

التجهيز البيانات

تحميل الداتا سيت

```
# =====
# STEP 1: Load the Dataset
# الهدف:
# Google Drive تحميل ملف البيانات الخام من
#
# المخرجات:
# DataFrame df باسم يحتوي جميع الإيميلات
# =====

import os
import pandas as pd

from google.colab import drive
drive.mount('/content/drive')

file_path = "/content/drive/MyDrive/Thesis_Project/emails.csv"

df = pd.read_csv(file_path)

print("Dataset Shape:", df.shape)

Mounted at /content/drive
Dataset Shape: (517401, 2)

import pandas as pd

# Google إذا كنت تستخدم Google Drive لل Mount خطوة اختيارية: تأكد من عمل 1.
# Colab
# from google.colab import drive
# drive.mount('/content/drive')

# تحديد مسار ملف البيانات
file_path = "/content/drive/MyDrive/Thesis_Project/emails.csv"

try:
    # 2. CSV قراءة ملف ال
    df = pd.read_csv(file_path)
    print("✅ تم تحميل البيانات بنجاح\n")

    # 3. عرض الأسطر الخمسة الأولى لمعينة شكل البيانات والأعمدة
    print("🔍 (First 5 Rows): نظرة أولية على البيانات [1]")
    # لعرض منسق، أو display(df.head()) يفضل استخدام Colab إذا كنت تستخدم
    print(df.head())
    print(df.head())
```

```
print("-" * 50)
```

```
# 4. معرفة أبعاد الداتا سيت (عدد النصوص/الإيميلات وعدد الخصائص)
print(f"📊 [2] حجم الداتا سيت: {df.shape[0]} سطر (إيميل) و {df.shape[1]} أعمدة.")
print("-" * 50)
```

```
# 5. وحجم الذاكرة المستهلكة (Data Types) تفاصيل الأعمدة، أنواع البيانات
print("📄 [3] معلومات الأعمدة وأنواع البيانات:")
df.info()
print("-" * 50)
```

```
# 6. في كل عمود لضمان جودة البيانات (Missing Values) فحص القيم المفقودة
print("❌ [4] عدد القيم المفقودة في كل عمود:")
print(df.isnull().sum())
print("-" * 50)
```

```
# 7. ملخص إحصائي سريع للأعمدة النصية والرقمية (التكرار، القيم الفريدة، إلخ)
print("📈 [5] الملخص الإحصائي الشامل:")
print(df.describe(include='all'))
print("-" * 50)
```

```
# 8. فحص أسماء الأعمدة بشكل مباشر
print("📁 [6] أسماء الأعمدة المتوفرة في الملف:")
print(df.columns.tolist())
```

```
except FileNotFoundError:
```

```
    print("❌ خطأ: لم يتم العثور على الملف في المسار المحدد. تأكد من أن  
    وصلة المسار Google Drive تفعيل اتصال الـ")
```

```
except Exception as e:
```

```
    print(f"⚠️ {e}: حدث خطأ أثناء معالجة الملف")
```

✅ تم تحميل البيانات بنجاح!

🔍 (First 5 Rows): نظرة أولية على البيانات [1]:

```
file
```

```
message
```

```
0    allen-p/_sent_mail/1. Message-ID:
<18782981.1075855378110.JavaMail.e...
1    allen-p/_sent_mail/10. Message-ID:
<15464986.1075855378456.JavaMail.e...
2    allen-p/_sent_mail/100. Message-ID:
<24216240.1075855687451.JavaMail.e...
3    allen-p/_sent_mail/1000. Message-ID:
<13505866.1075863688222.JavaMail.e...
4    allen-p/_sent_mail/1001. Message-ID:
<30922949.1075863688243.JavaMail.e...
```

📊 [2] حجم الداتا سيت: 517401 سطر (إيميل) و 2 أعمدة.

```
📄 [3] معلومات الأعمدة وأنواع البيانات:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 517401 entries, 0 to 517400
Data columns (total 2 columns):
#   Column    Non-Null Count  Dtype
---  -
```

```
0 file 517401 non-null object
1 message 517401 non-null object
dtypes: object(2)
memory usage: 7.9+ MB
```

❌ [4] عدد القيم المفقودة في كل عمود:

```
file 0
message 0
dtype: int64
```

📊 [5] الملخص الإحصائي الشامل:

```
file \
count 517401
unique 517401
top zufferli-j/sent_items/99.
freq 1
```

```
message
count 517401
unique 517401
top Message-ID: <28618979.1075842030037.JavaMail.e...
freq 1
```

📁 [6] أسماء الأعمدة المتوفرة في الملف:

```
['file', 'message']
```

```
import pandas as pd
import email
```

1. قراءة ملف البيانات

```
file_path = "/content/drive/MyDrive/Thesis_Project/emails.csv"
df = pd.read_csv(file_path)
```

2. دالة ذكية وسريعة لاستخراج الموضوع ونص الإيميل فقط

```
def extract_clean_email(raw_message):
    msg = email.message_from_string(raw_message)

    # استخراج الموضوع (إذا لم يكن موجوداً نضع نصاً فارغاً)
    subject = msg['Subject'] if msg['Subject'] else ""

    # استخراج نص الرسالة (Body)
    body = ""
    if msg.is_multipart():
        for part in msg.walk():
            if part.get_content_type() == 'text/plain':
                payload = part.get_payload(decode=True)
                if payload:
                    body = payload.decode(errors='ignore')
                    break
    else:
        payload = msg.get_payload(decode=True)
        if payload:
            body = payload.decode(errors='ignore')

    return pd.Series([subject.strip(), body.strip()])
```

```

print("⌚ جاري استخراج البيانات ودمجها... يرجى الانتظار قليلاً بسبب حجم
...البيانات")

# نصيحة: إذا كنت تريد تجربة الكود فوراً وبسرعة فائقة، يمكنك إلغاء التعليق عن السطر التالي
# لتجربته على أول 2000 إيميل فقط:
# df = df.head(2000)

# تطبيق الدالة لإنشاء عمود الموضوع وعمود نص الإيميل
df[['Subject', 'Body']] = df['message'].apply(extract_clean_email)

# إنشاء العمود الثالث الذي يجمع (الموضوع + نص الإيميل) معاً
df['Subject_and_Body'] = df['Subject'] + " " + df['Body']

# الاحتفاظ بالأعمدة الثلاثة المطلوبة فقط وحذف الباقي
df_final = df[['Subject', 'Body', 'Subject_and_Body']]

print("✅ انتهت المعالجة بنجاح! تم تجهيز الأعمدة الثلاثة")

# طباعة عينات عشوائية للتحقق من جودة النتيجة
print("🔍 ===== عينات للتحقق الفوري =====")
# سحب 3 عينات عشوائية من البيانات كاملة
samples = df_final.sample(3)

for i, (idx, row) in enumerate(samples.iterrows(), 1):
    print(f"📄 السطر الأصلي في الملف - {i} العينة العشوائية رقم {idx}:")
    print(f"💎 (Subject):\n -> {row['Subject']} if
row['Subject'] else '[موضوع فارغ]')")
    print(f"💎 (Body - 150 حرف فقط):\n ->
{row['Body'][:150]}...")
    print(f"💎 (Subject_and_Body - 150 حرف فقط):\n
-> {row['Subject_and_Body'][:150]}...")
    print("-" * 60)

⌚ جاري استخراج البيانات ودمجها... يرجى الانتظار قليلاً بسبب حجم
...البيانات
✅ انتهت المعالجة بنجاح! تم تجهيز الأعمدة الثلاثة

🔍 ===== عينات للتحقق الفوري =====

📄 [العينة العشوائية رقم 1 - السطر الأصلي في الملف: 379182]:
💎 (Subject):
-> Fwd: FW: Fwd[2]:EMBARRASSING MOMENTS IN SPORTS,IT IS TOO FUNNY!!!!
💎 (Body - 150 حرف فقط):
-> From: "Danielle Miller" <dmill002@hotmail.com>
To: dgioffre@hotmail.com
Subject: Fwd: FW: Fwd[2]:EMBARRASSING MOMENTS IN SPORTS,IT IS TOO FUNNY!!!!
Da...
💎 (Subject_and_Body - 150 حرف فقط):
-> Fwd: FW: Fwd[2]:EMBARRASSING MOMENTS IN SPORTS,IT IS TOO FUNNY!!!!
From: "Danielle Miller" <dmill002@hotmail.com>
To: dgioffre@hotmail.com
Subject: Fw...
-----

📄 [العينة العشوائية رقم 2 - السطر الأصلي في الملف: 13510]:

```

- ♦ 1. عمود الموضوع (Subject):
-> Re: Jason' Bachelor Party
- ♦ 2. (أول 150 حرف فقط - Body) عمود نص الإيميل:
-> Okay. If you decide to use driver/van, I will pay for it. I guess the real problem is the driver/van would merely sit at @ the dock for 7 hours. But, ...
- ♦ 3. (أول 150 حرف فقط - Subject_and_Body) عمود الدمج سوا:
-> Re: Jason' Bachelor Party Okay. If you decide to use driver/van, I will pay for it. I guess the real problem is the driver/van would merely sit at @ t...

 [العيونة العشوائية رقم 3 - السطر الأصلي في الملف: 348590]:

- ♦ 1. عمود الموضوع (Subject):
-> Enfolio Gas Purchase template
- ♦ 2. (أول 150 حرف فقط - Body) عمود نص الإيميل:
-> Could one of you please email me a copy of the standard template you use for Enfolio gas purchase agreements? As we're getting more and more of those...
- ♦ 3. (أول 150 حرف فقط - Subject_and_Body) عمود الدمج سوا:
-> Enfolio Gas Purchase template Could one of you please email me a copy of the standard template you use for Enfolio gas purchase agreements? As we're ...

تنظيف الداتا سبت

```
import pandas as pd
import re
import numpy as np
```

قد تم تشغيلها بنجاح في df_final ملاحظة سريعة: تأكد أن خلية الكود السابقة التي أنتجت هذا الجلسة.

NameError سنقوم الآن بعملية التنظيف وحساب الإحصائيات معاً في نفس الخلية لتجنب الـ

```
# -----
# df_cleaned أولاً: مرحلة التنظيف وإنشاء المتغير
# -----
```

```
df_cleaned = df_final.copy()
```

```
print("⌚ جاري تنظيف النصوص وتصفية الشوائب (يستغرق لحظات بسبب حجم البيانات...)"
```

```
def advanced_clean_text(text):
    if not isinstance(text, str):
        return ""
```

```
# 1. تحويل النصوص للأحرف الصغيرة
text = text.lower()
```

```
# 2. إزالة روابط الإنترنت
text = re.sub(r'https?://\s*\S+|www\.\S+', '', text)
```

```
# 3. Forward Headers (From:, To:, إلخ) تنظيف مخلفات الـ
text = re.sub(r'(from|to|subject|cc|bcc|sent):.*', '', text)
```

```

# 4. إزالة الرموز الخاصة وعلامات الترقيم الزائدة
text = re.sub(r'^[a-zA-Z0-9\s]', ' ', text)

# 5. إزالة المسافات الزائدة
text = re.sub(r'\s+', ' ', text).strip()

return text

# تطبيق التابع لإنشاء عمود النص النقي
df_cleaned['Cleaned_Text'] =
    df_cleaned['Subject_and_Body'].apply(advanced_clean_text)

# ثم حذفها صارماً NaN تحويل النصوص التي أصبحت فارغة بعد التنظيف إلى
df_cleaned['Cleaned_Text'] = df_cleaned['Cleaned_Text'].replace(r'^\s*$',
    pd.NA, regex=True)
df_cleaned = df_cleaned.dropna(subset=['Cleaned_Text'])

print("✅ اكتملت عملية معالجة وتنظيف البيانات بنجاح")
print("📊 جاري حساب وتوليد التقرير الإحصائي المقارن...\n")

# -----
# ثانياً: حساب الإحصائيات المقارنة (قبل وبعد المعالجة)
# -----

# إحصائيات ما قبل المعالجة (على عمود الدمج الأصلي)
total_before = len(df_final)
missing_before = df_final['Subject_and_Body'].isna().sum() or
    (df_final['Subject_and_Body'] == '').sum()
words_per_email_before =
    df_final['Subject_and_Body'].astype(str).apply(lambda x:
        len(x.split()))
avg_words_before = words_per_email_before.mean()
max_words_before = words_per_email_before.max()

# إحصائيات ما بعد المعالجة (على العمود المنظف الجديد)
total_after = len(df_cleaned)
missing_after = df_cleaned['Cleaned_Text'].isna().sum()
words_per_email_after = df_cleaned['Cleaned_Text'].apply(lambda x:
    len(x.split()))
avg_words_after = words_per_email_after.mean()
max_words_after = words_per_email_after.max()

# حساب تتبع التعديل والسجلات المحذوفة ونسبتها
dropped_records = total_before - total_after
dropped_percentage = (dropped_records / total_before) * 100

# -----
# ثالثاً: طباعة تقرير جودة البيانات وسجل التتبع المتبقي
# -----
print("📊 (Data Profiling Report) تقرير إحصائيات البيانات =====")
print(f"| (Original) المقياس الإحصائي | قبل المعالجة |")
print(f"| (Cleaned) المعالجة |")

```

```

print(f" |-----|-----|-----|
      |-----|")
print(f" | إجمالي السجلات المتوفرة | {total_before:<23} | {total_after:
<23} |")
print(f" | السجلات الفارغة/المفقودة | {missing_before:<23} |
{missing_after:<23} |")
print(f" | متوسط عدد الكلمات في الإيميل | {avg_words_before:<23.2f} |
{avg_words_after:<23.2f} |")
print(f" | أقصى عدد كلمات في إيميل واحد | {max_words_before:<23} |
{max_words_after:<23} |")
print("=====

print(f"\n📊 سجل تتبع التعديل (Data Retention Log):")
print(f" - سجل {total_after} : عدد السجلات المتبقية (السجل الباقي) -")
print(f" - سجل {dropped_records} : عدد السجلات المحذوفة كلياً -")
print(f" - {dropped_percentage:.2f}% من حجم : نسبة البيانات المحذوفة -
      .الداتا سيت الكلية")
print("=====

```

...جاري تنظيف النصوص وتصفية الشوائب (يستغرق لحظات بسبب حجم البيانات)

✅ اكتملت عملية معالجة وتنظيف البيانات بنجاح

...جاري حساب وتوليد التقرير الإحصائي المقارن

```

📊 ===== تقرير إحصائيات البيانات (Data Profiling Report)
=====
| بعد المعالجة | (Original) المقياس الإحصائي | قبل المعالجة |
| (Cleaned) | | |
|-----|-----|-----| |
|---|---|---|---|
| 516755 | | 517401 | إجمالي السجلات المتوفرة |
| | | 0 | السجلات الفارغة/المفقودة |
| 267.64 | | 263.64 | متوسط عدد الكلمات في الإيميل |
| 191378 | | 60530 | أقصى عدد كلمات في إيميل واحد |
|
=====

```

```

📊 (Data Retention Log) سجل تتبع التعديل:
- عدد السجلات المتبقية (السجل الباقي): 516755 سجل
- عدد السجلات المحذوفة كلياً: 646 سجل
- نسبة البيانات المحذوفة: 0.12% من حجم الداتا سيت الكلية
=====

```

تحديد مسار حفظ الملف الجديد المنظف

```

output_path =
    "/content/drive/MyDrive/Thesis_Project/emails_cleaned_final.csv"

```

```

print(f"📁 جاري حفظ الداتا سيت المنظمة في ملف جديد")
# حفظ فقط عمود النص المنظف لتقليل حجم الملف وسرعة التعامل معه لاحقاً
df_cleaned[['Cleaned_Text']].to_csv(output_path, index=False)
print(f"✅ تم الحفظ بنجاح في المسار {output_path}")

