National College of Ireland

Configuration Manual

MSc Research Project
MSc Data Analytics (MSCDAD_A)

Devanshu Singh
Student ID: x23267143

School of Computing
National College of Ireland

Supervisor: David Hamill

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Devanshu Singh
Student ID:	x23267143
Programme:	MSc. Data Analytics **Year:** 2024 - 2025
Module:	Research Practicum 2
Lecturer:	David Hamill
Submission Due Date:	11-08-2025
Project Title:	Pixels to Policy: A Multi-Modal AI Framework for Proactive, Sustainable Waste Management Across Urban & Rural Regions
Word Count:	**Page Count:**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Devanshu Singh

Date: 10-08-2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Devanshu Singh
Student ID: x23267143

SETUP/CONFIGURATION

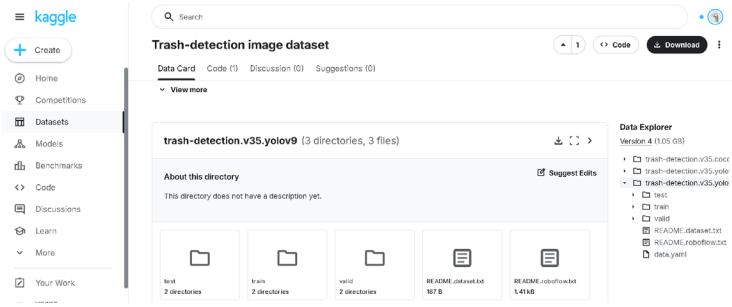
Section 1: Requirements

Google Colab: Used for executing Python code via browser.
Runtime Type: Python 3
Hardware Accelerator: L4 GPU

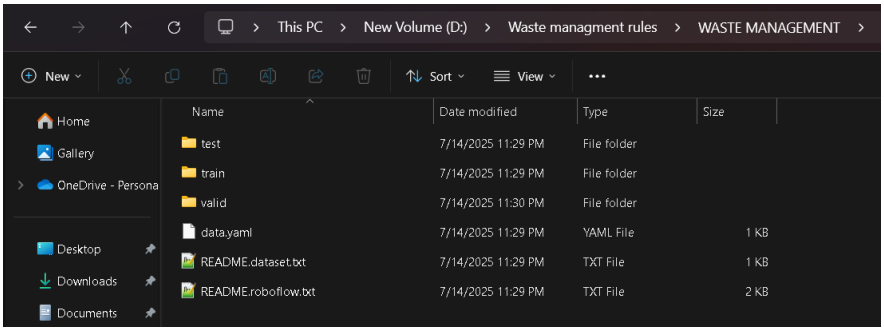
Supporting Tools: Google Drive
Deep Learning Frameworks: ultralytics (YOLOv8), TensorFlow/Keras
Libraries for Data Handling & Visualization: Pandas, Numpy, Matplotlib, Yaml, cv2 (OpenCV), Glob, os & shutil

Section 2: Dataset Collection

1. Download the Waste Image Dataset from Kaggle – https://www.kaggle.com/datasets/ahnaftahmeed/trash-detection-image-dataset/data?select=trash-detection.v35.yolov9

