDA error: Failed call to cuInit: UNKNOWN ERROR (303)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
flatten (Flatten)	(None, 82944)	0
dense (Dense)	(None, 128)	10,616,960
dense_1 (Dense)	(None, 6)	774

Total params: 10,637,126 (40.58 MB)

Trainable params: 10,637,126 (40.58 MB)

Non-trainable params: 0 (0.00 B)

In [2]:
```
import opendatasets as od
import os


dataset_url = 'https://www.kaggle.com/datasets/muratkokludataset/rice-image-dataset'

download_path = './rice_dataset'

if not os.path.exists(download_path):
    print("Downloading dataset (you will be prompted for Kaggle credentials)...")
    od.download(dataset_url, data_dir=download_path)
    print("Download complete.")
else:
    print(f"Dataset directory '{download_path}' already exists. Skipping download.")
```

```
Downloading dataset (you will be prompted for Kaggle credentials)...
Please provide your Kaggle credentials to download this dataset. Learn more: http://bit.ly/kaggle-creds
Your Kaggle username: Krithanaa
Your Kaggle Key: ········
Dataset URL: https://www.kaggle.com/datasets/muratkokludataset/rice-image-dataset
Download complete.
```

In [9]:
```
import pathlib
download_path=r"C:\Users\batch1\Downloads\ricedatset"
data_dir_root = pathlib.Path(download_path) / 'Rice_Image_Dataset'

if data_dir_root.exists():
    print(f"Successfully located data root directory: {data_dir_root}")
    # List the subdirectories (which are your class names)
    class_names = sorted([item.name for item in data_dir_root.glob('*') if item.is_dir()])
    print(f"Found {len(class_names)} classes: {class_names}")
else:
    print(f"Error: Directory not found at {data_dir_root}")
```

```
Successfully located data root directory: C:\Users\batch1\Downloads\ricedatset\Rice_Image_Dataset
Found 5 classes: ['Arborio', 'Basmati', 'Ipsala', 'Jasmine', 'Karacadag']
```

In [10]:
```
import tensorflow as tf

# Configuration parameters
IMAGE_SIZE = (224, 224) # Common size for CNN models
BATCH_SIZE = 32

# Load the training dataset using the confirmed path from Step 3
# We can also use validation_split here to create a separate validation set
train_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir_root,
    labels='inferred',
    label_mode='categorical',
    image_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    shuffle=True,
    validation_split=0.2, # Use 20% of data for validation
    subset='training',
    seed=123 # Seed for reproducible splits
)

# Load the validation dataset
val_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir_root,
    labels='inferred',
    label_mode='categorical',
    image_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    shuffle=False, # Often keep validation unshuffled
    validation_split=0.2,
    subset='validation',
    seed=123
)

print(f"\nTraining dataset created with {tf.data.experimental.cardinality(train_ds).numpy() * BATCH_SIZE} images (approx).")
print(f"Validation dataset created with {tf.data.experimental.cardinality(val_ds).numpy() * BATCH_SIZE} images (approx).")
print(f"Class names: {train_ds.class_names}")
```

```
Found 75000 files belonging to 5 classes.
Using 60000 files for training.
Found 75000 files belonging to 5 classes.
Using 15000 files for validation.

Training dataset created with 60000 images (approx).
Validation dataset created with 15008 images (approx).
Class names: ['Arborio', 'Basmati', 'Ipsala', 'Jasmine', 'Karacadag']
```

In [33]:
```
history = model.fit(
    train_data,
    validation_data=val_data,
    epochs=10
)
```