```

...جاري حفظ الداتا سيت المنظمة في ملف جديد

✅ تم الحفظ بنجاح في المسار:

/content/drive/MyDrive/Thesis_Project/emails_cleaned_final.csv

اعداد القاموس الخاص بالتصنيفات والاولويات

```
import json
```

```
keywords_data = {  
    "categories": {  
        "Legal": {  
            "contract": 5,  
            "agreement": 5,  
            "legal": 6,  
            "lawsuit": 8,  
            "attorney": 7,  
            "compliance": 7,  
            "regulation": 6,  
            "court": 8,  
            "litigation": 8,  
            "hearing": 7,  
            "regulatory": 6,  
            "filing": 5,  
            "license": 5,  
            "dispute": 7,  
            "settlement": 7,  
            "claim": 6,  
            "investigation": 7  
        },  
  
        "Financial": {  
            "invoice": 6,  
            "budget": 5,  
            "payment": 6,  
            "revenue": 7,  
            "expense": 6,  
            "finance": 6,  
            "financial": 7,  
            "profit": 7,  
            "cost": 5,  
            "cash flow": 8,  
            "forecast": 6,  
            "pricing": 6,  
            "credit": 7,  
            "investment": 6,  
            "funding": 6,  
            "trading": 7,  
            "trade": 7,  
            "cash": 7,  
            "hedge": 8,  
            "commodity": 7,  
            "exposure": 8,  
            "margin": 8,  
            "margin plan": 8,  
            "million": 7,  
        }  
    }  
}
```

```
    "billion": 8,  
    "financing": 8,  
    "fund": 7,  
    "loan": 7,  
    "debt": 8,  
    "asset": 6,  
    "liability": 8,  
    "balance": 6,  
    "accounting": 8,  
    "audit": 8,  
    "tax": 7,  
    "earnings": 8  
  },
```

```
"HR": {  
  "employee": 7,  
  "employees": 7,  
  "recruitment": 8,  
  "hiring": 8,  
  "salary": 8,  
  "vacation": 5,  
  "leave": 6,  
  "benefits": 7,  
  "training": 6,  
  "interview": 8,  
  "candidate": 7,  
  "staff": 6,  
  "personnel": 7,  
  "promotion": 5,  
  "termination": 8,  
  "performance": 8,  
  "performance feedback": 9,  
  "performance management": 9,  
  "feedback": 7,  
  "reviewer": 7,  
  "review": 5,  
  "management process": 6,  
  "career": 7,  
  "development": 6,  
  "workforce": 7,  
  "responsibility": 6  
},
```

```
"Market": {  
  "customer": 6,  
  "sales": 7,  
  "marketing": 7,  
  "campaign": 7,  
  "advertising": 7,  
  "promotion": 6,  
  "market": 7,  
  "client": 6,
```

```
"consumer": 6,  
"business development": 7,  
"product": 5,  
"service": 5,  
"competition": 6,  
"branding": 7,  
"energy": 7,  
"electricity": 7,  
"power": 7,  
"generation": 7,  
"gas": 7,  
"pipeline": 7,  
"customer demand": 8,  
"supply": 7,  
"demand": 7,  
"capacity": 6,  
"construction": 7,  
"bid": 7,  
"proposal": 6,  
"vendor": 6,  
"supplier": 6,  
"contractor": 7,  
"project": 5  
},
```

```
"Administration": {  
  "meeting": 6,  
  "schedule": 6,  
  "memo": 7,  
  "administrative": 7,  
  "office": 5,  
  "procedure": 7,  
  "policy": 7,  
  "conference call": 6,  
  "agenda": 6,  
  "minutes": 6,  
  "coordination": 6,  
  "planning": 5,  
  "project": 5,  
  "update": 4,  
  "report": 5,  
  "call": 5,  
  "attached": 4,  
  "attachment": 4,  
  "forwarded": 3,  
  "please review": 5,  
  "presentation": 5,  
  "document": 5,  
  "discussion": 5,  
  "process": 6,  
  "request": 5,  
  "approval": 6  
}
```

```
},
```

```
"priorities": {
```

```
  "High": {
    "urgent": 10,
    "asap": 10,
    "immediately": 10,
    "critical": 9,
    "emergency": 9,
    "deadline": 9,
    "time sensitive": 8,
    "action required": 8,
    "response required": 8,
    "high priority": 10,
    "important": 7,
    "must": 7,
    "required": 7,
    "need today": 8,
    "need by": 7,
    "final notice": 8,
    "termination": 8,
    "credit risk": 9,
    "default": 9,
    "outage": 8,
    "power shortage": 8,
    "market risk": 8,
    "financial exposure": 9,
    "lawsuit": 9,
    "litigation": 9,
    "hearing": 8,
    "regulatory filing": 8,
    "bankruptcy": 10,
    "compliance issue": 8,
    "immediate attention": 10,
    "approve immediately": 9,
    "sign immediately": 9,
    "customer complaint": 8,
    "incident": 8,
    "safety issue": 9
  },
```

```
  "Normal": {
    "meeting": 4,
    "conference call": 4,
    "schedule": 4,
    "discussion": 4,
    "review": 4,
    "update": 4,
    "follow up": 4,
    "report": 4,
```

```
    "forecast": 4,  
    "budget": 4,  
    "planning": 4,  
    "project": 4,  
    "proposal": 4,  
    "memo": 4,  
    "agenda": 4,  
    "minutes": 4,  
    "analysis": 4,  
    "presentation": 4,  
    "contract": 5,  
    "agreement": 5,  
    "negotiation": 5,  
    "approval": 5,  
    "weekly": 3,  
    "monthly": 3,  
    "status": 4,  
    "coordination": 4,  
    "committee": 4,  
    "strategy": 4,  
    "evaluation": 4  
  },  
  
  "Low": {  
    "fyi": 8,  
    "for your information": 8,  
    "thank you": 7,  
    "thanks": 7,  
    "appreciate": 6,  
    "congratulations": 8,  
    "welcome": 6,  
    "vacation": 8,  
    "holiday": 8,  
    "party": 9,  
    "superbowl": 9,  
    "tickets": 8,  
    "lunch": 7,  
    "dinner": 7,  
    "travel specials": 10,  
    "discount": 9,  
    "special offer": 10,  
    "hotel": 8,  
    "airline": 8,  
    "car rental": 8,  
    "newsletter": 8,  
    "announcement": 6,  
    "social event": 8,  
    "golf": 8,  
    "sports": 7,  
    "birthday": 8,  
    "weekend": 7,  
    "personal": 8,  
    "joke": 9,  
  },  
}
```

```

        "entertainment": 8
    }
}
}

```

```
json_file_path = "/content/drive/MyDrive/Thesis_Project/keywords_config.json"
```

```

with open(json_file_path, "w", encoding="utf-8") as f:
    json.dump(
        keywords_data,
        f,
        indent=4,
        ensure_ascii=False
    )

```

```

print("✅ JSON Saved Successfully")
print(json_file_path)

```

```

✅ JSON Saved Successfully
/content/drive/MyDrive/Thesis_Project/keywords_config.json

```

وسم ملف الداتا بعد عملية التنظيف

```

# =====
# IMPORT LIBRARIES
# =====

```

```

import pandas as pd
import json
import re
import os

```

```
from sklearn.preprocessing import MultiLabelBinarizer
```

```

# =====
# PATHS
# =====

```

الملف الأصلي المنظف

```
DATA_PATH = "/content/drive/MyDrive/Thesis_Project/emails_cleaned_final.csv"
```

ملف الكلمات والأوزان

```
CONFIG_PATH = "/content/drive/MyDrive/Thesis_Project/keywords_config.json"
```

الملف الناتج بعد إعادة التصنيف

```

OUTPUT_PATH =
    "/content/drive/MyDrive/Thesis_Project/emails_labeled_final_multilabel.csv"

```

```

# =====
# DELETE OLD LABELED FILE

```

```

# =====

# حذف الملف القديم إذا كان موجود
# الهدف منع بقاء نتائج تصنيف قديمة

if os.path.exists(OUTPUT_PATH):
    os.remove(OUTPUT_PATH)
    print("Old labeled file deleted")

# =====
# LOAD CLEAN DATASET
# =====

df = pd.read_csv(DATA_PATH)

print("="*60)
print("ORIGINAL DATA")
print("="*60)

print("Shape:", df.shape)

print(df.head())

# =====
# LOAD JSON CONFIGURATION
# =====

with open(CONFIG_PATH, "r", encoding="utf-8") as f:
    config = json.load(f)

category_rules = config["categories"]

priority_rules = config["priorities"]

print("\nCategories:")
print(list(category_rules.keys()))

print("\nPriorities:")
print(list(priority_rules.keys()))

# =====
# TEXT COLUMN
# =====

TEXT_COL = "Cleaned_Text"

```

```
# =====  
# TEXT NORMALIZATION  
# =====
```

```
def clean_text(text):
```

```
    text = str(text).lower()
```

```
    # حذف الروابط
```

```
    text = re.sub(  
        r"http\S+",  
        " ",  
        text  
    )
```

```
    # الاحتفاظ بالحروف فقط
```

```
    text = re.sub(  
        r"[^a-z\s]",  
        " ",  
        text  
    )
```

```
    # إزالة المسافات الزائدة
```

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

```
    return text
```

```
# إنشاء نسخة للتصنيف فقط
```

```
df["temp_text"] = df[TEXT_COL].apply(clean_text)
```

```
# =====  
# WEIGHTED KEYWORD SCORING  
# =====
```

```
def compute_score(text, rules):
```

```
    scores = {}
```

```
    for category, keywords in rules.items():
```

```
        total_score = 0
```

```
        for keyword, weight in keywords.items():
```

```
            if keyword.lower() in text:
```

```
                total_score += weight
```

```

        scores[category] = total_score

    return scores

# =====
# CATEGORY LABELING
# =====

# أقل مجموع نقاط حتى يعتبر التصنيف موجود
CATEGORY_THRESHOLD = 5

def label_category(text):

    scores = compute_score(
        text,
        category_rules
    )

    labels = []

    for category, score in scores.items():

        if score >= CATEGORY_THRESHOLD:

            labels.append(category)

    # إذا لم يجد أي تصنيف
    if len(labels) == 0:

        labels.append("Other")

    return labels

# =====
# PRIORITY LABELING
# =====

def label_priority(text):

    scores = compute_score(
        text,
        priority_rules

```

```

    )

    best_priority = max(
        scores,
        key=scores.get
    )

    if scores[best_priority] > 0:

        return best_priority

    return "Normal"

# =====
# APPLY LABELING
# =====

print("\nStarting labeling...")

df["category_list"] = df["temp_text"].apply(
    label_category
)

df["priority"] = df["temp_text"].apply(
    label_priority
)

print("Labeling Completed")

# =====
# MULTI LABEL ENCODING
# =====

mlb = MultiLabelBinarizer()

category_matrix = mlb.fit_transform(
    df["category_list"]
)

category_df = pd.DataFrame(
    category_matrix,
    columns=[
        "category_" + c

```

```

        for c in mlb.classes_
    ]
)

print("\nGenerated Categories:")
print(category_df.columns.tolist())

# =====
# FINAL DATASET
# =====

final_df = pd.concat(

    [

        df[
            [
                TEXT_COL,
                "priority",
                "category_list"
            ]
        ],

        category_df

    ],

    axis=1

)

# =====
# STATISTICS
# =====

print("\n")
print("="*60)
print("CATEGORY DISTRIBUTION")
print("="*60)

for col in category_df.columns:

    print(
        col,
        ":",
        final_df[col].sum()
    )

```

```

    )

print("\n")
print("="*60)
print("PRIORITY DISTRIBUTION")
print("="*60)

print(
    final_df["priority"].value_counts()
)

print("\n")
print("="*60)
print("NUMBER OF LABELS PER EMAIL")
print("="*60)

print(
    final_df[
        category_df.columns
    ]
    .sum(axis=1)
    .value_counts()
)

# =====
# SAVE FINAL FILE
# =====

final_df.to_csv(
    OUTPUT_PATH,
    index=False
)

print("\n")
print("="*60)
print("SAVE COMPLETE")
print("="*60)

print(
    "Saved File:",
    OUTPUT_PATH
)

```

```

=====
ORIGINAL DATA
=====
Shape: (516755, 1)

                                Cleaned_Text
0                                here is our forecast
1  re traveling to have a business meeting takes ...
2                                re test test successful way to go
3  randy can you send me a schedule of the salary...
4                                re hello let s shoot for tuesday at 11 45

Categories:
['Legal', 'Financial', 'HR', 'Market', 'Administration']

Priorities:
['High', 'Normal', 'Low']

Starting labeling...
Labeling Completed

Generated Categories:
['category_Administration', 'category_Financial', 'category_HR',
'category_Legal', 'category_Market', 'category_Other']

=====
CATEGORY DISTRIBUTION
=====
category_Administration : 335630
category_Financial : 201533
category_HR : 171381
category_Legal : 156003
category_Market : 250563
category_Other : 82944

=====
PRIORITY DISTRIBUTION
=====
priority
Normal    263752
Low       189017
High      63986
Name: count, dtype: int64

=====
NUMBER OF LABELS PER EMAIL
=====
1    198025
2    113222
3     92584
4     68787
5     44137
Name: count, dtype: int64

```

```

=====
SAVE COMPLETE
=====

Saved File:
/content/drive/MyDrive/Thesis_Project/emails_labeled_final_multilabel.csv

اعادة الوسم

# =====
# إعادة وسم الإيميلات (Categories + Priorities)
# مع الإحصائيات والتحقق من جودة الوسوم
# =====

import pandas as pd
import json
import re
from sklearn.preprocessing import MultiLabelBinarizer

# =====
# المسارات
# =====

DATA_PATH = "/content/drive/MyDrive/Thesis_Project/emails_cleaned_final.csv"

CONFIG_PATH = "/content/drive/MyDrive/Thesis_Project/keywords_config.json"

OUTPUT_PATH =
    "/content/drive/MyDrive/Thesis_Project/emails_labeled_analysis.csv"

# =====
# قراءة البيانات
# =====

df = pd.read_csv(DATA_PATH)

print("="*60)
print("ORIGINAL DATA")
print("="*60)

print("Shape:", df.shape)
print(df.head())

# =====
# قراءة JSON
# يحتوي:
# categories
# priorities
# =====

with open(CONFIG_PATH, "r", encoding="utf-8") as f:
    config = json.load(f)

```

```

category_rules = config["categories"]

priority_rules = config["priorities"]


print("\nCategories:")
print(list(category_rules.keys()))


print("\nPriorities:")
print(list(priority_rules.keys()))


# =====
# تنظيف النص
# =====

TEXT_COL = "Cleaned_Text"


def clean_text(text):

    text = str(text).lower()

    text = re.sub(
        r"http\S+",
        " ",
        text
    )

    text = re.sub(
        r"[^a-z\s]",
        " ",
        text
    )

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

    return text


df["processed_text"] = df[TEXT_COL].apply(clean_text)


# =====
# حساب النقاط بالوزن
# =====

```

```

def compute_score(text, rules):

    scores = {}

    for label, keywords in rules.items():

        total_score = 0

        for keyword, weight in keywords.items():

            if keyword.lower() in text:

                total_score += weight

        scores[label] = total_score

    return scores

# =====
# تصنيف Categories
# Multi Label
# =====

CATEGORY_THRESHOLD = 5

def classify_category(text):

    scores = compute_score(
        text,
        category_rules
    )

    labels = []

    for category, score in scores.items():

        if score >= CATEGORY_THRESHOLD:

            labels.append(category)

    if len(labels) == 0:

        labels.append("Other")