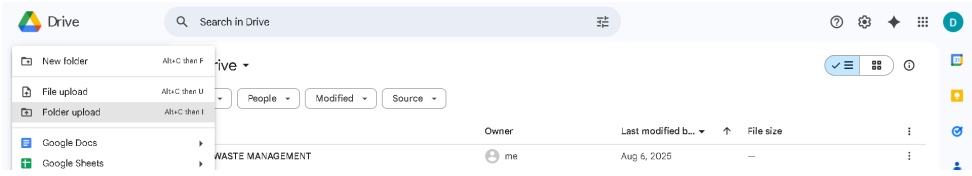


2. Extract and save the folder in your local system and then rename the folder to 'WASTE MANAGEMENT'.

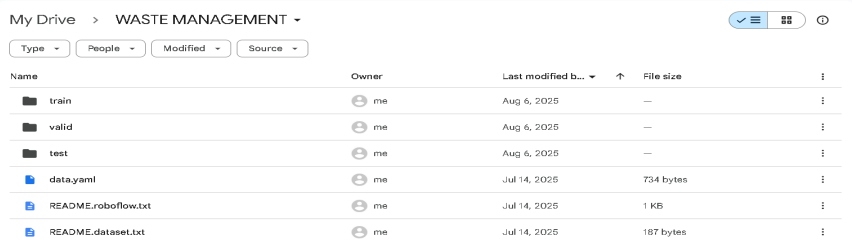


Section 3: Google Drive Setup

3. Open Google Drive; go to My Drive; click New; choose Folder upload; upload the WASTE MANAGEMENT folder.

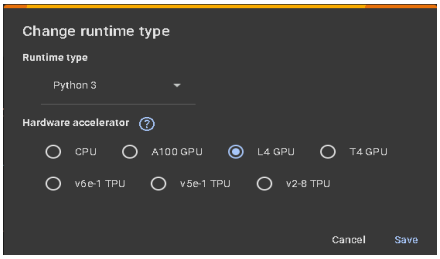


4. Locate the WASTE MANAGEMENT folder; open it; confirm the folder structure matches the image shown below.

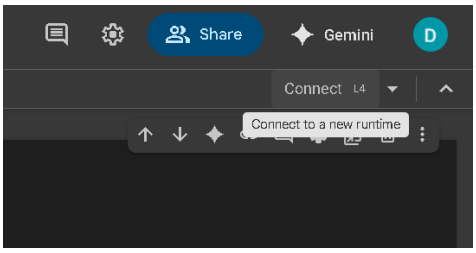


Section 4: Google Colab Setup

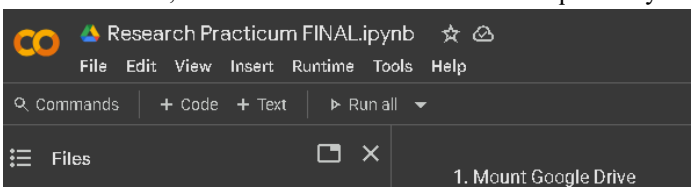
5. Open the code file in Google Colab; click Runtime; click Change runtime type; set the hardware accelerator to L4 GPU.



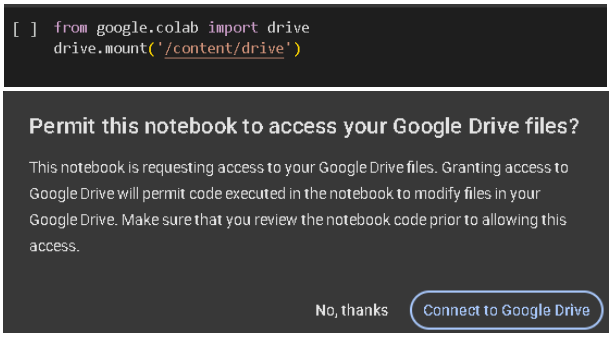
6. Once changed, click on 'Connect L4' to connect to the select L4 GPU runtime.



7. Once connected to the runtime; click Run all to execute all cells sequentially.



8. On the first cell's Drive access prompt, click 'Connect to Google Drive' and grant permission; once connected, let the notebook proceed to the next cells.



EXECUTION OF THE CODE

Section 5: Classification of Waste Types using YOLOv8s

Section 5.1: Preprocessing of Waste Image Data

9. Count YOLO label frequencies (train+valid) - Parses label files to tally instances per class across splits.

```python
import os
import glob
from collections import defaultdict

# Define label paths
label_dirs = [
    "/content/drive/MyDrive/WASTE MANAGEMENT/train/labels",
    "/content/drive/MyDrive/WASTE MANAGEMENT/valid/labels"
]

# Count class occurrences
class_counts = defaultdict(int)

for label_dir in label_dirs:
    label_files = glob.glob(os.path.join(label_dir, "*.txt"))
    for file in label_files:
        with open(file, "r") as f:
            for line in f:
                if line.strip():
                    class_id = int(line.split()[0])
                    class_counts[class_id] += 1

# Show class counts
print("Class Frequencies:")
for cls_id, count in sorted(class_counts.items()):
    print(f"Class {cls_id}: {count} instances")
```

