



L-Università  
ta' Malta

# Generative AI Usage Journal

Name x

Name x

Name x

Jamal Muhsen

ICS5110: Applied Machine Learning

By Dr Dylan Seychell

Due 31st of January 2026

## Using Claude Sonnet 4.5 to suggest the best ways to extract Maltese localities from unstructured text

If I want to extract Maltese city/village names from unstructured text, what is the best way to do it - using a list of localities and then matching to the text or using one of the NLP packages in python such as regex, spacy or nltk. Explain why

Based on your traffic accident project, I'll compare both approaches for extracting Maltese locality names from unstructured text.

### Recommended Approach: Hybrid Method (Gazetteer + NLP)

The best solution combines a curated list of Maltese localities with pattern-matching techniques. Here's why each pure approach has limitations:

#### Approach 1: Gazetteer-Based Matching (List-Based)

How it works:

```
python
# Example implementation
maltese_localities = [
    'Valletta', 'Sliema', 'St. Julian', 'Birkirkara',
    'Mosta', 'Qormi', 'Żabbar', 'San Ĝwara', 'Hamrun',
```

J

```
def extract_localities_simple(text):
    """Simple substring matching"""
    found = []
    text_lower = text.lower()
    for locality in maltese_localities:
        if locality.lower() in text_lower:
            found.append(locality)
    return found
```

Advantages:

- **High precision:** Only returns known Maltese localities
- **Simple to implement:** No complex dependencies
- **Fast execution:** O(n) complexity for matching
- **Malta-specific:** Handles unique Maltese orthography (ċ, ġ, h, ž)

Building regex to extract location from unstructured text. For this step Claude Sonnet 4.5 was mainly used to get an initial structure of the code and then used for bug fixes and iterative changes to improve extraction accuracy.

```

text_lower = text.lower()

# First, check for exact matches of localities
for locality in self.localities:
    # Create case-insensitive pattern
    # \b ensures we match whole words, not parts of words
    pattern = r'\b' + re.escape(locality.lower()) + r'\b'

    if re.search(pattern, text_lower):
        return locality

# Check for variations
for variation, standard_name in self.locality_variations.items():
    pattern = r'\b' + re.escape(variation.lower()) + r'\b'
    if re.search(pattern, text_lower):
        return standard_name

return None

# Initialize the extractor
location_extractor = LocationExtractor()

```

# Test on a single example first  
`test_text = police_df['content'].iloc[0]`



I know this is not complete but why isn't it working?

```

# Creating a function to extract location from the content of the
police/news article datasets by comparing it to df_malta_cities
dataset.
# Then extracting the location coordinates from the cities
dataset
def extract_city_and_coordinates(text_df,cities_df):

```

Show more

### Summary of fixes:

- Line 17: `row = ["content"]` → `row["content"]` (was assigning instead of accessing)
- Line 33: `r'/\b'` → `r'\b'` (backslash for regex, not forward slash)
- Line 41-56: The `if found_city:` block was indented inside the `for city` loop. It should be OUTSIDE the loop so it only runs after checking all cities.
- Line 49: `"lgn"` → `"lng"` (typo in column name)
- Line 56: Fixed quote escaping in f-string
- Line 58: Added `return pd.DataFrame(matched_results)` to actually return the results

The main issue was Fix #3 - the indentation of the `if found_city:` block. Because it was inside the loop, it was checking after every single city instead of after checking all cities. This would have caused it to either miss results or behave incorrectly.

```
# Creating a function to extract location from the content of the
police/news article datasets by comparing it to df_malta_cities
dataset.
# Then extracting the location coordinates from the cities
dataset
def extract_city_and_coordinates(text_df,cities_df):

    # Sort the list of cities by length. This is important so that
```

Show more

Great improvements! You've added tracking for unmatched results. However, there are a few issues to fix:

#### Key fixes:

1. **Line 71:** Changed from `extracted_df = extract_city_and_coordinates(...)` to `extracted_df, unmatched_df = extract_city_and_coordinates(...)`
  - Your function returns TWO values (a tuple), so you need to unpack them into two variables
  - Before, `extracted_df` was getting the tuple `(final_result, unfinished_result)` instead of just the DataFrame
2. **Line 61:** Fixed the f-string quotes: `{row["title"]}` → `{row['title']}`
3. **Added useful output** to show both matched and unmatched results