```

```

    return labels

# =====
# تصنيف Priority
# Single Label
# =====

def classify_priority(text):

    scores = compute_score(
        text,
        priority_rules
    )

    best_priority = max(
        scores,
        key=scores.get
    )

    if scores[best_priority] > 0:

        return best_priority

    return "Normal"

# =====
# تطبيق التصنيف
# =====

print("\nStarting labeling...")

df["category_list"] = df["processed_text"].apply(
    classify_category
)

df["priority"] = df["processed_text"].apply(
    classify_priority
)

print("Labeling completed")

# =====
# إلى أعمدة Multi-Label تحويل

```

```

# =====

mlb = MultiLabelBinarizer()

category_matrix = mlb.fit_transform(
    df["category_list"]
)

category_df = pd.DataFrame(
    category_matrix,
    columns=[
        "category_" + c
        for c in mlb.classes_
    ]
)

print("\nGenerated Categories:")
print(category_df.columns.tolist())

# =====
# دمج النتيجة
# =====

final_df = pd.concat(
    [
        df[
            [
                TEXT_COL,
                "priority",
                "category_list"
            ]
        ],
        category_df
    ],
    axis=1
)

# =====
# إحصائيات / Categories
# =====

print("\n")
print("="*60)
print("CATEGORY DISTRIBUTION")

```

```

print("="*60)

for col in category_df.columns:

    print(
        col,
        ":",
        final_df[col].sum()
    )

# =====
# إحصائيات Priority
# =====

print("\n")
print("="*60)
print("PRIORITY DISTRIBUTION")
print("="*60)

print(
    final_df["priority"]
    .value_counts()
)

# =====
# Email عدد التصنيفات لكل
# =====

print("\n")
print("="*60)
print("NUMBER OF LABELS PER EMAIL")
print("="*60)

label_count = (
    category_df.sum(axis=1)
)

print(
    label_count.value_counts()
)

# =====
# كشف الرسائل ذات 4 و5 تصنيفات
# =====

```

```

high_multi = final_df[
    label_count >= 4
]

print("\n")
print("="*60)
print("EMAILS WITH 4 OR MORE LABELS")
print("="*60)

print(
    "Count:",
    len(high_multi)
)

print("\nSample:")

print(
    high_multi[
        [
            TEXT_COL,
            "category_list",
            "priority"
        ]
    ].head(20)
)

# =====
# حفظ الملف
# =====

final_df.to_csv(
    OUTPUT_PATH,
    index=False
)

print("\n")
print("="*60)
print("SAVE COMPLETE")
print("="*60)

print(
    OUTPUT_PATH
)

=====
ORIGINAL DATA
=====

```

Shape: (516755, 1)

	Cleaned_Text
0	here is our forecast
1	re traveling to have a business meeting takes ...
2	re test test successful way to go
3	randy can you send me a schedule of the salary...
4	re hello let s shoot for tuesday at 11 45

Categories:

['Legal', 'Financial', 'HR', 'Market', 'Administration']

Priorities:

['High', 'Normal', 'Low']

Starting labeling...

Labeling completed

Generated Categories:

['category_Administration', 'category_Financial', 'category_HR',
'category_Legal', 'category_Market', 'category_Other']

=====

CATEGORY DISTRIBUTION

=====

category_Administration : 335630
category_Financial : 201533
category_HR : 171381
category_Legal : 156003
category_Market : 250563
category_Other : 82944

=====

PRIORITY DISTRIBUTION

=====

priority
Normal 263752
Low 189017
High 63986
Name: count, dtype: int64

=====

NUMBER OF LABELS PER EMAIL

=====

1 198025
2 113222
3 92584
4 68787
5 44137
Name: count, dtype: int64

=====

EMAILS WITH 4 OR MORE LABELS

=====

Count: 112924

Sample:

```

Cleaned_Text \
9    fw fixed forward or other collar floor gas pri...
23   san juan index forwarded by phillip k allen ho...
24   san juan index liane as we discussed yesterday...
25   investment structure forwarded by phillip k al...
26   investment structure forwarded by phillip k al...
40   george below is a list of questions that keith...
41   westgate proforma phillip allen xls forwarded ...
50   westgate proforma phillip allen xls forwarded ...
54   re sagewood etc jeff you would clearly receive...
56   westgate proforma phillip allen xls forwarded ...
59   larry just a note to touch base on the sagewoo...
62   wow forwarded by phillip k allen hou ect on 09...
117  gas fundamentals development website forwarded...
127  tim can you authorize access to the west power...
135  assoc for west desk forwarded by phillip k all...
187  re sw gas forwarded by phillip k allen hou ect...
188  sw gas laird did you meet with swg on april 27...
239  pira s california southwest gas pipeline study...
253  forwarded by phillip k allen hou ect on 02 04 ...
276  re 2 survey information email 5 14 01 outlook ...
```

```

category_list priority
9    [Financial, HR, Market, Administration]    High
23   [Financial, HR, Market, Administration]    Normal
24   [Financial, HR, Market, Administration]    Normal
25   [Legal, Financial, HR, Market, Administration] Normal
26   [Legal, Financial, HR, Market, Administration] Normal
40   [Legal, Financial, HR, Market, Administration] Normal
41   [Legal, Financial, HR, Market, Administration] Normal
50   [Legal, Financial, HR, Market, Administration] Normal
54   [Legal, Financial, Market, Administration] Normal
56   [Legal, Financial, HR, Market, Administration] Normal
59   [Financial, HR, Market, Administration]    Normal
62   [Legal, Financial, HR, Market, Administration] Normal
117  [Financial, HR, Market, Administration]    Normal
127  [Legal, Financial, Market, Administration] Normal
135  [Financial, HR, Market, Administration]    Low
187  [Legal, Financial, HR, Market, Administration] Normal
188  [Legal, Financial, Market, Administration] Low
239  [Financial, HR, Market, Administration]    Low
253  [Financial, HR, Market, Administration]    Normal
276  [Legal, HR, Market, Administration]        Low
```

```
=====
SAVE COMPLETE
=====
```

/content/drive/MyDrive/Thesis_Project/emails_labeled_analysis.csv

استبعاد الرسائل ذات التصنيفات الكثيرة

```
# =====
# استبعاد الرسائل ذات التصنيفات الكثيرة للتدريب
```

```
# مع معالجة نوع بيانات أعمدة التصنيفات
```

```
# =====
```

```
import pandas as pd
```

```
INPUT_PATH =
```

```
    "/content/drive/MyDrive/Thesis_Project/emails_labeled_analysis.csv"
```

```
TRAIN_PATH =
```

```
    "/content/drive/MyDrive/Thesis_Project/emails_labeled_training.csv"
```

```
# قراءة الملف
```

```
df = pd.read_csv(INPUT_PATH)
```

```
# استخراج أعمدة التصنيفات
```

```
category_columns = [  
    col for col in df.columns  
    if col.startswith("category_")  
]
```

```
print("Category columns:")
```

```
print(category_columns)
```

```
# تحويل أعمدة التصنيف إلى أرقام
```

```
for col in category_columns:  
    df[col] = pd.to_numeric(  
        df[col],  
        errors="coerce"  
    ).fillna(0)
```

```
# حساب عدد التصنيفات لكل إيميل
```

```
df["label_count"] = (  
    df[category_columns]  
    .sum(axis=1)  
)
```

```
# الإحصائيات قبل الاستبعاد
```

```
print("\n=====")  
print("LABEL COUNT BEFORE FILTER")  
print("=====")
```

```
print(  
    df["label_count"]  
    .value_counts()
```

```
        .sort_index()  
    )
```

الاحتفاظ بالإيميلات ذات 3-1 تصنيفات

```
df_train = df[  
    df["label_count"] <= 3  
].copy()
```

حذف العمود المؤقت

```
df_train.drop(  
    columns=["label_count"],  
    inplace=True  
)
```

حفظ ملف التدريب

```
df_train.to_csv(  
    TRAIN_PATH,  
    index=False  
)
```

```
print("\n=====")  
print("TRAINING DATA")  
print("=====")
```

```
print(  
    "Original:",  
    df.shape  
)
```

```
print(  
    "After filtering:",  
    df_train.shape  
)
```

```
print("\nSaved:")  
print(TRAIN_PATH)
```

```
Category columns:  
['category_list', 'category_Administration', 'category_Financial',  
'category_HR', 'category_Legal', 'category_Market', 'category_Other']
```

```
=====
LABEL COUNT BEFORE FILTER
=====
label_count
1.0      198025
2.0      113222
```

```
3.0      92584
4.0      68787
5.0      44137
Name: count, dtype: int64
```

```
=====
TRAINING DATA
=====
```

```
Original: (516755, 10)
After filtering: (403831, 9)
```

```
Saved:
/content/drive/MyDrive/Thesis_Project/emails_labeled_training.csv
```

تنظيف اضافي

```
# =====
# CLEAN MASTER LABELED DATASET
# =====
#
# المصدر:
# emails_labeled_training.csv
#
# الهدف:
# إنشاء نسخة نظيفة قبل استخراج العينة النهائية
#
# =====
```

```
import pandas as pd
```

```
INPUT_FILE =
    "/content/drive/MyDrive/Thesis_Project/emails_labeled_training.csv"
```

```
OUTPUT_FILE =
    "/content/drive/MyDrive/Thesis_Project/emails_labeled_training_clean.csv"
```

```
# =====
# LOAD DATA
# =====
```

```
df = pd.read_csv(INPUT_FILE)
```

```
print("=" * 60)
print("ORIGINAL DATASET")
print("=" * 60)
print("Shape:", df.shape)
```

```
# =====
# REMOVE MISSING TEXT
# =====
```

```
df = df.dropna(subset=["Cleaned_Text"])
```

```
# =====
# TEXT LENGTH
# =====
```

```

df["text_length"] = df["Cleaned_Text"].astype(str).str.len()

# =====
# REMOVE VERY SHORT EMAILS
# =====

df = df[df["text_length"] >= 10]

# =====
# REMOVE DUPLICATES
# =====

before = len(df)

df = df.drop_duplicates(
    subset=["Cleaned_Text"]
).reset_index(drop=True)

duplicates_removed = before - len(df)

# =====
# REMOVE EXTREME OUTLIERS
# =====

p995 = df["text_length"].quantile(0.995)

df = df[df["text_length"] <= p995]

# =====
# CLEANUP
# =====

df = df.drop(columns=["text_length"])

# =====
# SAVE CLEAN DATASET
# =====

df.to_csv(
    OUTPUT_FILE,
    index=False
)

# =====
# REPORT
# =====

print("\n" + "=" * 60)
print("CLEANING REPORT")
print("=" * 60)

print("Duplicates Removed :", duplicates_removed)
print("P99.5 Threshold      :", int(p995))
print("Final Shape           :", df.shape)

```

```

print("\nSaved File:")
print(OUTPUT_FILE)

=====
ORIGINAL DATASET
=====
Shape: (403831, 8)

=====
CLEANING REPORT
=====
Duplicates Removed : 211981
P99.5 Threshold    : 5965
Final Shape        : (190280, 8)

Saved File:
/content/drive/MyDrive/Thesis_Project/emails_labeled_training_clean.csv

```

استكشاف البيانات (Exploratory Data Analysis - EDA)

```

import pandas as pd

df = pd.read_csv(
    "/content/drive/MyDrive/Thesis_Project/emails_labeled_training_clean.csv"
)

lengths = df["Cleaned_Text"].astype(str).str.len()

print("Min    :", lengths.min())
print("Mean    :", round(lengths.mean(),2))
print("Median :", lengths.median())

print("\nPercentiles")
print("P95    :", lengths.quantile(0.95))
print("P99    :", lengths.quantile(0.99))
print("P99.5:", lengths.quantile(0.995))
print("P99.9:", lengths.quantile(0.999))

print("\nMax Length:", lengths.max())

Min    : 10
Mean   : 672.87
Median : 421.0

Percentiles
P95    : 2124.0
P99    : 3756.0
P99.5: 4474.60500000000105
P99.9: 5481.4420000000039

```

Max Length: 5964

للعينة (Descriptive Statistics) الإحصاء الوصفي

```
import pandas as pd

df = pd.read_csv(
    "/content/drive/MyDrive/Thesis_Project/emails_labeled_training_clean.csv"
)

category_cols = [
    "category_Administration",
    "category_Financial",
    "category_HR",
    "category_Legal",
    "category_Market",
    "category_Other"
]

print("="*60)
print("DATASET SIZE")
print("="*60)

print(len(df))

print("\n" + "="*60)
print("LABEL CARDINALITY")
print("="*60)

label_count = df[category_cols].sum(axis=1)

print("Average labels per email:")
print(label_count.mean())

print("\nDistribution:")
print(label_count.value_counts().sort_index())

print("\n" + "="*60)
print("CATEGORY DISTRIBUTION")
print("="*60)

for col in category_cols:

    count = int(df[col].sum())

    print(
        col,
        count,
        round(count/len(df)*100,2),
        "%"
    )

print("\n" + "="*60)
print("PRIORITY DISTRIBUTION")
```

```

print("="*60)

print(
    df["priority"]
      .value_counts(normalize=True)*100
)

=====
DATASET SIZE
=====
190280

=====
LABEL CARDINALITY
=====
Average labels per email:
1.7297929367248266

Distribution:
1    94389
2    52917
3    42974
Name: count, dtype: int64

=====
CATEGORY DISTRIBUTION
=====
category_Administration 103304 54.29 %
category_Financial 47397 24.91 %
category_HR 37257 19.58 %
category_Legal 33618 17.67 %
category_Market 67362 35.4 %
category_Other 40207 21.13 %

=====
PRIORITY DISTRIBUTION
=====
priority
Normal    51.464158
Low       39.288943
High       9.246899
Name: proportion, dtype: float64

```

"Summary Statistics of Dataset Word Count" (إحصاءات التلخيص لعدد كلمات مجموعة البيانات).

```

df["word_count"] = (
    df["Cleaned_Text"]
      .str.split()
      .str.len()
)

print(df["word_count"].describe())

count    190280.000000
mean      124.742795
std       137.452653
min        1.000000

```

```
25%          37.000000
50%          79.000000
75%         162.000000
max         1321.000000
Name: word_count, dtype: float64
```

حساب Percentiles

```
print(
    df["word_count"].quantile(
        [0.90,0.95,0.99,0.995,0.999]
    )
)
```

```
0.900    283.0
0.950    390.0
0.990    692.0
0.995    827.0
0.999   1037.0
Name: word_count, dtype: float64
```

```
# =====
# إحصائية رقم (1)
# Label Cardinality و Label Density حساب
#
# الهدف:
# معرفة متوسط عدد التصنيفات الموجودة في كل رسالة
# ومعرفة كثافة التصنيفات بالنسبة لعدد الفئات الكلية
# =====
```

```
import pandas as pd
```

```
# قراءة ملف التدريب النهائي
df = pd.read_csv(
    "/content/drive/MyDrive/Thesis_Project/emails_labeled_training_clean.csv"
)
```

```
# أعمدة التصنيفات
category_columns = [
    "category_Administration",
    "category_Financial",
    "category_HR",
    "category_Legal",
    "category_Market",
    "category_Other"
]
```

```
# حساب عدد التصنيفات لكل إيميل
label_count = df[category_columns].sum(axis=1)
```

```
# Label Cardinality حساب
label_cardinality = label_count.mean()
```

```
# Label Density حساب
label_density = (
```

```

        label_cardinality /
        len(category_columns)
    )

print("=" * 60)
print("LABEL CARDINALITY & LABEL DENSITY")
print("=" * 60)

print(
    "Average Labels Per Email:",
    round(label_cardinality, 3)
)

print(
    "Label Density:",
    round(label_density, 3)
)

print("\nDistribution:")

print(
    label_count
    .value_counts()
    .sort_index()
)

=====
LABEL CARDINALITY & LABEL DENSITY
=====
Average Labels Per Email: 1.73
Label Density: 0.288

Distribution:
1    94389
2    52917
3    42974
Name: count, dtype: int64

# =====
# إحصائية رقم (2)
# Category Distribution + Imbalance Ratio
#
# الهدف:
# معرفة حجم كل فئة داخل البيانات -1
# معرفة نسبة كل فئة من إجمالي البيانات -2
# (Imbalance Ratio) قياس عدم التوازن -3
#
# ملاحظة:
# من 1 كان التوازن أفضل IR كلما اقتربت قيمة
# لها 1 = IR الفئة الأكبر دائماً سيكون
# =====

import pandas as pd

# قراءة ملف التدريب النهائي