10. Copy dataset from Drive to Colab - Moves data to local /content for faster I/O during training.

```python
import shutil

# Source and destination paths
drive_base = "/content/drive/MyDrive/WASTE MANAGEMENT"
local_base = "/content/WASTE_LOCAL"

# List of folders to copy
folders_to_copy = ["train", "valid", "test"]

# Copy each folder recursively
for folder in folders_to_copy:
    src = os.path.join(drive_base, folder)
    dst = os.path.join(local_base, folder)
    shutil.copytree(src, dst, dirs_exist_ok=True)

print("Dataset copied to Colab local environment.")
```

11. Resize dataset images to 640×640 - Batch-rescales images to YOLO's target size, keeping labels aligned.

```python
import cv2
from glob import glob

# Resize the dataset images
def resize_images(image_folder, size=(640, 640)):
    image_paths = glob(os.path.join(image_folder, "*.jpg")) + glob(os.path.join(image_folder, "*.png"))
    for img_path in image_paths:
        img = cv2.imread(img_path)
        if img is None:
            continue
        resized_img = cv2.resize(img, size)
        cv2.imwrite(img_path, resized_img)

# Apply to all splits
resize_images("/content/WASTE_LOCAL/train/images")
resize_images("/content/WASTE_LOCAL/valid/images")
resize_images("/content/WASTE_LOCAL/test/images")
```

12. Tiered oversampling for underrepresented classes - Duplicates or augments scarce classes to reduce imbalance.

```python
from collections import defaultdict
import shutil
import os
import numpy as np

def tiered_oversample_classes(label_dir, image_dir, output_label_dir, output_image_dir):
    os.makedirs(output_label_dir, exist_ok=True)
    os.makedirs(output_image_dir, exist_ok=True)

    class_to_files = defaultdict(list)

    # Map files to class
    for label_file in os.listdir(label_dir):
        if label_file.endswith(".txt"):
            label_path = os.path.join(label_dir, label_file)
            try:
                with open(label_path, 'r') as f:
                    for line in f:
                        class_id = int(float(line.strip().split()[0]))
                        class_to_files[class_id].append(label_file)
                        break
            except:
                continue
```

13. Write YOLO data.yaml for local dataset - Creates a new config pointing to the local resized/oversampled data.

```
yaml_text = """
path: /content/WASTE_LOCAL
train: train/images
val: valid/images
test: test/images

nc: 29
names: ['Aluminium foil', 'Bottle cap', 'Broken glass', 'Cigarette', 'Clear plastic bottle', 'Crisp packet',
        'Cup', 'Drink can', 'Food Carton', 'Food container', 'Food waste', 'Garbage bag', 'Glass bottle', 'Lid',
        'Other Carton', 'Other can', 'Other container', 'Other plastic', 'Other plastic bottle', 'Other plastic wrapper',
        'Paper', 'Paper bag', 'Plastic bag wrapper', 'Plastic film', 'Pop tab', 'Single-use carrier bag',
        'Straw', 'Styrofoam piece', 'Unlabeled litter']
""".strip()

with open("/content/WASTE_LOCAL/data.yaml", "w") as f:
    f.write(yaml_text)
```

14. Install/upgrade Ultralytics - Ensures the latest YOLOv8 package and dependencies are available.

```
!pip install ultralytics --upgrade -q
```

Section 5.2: Training of the YOLOv8s Model

15. Train YOLOv8 model on waste dataset - Launches training with chosen hyperparameters and logs.

```
from ultralytics import YOLO

# Load pretrained YOLOv8s model
model = YOLO("yolov8s.pt")

# Train configuration
model.train(
    data='/content/WASTE_LOCAL/data.yaml',
    epochs=100,
    imgsz=640,
    batch=64,
    lr0=0.001,
    optimizer='SGD',
    cos_lr=True,
    project='waste_detection_yolo',
    name='high_precision_run',
    augment=True,
    verbose=True
)
```