Now when you run this, you'll get:

- `extracted_df`: DataFrame with all records that had a location found
- `unmatched_df`: DataFrame with all records where no location was found

This is perfect for debugging - you can look at the unmatched results to see if you need to add more cities or adjust your matching logic!

does this make sense?

```
# Creating helper functions

def sort_cities (cities_df, column_name):
    """
        Function: Returns the column with cities as a list starting
        from longest to shortest.
```

Show more

### Main Issues Fixed:

1. **Indentation Error (Line 23):** Your `search_for_city` function was indented inside `sort_cities`. Each function should be at the same level.
2. **Undefined Variables:**
  - You used `lowercase_text` but never defined it → Changed to `text_lower`
  - You used `found_city` before defining it → Initialize it as `None`
3. **Function Does Too Much:** Your `search_for_city` function was:
  - Searching for cities ✓
  - Getting city info from database X
  - Appending to results lists X
  - Using variables from outer scope (`row`, `idx`, `matched_results`) XFunctions should do ONE thing! Just search and return the city name.
4. **Missing Variables:** The function tried to use `cities_df`, `row`, `idx`, `matched_results`, `unmatched_results` which don't exist inside the function.

### Key Principle: Single Responsibility

Each function should have ONE clear job:

- `sort_cities()` → Sort a column
- `search_for_city()` → Find a city in text ↓

not have permit\*

in this case, why didn't it find st Paul's bay in the title? This name is in the city\_other\_name column

```

# Convert text to lowercase and normalize abbreviations
lowercase_text = text.lower()

# Normalize common variations BEFORE searching
# This handles: "St Paul's" vs "St. Paul's" vs "Saint Paul's"
lowercase_text = lowercase_text.replace('st.', 'st') # Remove periods from St.
lowercase_text = lowercase_text.replace('saint ', 'st ') # Saint → St

found_city = None # Initialising found city variable

for city in city_list:
    # Also normalize the city name for comparison
    city_normalized = city.lower().replace('st.', 'st').replace('saint ', 'st ')

    # Create a search algorithm pattern
    pattern = r'(?<!from )\b' + re.escape(city_normalized) + r'\b'

    # Searches for the above pattern from the lowercase text
    if re.search(pattern, lowercase_text):
        found_city = city # Return original city name (not normalized)
        break # Stops as soon as it finds the first city

return found_city # Returns found city

```

#### Explanation of changes:

- Line 18: Remove periods: "St." → "St"
- Line 19: Normalize Saint to St: "Saint Paul's" → "St Paul's"
- Line 24: Apply same normalization to city names from database
- Now all these will match: "St Paul's Bay", ↓ paul's Bay", "Saint Paul's Bay"

#### Claude Sonnet 4.5 used to recommend API for historical weather retrieval

what package is best for historical weather data retrieval in Malta given that I have the date and coordinates for the location

### Recommendation: Open-Meteo is the Best Choice for Your Project

For your ICS5110 Applied Machine Learning project on traffic accidents in Malta, **Open-Meteo's Historical Weather API** is the optimal choice. Here's a detailed comparison and explanation:

#### Comparison of Available Options

Feature	Open-Meteo	Meteostat	Visual Crossing
API Key Required	✗ No	✗ No	<input checked="" type="checkbox"/> Yes (free tier available)
Cost	Free for non-commercial use	Free (CC BY-NC 4.0)	1000 free requests/day
Data Source	ERA5/ECMWF reanalysis models	Weather station observations	Multiple sources
Coverage for Malta	<input checked="" type="checkbox"/> Complete (global grid-based)	⚠ Limited (depends on station proximity)	<input checked="" type="checkbox"/> Complete
Missing Data	<input checked="" type="checkbox"/> None (model-based)	⚠ Possible gaps	<input checked="" type="checkbox"/> Filled
Historical Range	1940–present	Variable by station	50+ years
Spatial Resolution	9–25 km	Point-based (station)	Variable
Academic Citation	<input checked="" type="checkbox"/> Zenodo DOI available	⚠ No formal citation	⚠ Commercial
Python Package	<a href="#">openmeteo-requests</a>		API calls via <a href="#">requests</a>

### Claude Sonnet 4.5 used for an initial random forest hyperparameter grid

I am creating parameters for my random forest algorithm. My dataset has the following dimensions:

Dimensions for training dataset are: (208, 34) Dimensions for training labels are: (208,) Dimensions for testing dataset are: (53, 34) Dimensions for training labels are: (53,)

What do you suggest should be my starting parameters from my parameter grid? I want to include 2-3 optional parameters for each of the following:  
max\_dept  
min\_samples\_split  
min\_samples\_leaf  
max\_features  
n\_estimators

explain why and suggest if I should include other parameters

Show less

## Rationale for Each Parameter

### 1. n\_estimators: [50, 100, 200]

Why these values:

- **50 trees:** Lower bound that still captures ensemble benefits
- **100 trees:** Often sufficient for small datasets [1]
- **200 trees:** Upper bound before diminishing returns

With 208 samples, you don't need 500+ trees. Research shows that performance plateaus around 100-200 trees for datasets of this size [2].

### 2. max\_depth: [5, 10, None]

Why these values:

- **5:** Shallow trees prevent overfitting (critical with only 208 samples)
- **10:** Moderate depth balances complexity and generalization
- **None:** Unrestricted growth (baseline comparison)

With your small training set, deep trees (15-20+ levels) will likely memorize noise rather than learn patterns [3].

### 3. min\_samples\_split: [2, 5, 10]

Why these values:

- **2:** Default, allows maximum tree growth
- **5:** Requires ~2.4% of training data for splits
- **10:** Conservative, requires ~4.8% of training data



Higher values act as regularization, preventing splits based on very few samples [4].

### 4. min\_samples\_leaf: [1, 2, 4]

Why these values:

- **1:** Default, no restriction
- **2:** Ensures each leaf has minimum support
- **4:** Stronger regularization (~2% of training data)

This parameter directly controls overfitting by preventing leaves with negligible samples [5].

### 5. max\_features: ['sqrt', 'log2', None]

Why these values:

- **'sqrt':** ~6 features ( $\sqrt{34} \approx 5.83$ ) - standard recommendation for classification
- **'log2':** ~5 features ( $\log_2 34 \approx 5.09$ ) - more restrictive, increases tree diversity
- **None:** All 34 features - reduces randomness but may decrease ensemble diversity

This controls the randomness/diversity tradeoff in your forest [6].

---

## Additional Parameters You Should Consider

**class\_weight: 'balanced' or {0: 1, 1: w}** CRITICAL

does this make sense?

```
# Defining parameters for random forest
# Refer to: https://scikit-
learn.org/stable/modules/generated/sklearn.ensemble.RandomForestC
lassifier.html

param_grid = {
    "bootstrap": [True, False] # If true, sub-sample size is
```

Show more

Your parameter grid is **mostly excellent**, but you're **missing one critical parameter** I strongly recommended. Here's my analysis:

### ✓ What You Got Right

1. `bootstrap` - Good addition! Testing this is valuable for small datasets
2. `n_estimators` - Perfect range [50, 100, 200]
3. `min_samples_split` - Excellent choices [2, 5, 10]
4. `min_samples_leaf` - Perfect [1, 2, 4]
5. `max_features` - Good choices ["sqrt", "log2", None]
6. `class_weight` - Smart to test both given our discussion
7. **Comments** - Very clear and helpful

### ⚠ Critical Missing Parameter: `max_depth`

You're missing `max_depth`, which I specifically recommended as **one of the most important regularization parameters** for preventing overfitting on your small dataset (208 samples).

#### Why This Matters

Without controlling `max_depth`, your trees can grow extremely deep (especially when `min_samples_leaf=1` and `min_samples_split=2`), leading to:

- **Overfitting** on training data
- **Poor generalization** to test data
- **Memorization** of noise rather than learning patterns

Claude Sonnet 4.5 used to suggest ways to improve model performance

Improvement	Expected Gain	Your Priority
Threshold optimization (0.5→0.4)	+15-25% recall	✓ DONE (74%→94%)
Fix parameter grid (remove max_depth=None)	+5-10% accuracy	🔥 HIGH
Remove data leakage	Real operational performance	🔥 CRITICAL
Imputation (vs dropna)	+5-8% recall	✓ DONE
Temporal risk features	+2-5% recall	💡 RECOMMENDED
Optimize for recall (not accuracy)	+10-15% recall	🔥 HIGH
Remove zero-variance features	Cleaner model	✓ DONE