```

```
FILE = "/content/drive/MyDrive/Thesis_Project/emails_labeled_training_clean.csv"
```

```
df = pd.read_csv(FILE)
```

```
# أعمدة التصنيفات
```

```
category_columns = [  
    "category_Administration",  
    "category_Financial",  
    "category_HR",  
    "category_Legal",  
    "category_Market",  
    "category_Other"  
]
```

```
print("=" * 60)  
print("CATEGORY DISTRIBUTION")  
print("=" * 60)
```

```
category_counts = {}
```

```
# حساب حجم كل فئة
```

```
for col in category_columns:  
  
    count = int(df[col].sum())  
  
    percentage = round(  
        count / len(df) * 100,  
        2  
    )  
  
    category_counts[col] = count  
  
    print(  
        f"{col:<25}",  
        f"Count = {count:<10}",  
        f"Percentage = {percentage}%"  
    )
```

```
# =====  
# حساب Imbalance Ratio  
# =====
```

```
print("\n")  
print("=" * 60)  
print("IMBALANCE RATIO")  
print("=" * 60)
```

```
max_count = max(  
    category_counts.values()  
)
```

```
for col, count in category_counts.items():
```

```
    ir = round(  
        (max_count - count) / max_count * 100,  
        2  
    )
```

```

        max_count / count,
        2
    )

    print(
        f"{col:<25}",
        f"IR = {ir}"
    )

# =====
# ملخص سريع
# =====

largest_category = max(
    category_counts,
    key=category_counts.get
)

smallest_category = min(
    category_counts,
    key=category_counts.get
)

print("\n")
print("=" * 60)
print("SUMMARY")
print("=" * 60)

print("Largest Category :", largest_category)
print("Smallest Category:", smallest_category)

print(
    "Largest Count      :",
    category_counts[largest_category]
)

print(
    "Smallest Count     :",
    category_counts[smallest_category]
)

=====
CATEGORY DISTRIBUTION
=====
category_Administration    Count = 103304      Percentage = 54.29%
category_Financial         Count = 47397       Percentage = 24.91%
category_HR                Count = 37257       Percentage = 19.58%
category_Legal             Count = 33618       Percentage = 17.67%
category_Market            Count = 67362       Percentage = 35.4%
category_Other             Count = 40207       Percentage = 21.13%

=====
IMBALANCE RATIO
=====

```

category_Administration	IR = 1.0
category_Financial	IR = 2.18
category_HR	IR = 2.77
category_Legal	IR = 3.07
category_Market	IR = 1.53
category_Other	IR = 2.57

```
=====
SUMMARY
=====
Largest Category : category_Administration
Smallest Category: category_Legal
Largest Count    : 103304
Smallest Count   : 33618
```

تحليل الترابط بين الفئات (Category Co-occurrence Matrix)

```
# =====
# إحصائية رقم (3)
# Category Co-occurrence Matrix
#
# الهدف:
# حساب عدد المرات التي تظهر فيها كل فئتين معاً
# داخل نفس البريد الإلكتروني.
#
# مثال:
# Financial + Market
# HR + Administration
# Legal + Financial
#
# كلما زاد الرقم كانت العلاقة بين الفئتين أقوى.
# =====
```

```
import pandas as pd
```

```
# =====
# قراءة ملف التدريب
# =====
```

```
FILE = "/content/drive/MyDrive/Thesis_Project/emails_labeled_training_clean.csv"
```

```
df = pd.read_csv(FILE)
```

```
# =====
# أعمدة التصنيفات
# =====
```

```
category_columns = [
    "category_Administration",
    "category_Financial",
    "category_HR",
    "category_Legal",
    "category_Market",
    "category_Other"
```

```

]

# =====
# إنشاء مصفوفة الترابط
# =====

co_matrix = pd.DataFrame(
    index=category_columns,
    columns=category_columns
)

# =====
# حساب عدد مرات ظهور كل زوج من الفئات معاً
# =====

for cat1 in category_columns:

    for cat2 in category_columns:

        co_matrix.loc[cat1, cat2] = (
            (
                df[cat1] == 1
            ) &
            (
                df[cat2] == 1
            )
        ).sum()

# =====
# عرض المصفوفة
# =====

print("=" * 70)
print("CATEGORY CO-OCCURRENCE MATRIX")
print("=" * 70)

print(co_matrix)

# =====
# استخراج أقوى العلاقات بين الفئات
# (باستثناء القطر الرئيسي)
# =====

print("\n")
print("=" * 70)
print("TOP CATEGORY RELATIONSHIPS")
print("=" * 70)

relationships = []

for i in range(len(category_columns)):

    for j in range(i + 1, len(category_columns)):

        cat1 = category_columns[i]

```

```

        cat2 = category_columns[j]

        count = (
            (
                df[cat1] == 1
            ) &
            (
                df[cat2] == 1
            )
        ).sum()

        relationships.append(
            (cat1, cat2, count)
        )

relationships = sorted(
    relationships,
    key=lambda x: x[2],
    reverse=True
)

for rel in relationships[:15]:

    print(
        f"{rel[0]} + {rel[1]} = {rel[2]}"
    )

# =====
# عدد العلاقات المحسوبة
# =====

print("\n")
print("=" * 70)
print("SUMMARY")
print("=" * 70)

print(
    "Number of Categories:",
    len(category_columns)
)

print(
    "Number of Category Pairs:",
    len(relationships)
)

=====
CATEGORY CO-OCCURRENCE MATRIX
=====

```

	category_Administration	category_Financial \
category_Administration	103304	27690
category_Financial	27690	47397
category_HR	24599	8033
category_Legal	18787	8246
category_Market	40952	21835

category_Other	0	0	
	category_HR	category_Legal	category_Market \
category_Administration	24599	18787	40952
category_Financial	8033	8246	21835
category_HR	37257	7171	11624
category_Legal	7171	33618	12902
category_Market	11624	12902	67362
category_Other	0	0	0

	category_Other
category_Administration	0
category_Financial	0
category_HR	0
category_Legal	0
category_Market	0
category_Other	40207

===== TOP CATEGORY RELATIONSHIPS =====

category_Administration + category_Market = 40952
category_Administration + category_Financial = 27690
category_Administration + category_HR = 24599
category_Financial + category_Market = 21835
category_Administration + category_Legal = 18787
category_Legal + category_Market = 12902
category_HR + category_Market = 11624
category_Financial + category_Legal = 8246
category_Financial + category_HR = 8033
category_HR + category_Legal = 7171
category_Administration + category_Other = 0
category_Financial + category_Other = 0
category_HR + category_Other = 0
category_Legal + category_Other = 0
category_Market + category_Other = 0

===== SUMMARY =====

Number of Categories: 6
Number of Category Pairs: 15

فحص جودة البيانات والتحقق من مصادر الانحياز قبل استخراج عينة التدريب

```
# =====
# إحصائية رقم (4)
# Data Quality & Bias Check
#
# الهدف:
# التأكد من سلامة البيانات قبل استخراج عينة التدريب
# والكشف عن أي مشكلة قد تسبب انحيازاً للنموذج.
# =====
```

```

import pandas as pd

# =====
# قراءة ملف التدريب
# =====

FILE = "/content/drive/MyDrive/Thesis_Project/emails_labeled_training_clean.csv"

df = pd.read_csv(FILE)

# =====
# أعمدة التصنيفات
# =====

category_columns = [
    "category_Administration",
    "category_Financial",
    "category_HR",
    "category_Legal",
    "category_Market",
    "category_Other"
]

# =====
# حجم البيانات
# =====

print("=" * 70)
print("DATASET OVERVIEW")
print("=" * 70)

print("Rows :", len(df))
print("Columns :", len(df.columns))

# =====
# القيم الفارغة
# =====

print("\n")
print("=" * 70)
print("MISSING VALUES")
print("=" * 70)

missing = df.isnull().sum()

print(missing)

print("\nTotal Missing Values:")
print(missing.sum())

# =====
# النصوص الفارغة
# =====

print("\n")

```

```

print("=" * 70)
print("EMPTY TEXT CHECK")
print("=" * 70)

empty_text = (
    df["Cleaned_Text"]
    .fillna("")
    .str.strip()
    .eq("")
    .sum()
)

print("Empty Emails:", empty_text)

# =====
# التكرار
# =====

print("\n")
print("=" * 70)
print("DUPLICATE EMAIL CHECK")
print("=" * 70)

duplicates = df.duplicated(
    subset=["Cleaned_Text"]
).sum()

print("Duplicate Emails:", duplicates)

# =====
# عدد التصنيفات لكل رسالة
# =====

print("\n")
print("=" * 70)
print("LABEL COUNT DISTRIBUTION")
print("=" * 70)

df["label_count"] = (
    df[category_columns]
    .sum(axis=1)
)

print(
    df["label_count"]
    .value_counts()
    .sort_index()
)

# =====
# توزيع الأولويات
# =====

print("\n")

```

```

print("=" * 70)
print("PRIORITY DISTRIBUTION")
print("=" * 70)

print(
    df["priority"]
    .value_counts(normalize=True)
    * 100
)

# =====
# التحقق من قيم التصنيفات
# =====

print("\n")
print("=" * 70)
print("CATEGORY VALUE VALIDATION")
print("=" * 70)

for col in category_columns:

    unique_values = sorted(
        df[col]
        .dropna()
        .unique()
        .tolist()
    )

    print(col, "=>", unique_values)

# =====
# تحليل أطوال الرسائل
# =====

print("\n")
print("=" * 70)
print("EMAIL LENGTH ANALYSIS")
print("=" * 70)

df["word_count"] = (
    df["Cleaned_Text"]
    .fillna("")
    .str.split()
    .str.len()
)

print(
    df["word_count"]
    .describe()
)

print("\nPercentiles:")

print(

```

```

        df["word_count"]
        .quantile(
            [0.90, 0.95, 0.99, 0.995, 0.999]
        )
    )

# =====
# رسائل قصيرة جداً
# =====

print("\n")
print("=" * 70)
print("SHORT EMAIL CHECK")
print("=" * 70)

short_emails = (
    df["word_count"] < 5
).sum()

print("Emails < 5 words :", short_emails)

print(
    "Percentage :",
    round(
        short_emails / len(df) * 100,
        2
    ),
    "%"
)

# =====
# رسائل طويلة جداً
# =====

print("\n")
print("=" * 70)
print("LONG EMAIL CHECK")
print("=" * 70)

long_emails = (
    df["word_count"] > 2000
).sum()

print("Emails > 2000 words :", long_emails)

print(
    "Percentage :",
    round(
        long_emails / len(df) * 100,
        4
    ),
    "%"
)

```

```
# =====  
# ملخص نهائي  
# =====
```

```
print("\n")  
print("=" * 70)  
print("FINAL QUALITY SUMMARY")  
print("=" * 70)  
  
print("Missing Values :", missing.sum())  
print("Duplicate Emails :", duplicates)  
print("Empty Emails :", empty_text)  
print("Short Emails :", short_emails)  
print("Long Emails :", long_emails)  
  
if (  
    missing.sum() == 0  
    and empty_text == 0  
):  
    print("\nDataset Passed Basic Quality Checks")  
else:  
    print("\nDataset Requires Additional Cleaning")
```

```
=====
```

DATASET OVERVIEW

```
=====
```

Rows : 190280
Columns : 8

```
=====
```

MISSING VALUES

```
=====
```

Cleaned_Text	0
priority	0
category_Administration	0
category_Financial	0
category_HR	0
category_Legal	0
category_Market	0
category_Other	0

dtype: int64

Total Missing Values:
0

```
=====
```

EMPTY TEXT CHECK

```
=====
```

Empty Emails: 0

```
=====
```

DUPLICATE EMAIL CHECK

```
=====
```

Duplicate Emails: 0

```
=====
LABEL COUNT DISTRIBUTION
=====
label_count
1      94389
2      52917
3      42974
Name: count, dtype: int64
```

```
=====
PRIORITY DISTRIBUTION
=====
priority
Normal    51.464158
Low       39.288943
High      9.246899
Name: proportion, dtype: float64
```

```
=====
CATEGORY VALUE VALIDATION
=====
category_Administration => [0, 1]
category_Financial => [0, 1]
category_HR => [0, 1]
category_Legal => [0, 1]
category_Market => [0, 1]
category_Other => [0, 1]
```

```
=====
EMAIL LENGTH ANALYSIS
=====
count      190280.000000
mean        124.742795
std         137.452653
min          1.000000
25%         37.000000
50%         79.000000
75%        162.000000
max        1321.000000
Name: word_count, dtype: float64
```

```
Percentiles:
0.900      283.0
0.950      390.0
0.990      692.0
0.995      827.0
0.999     1037.0
Name: word_count, dtype: float64
```

```
=====
```

SHORT EMAIL CHECK

Emails < 5 words : 1261

Percentage : 0.66 %

LONG EMAIL CHECK

Emails > 2000 words : 0

Percentage : 0.0 %

FINAL QUALITY SUMMARY

Missing Values : 0

Duplicate Emails : 0

Empty Emails : 0

Short Emails : 1261

Long Emails : 0

Dataset Passed Basic Quality Checks

تحليل التكرار الحقيقي

```
# =====  
# تحليل التكرار الحقيقي  
# =====
```

```
unique_emails = df["Cleaned_Text"].nunique()
```

```
print("=" * 60)
```

```
print("UNIQUE EMAIL ANALYSIS")
```

```
print("=" * 60)
```

```
print("Total Emails :", len(df))
```

```
print("Unique Emails:", unique_emails)
```

```
print(  
    "Duplicate Rate:",  
    round(  
        (1 - unique_emails / len(df)) * 100,  
        2  
    ),  
    "%"  
)
```

UNIQUE EMAIL ANALYSIS

Total Emails : 190280

Unique Emails: 190280

Duplicate Rate: 0.0 %

تحليل التكرارات (Duplicate Frequency Analysis)

```

# =====
# إحصائية رقم (5)
# Duplicate Frequency Analysis
#
# الهدف:
# دراسة التكرارات داخل البيانات
# ومعرفة هل يوجد عدد قليل من الرسائل يسيطر على جزء كبير
# من مجموعة البيانات.
# =====

import pandas as pd

# =====
# حساب عدد مرات ظهور كل رسالة
# =====

duplicate_counts = (
    df["Cleaned_Text"]
    .value_counts()
)

# =====
# ملخص عام
# =====

print("=" * 70)
print("DUPLICATE ANALYSIS")
print("=" * 70)

print("Total Emails:")
print(len(df))

print("\nUnique Emails:")
print(df["Cleaned_Text"].nunique())

print("\nRepeated Emails:")
print(
    (duplicate_counts > 1).sum()
)

# =====
# أكثر الرسائل تكراراً
# =====

print("\n")
print("=" * 70)
print("TOP 20 MOST REPEATED EMAILS")
print("=" * 70)

print(
    duplicate_counts.head(20)
)

# =====