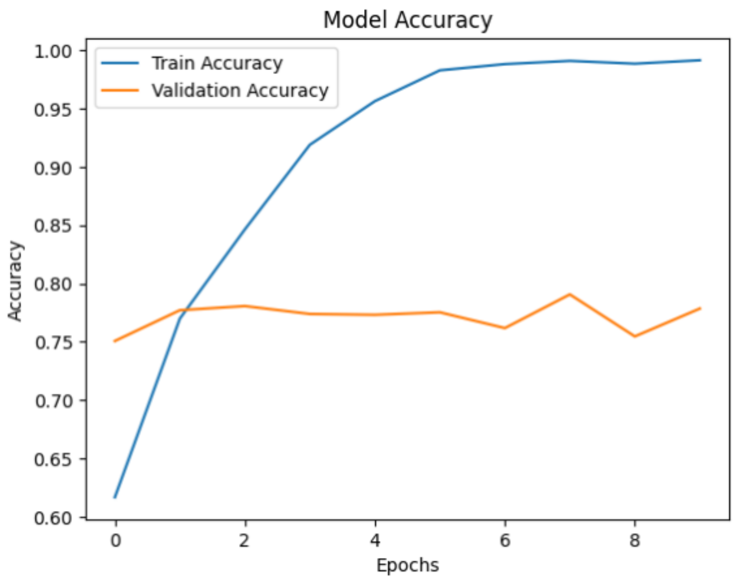
```
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyData
set` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessin
g`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
Epoch 1/10
351/351 ━━━━━━━━━━━━━━━━━━━━ 243s 688ms/step - accuracy: 0.5138 - loss: 1.5153 - val_accuracy: 0.7507 - val_loss: 0.6953
Epoch 2/10
351/351 ━━━━━━━━━━━━━━━━━━━━ 235s 668ms/step - accuracy: 0.7594 - loss: 0.6616 - val_accuracy: 0.7771 - val_loss: 0.6493
Epoch 3/10
351/351 ━━━━━━━━━━━━━━━━━━━━ 232s 662ms/step - accuracy: 0.8470 - loss: 0.4451 - val_accuracy: 0.7807 - val_loss: 0.6590
Epoch 4/10
351/351 ━━━━━━━━━━━━━━━━━━━━ 232s 661ms/step - accuracy: 0.9211 - loss: 0.2337 - val_accuracy: 0.7739 - val_loss: 0.7882
Epoch 5/10
351/351 ━━━━━━━━━━━━━━━━━━━━ 232s 662ms/step - accuracy: 0.9604 - loss: 0.1263 - val_accuracy: 0.7732 - val_loss: 0.8878
Epoch 6/10
351/351 ━━━━━━━━━━━━━━━━━━━━ 263s 665ms/step - accuracy: 0.9862 - loss: 0.0641 - val_accuracy: 0.7753 - val_loss: 0.9760
Epoch 7/10
351/351 ━━━━━━━━━━━━━━━━━━━━ 232s 661ms/step - accuracy: 0.9908 - loss: 0.0412 - val_accuracy: 0.7618 - val_loss: 0.9884
Epoch 8/10
351/351 ━━━━━━━━━━━━━━━━━━━━ 233s 663ms/step - accuracy: 0.9929 - loss: 0.0356 - val_accuracy: 0.7907 - val_loss: 1.0646
Epoch 9/10
351/351 ━━━━━━━━━━━━━━━━━━━━ 260s 656ms/step - accuracy: 0.9903 - loss: 0.0375 - val_accuracy: 0.7546 - val_loss: 1.2407
Epoch 10/10
351/351 ━━━━━━━━━━━━━━━━━━━━ 230s 655ms/step - accuracy: 0.9953 - loss: 0.0269 - val_accuracy: 0.7785 - val_loss: 1.1431
```

In [34]:
```
prediction = model.predict(img_reshaped)
predicted_class = list(train_data.class_indices.keys())[np.argmax(prediction)]

print("Predicted Class:", predicted_class)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 104ms/step
Predicted Class: forest
```

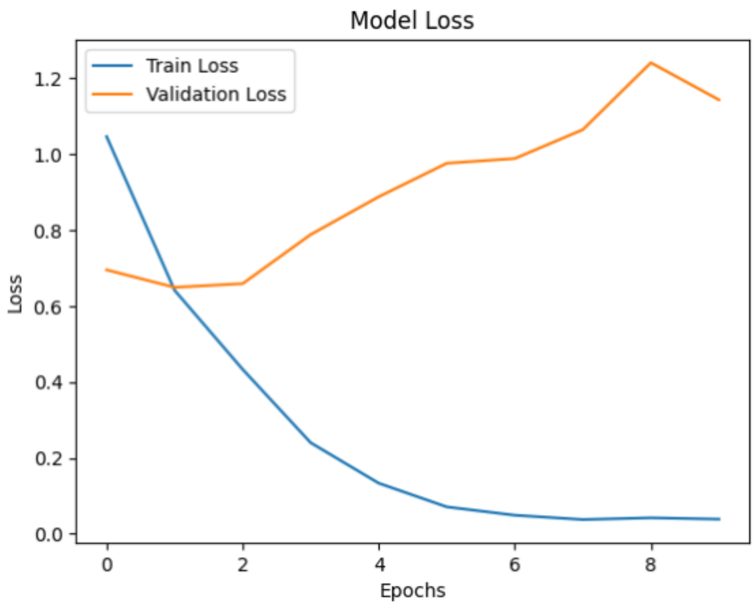
In [35]:

```
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Model Accuracy')
plt.show()
```



