# Step 1: Load Libraries

**Download necessary NLTK data files for tokenisation and stopwords**

nltk.download('punkt')

nltk.download('stopwords')

**- Imports necessary libraries for data manipulation, natural language processing, machine learning, and visualisation.**

**- Downloads necessary NLTK data files for tokenisation and stopwords**

# Step 2: Load the dataset from CSV files in the raw data path

raw\_data\_path = "/Users/lanahumphrys/Documents/Project1/ForkNatNew/data/raw"

processed\_data\_path = "/Users/lanahumphrys/Documents/Project1/ForkNatNew/data/processed"

files = [os.path.join(raw\_data\_path, file) for file in os.listdir(raw data\_path) if file ends with('.csv')]

data\_frames = []

for file in files:

try:

df = pd.read\_csv(file, on\_bad\_lines='skip', delimiter=',')

**Adjust delimiter if necessary**

data\_frames.append(df)

except pd.errors.ParserError as e:

print(f"Error parsing {file}: {e}")

data = pd.concat(data\_frames, ignore\_index=True)

**- Defines the paths to the raw and processed data.**

**- Loads the dataset from CSV files in the raw data path.**

**- Concatenates all data frames into a single data frame.**

# Step 3: Preprocess the text data and include additional features like category and tone

if 'category' not in data.columns:

data['category'] =

if 'tone' not in data.columns:

data['tone'] = ""

def preprocess\_text(text):

if isinstance(text, str):

text = text.lower()

text = re.sub(r'\\d+', '', text)

text = re.sub(r'\\s+', ' ', text).strip()

else:

text = ""

return text

data['text'] = data['text'].apply(preprocess\_text)

data['category'] = data['category'].apply(preprocess\_text)

data['tone'] = data['tone'].apply(preprocess\_text)

**- Checks if 'category' and 'tone' columns exist, if not create them with empty strings.**

**- Defines a function to preprocess text by converting to lowercase, removing digits, and extra spaces.**

**- Applies preprocessing to relevant columns.**

# Step 4: Load the spaCy model for named entity recognition (NER) and annotate the text data using NER

nlp = spacy.load('en\_core\_web\_sm')

data['annotations'] = data['text'].apply(lambda x: [(ent.text, ent.label\_) for ent in nlp(x).ents])

**Explanation:**

**- Loads the spaCy model for named entity recognition (NER).**

**- Annotates the text data using NER by extracting entities and their labels**.

# Step 5: Exploratory Data Analysis (EDA)

def plot\_class\_distribution(data):

fig = px.histogram(data, x='category', title='Class Distribution')

fig.show()

if 'category' in data.columns:

plot\_class\_distribution(data)

**Explanation:**

**- Defines a function to plot class distribution using Plotly Express.**

**- Checks if 'category' column is present in the data and plots class distribution.**

**A blue rectangle with white text

AI-generated content may be incorrect.**

# Step 6: Model Selection and Design

data['clean\_text'] = data['clean\_text'].fillna("")

def assign\_category(text):

if 'contract' in text:

return 'Contract'

elif 'royalty' in text:

return 'Royalty'

elif 'editor' in text:

return 'Editor'

elif 'impact factor' in text:

return 'Impact Factor'

elif 'meeting' in text:

return 'Meeting'

else:

return 'Other'

data['category'] = data['text'].apply(assign\_category)

vectorizer = TfidfVectorizer(max\_features=1000)

X = vectorizer.fit\_transform(data['clean\_text'])

y = data['category']

unique\_categories = y.unique()

print(f"Unique categories in 'category' column: {unique\_categories}")

if len(unique\_categories) < 2:

raise ValueError("The 'category' column needs to have at least two unique classes for model training.")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**Explanation:**

**- Replaces NaN values with empty strings in 'clean\_text' column to avoid ValueError.**

**- Defines a rule-based categorisation function to assign categories based on keywords.**

**- Applies categorisation to the 'text' column.**

**- Uses TfidfVectorizer to convert text data into numerical features.**

**- Checks unique values in the 'category' column and ensures there are at least two unique classes.**

**- Splits the data into training and testing sets.**

A screenshot of a computer

AI-generated content may be incorrect.