```

```

# توزيع التكرار
# =====

print("\n")
print("=" * 70)
print("DUPLICATE FREQUENCY DISTRIBUTION")
print("=" * 70)

freq_distribution = (
    duplicate_counts
    .value_counts()
    .sort_index()
)

print(freq_distribution.head(20))

# =====
# نسب الرسائل حسب عدد مرات التكرار
# =====

print("\n")
print("=" * 70)
print("EMAIL REPETITION GROUPS")
print("=" * 70)

once = (duplicate_counts == 1).sum()
two_to_five = ((duplicate_counts >= 2) & (duplicate_counts <= 5)).sum()
six_to_ten = ((duplicate_counts >= 6) & (duplicate_counts <= 10)).sum()
more_than_ten = (duplicate_counts > 10).sum()

print("Appeared Once      :", once)
print("Appeared 2-5 Times  :", two_to_five)
print("Appeared 6-10 Times :", six_to_ten)
print("Appeared >10 Times  :", more_than_ten)

# =====
# نسب مئوية
# =====

total_unique = len(duplicate_counts)

print("\nPercentages:")

print(
    "Once:",
    round(once / total_unique * 100, 2),
    "%"
)

print(
    "2-5:",
    round(two_to_five / total_unique * 100, 2),
    "%"
)

```

```

print(
    "6-10:",
    round(six_to_ten / total_unique * 100, 2),
    "%"
)

print(
    ">10:",
    round(more_than_ten / total_unique * 100, 2),
    "%"
)

```

```

=====
DUPLICATE ANALYSIS
=====

```

```

Total Emails:
190280

```

```

Unique Emails:
190280

```

```

Repeated Emails:
0

```

```

=====
TOP 20 MOST REPEATED EMAILS
=====

```

```

Cleaned_Text

```

```

re ali s essays i think the ymca has a class that is for people recovering
from heart attacks i remeber something about that original message i don t
know about the heart classes i ll look into it but her dr ravi isn t offering
up any suggestions or anything she saw him before the surgery in august and
he said things were okay i really don t think he s too helpful she is lazy
but it really frustrates me that she doesn t want to help herself i told her
that not walking is like not taking her heart medication that didn t seem to
resonate dad is going to go to the ymca tomorrow and maybe get a membership
for both of them they have a walking track there at least it s something to
do in the winter when she was down this weekend we walked around the craft
show at the exhibition place and she said that was a lot of exercise 2 hrs
the only problem is that we were just strolling and not really walking very
fast john zufferli enron com 11 26 2001 01 41 pm just send the essay at home
i don t know what to do about mom i don t think fear is the only thing
holding her back i think she is lazy is there a heart health class in sudbury
that has excercise regimines as well as diets when is the last time she saw
her doctor original message hi john how was thanksgiving was the baby shower
fun i was wondering if you d have some time to read over ali s chicago essays
later tonight he s going to submit them on wednesday let me know if that s
okay do you have a printer at home can i send them to your home account i don
t think ali will be done before about 8pm or so tonight ps we need to talk
about mom i saw her this weekend she s gained a lot of weight and hasn t been
exercising at all dad s pretty frustrated because all she does is watch tv i
had a talk with her yesterday telling her that she has more risk of having a
heart attack if she doesn t walk exercise than if she exerts herself when
exercising i think she s afraid of having a heart attack while exercising we

```

need to do something she s 170lbs now and should be at around 140lbs to be healthy livia this e mail is the property of enron corp and or its relevant affiliate and may contain confidential and privileged material for the sole use of the intended recipient s any review use distribution or disclosure by others is strictly prohibited if you are not the intended recipient or authorized to receive for the recipient please contact the sender or reply to enron corp at enron messaging administration enron com and delete all copies of the message this e mail and any attachments hereto are not intended to be an offer or an acceptance and do not create or evidence a binding and enforceable contract between enron corp or any of its affiliates and the intended recipient or any other party and may not be relied on by anyone as the basis of a contract by estoppel or otherwise thank you 1

here is our forecast

1

re traveling to have a business meeting takes the fun out of the trip especially if you have to prepare a presentation i would suggest holding the business plan meetings here then take a trip without any formal business meetings i would even try and get some honest opinions on whether a trip is even desired or necessary as far as the business meetings i think it would be more productive to try and stimulate discussions across the different groups about what is working and what is not too often the presenter speaks and the others are quiet just waiting for their turn the meetings might be better if held in a round table discussion format my suggestion for where to go is austin play golf and rent a ski boat and jet ski s flying somewhere takes too much time

1

re test test successful way to go

1

randy can you send me a schedule of the salary and level of everyone in the scheduling group plus your thoughts on any changes that need to be made patti s for example phillip

1

re hello let s shoot for tuesday at 11 45

1

re hello greg how about either next tuesday or thursday phillip

1

please cc the following distribution list with updates phillip allen pallen enron com mike grigsby mike grigsby enron com keith holst kholst enron com monique sanchez frank ermis john lavorato thank you for your help phillip allen

1

fw i newco file inventory original message everything from ops trading marketing and development in calgary plus mike s inputs from toronto has been consolidated into one master index the file is located at i newco 00 readme xls if you have additional items to add to your directories or i have not yet received inputs from you please update the master index there is one tab for each department thanks richard c mckeel enron canada corp phone 403 974 6742 fax 403 974 6706

1

fw original message hey there dad told me that you got your bonus congratulations that s a piece of good news at least i got some bad news about bonuses today we usually get them paid out this week but we just got an email from our ceo saying that only a few exceptional performers will get some sort of bonus but they won t disclose the information until the end of january given that i have to have a new place to live for february i don t think that i ll be able buy a condo anymore the bonus money would have been for the down payment anyway kind of annoying but i guess i should be grateful

that i have a job mom s size is large she s 5 4 and about 145 pounds could you pass this on to jess by the way our flight comes in at around 10pm on saturday but i ll call to confirm livia

1

re market risk meeting today john zufferli i am waiting on the conference call and no one is on it yet original message importance high hi the market risk meeting has been scheduled for today at 5 15 p m houston time the dial in information is as follows dial in number 203 719 0070 freephone within the u s 877 315 5218 pin number 808787 mercy please reserve a conference room for those participants in houston thanks jm visit our website at this message contains confidential information and is intended only for the individual named if you are not the named addressee you should not disseminate distribute or copy this e mail please notify the sender immediately by e mail if you have received this e mail by mistake and delete this e mail from your system e mail transmission cannot be guaranteed to be secure or error free as information could be intercepted corrupted lost destroyed arrive late or incomplete or contain viruses the sender therefore does not accept liability for any errors or omissions in the contents of this message which arise as a result of e mail transmission if verification is required please request a hard copy version this message is provided for informational purposes and should not be construed as a solicitation or offer to buy or sell any securities or related financial instruments

1

gas and power eol access here is the update list that you requested mike cowan chris lambie chad clark chris dorland lon draper cooper richie john zufferli also needs access to the nymex eol website mike cowan chris lambie chad clark chris dorland need access to us west and east basis markets as well

1

re new company online trader access stack manager website original message importance high john i understand you have been approached separately to provide this information if so could i please have a copy of your worksheets thanks nella original message importance high rob please populate the attached worksheets for stack manager website access i have added the canada product types and a drop down for each user product type to populate with read execute or none for read only website id s additional population is not necessary please return to me once complete let me know if you have any questions file can gas stack website for new version 121901 xls thank you stephanie sever enrononline 713 853 3465

1

re quick question you should talk to jess about that she is due march 2 but i am not sure when her mom is coming and going i would think that the 2 week in march is good though look forward to seeing you tomorrow original message when is jessica due i want to book tickets out to see you guys in feb march but i want to make sure that you have time to get settled before i come i have to order tickets by dec 24th as ali has a special 2 for 1 coupon so he can come with me too what timing is best

1

lethbrige ironworks you can increase the direct sale to include dec 31 2001 thank you john zufferli

1

re w c deficiency rider yes original message importance high this is the first that i have heard of this something for mike to look at mark original message wc deficiency notice doc file wc deficiency notice doc eal deferral summary november 2001 doc file eal deferral summary november 2001 doc

1

training i am interested in a program for myself i am 33 years old have been

inactive for 6 months i am 6'1" and 225 lbs i can easily ride 50 km at about 27 km/h on flat terrain i would like to lose 15 lbs improve my speed and definitely improve my hill climbing ability i live at 3800 ft altitude and hill climbing is very difficult i hit lactate threshold fairly quick thanks
1

re dad asked if i wanted to lend you some money but i am reluctant to do it at this point in time since a) i am not completely at liberty to touch all the cash that i have received lately for concern that it may be taken bank under potential creditor action against enron unfortunately without this cash i really can't afford to lend any the cash that i received recently represents 80% of the cash that i have and i am not sure when i can count on this cash with any degree of certainty therefore i am reluctant to touch this money for at least 6 months b) as of right now i don't have a job after feb 28 the new company deal that enron is trying to structure with its traders and a new buyer will not be done until the month if this deal falls through then i have to find a job with another company in this case it is most likely that i will have to move back to the us since my contacts and the most lucrative opportunities i.e. compared to the new enron deal are there c) if i had the cash to give dad asked that i could give you a discount to the bank mortgage with discount rates near 5% even if i gave you a mortgage at 4% i don't think the monthly interest saving for you would be material enough d) i think it is good for you to develop a credit rating with a bank through a mortgage it is very beneficial for later on in life paying visa bills does help you a bit in this category but not like a mortgage hopefully you can get the financing from a bank i think that this is the best solution in the long run original message hi john wanted to talk to you about the house stuff dad told me that he was talking to you about financing options i just wanted to say that i don't want you to feel compelled to write a mortgage for me dad's just trying to explore different options i know it's a lot of cash and i would never want to impose on you i'm sure that i can get a mortgage from the bank at a decent rate as long as i can put down the 25% down payment so that i can avoid the mortgage insurance which is costly i'm cashing in some gics and other bonds and ali's going to lend me 10k so i will be able to come up with the 55k for the down payment if writing the mortgage is something that you would like to do for a stable investment then we can talk too my time horizon on the condo is 5-7 years so i recognize that that would hold up your capital for a long while anyway i have to go ahead and get the guarantee of financing within 5 days or so so i have to make a decision in the next day either way maybe you can call or email me livia
1

re if the legacy deals move to netco then we need the ability to trade canadian gas canada would not be out of business if we only traded us denominated financial products but it reduce our daily transactions by 75% original message john dealing with canadian issues and netco original thought was that legacy deals go away and we can do in canada as we do in us unfortunately feedback is that currency issue will still exist in your minds eye will it put canada out of biz if we require that during first 45 days only us like transactions occur your feedback would be great thanks frank
1

re market risk meeting today john zufferli i waited for 15 min i guess the conference call was cancelled original message importance high hi the market risk meeting has been scheduled for today at 5:15 p.m. houston time the dial in information is as follows dial in number 203 719 0070 freephone within the u.s. 877 315 5218 pin number 808787 mercy please reserve a conference room for those participants in houston thanks jm visit our website at this message contains confidential information and is intended only for the individual named if you are not the named addressee you should not disseminate

distribute or copy this e mail please notify the sender immediately by e mail if you have received this e mail by mistake and delete this e mail from your system e mail transmission cannot be guaranteed to be secure or error free as information could be intercepted corrupted lost destroyed arrive late or incomplete or contain viruses the sender therefore does not accept liability for any errors or omissions in the contents of this message which arise as a result of e mail transmission if verification is required please request a hard copy version this message is provided for informational purposes and should not be construed as a solicitation or offer to buy or sell any securities or related financial instruments

1

Name: count, dtype: int64

```
=====
DUPLICATE FREQUENCY DISTRIBUTION
=====
```

count

1 190280

Name: count, dtype: int64

```
=====
EMAIL REPETITION GROUPS
=====
```

Appeared Once : 190280

Appeared 2-5 Times : 0

Appeared 6-10 Times : 0

Appeared >10 Times : 0

Percentages:

Once: 100.0 %

2-5: 0.0 %

6-10: 0.0 %

>10: 0.0 %

إنشاء عينة تدريب ممثلة (Stratified Training Sample)

```
# =====
```

```
# إحصائية رقم (6)
```

```
# Stratified Training Sample Extraction
```

```
#
```

```
# الهدف:
```

```
# إنشاء عينة تدريب ممثلة للبيانات الأصلية
```

```
# مع الحفاظ على:
```

```
# 1- (Priority) توزيع الأولويات
```

```
# 2- (Label Count) توزيع عدد التصنيفات
```

```
#
```

```
# ملاحظة:
```

```
# داخل الملف النهائي label_count يتم الاحتفاظ بعمود
```

```
# لأغراض التحليل الإحصائي فقط، ولن يستخدم كميزة
```

```

# أثناء تدريب النماذج (Feature).
#
# حجم العينة:
# 50,000 سجل
# =====

import pandas as pd
from sklearn.model_selection import train_test_split

# =====
# قراءة ملف التدريب النهائي
# =====

FILE = "/content/drive/MyDrive/Thesis_Project/emails_labeled_training_clean.csv"

df = pd.read_csv(FILE)

print("=" * 60)
print("ORIGINAL DATASET")
print("=" * 60)

print(df.shape)

# =====
# أعمدة التصنيفات
# =====

category_columns = [
    "category_Administration",
    "category_Financial",
    "category_HR",
    "category_Legal",
    "category_Market",
    "category_Other"
]

# =====
# حساب عدد التصنيفات لكل رسالة
# =====

df["label_count"] = (
    df[category_columns]
    .sum(axis=1)
)

# =====
# إنشاء طبقة Stratification
#
# جمع:
# - الأولوية
# - عدد التصنيفات
#
# أمثلة:
# Normal_1

```

```

# High_2
# Low_3
# =====

df["stratify_group"] = (
    df["priority"].astype(str)
    + "_"
    + df["label_count"].astype(str)
)

# =====
# حجم العينة المطلوبة
# =====

SAMPLE_SIZE = 50000

# =====
# استخراج العينة الطبقية
# =====

sample_df, _ = train_test_split(
    df,
    train_size=SAMPLE_SIZE,
    stratify=df["stratify_group"],
    random_state=42
)

# =====
# حذف عمود التقسيم المؤقت فقط
# =====

sample_df = sample_df.drop(
    columns=["stratify_group"]
)

# =====
# حفظ العينة
# =====

OUTPUT_FILE = (
    "/content/drive/MyDrive/Thesis_Project/"
    "emails_training_sample_50k.csv"
)

sample_df.to_csv(
    OUTPUT_FILE,
    index=False
)

# =====
# تقرير نهائي
# =====

print("\n")

```

```

print("=" * 60)
print("SAMPLE CREATED")
print("=" * 60)

print("Original Dataset :", len(df))
print("Sample Size      :", len(sample_df))

print("\nSaved File:")
print(OUTPUT_FILE)

# =====
# التحقق من توزيع الأولويات
# =====

print("\n")
print("=" * 60)
print("PRIORITY DISTRIBUTION CHECK")
print("=" * 60)

print(
    (
        sample_df["priority"]
        .value_counts(normalize=True)
        * 100
    ).round(3)
)

# =====
# التحقق من توزيع عدد التصنيفات
# =====

print("\n")
print("=" * 60)
print("LABEL COUNT DISTRIBUTION CHECK")
print("=" * 60)

print(
    (
        sample_df["label_count"]
        .value_counts(normalize=True)
        .sort_index()
        * 100
    ).round(3)
)

# =====
# التحقق من عدم وجود تكرارات
# =====

total_records = len(sample_df)
unique_emails = sample_df["Cleaned_Text"].nunique()

print("\n")
print("=" * 60)