Section 5.3: Saving and Loading Best Model to and from Google Drive

16. Save best YOLO weights to Google Drive - Copies best_model_path.pt (best checkpoint) back to Drive for safekeeping.

```
from pathlib import Path
import shutil

# Locate best model after training
best_model_path = Path('waste_detection_yolo/high_precision_run/weights/best.pt')

# Define target save location in Google Drive
target_path = Path('/content/drive/MyDrive/WASTE MANAGEMENT/yolo_best_model.pt')

# Copy best model to Google Drive
shutil.copy(best_model_path, target_path)
print(f"Best YOLO model saved to Google Drive at:\n{target_path}")
```

17. Load trained YOLO model (best_model_path.pt) - Instantiates a YOLO predictor from the saved checkpoint.

```
from ultralytics import YOLO

# Load the best model from training
model = YOLO('waste_detection_yolo/high_precision_run/weights/best.pt')
```

Section 5.4: Running Inference on Test Data

18. Run YOLO inference on test set and save predictions - Generates predictions and writes results/plots to disk.

```python
from ultralytics import YOLO
import os
from glob import glob
import shutil

# Set test images directory
test_img_dir = '/content/WASTE_LOCAL/test/images'
output_dir = '/content/test_predictions'

# Clean and recreate output dir
if os.path.exists(output_dir):
    shutil.rmtree(output_dir)
os.makedirs(output_dir, exist_ok=True)

# Run predictions
results = model.predict(
    source=test_img_dir,
    save=True,
    save_txt=True,
    save_conf=True,
    conf=0.25,
    iou=0.45,
    max_det=300,
    project=output_dir,
    name='results',
    exist_ok=True
)
```

Section 6: Forecasting using LSTM

Section 6.1: Curating, Saving & Loading and Preprocessing Data

19. Curate waste forecast metadata via YOLO inference - Uses detections to build time/location aggregates.

```python
image_dir = "/content/drive/MyDrive/WASTE MANAGEMENT/train/images"
model_path = "/content/drive/MyDrive/WASTE MANAGEMENT/yolo_best_model.pt"
output_csv = "/content/drive/MyDrive/WASTE MANAGEMENT/waste_forecast_metadata_enhanced.csv"
n_days = 1000

# Location settings
locations = [
    {"lat": 53.338353, "long": -6.258680, "place_type": "Park", "time_range": (11, 17)},
    {"lat": 53.344062, "long": -6.254571, "place_type": "College", "time_range": (9, 19)},
    {"lat": 53.348341, "long": -6.246732, "place_type": "Hotel", "time_range": (6, 23)}
]

# Load model
model = YOLO(model_path)
class_names = model.names

# Sample exactly 3000 images
all_images = sorted([f for f in os.listdir(image_dir) if f.lower().endswith(('.jpg', '.jpeg', '.png'))])
if len(all_images) < 3000:
    raise ValueError(f"Not enough images found! Needed 3000 but found {len(all_images)}")

sampled_images = random.sample(all_images, 3000)

# Split into 1000 days x 3 locations
grouped_images = [sampled_images[i:i + 3] for i in range(0, 3000, 3)]
```

20. Load and preview enhanced forecast dataset - Reads the engineered dataset and shows head/sample.

```python
import pandas as pd

# Load the enhanced dataset
df = pd.read_csv('/content/drive/MyDrive/WASTE MANAGEMENT/waste_forecast_metadata_enhanced.csv')

# Show the first few rows of the dataset
df.head()
```

21. Initial EDA: shape, info, missing, duplicates, stats – Quick health check of data quality and distributions.

```python
# Number of rows and columns
print(f"Dataset shape: {df.shape}")

# Column data types and non-null counts
print("\n--- Info ---")
df.info()

# Check for missing values
print("\n--- Missing Values ---")
print(df.isnull().sum())

# Check for duplicates
print("\n--- Duplicate Rows ---")
print(df.duplicated().sum())

# Summary statistics for numeric columns
print("\n--- Summary Statistics ---")
print(df.describe())

# Preview unique values in categorical columns
categorical_cols = ['day', 'season', 'place_type']
for col in categorical_cols:
    print(f"\nUnique values in '{col}':", df[col].unique())
```

22. Build daily feature matrix with one-hot waste targets - Creates model-ready features and multi-target labels.

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import MultiLabelBinarizer

# Combine date & time
df['datetime'] = pd.to_datetime(df['date'] + ' ' + df['time'])
df.set_index('datetime', inplace=True)
df = df.sort_index()