In [36]:

```
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Model Loss')
plt.show()
```



In [38]:

```
test_img_path = "/kaggle/input/intel-image-classification/seg_pred/seg_pred/10004.jpg"

img = cv2.imread(test_img_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (150,150))
img = img / 255.0
img = img.reshape(1,150,150,3)

prediction = model.predict(img)
predicted_class = list(train_data.class_indices.keys())[np.argmax(prediction)]

plt.imshow(img[0])
plt.title(f"Predicted: {predicted_class}")
plt.axis("off")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
```

Out[38]: (-0.5, 149.5, 149.5, -0.5)

Predicted: street

In [39]:

```
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np

y_true = []
y_pred = []

for i in range(len(val_data)):
    x, y = val_data[i]
    preds = model.predict(x)
    y_true.extend(np.argmax(y, axis=1))
    y_pred.extend(np.argmax(preds, axis=1))

print(classification_report(y_true, y_pred))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 258ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 211ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 203ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 203ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 220ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 236ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 203ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 200ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 206ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 207ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 205ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 206ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 204ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 202ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 203ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 200ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 206ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 203ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 204ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 201ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 202ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 203ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 209ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 211ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 210ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 203ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 202ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 209ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 203ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 204ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 204ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 210ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 202ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 204ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 206ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 204ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 139ms/step
              precision    recall  f1-score   support

           0       0.69      0.75      0.72       438
           1       0.95      0.94      0.94       454
           2       0.75      0.76      0.76       480
           3       0.79      0.67      0.73       502
           4       0.72      0.73      0.73       454
           5       0.78      0.82      0.80       476

    accuracy                           0.78      2804
   macro avg       0.78      0.78      0.78      2804
weighted avg       0.78      0.78      0.78      2804
```

Lab2:

In [2]:
```
!pip install nltk
```

```
Requirement already satisfied: nltk in c:\users\g anbalagan\anaconda3\lib\site-packages (3.7)
Requirement already satisfied: tqdm in c:\users\g anbalagan\anaconda3\lib\site-packages (from nltk) (4.64.1)
Requirement already satisfied: joblib in c:\users\g anbalagan\anaconda3\lib\site-packages (from nltk) (1.5.2)
Requirement already satisfied: click in c:\users\g anbalagan\anaconda3\lib\site-packages (from nltk) (8.0.4)
Requirement already satisfied: regex>=2021.8.3 in c:\users\g anbalagan\anaconda3\lib\site-packages (from nltk) (2022.7.9)
Requirement already satisfied: colorama in c:\users\g anbalagan\anaconda3\lib\site-packages (from click->nltk) (0.4.6)
```

In [3]:
```
nltk.download('names')
```

```
[nltk_data] Downloading package names to C:\Users\G
[nltk_data]     ANBALAGAN\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\names.zip.
```

Out[3]: True

In [4]:
```
from nltk.corpus import names
names.words()
print(len(names.words()))
```

```
7944
```