```

```

print("UNIQUE EMAIL ANALYSIS")
print("=" * 60)

print("Total Records :", total_records)
print("Unique Emails :", unique_emails)
print("Duplicates      :", total_records - unique_emails)

=====
ORIGINAL DATASET
=====
(190280, 8)

=====
SAMPLE CREATED
=====
Original Dataset : 190280
Sample Size      : 50000

Saved File:
/content/drive/MyDrive/Thesis_Project/emails_training_sample_50k.csv

=====
PRIORITY DISTRIBUTION CHECK
=====
priority
Normal    51.462
Low       39.290
High      9.248
Name: proportion, dtype: float64

=====
LABEL COUNT DISTRIBUTION CHECK
=====
label_count
1    49.606
2    27.810
3    22.584
Name: proportion, dtype: float64

=====
UNIQUE EMAIL ANALYSIS
=====
Total Records : 50000
Unique Emails : 50000
Duplicates    : 0

تقسيم العينة إلى (Train / Validation / Hold-out)

# =====
# تقسيم عينة الـ 50 ألف إيميل إلى:
# Train (70%) + Validation (10%) + Hold-out Test (20%)
# =====
#

```

```

# الهدف:
# إنشاء مجموعات البيانات النهائية المستخدمة في المشروع
#
# المنهجية:
# للحفاظ على توزيع البيانات Stratified Split استخدام 1-
# الاعتماد على 2-:
# Priority + Label Count
# Stratification كمتغير
# تقسيم البيانات إلى 3-:
# Train Set = 70% (35,000)
# Validation Set = 10% (5,000)
# Hold-out Test = 20% (10,000)
# بالكامل وعدم استخدامها أثناء Hold-out Test عزل 4-:
# - التدريب
# - ضبط المعلمات
# - اختيار النموذج
# استخدامها فقط للتقييم النهائي والمقارنة بين 5-:
# Baseline ML Models
# BERT Model
#
# ملاحظة:
# Label_count يستخدم لأغراض التحليل والتقسيم فقط،
# أثناء التدريب (Feature) ولن يستخدم كميزة
#
# =====

import pandas as pd
from sklearn.model_selection import train_test_split

# =====
# تحميل العينة النهائية
# =====

file_path = (
    "/content/drive/MyDrive/Thesis_Project/"
    "emails_training_sample_50k.csv"
)

df = pd.read_csv(file_path)

print("=" * 60)
print("WORKING DATASET")
print("=" * 60)
print("Dataset Shape:", df.shape)

# =====
# Label_count التحقق من وجود
# =====

if "label_count" not in df.columns:
    raise ValueError(
        "Column 'label_count' not found in dataset."
    )

```

```

# =====
# إنشاء عمود Stratification
# =====
#
# دمج:
# Priority + Label Count
#
# أمثلة:
# Normal_1
# Normal_2
# High_1
# Low_3
#
# =====

df["stratify_col"] = (
    df["priority"].astype(str)
    + "_"
    + df["label_count"].astype(str)
)

# =====
# المرحلة الأولى:
# إنشاء Hold-out Test Set (20%)
# =====

train_val_df, holdout_df = train_test_split(
    df,
    test_size=0.20,
    random_state=42,
    stratify=df["stratify_col"]
)

# =====
# المرحلة الثانية:
# إنشاء Train (70%) و Validation (10%)
#
# 80 = المتبقي%
# Validation 10 = المطلوبة%
#
#  $10 / 80 = 0.125$ 
#
# =====

train_df, val_df = train_test_split(
    train_val_df,
    test_size=0.125,
    random_state=42,
    stratify=train_val_df["stratify_col"]
)

# =====
# المؤقت فقط Stratification حذف عمود
# =====

```

```

for dataset in [train_df, val_df, holdout_df]:
    dataset.drop(
        columns=["stratify_col"],
        inplace=True
    )

# =====
# أحجام المجموعات النهائية
# =====

print("\n" + "=" * 60)
print("FINAL DATA SPLIT")
print("=" * 60)

print(f"Train Set      : {len(train_df):,}")
print(f"Validation Set : {len(val_df):,}")
print(f"Hold-out Test  : {len(holdout_df):,}")

print("\nPercentages:")

print(
    f"Train      : {(len(train_df)/len(df))*100:.2f}%"
)

print(
    f"Validation : {(len(val_df)/len(df))*100:.2f}%"
)

print(
    f"Hold-out   : {(len(holdout_df)/len(df))*100:.2f}%"
)

# =====
# التحقق من توزيع الأولوية
# =====

print("\n" + "=" * 60)
print("PRIORITY DISTRIBUTION CHECK")
print("=" * 60)

print("\nTRAIN SET")
print(
    (
        train_df["priority"]
        .value_counts(normalize=True)
        * 100
    ).round(3)
)

print("\nVALIDATION SET")
print(
    (
        val_df["priority"]

```

```

        .value_counts(normalize=True)
        * 100
    ).round(3)
)

print("\nHOLD-OUT TEST SET")
print(
    (
        holdout_df["priority"]
        .value_counts(normalize=True)
        * 100
    ).round(3)
)

# =====
# التحقق من توزيع عدد التصنيفات
# =====

print("\n" + "=" * 60)
print("LABEL COUNT DISTRIBUTION CHECK")
print("=" * 60)

print("\nTRAIN SET")
print(
    (
        train_df["label_count"]
        .value_counts(normalize=True)
        .sort_index()
        * 100
    ).round(3)
)

print("\nVALIDATION SET")
print(
    (
        val_df["label_count"]
        .value_counts(normalize=True)
        .sort_index()
        * 100
    ).round(3)
)

print("\nHOLD-OUT TEST SET")
print(
    (
        holdout_df["label_count"]
        .value_counts(normalize=True)
        .sort_index()
        * 100
    ).round(3)
)

# =====
# Data Leakage التحقق من عدم وجود

```

```

# =====

train_texts = set(train_df["Cleaned_Text"])
val_texts = set(val_df["Cleaned_Text"])
holdout_texts = set(holdout_df["Cleaned_Text"])

print("\n" + "=" * 60)
print("DATA LEAKAGE CHECK")
print("=" * 60)

print(
    "Train vs Validation :",
    len(train_texts & val_texts)
)

print(
    "Train vs Holdout      :",
    len(train_texts & holdout_texts)
)

print(
    "Validation vs Holdout :",
    len(val_texts & holdout_texts)
)

# =====
# حفظ الملفات
# =====

train_df.to_csv(
    "/content/drive/MyDrive/Thesis_Project/train_set.csv",
    index=False
)

val_df.to_csv(
    "/content/drive/MyDrive/Thesis_Project/validation_set.csv",
    index=False
)

holdout_df.to_csv(
    "/content/drive/MyDrive/Thesis_Project/holdout_test_set.csv",
    index=False
)

# =====
# تأكيد الحفظ
# =====

print("\n" + "=" * 60)
print("FILES SAVED SUCCESSFULLY")
print("=" * 60)

print("train_set.csv")

```

```
print("validation_set.csv")
print("holdout_test_set.csv")
```

```
=====
WORKING DATASET
=====
```

```
Dataset Shape: (50000, 9)
```

```
=====
FINAL DATA SPLIT
=====
```

```
Train Set      : 35,000
Validation Set  : 5,000
Hold-out Test   : 10,000
```

```
Percentages:
```

```
Train      : 70.00%
Validation : 10.00%
Hold-out    : 20.00%
```

```
=====
PRIORITY DISTRIBUTION CHECK
=====
```

```
TRAIN SET
```

```
priority
```

```
Normal    51.463
```

```
Low       39.289
```

```
High      9.249
```

```
Name: proportion, dtype: float64
```

```
VALIDATION SET
```

```
priority
```

```
Normal    51.46
```

```
Low       39.30
```

```
High      9.24
```

```
Name: proportion, dtype: float64
```

```
HOLD-OUT TEST SET
```

```
priority
```

```
Normal    51.46
```

```
Low       39.29
```

```
High      9.25
```

```
Name: proportion, dtype: float64
```

```
=====
LABEL COUNT DISTRIBUTION CHECK
=====
```

```
TRAIN SET
```

```
label_count
```

```
1    49.609
```

```
2    27.809
```

```
3    22.583
```

```
Name: proportion, dtype: float64
```

```
VALIDATION SET
```

```
label_count
1    49.60
2    27.82
3    22.58
Name: proportion, dtype: float64
```

HOLD-OUT TEST SET

```
label_count
1    49.60
2    27.81
3    22.59
Name: proportion, dtype: float64
```

```
=====
DATA LEAKAGE CHECK
=====
```

```
Train vs Validation : 0
Train vs Holdout    : 0
Validation vs Holdout : 0
```

```
=====
FILES SAVED SUCCESSFULLY
=====
```

```
train_set.csv
validation_set.csv
holdout_test_set.csv
```

تقرير مقارنة

```
# =====
# DATA SPLIT VALIDATION REPORT
# =====
```

```
import pandas as pd
```

```
# -----
# LOAD FILES
# -----
```

```
full_df = pd.read_csv(
    "/content/drive/MyDrive/Thesis_Project/emails_training_sample_50k.csv"
)
```

```
train_df = pd.read_csv(
    "/content/drive/MyDrive/Thesis_Project/train_set.csv"
)
```

```
val_df = pd.read_csv(
    "/content/drive/MyDrive/Thesis_Project/validation_set.csv"
)
```

```
holdout_df = pd.read_csv(
    "/content/drive/MyDrive/Thesis_Project/holdout_test_set.csv"
)
```

```
# -----
```

```
# BASIC INFORMATION
```

```
# -----
```

```
datasets = {  
    "Original": full_df,  
    "Train": train_df,  
    "Validation": val_df,  
    "Holdout": holdout_df  
}
```

```
print("=" * 70)  
print("DATASET SIZE COMPARISON")  
print("=" * 70)
```

```
for name, df in datasets.items():  
    print(f"{name:<12} {len(df):>8,}")
```

```
# -----
```

```
# TEXT LENGTH COMPARISON
```

```
# -----
```

```
print("\n" + "=" * 70)  
print("TEXT LENGTH COMPARISON")  
print("=" * 70)
```

```
for name, df in datasets.items():
```

```
    lengths = df["Cleaned_Text"].astype(str).str.len()
```

```
    print(f"\n{name}")  
    print(f"Mean Length : {lengths.mean():.2f}")  
    print(f"Median      : {lengths.median():.2f}")  
    print(f"Min Length  : {lengths.min()}")  
    print(f"Max Length  : {lengths.max()}")
```

```
# -----
```

```
# PRIORITY DISTRIBUTION
```

```
# -----
```

```
print("\n" + "=" * 70)  
print("PRIORITY DISTRIBUTION (%)")  
print("=" * 70)
```

```
for name, df in datasets.items():
```

```
    print(f"\n{name}")  
    print(  
        (  
            df["priority"]  
            .value_counts(normalize=True)  
            .sort_index()  
            * 100  
        ).round(2)  
    )
```

```

# -----
# CATEGORY DISTRIBUTION
# -----

category_cols = [
    "category_Administration",
    "category_Financial",
    "category_HR",
    "category_Legal",
    "category_Market",
    "category_Other"
]

print("\n" + "=" * 70)
print("CATEGORY DISTRIBUTION (%)")
print("=" * 70)

for name, df in datasets.items():

    print(f"\n{name}")

    dist = (
        df[category_cols]
        .mean()
        .sort_values(ascending=False)
        * 100
    )

    print(dist.round(2))

# -----
# MULTI-LABEL DISTRIBUTION
# -----

print("\n" + "=" * 70)
print("LABEL COUNT DISTRIBUTION (%)")
print("=" * 70)

for name, df in datasets.items():

    label_count = df[category_cols].sum(axis=1)

    print(f"\n{name}")

    print(
        (
            label_count
            .value_counts(normalize=True)
            .sort_index()
            * 100
        ).round(2)
    )

```

```

# -----
# DATA LEAKAGE CHECK
# -----

print("\n" + "=" * 70)
print("DATA LEAKAGE CHECK")
print("=" * 70)

train_texts = set(train_df["Cleaned_Text"])
val_texts = set(val_df["Cleaned_Text"])
holdout_texts = set(holdout_df["Cleaned_Text"])

train_val_overlap = len(train_texts & val_texts)
train_holdout_overlap = len(train_texts & holdout_texts)
val_holdout_overlap = len(val_texts & holdout_texts)

print("Train vs Validation :", train_val_overlap)
print("Train vs Holdout      :", train_holdout_overlap)
print("Validation vs Holdout:", val_holdout_overlap)

# -----
# FINAL SUMMARY
# -----

print("\n" + "=" * 70)
print("FINAL CHECK")
print("=" * 70)

total_split = len(train_df) + len(val_df) + len(holdout_df)

print("Original Records :", len(full_df))
print("Split Records      :", total_split)

if total_split == len(full_df):
    print("PASS: All records accounted for.")
else:
    print("WARNING: Record count mismatch.")

=====
DATASET SIZE COMPARISON
=====
Original      50,000
Train         35,000
Validation     5,000
Holdout       10,000

=====
TEXT LENGTH COMPARISON
=====

Original
Mean Length : 673.16
Median      : 421.00
Min Length  : 10
Max Length  : 5963

```

Train
Mean Length : 675.88
Median : 422.00
Min Length : 10
Max Length : 5963

Validation
Mean Length : 684.32
Median : 427.00
Min Length : 10
Max Length : 5928

Holdout
Mean Length : 658.05
Median : 415.00
Min Length : 10
Max Length : 5944

=====
PRIORITY DISTRIBUTION (%)
=====

Original
priority
High 9.25
Low 39.29
Normal 51.46
Name: proportion, dtype: float64

Train
priority
High 9.25
Low 39.29
Normal 51.46
Name: proportion, dtype: float64

Validation
priority
High 9.24
Low 39.30
Normal 51.46
Name: proportion, dtype: float64

Holdout
priority
High 9.25
Low 39.29
Normal 51.46
Name: proportion, dtype: float64

=====
CATEGORY DISTRIBUTION (%)
=====

Original
category_Administration 54.25

category_Market	35.50
category_Financial	24.94
category_Other	21.10
category_HR	19.53
category_Legal	17.65

dtype: float64

Train

category_Administration	54.21
category_Market	35.29
category_Financial	24.97
category_Other	21.16
category_HR	19.66
category_Legal	17.69

dtype: float64

Validation

category_Administration	54.26
category_Market	35.70
category_Financial	25.56
category_Other	20.98
category_HR	19.96
category_Legal	16.52

dtype: float64

Holdout

category_Administration	54.35
category_Market	36.14
category_Financial	24.56
category_Other	20.95
category_HR	18.89
category_Legal	18.10

dtype: float64

=====

LABEL COUNT DISTRIBUTION (%)

=====

Original

1	49.61
2	27.81
3	22.58

Name: proportion, dtype: float64

Train

1	49.61
2	27.81
3	22.58

Name: proportion, dtype: float64

Validation

1	49.60
2	27.82
3	22.58

Name: proportion, dtype: float64

Holdout

```
1    49.60
2    27.81
3    22.59
Name: proportion, dtype: float64
```

```
=====
DATA LEAKAGE CHECK
=====
```

```
Train vs Validation : 0
Train vs Holdout    : 0
Validation vs Holdout: 0
```

```
=====
FINAL CHECK
=====
```

```
Original Records : 50000
Split Records    : 50000
PASS: All records accounted for.
```