# Convert stringified waste_types
if isinstance(df['waste_types'].iloc[0], str):
    df['waste_types'] = df['waste_types'].apply(eval)

# One-hot encode waste_types
mlb = MultiLabelBinarizer()
waste_encoded = pd.DataFrame(mlb.fit_transform(df['waste_types']),
                             columns=mlb.classes_,
                             index=df.index)

# Prepare features from metadata
df_features = df[['place_type', 'season']].copy()
df_features = pd.get_dummies(df_features, columns=['place_type', 'season'])

# Combine metadata and target (waste types)
full_df = df_features.join(waste_encoded)

# Resample to daily frequency
daily_df = full_df.resample('D').sum()

# Remove days with zero total waste
daily_df = daily_df[daily_df.sum(axis=1) > 0]
```

23. Create 7-day sequences for forecasting - Windows the data into rolling sequences for sequence models.

```python
import numpy as np

def create_sequences(data, lookback=7):
    X, y = [], []
    data_values = data.values

    for i in range(len(data_values) - lookback):
        X.append(data_values[i:i + lookback])
        y.append(data_values[i + lookback])

    return np.array(X), np.array(y)

# Apply the function to your daily_df DataFrame
lookback = 7
X, y = create_sequences(daily_df, lookback=lookback)

# Print resulting shapes
print("X shape:", X.shape, "| y shape:", y.shape)
```

24. Define LSTM model for multivariate next-day prediction - Builds a neural net to predict next-day waste counts.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

model = Sequential([
    LSTM(128, input_shape=(lookback, X.shape[2]), return_sequences=False),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dense(y.shape[1])
])

model.compile(optimizer='adam', loss='mse')
model.summary()
```

Section 6.2: Training LSTM Model on Spatio-Temporal Data

25. Train LSTM with early stopping and save best model - Fits the model; saves the top validation checkpoint.

```python
from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=50,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)

# Save the best model to Google Drive
model_save_path = '/content/drive/MyDrive/WASTE MANAGEMENT/lstm_best_model.h5'
model.save(model_save_path)
print(f"Best LSTM model saved to Google Drive at:\n{model_save_path}")
```

Section 6.3: Evaluation of the LSTM model

26. Evaluate LSTM predictions per waste type (RMSE/MAE) – Computes error metrics by class for granular insight.

```
y_pred = model.predict(X_test)

# Evaluate per waste type
from sklearn.metrics import mean_squared_error, mean_absolute_error

waste_cols = mlb.classes_
print("Evaluation per waste type:")
for i, col in enumerate(waste_cols):
    rmse = np.sqrt(mean_squared_error(y_test[:, i], y_pred[:, i]))
    mae = mean_absolute_error(y_test[:, i], y_pred[:, i])
    print(f"{col:<25} | RMSE: {rmse:.2f} | MAE: {mae:.2f}")
```

27. Discretize predictions into quantile-based risk levels – Converts continuous forecasts to low/med/high risk bins.

```
quantiles = [0.0, 0.4, 0.7, 0.9, 0.98, 1.0]  # bin boundaries
levels = [0, 1, 2, 3, 4]                     # integer levels per bin

def class_bins_by_quantiles(y_pred_cont, q, levels):
    C = y_pred_cont.shape[1]
    all_bins = []
    for j in range(C):
        edges = np.quantile(y_pred_cont[:, j], q)
        edges[0] = 0.0
        all_bins.append((edges, np.array(levels, int)))
    return all_bins

def apply_class_bins(y_pred_cont, class_bins):
    C = y_pred_cont.shape[1]
    out = np.zeros_like(y_pred_cont, dtype=int)
    for j in range(C):
        edges, levs = class_bins[j]
        idx = np.digitize(y_pred_cont[:, j], edges) - 1
        idx = np.clip(idx, 0, len(levs)-1)
        out[:, j] = levs[idx]
    return out

# BUILD BINS FROM VALIDATION
y_pred_val_clipped = np.clip(y_pred, 0, None)[:, :len(waste_cols)]
class_bins = class_bins_by_quantiles(y_pred_val_clipped, quantiles, levels)

# APPLY TO FORECAST
y_pred_clipped = np.clip(y_pred, 0, None)[:, :len(waste_cols)]
y_pred_rounded = apply_class_bins(y_pred_clipped, class_bins)
```

