Your Kaggle Notebook — Code Explained (with Snippets)

Use this as a handout during the viva. Each section has (1) what it does, (2) how to explain it in one line, and (3) exact code you wrote.

---

0) High-level pipeline (say this first)

One-liner: "Load labels → build/clean contexts from XML → TF-IDF features → Logistic Regression (class-balanced) → predict mentions in test XML → submission.csv."

---

1) Setup & imports

What it does: Brings in libraries for data, XML parsing, features, model, and split.

```
import os
import re
import pandas as pd
import xml.etree.ElementTree as ET
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

How to say it: "Pandas for CSV, ElementTree for XML, TF-IDF for text features, Logistic Regression for classification."

---

2) Load the competition labels

What it does: Reads Kaggle's training labels (article_id, dataset_id, type).

```
data_path = "/kaggle/input/make-data-count-finding-data-references"
df = pd.read_csv(f"{data_path}/train_labels.csv")
```

How to say it: "This is the official train_labels.csv from the competition."

---

3) (Option A) Fast baseline using dataset_id text as proxy context

What it does: Simple baseline that vectorizes the dataset_id string and learns a classifier.

```
df = df.dropna()
X = df["dataset_id"].astype(str)
y = df["type"]

vectorizer = TfidfVectorizer(stop_words="english", max_features=3000,
ngram_range=(1, 2))
X_vec = vectorizer.fit_transform(X)

model = LogisticRegression(max_iter=200, class_weight='balanced')
model.fit(X_vec, y)
```

How to say it: "Quick, reproducible baseline: TF-IDF (1–2 grams, 3k features) + multinomial Logistic Regression with class balancing."

---

4) (Option B) Context-aware training set from XML

What it does: Scans training XML files, finds where a labeled DOI appears, and stores the sentence as context + label.

```
def build_training_data(df, xml_folder):
    import xml.etree.ElementTree as ET
    dataset = []

    for idx, row in df.iterrows():
        article_id = row['article_id']
        dataset_id = row['dataset_id']
        label = row['type']

        if dataset_id == "Missing" or label == "Missing":
            continue

        clean_doi = dataset_id.replace("https://doi.org/", "").lower()
        xml_path = f"{xml_folder}/{article_id}.xml"

        try:
            tree = ET.parse(xml_path)
            root = tree.getroot()
            for elem in root.iter():
                if elem.text and clean_doi in elem.text.lower():
                    context = elem.text.strip()
```

```
                    dataset.append({
                        "article_id": article_id,
                        "context": context,
                        "dataset_id": dataset_id,
                        "label": label
                    })
            except:
                continue

    return pd.DataFrame(dataset)
```

How to say it: "I locate labeled DOIs inside XML text and capture the surrounding sentence as training context."

---

5) Keep only informative contexts

What it does: Drops very short sentences (noise like bare IDs).

```
train_data_cleaned = train_data[train_data['context'].str.split().str.len() >= 5]
```

How to say it: "<5 tokens is usually non-informative; filtering improves signal."

---

6) Vectorize contexts and split

What it does: TF-IDF feature extraction and 80/20 split with a fixed seed.

```
X_texts = train_data_cleaned['context']
y_labels = train_data_cleaned['label']

vectorizer = TfidfVectorizer(stop_words='english', max_features=3000)
X = vectorizer.fit_transform(X_texts)

X_train, X_test, y_train, y_test = train_test_split(
    X, y_labels, test_size=0.2, random_state=42
)
```

How to say it: "Same TF-IDF idea, but now on real contexts; 80/20 split for validation."

---

7) Train the classifier (balanced)

What it does: Fits Logistic Regression; the balanced option handles class imbalance.

```
model = LogisticRegression(max_iter=200, class_weight='balanced')
model.fit(X_train, y_train)
```

How to say it: "Balanced weighting prevents the model from ignoring rare 'Primary' cases."

---

8) Evaluate with macro metrics

What it does: Reports precision/recall/F1 with all classes weighted equally (macro).

```
from sklearn.metrics import classification_report

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

How to say it: "Macro F1 treats each class fairly, which matters under imbalance."

---

9) Mention extraction for test XMLs

What it does: Uses regex to find DOI/accession-like strings in each test XML file.

```
def extract_mentions(xml_file):
    doi_pattern = re.compile(r'(10\.\d{4,9}/[^\s";]+)', re.IGNORECASE)
    acc_pattern = re.compile(r'\b(GSE\d+|E-\w+-\d+|PRJ\w+\d+|CHEMBL\d+|PDB\s+
\w+)\b', re.IGNORECASE)
    mentions = set()
    try:
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for elem in root.iter():
            if elem.text:
                text = elem.text.strip()
                mentions.update(doi_pattern.findall(text))
                mentions.update(acc_pattern.findall(text))
    except:
        pass
    return mentions
```

How to say it: “Regex finds likely dataset IDs; we’ll classify each mention.”

---

10) Predict and build `submission.csv`

What it does: Loops through test XMLs, predicts a label for every mention, and writes the file Kaggle expects.

```
test_xml_dir = f"{data_path}/test/XML"
predictions = []
row_id = 0

for fname in os.listdir(test_xml_dir):
    if not fname.endswith(".xml"):
        continue
    article_id = fname.replace(".xml", "")
    mentions = extract_mentions(os.path.join(test_xml_dir, fname))
    for m in mentions:
        if m.startswith("10."):
            dataset_id = "https://doi.org/" + m.lower()
        else:
            dataset_id = m.upper().strip()
        context_text = m  # proxy context at inference
        X_test = vectorizer.transform([context_text])
        pred = model.predict(X_test)[0]
        predictions.append({
            "row_id": row_id,
            "article_id": article_id,
            "dataset_id": dataset_id,
            "type": pred
        })
        row_id += 1

submission = pd.DataFrame(predictions)
submission = submission.drop_duplicates(subset=["article_id", "dataset_id",
"type"])
submission.to_csv("submission.csv", index=False)
```

How to say it: “For each XML, extract mentions → vectorize → predict → deduplicate → save submission.csv.”

---

11) One-minute viva script

- **Competition:** Make Data Count – classify dataset mentions as Primary/Secondary/Missing.

- **Data:** train_labels.csv + article XMLs; I build sentence-level contexts around labeled DOIs.
- **Features & Model:** TF-IDF (1–2 grams, 3k features) + Logistic Regression with class_weight='balanced'.
- **Metrics:** Macro precision/recall/F1; ‘Missing’ is easiest, ‘Primary’ is hardest due to rarity and subtle wording.
- **Inference:** Regex-find DOIs/accessions in test XML, classify each mention, write submission.csv.
- **Improvements:** Larger context windows, better regex/NER, and fine-tuned SciBERT/BERT.

---

12) What to say if asked “why this design?”

- **Simplicity & speed:** Strong baseline that trains fast on Kaggle.
- **Interpretability:** Linear model on TF-IDF is easy to explain.
- **Class imbalance addressed:** `class_weight='balanced'`.
- **Clear upgrade path:** Drop-in replacement with transformer models later.

---

13) Quick glossary for beginners

- **TF-IDF:** Weighs words by how specific they are to a sentence vs common overall.
- **Logistic Regression (multiclass):** A linear classifier that outputs class probabilities.
- **Macro F1:** Average F1 across classes; treats them equally.
- **Context window:** The sentence (or nearby text) around a detected DOI/ID.