TF-IDF + OneVsRest Logistic Regression

Multi-label Classification Baseline Models

TF-IDF + Logistic Regression

```
# =====
# BASELINE MODEL #1
# TF-IDF + Logistic Regression
# =====
#
# الهدف:
# BERT للمقارنة لاحقاً مع Baseline إنشاء أول نموذج
#
# المنهجية:
#
# 1- استخدام مجموعة التدريب فقط للتعلم.
# 2- للتقييم Validation استخدام مجموعة.
# 3- تحويل النصوص إلى متجهات رقمية باستخدام TF-IDF.
# 4- Multi-Label Classification لمعالجة One-vs-Rest تطبيق.
# 5- Logistic Regression تدريب.
# 6- حساب المقاييس الأساسية.
#
# =====
```

```
import pandas as pd
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.multiclass import OneVsRestClassifier
```

```
from sklearn.metrics import (
```

```

        accuracy_score,
        precision_score,
        recall_score,
        f1_score,
        hamming_loss,
        jaccard_score
    )

# =====
# تحميل البيانات
# =====

train_df = pd.read_csv(
    "/content/drive/MyDrive/Thesis_Project/train_set.csv"
)

val_df = pd.read_csv(
    "/content/drive/MyDrive/Thesis_Project/validation_set.csv"
)

print("=" * 70)
print("DATASETS LOADED")
print("=" * 70)

print("Train Shape      :", train_df.shape)
print("Validation Shape :", val_df.shape)

# =====
# أعمدة التصنيف المستهدفة
# =====

category_columns = [
    "category_Administration",
    "category_Financial",
    "category_HR",
    "category_Legal",
    "category_Market",
    "category_Other"
]

# =====
# تقسيم Features و Labels
# =====

X_train = train_df["Cleaned_Text"]

X_val = val_df["Cleaned_Text"]

y_train = train_df[category_columns]

y_val = val_df[category_columns]

print("\n" + "=" * 70)
print("FEATURES AND LABELS READY")

```

```

print("=" * 70)

print("Training Samples :", len(X_train))
print("Validation Samples :", len(X_val))

# =====
# TF-IDF Vectorization
# =====
#
# max_features:
# الحد الأقصى للمفردات المستخدمة.
#
# ngram_range:
# استخدام الكلمات المفردة والثنائيات.
#
# min_df:
# تجاهل الكلمات النادرة جداً.
#
# max_df:
# تجاهل الكلمات الشائعة جداً.
#
# sublinear_tf:
# تحسين حساب التكرار.
#
# =====

vectorizer = TfidfVectorizer(
    max_features=20000,
    ngram_range=(1, 2),
    min_df=5,
    max_df=0.95,
    sublinear_tf=True
)

X_train_tfidf = vectorizer.fit_transform(X_train)

X_val_tfidf = vectorizer.transform(X_val)

print("\n" + "=" * 70)
print("TF-IDF COMPLETED")
print("=" * 70)

print("Vocabulary Size :", len(vectorizer.vocabulary_))
print("Train Matrix      :", X_train_tfidf.shape)
print("Validation Matrix :", X_val_tfidf.shape)

# =====
# Logistic Regression بناء نموذج
# =====
#
# OneVsRest:
# تحويل Logistic Regression إلى Multi-Label Classifier
#
# class_weight='balanced'

```

```

# للتعامل مع عدم توازن الفئات.
#
# =====

model = OneVsRestClassifier(
    LogisticRegression(
        max_iter=3000,
        class_weight="balanced",
        random_state=42
    )
)

print("\n" + "=" * 70)
print("TRAINING MODEL")
print("=" * 70)

model.fit(
    X_train_tfidf,
    y_train
)

print("Training Completed")

# =====
# Validation Set التنبؤ على
# =====

y_pred = model.predict(
    X_val_tfidf
)

# =====
# حساب المقاييس
# =====

subset_accuracy = accuracy_score(
    y_val,
    y_pred
)

micro_precision = precision_score(
    y_val,
    y_pred,
    average="micro",
    zero_division=0
)

micro_recall = recall_score(
    y_val,
    y_pred,
    average="micro",
    zero_division=0
)

```

```

micro_f1 = f1_score(
    y_val,
    y_pred,
    average="micro",
    zero_division=0
)

macro_precision = precision_score(
    y_val,
    y_pred,
    average="macro",
    zero_division=0
)

macro_recall = recall_score(
    y_val,
    y_pred,
    average="macro",
    zero_division=0
)

macro_f1 = f1_score(
    y_val,
    y_pred,
    average="macro",
    zero_division=0
)

ham_loss = hamming_loss(
    y_val,
    y_pred
)

jaccard = jaccard_score(
    y_val,
    y_pred,
    average="samples",
    zero_division=0
)

# =====
# عرض النتائج
# =====

print("\n")
print("=" * 70)
print("BASELINE MODEL RESULTS")
print("=" * 70)

print(f"Subset Accuracy : {subset_accuracy:.4f}")

print("\nMICRO METRICS")
print(f"Precision : {micro_precision:.4f}")
print(f"Recall : {micro_recall:.4f}")

```

```

print(f"F1 Score : {micro_f1:.4f}")

print("\nMACRO METRICS")
print(f"Precision : {macro_precision:.4f}")
print(f"Recall : {macro_recall:.4f}")
print(f"F1 Score : {macro_f1:.4f}")

print("\nOTHER METRICS")
print(f"Hamming Loss : {ham_loss:.4f}")
print(f"Jaccard Score: {jaccard:.4f}")

print("\n" + "=" * 70)
print("BASELINE #1 COMPLETED")
print("=" * 70)

=====
DATASETS LOADED
=====
Train Shape      : (35000, 9)
Validation Shape : (5000, 9)

=====
FEATURES AND LABELS READY
=====
Training Samples : 35000
Validation Samples : 5000

=====
TF-IDF COMPLETED
=====
Vocabulary Size : 20000
Train Matrix    : (35000, 20000)
Validation Matrix : (5000, 20000)

=====
TRAINING MODEL
=====
Training Completed

=====
BASELINE MODEL RESULTS
=====
Subset Accuracy : 0.6840

MICRO METRICS
Precision : 0.8711
Recall    : 0.8925
F1 Score  : 0.8817

MACRO METRICS
Precision : 0.8601
Recall    : 0.8958
F1 Score  : 0.8751

OTHER METRICS

```

Hamming Loss : 0.0691
Jaccard Score: 0.8280

```
=====
BASELINE #1 COMPLETED
=====
```

TF-IDF + Linear Support Vector Machine (Linear SVM)

```
# =====
# BASELINE MODEL #2
# TF-IDF + Linear Support Vector Machine (Linear SVM)
# =====
#
# الهدف:
# TF-IDF باستخدام نفس تمثيل Linear SVM تدريب نموذج
# Logistic Regression. المستخدم مع
#
# سبب الاختيار:
# من أقوى النماذج التقليدية Linear SVM يعتبر
# في تصنيف النصوص عالية الأبعاد
#
# سيتم استخدام:
# - نفس بيانات التدريب
# - نفس بيانات التحقق
# - نفس إعدادات TF-IDF
#
# Baseline #1. لضمان عدالة المقارنة مع
#
# =====

import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.svm import LinearSVC

from sklearn.multiclass import OneVsRestClassifier

from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    hamming_loss,
    jaccard_score
)

# =====
# تحميل البيانات
# =====

train_df = pd.read_csv(
    "/content/drive/MyDrive/Thesis_Project/train_set.csv"
)
```

```

val_df = pd.read_csv(
    "/content/drive/MyDrive/Thesis_Project/validation_set.csv"
)

print("=" * 70)
print("DATASETS LOADED")
print("=" * 70)

print("Train Shape      :", train_df.shape)
print("Validation Shape :", val_df.shape)

# =====
# أعمدة التصنيفات المستهدفة
# =====

category_columns = [
    "category_Administration",
    "category_Financial",
    "category_HR",
    "category_Legal",
    "category_Market",
    "category_Other"
]

# =====
# تجهيز Features و Labels
# =====

X_train = train_df["Cleaned_Text"]

X_val = val_df["Cleaned_Text"]

y_train = train_df[category_columns]

y_val = val_df[category_columns]

# =====
# TF-IDF Vectorization
# =====
#
# يتم استخدام نفس الإعدادات المستخدمة
# لضمان عدالة المقارنة Logistic Regression في
#
# =====

vectorizer = TfidfVectorizer(
    max_features=20000,
    ngram_range=(1, 2),
    min_df=5,
    max_df=0.95,
    sublinear_tf=True
)

```

```

X_train_tfidf = vectorizer.fit_transform(X_train)

X_val_tfidf = vectorizer.transform(X_val)

print("\n" + "=" * 70)
print("TF-IDF COMPLETED")
print("=" * 70)

print("Vocabulary Size :", len(vectorizer.vocabulary_))
print("Train Matrix   :", X_train_tfidf.shape)
print("Validation Matrix :", X_val_tfidf.shape)

# =====
# بناء Linear SVM نموذج
# =====
#
# إلى Linear SVM يحول OneVsRest
# Multi-Label Classifier.
#
# =====

model = OneVsRestClassifier(
    LinearSVC(
        random_state=42
    )
)

print("\n" + "=" * 70)
print("TRAINING MODEL")
print("=" * 70)

model.fit(
    X_train_tfidf,
    y_train
)

print("Training Completed")

# =====
# Validation Set التنبؤ على
# =====

y_pred = model.predict(
    X_val_tfidf
)

# =====
# حساب المقاييس
# =====

subset_accuracy = accuracy_score(
    y_val,
    y_pred
)

```

```
micro_precision = precision_score(  
    y_val,  
    y_pred,  
    average="micro",  
    zero_division=0  
)
```

```
micro_recall = recall_score(  
    y_val,  
    y_pred,  
    average="micro",  
    zero_division=0  
)
```

```
micro_f1 = f1_score(  
    y_val,  
    y_pred,  
    average="micro",  
    zero_division=0  
)
```

```
macro_precision = precision_score(  
    y_val,  
    y_pred,  
    average="macro",  
    zero_division=0  
)
```

```
macro_recall = recall_score(  
    y_val,  
    y_pred,  
    average="macro",  
    zero_division=0  
)
```

```
macro_f1 = f1_score(  
    y_val,  
    y_pred,  
    average="macro",  
    zero_division=0  
)
```

```
ham_loss = hamming_loss(  
    y_val,  
    y_pred  
)
```

```
jaccard = jaccard_score(  
    y_val,  
    y_pred,  
    average="samples",  
    zero_division=0  
)
```

```

# =====
# عرض النتائج
# =====

print("\n")
print("=" * 70)
print("BASELINE MODEL RESULTS")
print("=" * 70)

print(f"Subset Accuracy : {subset_accuracy:.4f}")

print("\nMICRO METRICS")
print(f"Precision : {micro_precision:.4f}")
print(f"Recall : {micro_recall:.4f}")
print(f"F1 Score : {micro_f1:.4f}")

print("\nMACRO METRICS")
print(f"Precision : {macro_precision:.4f}")
print(f"Recall : {macro_recall:.4f}")
print(f"F1 Score : {macro_f1:.4f}")

print("\nOTHER METRICS")
print(f"Hamming Loss : {ham_loss:.4f}")
print(f"Jaccard Score: {jaccard:.4f}")

print("\n" + "=" * 70)
print("BASELINE #2 COMPLETED")
print("=" * 70)

=====
DATASETS LOADED
=====

Train Shape : (35000, 9)
Validation Shape : (5000, 9)

=====
TF-IDF COMPLETED
=====

Vocabulary Size : 20000
Train Matrix : (35000, 20000)
Validation Matrix : (5000, 20000)

=====
TRAINING MODEL
=====

Training Completed

=====
BASELINE MODEL RESULTS
=====

Subset Accuracy : 0.8092

MICRO METRICS
Precision : 0.9691

```

Recall : 0.8993
F1 Score : 0.9329

MACRO METRICS

Precision : 0.9697
Recall : 0.8884
F1 Score : 0.9267

OTHER METRICS

Hamming Loss : 0.0373
Jaccard Score: 0.8870

=====

BASELINE #2 COMPLETED

=====

TF-IDF + Random Forest

```
# =====
# BASELINE MODEL #3
# TF-IDF + Random Forest
# =====
#
# الهدف:
# كخط أساس ثالث للمقارنة Random Forest بناء نموذج
# مع Logistic Regression و Linear SVM.
#
# المنهجية:
#
# 1- Train Set و Validation Set استخدام نفس
# 2- TF-IDF استخدام نفس إعدادات
# 3- Multi-Label Classification لدعم One-vs-Rest استخدام
# 4- حساب نفس المقاييس المستخدمة سابقاً
#
# ملاحظة:
# لا يُعد الخيار الأفضل عادةً لتصنيف Random Forest
# النصوص عالية الأبعاد، ولكنه يُستخدم كمرجع للمقارنة
#
# =====
```

```
import pandas as pd
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.multiclass import OneVsRestClassifier
```

```
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    hamming_loss,
    jaccard_score
```

```

)

# =====
# تحميل البيانات
# =====

train_df = pd.read_csv(
    "/content/drive/MyDrive/Thesis_Project/train_set.csv"
)

val_df = pd.read_csv(
    "/content/drive/MyDrive/Thesis_Project/validation_set.csv"
)

print("=" * 70)
print("DATASETS LOADED")
print("=" * 70)

print("Train Shape      :", train_df.shape)
print("Validation Shape :", val_df.shape)

# =====
# أعمدة التصنيف المستهدفة
# =====

category_columns = [
    "category_Administration",
    "category_Financial",
    "category_HR",
    "category_Legal",
    "category_Market",
    "category_Other"
]

# =====
# تجهيز Features و Labels
# =====

X_train = train_df["Cleaned_Text"]

X_val = val_df["Cleaned_Text"]

y_train = train_df[category_columns]

y_val = val_df[category_columns]

# =====
# TF-IDF Vectorization
# =====
#
# Baseline Models نفس الإعدادات المستخدمة في جميع
# لضمان عدالة المقارنة.
#
# =====

```

```

vectorizer = TfidfVectorizer(
    max_features=20000,
    ngram_range=(1, 2),
    min_df=5,
    max_df=0.95,
    sublinear_tf=True
)

X_train_tfidf = vectorizer.fit_transform(X_train)

X_val_tfidf = vectorizer.transform(X_val)

print("\n" + "=" * 70)
print("TF-IDF COMPLETED")
print("=" * 70)

print("Vocabulary Size :", len(vectorizer.vocabulary_))
print("Train Matrix      :", X_train_tfidf.shape)
print("Validation Matrix :", X_val_tfidf.shape)

# =====
# بناء Random Forest نموذج
# =====
#
# n_estimators:
# عدد الأشجار داخل الغابة.
#
# max_depth:
# تقييد العمق لتقليل استهلاك الذاكرة والوقت.
#
# n_jobs=-1:
# استخدام جميع الأنوية المتاحة.
#
# =====

model = OneVsRestClassifier(
    RandomForestClassifier(
        n_estimators=200,
        max_depth=30,
        random_state=42,
        n_jobs=-1
    )
)

print("\n" + "=" * 70)
print("TRAINING MODEL")
print("=" * 70)

model.fit(
    X_train_tfidf,
    y_train
)

```

```

print("Training Completed")

# =====
# Validation Set
# =====

y_pred = model.predict(
    X_val_tfidf
)

# =====
# حساب المقاييس
# =====

subset_accuracy = accuracy_score(
    y_val,
    y_pred
)

micro_precision = precision_score(
    y_val,
    y_pred,
    average="micro",
    zero_division=0
)

micro_recall = recall_score(
    y_val,
    y_pred,
    average="micro",
    zero_division=0
)

micro_f1 = f1_score(
    y_val,
    y_pred,
    average="micro",
    zero_division=0
)

macro_precision = precision_score(
    y_val,
    y_pred,
    average="macro",
    zero_division=0
)

macro_recall = recall_score(
    y_val,
    y_pred,
    average="macro",
    zero_division=0
)