# Step 7: Hyperparameter Tuning and Model Training

models = {

'RandomForest': RandomForestClassifier(n\_estimators=100, random\_state=42),

'LogisticRegression': LogisticRegression(random\_state=42),

'SVM': SVC(probability=True, random\_state=42),

'GradientBoosting': GradientBoostingClassifier(random\_state=42)

}

param\_grid = {

'RandomForest': {'n\_estimators': [50, 100, 200]},

'LogisticRegression': {'C': [0.1, 1, 10]},

'SVM': {'C': [0.1, 1, 10]},

'GradientBoosting': {'n\_estimators': [50, 100, 200]}

}

best\_models = {}

for model\_name, model in models.items():

grid\_search = GridSearchCV(model, param\_grid[model\_name], cv=5, scoring='accuracy')

grid\_search.fit(X\_train, y\_train)

best\_models[model\_name] = grid\_search.best\_estimator\_

**- Defines a dictionary of models to be trained.**

**- Defines a parameter grid for hyperparameter tuning.**

**- Performs grid search with cross-validation to find the best model for each algorithm.**

# Step 8: Model Evaluation with adjusted precision metric

for model\_name, model in best\_models.items():

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

print(f"Model: {model\_name}")

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred)}")

print(f"Precision: {precision\_score(y\_test, y\_pred, average='weighted')}")

print(f"Recall: {recall\_score(y\_test, y\_pred, average='weighted')}")

print(f"F1-Score: {f1\_score(y\_test, y\_pred, average='weighted')}")

if hasattr(model, "predict\_proba"):

try:

roc\_auc = roc\_auc\_score(y\_test, model.predict\_proba(X\_test), multi\_class='ovr', average='weighted')

print(f"ROC-AUC: {roc\_auc}")

except ValueError as e:

print(f"ROC-AUC calculation error: {e}")

print(classification\_report(y\_test, y\_pred))

**- Evaluates the best models using the test data.**

**- Prints accuracy, precision, recall, F1-score, and ROC-AUC for each model.**

# Step 9: Visualise Classification Results

def plot\_classification\_results(y\_test, y\_pred):

plt.figure(figsize=(10, 6))

plt.hist([y\_test, y\_pred], label=['True Labels', 'Predicted Labels'], bins=10, alpha=0.7)

plt.legend(loc='upper right')

plt.title('Classification Results')

plt.show()

plot\_classification\_results(y\_test, best\_models['RandomForest'].predict(X\_test))

Explanation:

**- Defines a function to plot classification results using histograms.**

**- Plots the classification results for one of the models (e.g., RandomForest).**

A screenshot of a computer

AI-generated content may be incorrect.

# Step 10: Email Template Generation (Example)

def generate\_email\_template(subject, body\_text):

template = f"Subject: {subject}\\n\\n{body\_text}"

return template

subject\_example = "Meeting Request"

body\_text\_example = "Dear [Recipient],\\n\\nI would like to schedule a meeting to discuss our upcoming project.\\n\\nBest regards,\\n[Your Name]"

email\_template\_example = generate\_email\_template(subject\_example, body\_text\_example)

print(email\_template\_example)

**Explanation:**

**- Defines a function to generate email templates.**

**- Provides an example usage of email template generation.**

**A black screen with white text

AI-generated content may be incorrect.**

**Rule-Based Categorisation**

**Defining Categories:** Identify the categories that we would like to assign to the text data. For example, in the context of email responses, categories might include "Contract," "Royalty," "Editor," "Impact Factor," "Meeting," and "Other."

**Creating Rules:** Developing a set of rules that map specific keywords or phrases to each category. These rules are based on the presence of certain words or patterns in the text.

**Assigning Categories:** Applying the rules to the text data to assign categories. If a text contains a keyword or phrase that matches a rule, it is assigned the corresponding category.

**Example of Rule-Based Categorisation**

def assign\_category(text):

if 'contract' in text:

return 'Contract'

elif 'royalty' in text:

return 'Royalty'

elif 'editor' in text:

return 'Editor'

elif 'impact factor' in text:

return 'Impact Factor'

elif 'meeting' in text:

return 'Meeting'

else:

return 'Other'

**Explanation of the Function**

* **Function Definition**: The function assign\_category takes a single argument text, which is the text data to be categorised.
* **Keyword Matching:** The function checks if specific keywords are present in the text using the in operator. The keywords are chosen based on their relevance to the categories:
  + **If the text contains the keyword 'contract', it is assigned the category 'Contract'.**
  + **If the text contains the keyword 'royalty', it is assigned the category 'Royalty'.**
  + **If the text contains the keyword 'editor', it is assigned the category 'Editor'.**
  + **If the text contains the keyword 'impact factor', it is assigned the category 'Impact Factor'.**
  + **If the text contains the keyword 'meeting', it is assigned the category 'Meeting'.**
* **Default Category**: If none of the keywords match, the text is assigned the default category 'Other'.

**Applying the Function**

The function is applied to the text data using the apply method in pandas:

data['category'] = data['text'].apply(assign\_category)

This line of code applies the assign\_category function to each element in the text column of the data DataFrame and stores the resulting categories in the category column.

**Disadvantages**

* **Limited Flexibility**: May not handle complex or nuanced text data well.
* **Scalability:** Difficult to scale for large datasets with diverse categories.
* **Accuracy:** May not be as accurate as machine learning-based approaches, especially for ambiguous or context-dependent text.

Rule-based categorisation may not be suitable for more complex or large-scale text classification problems. For more advanced categorisation labelled data can provide better accuracy and flexibility.