In [5]:
```
labeled_names=([(name,'male') for name in names.words("male.txt")]+[ (name,'female') for name in names.words("female.txt")]
print(labeled_names)
```

```
[('Aamir', 'male'), ('Aaron', 'male'), ('Abbey', 'male'), ('Abbie', 'male'), ('Abbot', 'male'), ('Abbott', 'male'), ('Abby',
'male'), ('Abdel', 'male'), ('Abdul', 'male'), ('Abdulkarim', 'male'), ('Abdullah', 'male'), ('Abe', 'male'), ('Abel', 'mal
e'), ('Abelard', 'male'), ('Abner', 'male'), ('Abraham', 'male'), ('Abram', 'male'), ('Ace', 'male'), ('Adair', 'male'), ('Ad
am', 'male'), ('Adams', 'male'), ('Addie', 'male'), ('Adger', 'male'), ('Aditya', 'male'), ('Adlai', 'male'), ('Adnan', 'mal
e'), ('Adolf', 'male'), ('Adolfo', 'male'), ('Adolph', 'male'), ('Adolphe', 'male'), ('Adolpho', 'male'), ('Adolphus', 'mal
e'), ('Adrian', 'male'), ('Adrick', 'male'), ('Adrien', 'male'), ('Agamemnon', 'male'), ('Aguinaldo', 'male'), ('Aguste', 'ma
le'), ('Agustin', 'male'), ('Aharon', 'male'), ('Ahmad', 'male'), ('Ahmed', 'male'), ('Ahmet', 'male'), ('Ajai', 'male'), ('A
jay', 'male'), ('Al', 'male'), ('Alaa', 'male'), ('Alain', 'male'), ('Alan', 'male'), ('Alasdair', 'male'), ('Alastair', 'mal
e'), ('Albatros', 'male'), ('Albert', 'male'), ('Alberto', 'male'), ('Albrecht', 'male'), ('Alden', 'male'), ('Aldis', 'mal
e'), ('Aldo', 'male'), ('Aldric', 'male'), ('Aldrich', 'male'), ('Aldus', 'male'), ('Aldwin', 'male'), ('Alec', 'male'), ('Al
eck', 'male'), ('Alejandro', 'male'), ('Aleks', 'male'), ('Aleksandrs', 'male'), ('Alessandro', 'male'), ('Alex', 'male'),
('Alexander', 'male'), ('Alexei', 'male'), ('Alexis', 'male'), ('Alf', 'male'), ('Alfie', 'male'), ('Alfonse', 'male'), ('Alf
onso', 'male'), ('Alfonzo', 'male'), ('Alford', 'male'), ('Alfred', 'male'), ('Alfredo', 'male'), ('Algernon', 'male'), ('Al
i', 'male'), ('Alic', 'male'), ('Alister', 'male'), ('Alix', 'male'), ('Allah', 'male'), ('Allan', 'male'), ('Allen', 'mal
e'), ('Alley', 'male'), ('Allie', 'male'), ('Allin', 'male'), ('Allyn', 'male'), ('Alonso', 'male'), ('Alonzo', 'male'), ('Al
oysius', 'male'), ('Alphonse', 'male'), ('Alphonso', 'male'), ('Alston', 'male'), ('Alton', 'male'), ('Alvin', 'male'), ('Alw
in', 'male'), ('Amadeus', 'male'), ('Ambros', 'male'), ('Ambrose', 'male'), ('Ambrosi', 'male'), ('Ambrosio', 'male'), ('Ambr
osius', 'male'), ('Amery', 'male'), ('Amory', 'male'), ('Amos', 'male'), ('Anatol', 'male'), ('Anatole', 'male'), ('Anatoll
o', 'male'), ('Anatoly', 'male'), ('Anders', 'male'), ('Andie', 'male'), ('Andonis', 'male'), ('Andre', 'male'), ('Andrea',
'male'), ('Andreas', 'male'), ('Andrej', 'male'), ('Andres', 'male'), ('Andrew', 'male'), ('Andrey', 'male'), ('Andri', 'mal
```

In [11]:

```
import random
random.shuffle(labeled_names)
featuresets = [(gender_features(n), gender) for (n,gender) in labeled_names]
train_set, test_set = featuresets[5000:], featuresets[:2000]
import nltk
classifier= nltk.NaiveBayesClassifier.train(train_set)
classifier.classify(gender_features('Jhon'))
classifier.classify(gender_features('Yann Lecun'))

print(nltk.classify.accuracy(classifier,test_set))
```

```
0.771
```

Six Jars Model

1️⃣ Data Jar

Contains the input data used for training and testing

Includes data collection, cleaning, preprocessing, and feature extraction

Quality data directly impacts model performance

2️⃣ Task Jar

Defines what problem the model is solving

Examples: classification, regression, clustering

A clear task ensures correct model selection and evaluation

3️⃣ Model Jar

Refers to the algorithm or architecture used

Examples: Linear Regression, SVM, CNN, Transformer

The model learns patterns from data

4️⃣ Loss Jar

Measures how wrong the model's predictions are

Guides the learning process

Examples: Mean Squared Error, Cross-Entropy Loss

5️⃣ Learning Jar

Describes how the model improves using optimization techniques

Includes learning rate, gradient descent, backpropagation

Adjusts model parameters to minimize loss

6️⃣ Accuracy Jar

Evaluates how well the trained model performs

Measures correctness of predictions on unseen data

Can include accuracy, precision, recall, F1-score

Lab 3:

In [3]:

```
import pandas as pd
df=pd.read_csv(r"C:\Users\batch1\Downloads\logs_dataset.csv")
df.head()
```

Out[3]:

	@timestamp	_id	ip_address
0	July 8th 2019, 14:43:03.000	XswJ0msBoTGddM7vxMDB	10.1.1.285
1	July 8th 2019, 14:43:01.000	dKQJ0msB7mP0GwVzvJjz	10.1.2.389
2	July 8th 2019, 14:42:59.000	CcwJ0msBoTGddM7vtb8y	10.1.1.415
3	July 8th 2019, 14:42:57.000	bKQJ0msB7mP0GwVzrZdT	10.1.1.79
4	July 8th 2019, 14:42:55.000	L6QJ0msB7mP0GwVzpZeI	10.1.1.60

In [4]:

```
df['@timestamp'] = (
    df['@timestamp']
    .str.replace(r'(\d+)(st|nd|rd|th)', r'\1', regex=True)
)
df['@timestamp'] = pd.to_datetime(
    df['@timestamp'],
    format='%B %d %Y, %H:%M:%S.%f'
)
```

In [6]:

```
df.sort_values(['ip_address', '@timestamp'], inplace=True)
df['shift_time'] = df.groupby(['ip_address'])['@timestamp'].shift(1)
df.head()
```

Out[6]:

	@timestamp	_id	ip_address	shift_time
721473	2019-06-09 00:06:09	DBuOOWsB7mP0GwVzhZ9U	10.1.1.1	NaT
720483	2019-06-09 01:28:39	bB7aOWsB7mP0GwVzDY5G	10.1.1.1	2019-06-09 00:06:09
719233	2019-06-09 03:12:49	R0w5OmsBoTGddM7vayZT	10.1.1.1	2019-06-09 01:28:39
719222	2019-06-09 03:13:45	U0w6OmsBoTGddM7vRi8R	10.1.1.1	2019-06-09 03:12:49
718875	2019-06-09 03:42:39	z01UOmsBoTGddM7vuzyC	10.1.1.1	2019-06-09 03:13:45

In [14]:

```
df['time_diff'] = (df['@timestamp']-df['shift_time']).dt.seconds//60
df['date'] = df['@timestamp'].dt.date
df['weekday'] = df['@timestamp'].dt.weekday
df['hour'] = df['@timestamp'].dt.hour
df['is_weekend'] = ((df['weekday'] == 5) | (df['weekday'] == 6)).astype(int)
df['hour_buckets']=df['hour']//4
ip_addr='ip_address'
ip_counts = df.groupby(ip_addr)['@timestamp'].count().reset_index()
ip_counts.head()
ip_counts = ip_counts.rename(columns={'@timestamp':'total_count'})
df.head()
```

Out[14]:

	@timestamp	_id	ip_address	shift_time	time_diff	date	weekday	hour	is_weekend	hour_buckets
721473	2019-06-09 00:06:09	DBuOOWsB7mP0GwVzhZ9U	10.1.1.1	NaT	NaN	2019-06-09	6	0	1	0
720483	2019-06-09 01:28:39	bB7aOWsB7mP0GwVzDY5G	10.1.1.1	2019-06-09 00:06:09	82.0	2019-06-09	6	1	1	0
719233	2019-06-09 03:12:49	R0w5OmsBoTGddM7vayZT	10.1.1.1	2019-06-09 01:28:39	104.0	2019-06-09	6	3	1	0
719222	2019-06-09 03:13:45	U0w6OmsBoTGddM7vRi8R	10.1.1.1	2019-06-09 03:12:49	0.0	2019-06-09	6	3	1	0
718875	2019-06-09 03:42:39	z01UOmsBoTGddM7vuzyC	10.1.1.1	2019-06-09 03:13:45	28.0	2019-06-09	6	3	1	0

In [17]:

```
daily_counts = df.groupby([ip_addr,'date'])['@timestamp'].count().reset_index()
daily_counts_avg=daily_counts.groupby([ip_addr,'date'])['@timestamp'].count().reset_index()
daily_counts_avg.head(5)
weekend_counts = df.groupby([ip_addr, 'is_weekend'])['@timestamp'].count().reset_index()
weekend_counts
```

Out[17]:

	ip_address	is_weekend	@timestamp
0	10.1.1.1	0	975
1	10.1.1.1	1	471
2	10.1.1.100	0	1960
3	10.1.1.100	1	900
4	10.1.1.101	0	1006
...	...	...	...
767	10.1.2.90	1	871
768	10.1.2.95	0	1973
769	10.1.2.95	1	895
770	10.1.2.99	0	978
771	10.1.2.99	1	445

772 rows × 3 columns