28. Install tabulate - Adds pretty table printing for cleaner metric/policy displays.

```
from tabulate import tabulate

y_pred_rounded = y_pred_rounded[:, :len(waste_cols)]

# Attach forecast dates
forecast_dates = daily_df.index[-len(y_pred_rounded):]

# Create prediction DataFrame
pred_df = pd.DataFrame(y_pred_rounded, columns=waste_cols)
pred_df['datetime'] = forecast_dates

# Match place_type using df index
df_reset = df.reset_index()

# Extract place_type context matching forecast dates
context_df = df_reset[['datetime', 'place_type', 'time', 'season', 'day', 'image_id']].drop_duplicates().sort_values('datetime')
context_df = context_df[-len(pred_df):].reset_index(drop=True)

# Merge predictions with context
pred_df = pd.concat([pred_df, context_df[['place_type',  'time', 'season', 'day', 'image_id']]], axis=1)

# Filter forecast for next 7 days
forecast_next_7 = pred_df.tail(7)

# Group forecast by place_type and summarize waste counts
weekly_summary = forecast_next_7.groupby('place_type')[waste_cols].sum()

# Clean column names for display
weekly_summary = weekly_summary[waste_cols]  # Ensure correct order
weekly_summary.columns = [col.replace('waste_types_', '').replace('_', ' ').title()
                          for col in weekly_summary.columns]
```

Section 6.4: Saving & Loading the LSTM results in Google Drive

29. Export 7-day forecast to CSV - Saves forecasts for reuse, sharing, or downstream steps.

```
[ ]  # Save to CSV
     csv_path = "/content/drive/MyDrive/WASTE MANAGEMENT/LSTM_RES.csv"
     forecast_next_7.to_csv(csv_path, index=False)
```

30. Load and preview saved 7-day forecast CSV - Sanity-checks the exported file and schema.

```
import pandas as pd

# Load the enhanced dataset
df_forecast_week = pd.read_csv('/content/drive/MyDrive/WASTE MANAGEMENT/LSTM_RES.csv')

# Show the first few rows of the dataset
df_forecast_week.head()
```

Section 7: Policy Generation using Zephyr

Section 7.1: Query/Prompt Generation from LSTM results

31. Install Transformers, Accelerate, and bitsandbytes - Prepares lightweight LLM inference environment.

```
!pip install transformers accelerate bitsandbytes
```

32. Load Zephyr-7B model for text generation - Initializes an instruction-tuned LLM for policy drafting.

```python
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline

# Zephyr-7B (Open Access, No Token Needed)
model_id = "HuggingFaceH4/zephyr-7b-beta"

# Load tokenizer and model
tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(
    model_id,
    device_map="auto",
    torch_dtype="auto"
)

# Create text generation pipeline
generator = pipeline("text-generation", model=model, tokenizer=tokenizer)
```

33. Build place-wise natural-language prompts from forecast - Converts data signals into prompt text per location.

```python
import pandas as pd
from collections import Counter
import random

# Identify waste and metadata columns
non_waste_cols = ['datetime', 'place_type', 'time', 'season', 'day', 'image_id']
waste_cols = [col for col in df_forecast_week.columns if col not in non_waste_cols]

# Function to extract a summary prompt for each place_type
def generate_summary_query(group_df, place):
    # Aggregate waste type counts
    waste_counts = {}
    for col in waste_cols:
        total = group_df[col].sum()
        if total > 0:
            waste_name = col.replace("waste_types_", "").replace("_", " ").title()
            waste_counts[waste_name] = total

    # Sort waste types by frequency
    sorted_waste = sorted(waste_counts.items(), key=lambda x: x[1], reverse=True)
    top_waste = [w[0] for w in sorted_waste[:4]]  # Most common

    # Randomly select 5 from the rest
    rare_pool = [w[0] for w in sorted_waste[4:]]
    rare_waste = random.sample(rare_pool, min(5, len(rare_pool))) if rare_pool else []

    # Time and day analysis
    days = group_df['day'].value_counts().index.tolist()
    seasons = group_df['season'].value_counts().index.tolist()
```

Section 7.2: Policy Generation using Zephyr

34. Generate policy drafts using Zephyr-7B from prompts – Produces tailored policy suggestions for each place.

```python
# Dictionary to store results
generated_policies = {}

# Generate response
for place, prompt in grouped_queries.items():

  response = generator(
      prompt,
      max_new_tokens=1024,
      do_sample=True,
      temperature=0.7,
      top_p=0.9,
      pad_token_id=tokenizer.eos_token_id
  )

  # Print result
  print("Generated Policy:\n", response[0]["generated_text"])

  # Store the full generated text for post-processing
  generated_policies[place] = response[0]["generated_text"]
```

35. Extract and list policy bullets from LLM outputs – Parses raw generations into clean bullet points.

```python
import re

# Format and display policies
for place, full_text in generated_policies.items():
    print(f"\n=== Generated Policy for {place.upper()} ===\n")

    # Remove the prompt from the response if repeated
    prompt = grouped_queries[place]
    if full_text.startswith(prompt):
        generated_only = full_text[len(prompt):].strip()
    else:
        generated_only = full_text.replace(prompt, "", 1).strip()

    # Regex to match "1. Policy text" or "1) Policy text" where number is 1-2 digits
    pattern = re.compile(r'^\s*\d{1,2}[.)]\s+(.*?)(?=\n\s*\d{1,2}[.)]|\Z)', re.DOTALL | re.MULTILINE)
    policies = [p.strip() for p in pattern.findall(generated_only)]

    # Limit to first 10 policies max
    policies = policies[:10]

    # Print each policy without numbering
    for pol in policies:
        print(pol)
```

36. Define baseline policy library for reuse - Curates re-usable best-practice policy templates.

```
baseline_policies = [
    "Install and maintain color-coded segregated bins (general, recyclables, glass) in high-footfall public areas.",
    "Conduct public awareness campaigns and educational initiatives in schools and communities to reduce litter and promote recycling.",
    "Enforce weekly street cleaning schedules tailored for residential, commercial, and tourist-heavy zones.",
    "Deploy litter wardens to patrol public areas, issue fines, and promote anti-littering behavior.",
    "Plan waste management in advance for public events like festivals or weddings, with temporary bins and post-event cleanup.",
    "Support local clean-up days and Tidy Towns initiatives by providing supplies and logistical assistance.",
    "Enforce waste presentation standards for businesses and encourage them to reduce packaging and improve recycling practices.",
    "Install solar-powered smart bins with fill-level sensors to optimize waste collection efficiency.",
    "Maintain rapid response units to deal with illegal dumping, bulky waste, and graffiti issues.",
    "Promote environmental education through school-based green programs focused on waste separation and litter awareness."
]
```

Section 7.3: Dynamic Scoring of the top Generated Suggestive Policies

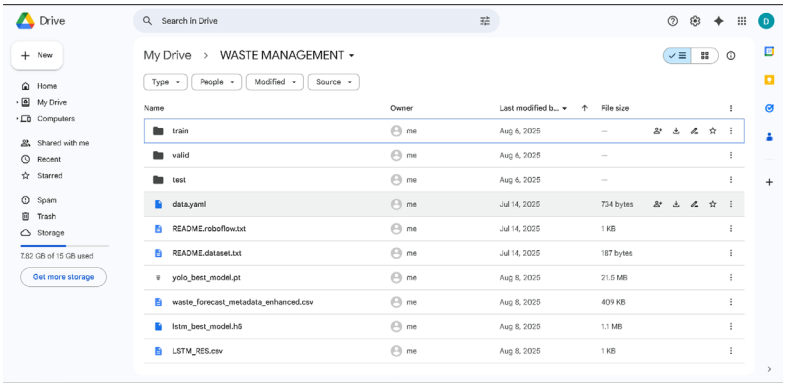
37. Score policies with SBERT and NLP heuristics – Rates drafts on relevance, clarity, feasibility, innovation, etc.

```
# Load enhanced SBERT and spacy models
sbert = SentenceTransformer("all-mpnet-base-v2")
nlp = spacy.load("en_core_web_sm")

# Load Data
df = pd.read_csv("/content/drive/MyDrive/WASTE MANAGEMENT/LSTM_RES.csv")
non_waste_cols = ['datetime', 'place_type', 'time', 'season', 'day', 'image_id']  # Removed 'occasion'
waste_cols = [col for col in df.columns if col not in non_waste_cols]

# Extract contextual details
place_info = {}
for place, group in df.groupby("place_type"):
    common_items = group[waste_cols].sum().sort_values(ascending=False)
    top_items = [col.replace("waste_types_", "").replace("_", " ").lower() for col in common_items.head(4).index]
    rare_items = [col.replace("waste_types_", "").replace("_", " ").lower() for col in common_items.tail(3).index]
    context = {
        "season": group["season"].mode()[0].lower(),
        "day": group["day"].mode()[0].lower(),
        "place": place.lower()
    }

    detailed_context_description = (
        f"In a {context['place']} environment, waste generation is typically higher in the "
        f"{context['season']} season, especially on {context['day']}s. "
        f"Common waste includes {', '.join(top_items)}, while rare but present items include {', '.join(rare_items)}."
    )
```

38. Select top 3 policies per place - Filters to the best candidates per location for decisioning.

```
# Get top 3 policies for each place by final score
top3_per_place = (
    score_df
    .sort_values(["place", "final_score"], ascending=[True, False])
    .groupby("place")
    .head(3)
)

# Display the result
display(top3_per_place)
```

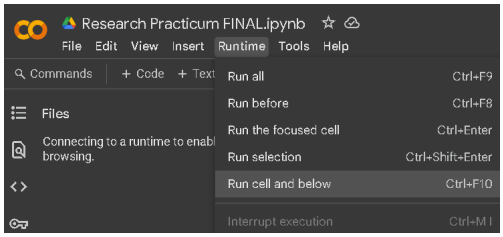
39. In the end the Google Drive WASTE MANAGEMENT folder should look like this.



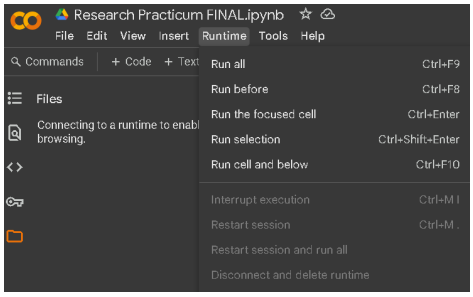
ERROR HANDLING (IN CASE OF OCCURANCE)

Section 8: Handling Google Colab runtime issues.

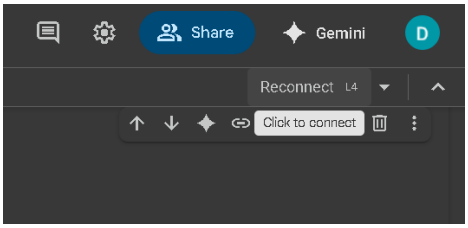
Error 1: When the training code for the YOLOv8s is executed (Step 14), sometimes during runtime Google Colab, Disconnects and Reconnects the session automatically. If this happens, we just have to execute that cell and the following cells one by one, OR select that cell, click on 'Runtime'; select 'Run cell and below' option, which will continue the execution of the rest of the cells automatically. This will fix that runtime issue and run the code normally.



Error 2: Upon execution of the LSTM training code, similar to Error 1, the cell might fail and show and error something along the line of CUDA not detected. If it happens, then click on 'Runtime'; click on 'Disconnect and delete runtime'.



Once the runtime is disconnected, then connect the runtime back again by clicking 'Reconnect L4'.



Once reconnected, select the cell in Step 20, and click on 'Runtime'; select 'Run cell and below' option, which will continue the execution of the rest of the cells automatically. This will fix that runtime issue and run the code normally.

Error 3: Upon executing the code in Step 31, it can fail to install Transformers, Accelerate, and bitsandbytes. So, in such cases follow the same steps as mentioned in Error 2 but select the cell in Step 30 and and click on 'Runtime'; select 'Run cell and below' option, which will continue the execution of the rest of the cells automatically. This will fix that runtime issue and run the code normally.