```

```

macro_f1 = f1_score(
    y_val,
    y_pred,
    average="macro",
    zero_division=0
)

ham_loss = hamming_loss(
    y_val,
    y_pred
)

jaccard = jaccard_score(
    y_val,
    y_pred,
    average="samples",
    zero_division=0
)

# =====
# عرض النتائج
# =====

print("\n")
print("=" * 70)
print("BASELINE MODEL RESULTS")
print("=" * 70)

print(f"Subset Accuracy : {subset_accuracy:.4f}")

print("\nMICRO METRICS")
print(f"Precision : {micro_precision:.4f}")
print(f"Recall : {micro_recall:.4f}")
print(f"F1 Score : {micro_f1:.4f}")

print("\nMACRO METRICS")
print(f"Precision : {macro_precision:.4f}")
print(f"Recall : {macro_recall:.4f}")
print(f"F1 Score : {macro_f1:.4f}")

print("\nOTHER METRICS")
print(f"Hamming Loss : {ham_loss:.4f}")
print(f"Jaccard Score: {jaccard:.4f}")

print("\n" + "=" * 70)
print("BASELINE #3 COMPLETED")
print("=" * 70)

=====
DATASETS LOADED
=====
Train Shape : (35000, 9)
Validation Shape : (5000, 9)
=====

```

TF-IDF COMPLETED

Vocabulary Size : 20000
Train Matrix : (35000, 20000)
Validation Matrix : (5000, 20000)

TRAINING MODEL

Training Completed

BASELINE MODEL RESULTS

Subset Accuracy : 0.3442

MICRO METRICS

Precision : 0.9005
Recall : 0.5524
F1 Score : 0.6848

MACRO METRICS

Precision : 0.9501
Recall : 0.4700
F1 Score : 0.6024

OTHER METRICS

Hamming Loss : 0.1466
Jaccard Score: 0.4980

BASELINE #3 COMPLETED

Priority Classification Baseline Models

(Priority Classification) باستخدام TF-IDF + Logistic Regression

```
# =====  
# PHASE 3B  
# PRIORITY CLASSIFICATION  
#  
# BASELINE MODEL #1  
# TF-IDF + LOGISTIC REGRESSION  
# =====  
#  
# الهدف:  
# لتحديد أولوية البريد الإلكتروني Baseline بناء أول نموذج  
#  
# المهمة:  
#  
# Email  
# ↓  
# Low / Normal / High  
#
```

```

# نوع المشكلة:
#
# Multi-Class Classification
#
# لأن كل رسالة تمتلك أولوية واحدة فقط.
#
# سيتم استخدام:
#
# Train Set      = 35,000
# Validation Set = 5,000
#
# حالياً Hold-out Test ولن يتم استخدام
#
# =====

import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

# =====
# تحميل البيانات
# =====

train_df = pd.read_csv(
    "/content/drive/MyDrive/Thesis_Project/train_set.csv"
)

val_df = pd.read_csv(
    "/content/drive/MyDrive/Thesis_Project/validation_set.csv"
)

print("=" * 70)
print("DATASETS LOADED")
print("=" * 70)

print("Train Shape      :", train_df.shape)
print("Validation Shape :", val_df.shape)

# =====
# تجهيز النصوص
# =====

```

```

X_train = train_df["Cleaned_Text"]

X_val = val_df["Cleaned_Text"]

# =====
# تجهيز الأولوية
# =====
#
# تحويل:
#
# Low
# Normal
# High
#
# إلى أرقام.
#
# =====

encoder = LabelEncoder()

y_train = encoder.fit_transform(
    train_df["priority"]
)

y_val = encoder.transform(
    val_df["priority"]
)

print("\nPriority Classes:")
print(encoder.classes_)

# =====
# TF-IDF
# =====
#
# نفس الإعدادات المستخدمة سابقاً
# لضمان عدالة المقارنة.
#
# =====

vectorizer = TfidfVectorizer(
    max_features=20000,
    ngram_range=(1, 2),
    min_df=5,
    max_df=0.95,
    sublinear_tf=True
)

X_train_tfidf = vectorizer.fit_transform(
    X_train
)

X_val_tfidf = vectorizer.transform(

```

```

        X_val
    )

    print("\n" + "=" * 70)
    print("TF-IDF COMPLETED")
    print("=" * 70)

    print("Vocabulary Size :", len(vectorizer.vocabulary_))
    print("Train Matrix      :", X_train_tfidf.shape)
    print("Validation Matrix :", X_val_tfidf.shape)

    # =====
    # بناء النموذج
    # =====

    model = LogisticRegression(
        max_iter=2000,
        random_state=42,
        n_jobs=-1
    )

    print("\n" + "=" * 70)
    print("TRAINING MODEL")
    print("=" * 70)

    model.fit(
        X_train_tfidf,
        y_train
    )

    print("Training Completed")

    # =====
    # التنبؤ
    # =====

    y_pred = model.predict(
        X_val_tfidf
    )

    # =====
    # المقاييس
    # =====

    accuracy = accuracy_score(
        y_val,
        y_pred
    )

    precision = precision_score(
        y_val,
        y_pred,
        average="weighted"
    )

```

```

recall = recall_score(
    y_val,
    y_pred,
    average="weighted"
)

f1 = f1_score(
    y_val,
    y_pred,
    average="weighted"
)

# =====
# النتائج
# =====

print("\n")
print("=" * 70)
print("BASELINE MODEL RESULTS")
print("=" * 70)

print(f"Accuracy : {accuracy:.4f}")
print(f"Precision : {precision:.4f}")
print(f"Recall : {recall:.4f}")
print(f"F1 Score : {f1:.4f}")

# =====
# تقرير تفصيلي
# =====

print("\n")
print("=" * 70)
print("CLASSIFICATION REPORT")
print("=" * 70)

print(
    classification_report(
        y_val,
        y_pred,
        target_names=encoder.classes_
    )
)

# =====
# مصفوفة الالتباس
# =====

print("\n")
print("=" * 70)
print("CONFUSION MATRIX")
print("=" * 70)

print(

```

```

        confusion_matrix(
            y_val,
            y_pred
        )
    )

print("\n" + "=" * 70)
print("PRIORITY BASELINE #1 COMPLETED")
print("=" * 70)

=====
DATASETS LOADED
=====
Train Shape      : (35000, 9)
Validation Shape : (5000, 9)

Priority Classes:
['High' 'Low' 'Normal']

=====
TF-IDF COMPLETED
=====
Vocabulary Size : 20000
Train Matrix    : (35000, 20000)
Validation Matrix : (5000, 20000)

=====
TRAINING MODEL
=====
Training Completed

=====
BASELINE MODEL RESULTS
=====
Accuracy : 0.8716
Precision : 0.8711
Recall    : 0.8716
F1 Score  : 0.8640

=====
CLASSIFICATION REPORT
=====

```

	precision	recall	f1-score	support
High	0.86	0.43	0.57	462
Low	0.87	0.89	0.88	1965
Normal	0.88	0.94	0.91	2573
accuracy			0.87	5000
macro avg	0.87	0.75	0.79	5000
weighted avg	0.87	0.87	0.86	5000

```
=====
CONFUSION MATRIX
=====
```

```
[[ 198  128  136]
 [   18 1744  203]
 [   13  144 2416]]
```

```
=====
PRIORITY BASELINE #1 COMPLETED
=====
```

```
PRIORITY CLASSIFICATION TF-IDF + LINEAR SVM
```

```
# =====
# PHASE 3B
# PRIORITY CLASSIFICATION
#
# BASELINE MODEL #2
# TF-IDF + LINEAR SVM
# =====
#
# الهدف:
# Linear Support Vector Machine تدريب نموذج
# لتحديد أولوية البريد الإلكتروني.
#
# المهمة:
#
# Email
# ↓
# Low / Normal / High
#
# نوع المشكلة:
#
# Multi-Class Classification
#
# سيتم استخدام نفس:
#
# Train Set      = 35,000
# Validation Set = 5,000
#
# المستخدمة في TF-IDF ونفس إعدادات
# Logistic Regression المقارنة لضمن عدالة.
#
# =====
```

```
import pandas as pd
```

```
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.svm import LinearSVC
```

```
from sklearn.metrics import (
    accuracy_score,
```

```

        precision_score,
        recall_score,
        f1_score,
        classification_report,
        confusion_matrix
    )

# =====
# تحميل البيانات
# =====

train_df = pd.read_csv(
    "/content/drive/MyDrive/Thesis_Project/train_set.csv"
)

val_df = pd.read_csv(
    "/content/drive/MyDrive/Thesis_Project/validation_set.csv"
)

print("=" * 70)
print("DATASETS LOADED")
print("=" * 70)

print("Train Shape      :", train_df.shape)
print("Validation Shape :", val_df.shape)

# =====
# النصوص
# =====

X_train = train_df["Cleaned_Text"]

X_val = val_df["Cleaned_Text"]

# =====
# ترميز الأولويات
# =====

encoder = LabelEncoder()

y_train = encoder.fit_transform(
    train_df["priority"]
)

y_val = encoder.transform(
    val_df["priority"]
)

print("\nPriority Classes:")
print(encoder.classes_)

# =====
# TF-IDF
# =====

```

```

vectorizer = TfidfVectorizer(
    max_features=20000,
    ngram_range=(1, 2),
    min_df=5,
    max_df=0.95,
    sublinear_tf=True
)

X_train_tfidf = vectorizer.fit_transform(
    X_train
)

X_val_tfidf = vectorizer.transform(
    X_val
)

print("\n" + "=" * 70)
print("TF-IDF COMPLETED")
print("=" * 70)

print("Vocabulary Size :", len(vectorizer.vocabulary_))
print("Train Matrix      :", X_train_tfidf.shape)
print("Validation Matrix :", X_val_tfidf.shape)

# =====
# بناء Linear SVM
# =====
#
# هو الخيار القياسي للنصوص
# TF-IDF عالية الأبعاد الناتجة من.
#
# =====

model = LinearSVC(
    C=1.0,
    random_state=42
)

print("\n" + "=" * 70)
print("TRAINING MODEL")
print("=" * 70)

model.fit(
    X_train_tfidf,
    y_train
)

print("Training Completed")

# =====
# التنبؤ
# =====

y_pred = model.predict(

```

```

        X_val_tfidf
    )

# =====
# حساب المقاييس
# =====

accuracy = accuracy_score(
    y_val,
    y_pred
)

precision = precision_score(
    y_val,
    y_pred,
    average="weighted"
)

recall = recall_score(
    y_val,
    y_pred,
    average="weighted"
)

f1 = f1_score(
    y_val,
    y_pred,
    average="weighted"
)

# =====
# النتائج
# =====

print("\n")
print("=" * 70)
print("BASELINE MODEL RESULTS")
print("=" * 70)

print(f"Accuracy : {accuracy:.4f}")
print(f"Precision : {precision:.4f}")
print(f"Recall : {recall:.4f}")
print(f"F1 Score : {f1:.4f}")

# =====
# التقرير التفصيلي
# =====

print("\n")
print("=" * 70)
print("CLASSIFICATION REPORT")
print("=" * 70)

print(

```

```

        classification_report(
            y_val,
            y_pred,
            target_names=encoder.classes_
        )
    )

# =====
# مصفوفة الالتباس
# =====

print("\n")
print("=" * 70)
print("CONFUSION MATRIX")
print("=" * 70)

print(
    confusion_matrix(
        y_val,
        y_pred
    )
)

print("\n" + "=" * 70)
print("PRIORITY BASELINE #2 COMPLETED")
print("=" * 70)

=====
DATASETS LOADED
=====
Train Shape      : (35000, 9)
Validation Shape : (5000, 9)

Priority Classes:
['High' 'Low' 'Normal']

=====
TF-IDF COMPLETED
=====
Vocabulary Size : 20000
Train Matrix    : (35000, 20000)
Validation Matrix : (5000, 20000)

=====
TRAINING MODEL
=====
Training Completed

=====
BASELINE MODEL RESULTS
=====
Accuracy : 0.8988
Precision : 0.8981
Recall : 0.8988
F1 Score : 0.8968

```

```
=====
CLASSIFICATION REPORT
=====
```

	precision	recall	f1-score	support
High	0.87	0.66	0.75	462
Low	0.90	0.90	0.90	1965
Normal	0.90	0.94	0.92	2573
accuracy			0.90	5000
macro avg	0.89	0.83	0.86	5000
weighted avg	0.90	0.90	0.90	5000

```
=====
CONFUSION MATRIX
=====
```

```
[[ 304   73   85]
 [  27 1764  174]
 [  18  129 2426]]
```

```
=====
PRIORITY BASELINE #2 COMPLETED
=====
```

Transformer-Based Deep Learning Models Evaluation

تقييم نماذج المحولات (Transformers) للتصنيف متعدد الفئات

```
# =====
# STEP 1: MOUNT GOOGLE DRIVE  إعادة الوصول إلى Google Drive البيانات
# =====
```

```
from google.colab import drive
```

```
drive.mount('/content/drive')
```

```
print("Google Drive mounted successfully")
```

```
Mounted at /content/drive
Google Drive mounted successfully
```

```
# =====
# STEP 2: IMPORT LIBRARIES  استيراد المكتبات الأساسية
# =====
```

```
import pandas as pd
import numpy as np
```

```
import os

from sklearn.model_selection import train_test_split

print("Libraries imported successfully")
```

Libraries imported successfully

DISTILBERT MULTI-LABEL CLASSIFICATION

بناء النموذج

```
# =====
# DistilBERT Multi-Task Email Classification Model
# الهدف:
# 1) Email (Multi-Label) لكل Category توقع أكثر من 1
# 2) Email (Single-Label) واحدة لكل Priority توقع
#
# Input:
#     Cleaned_Text فقط
#
# Outputs:
#     Category probabilities (6 classes)
#     Priority class (High/Normal/Low)
# =====

# =====
# 1) Install Libraries
# =====

!pip install transformers accelerate -q

# =====
# 2) Imports
# =====

import torch
import torch.nn as nn

import pandas as pd
import numpy as np

from torch.utils.data import Dataset

from transformers import (
    AutoTokenizer,
    AutoModel
)

from sklearn.preprocessing import LabelEncoder

# =====
```

```

# 3) Check GPU
# =====

device = torch.device(
    "cuda" if torch.cuda.is_available()
    else "cpu"
)

print("Using device:", device)

# =====
# 4) Define Labels
# =====
#
# هذه الأعمدة ليست Input
# هي الإجابات الصحيحة التي يتعلم منها النموذج
#

category_cols = [
    "category_Administration",
    "category_Financial",
    "category_HR",
    "category_Legal",
    "category_Market",
    "category_Other"
]

# Multi-Label Category Targets

y_train_category = (
    train_df[category_cols]
    .values
    .astype(float)
)

y_val_category = (
    val_df[category_cols]
    .values
    .astype(float)
)

# =====
# Priority Encoding
# =====

priority_encoder = LabelEncoder()

y_train_priority = priority_encoder.fit_transform(
    train_df["priority"]
)

```

```

)

y_val_priority = priority_encoder.transform(
    val_df["priority"]
)

print(
    "Priority classes:",
    priority_encoder.classes_
)

# =====
# 5) DistilBERT Tokenizer
# =====

MODEL_NAME = "distilbert-base-uncased"

tokenizer = AutoTokenizer.from_pretrained(
    MODEL_NAME
)

# =====
# 6) Dataset Class
# =====
#
# النموذج يستقبل:
# النص فقط
#
# ويرجع:
# Labels للتعلم
#

class EmailDataset(Dataset):

    def __init__(
        self,
        texts,
        category_labels,
        priority_labels,
        tokenizer,
        max_length=256
    ):

        self.texts = texts.reset_index(drop=True)

        self.category_labels = category_labels

        self.priority_labels = priority_labels

```

```

        self.tokenizer = tokenizer

        self.max_length = max_length

def __len__(self):

    return len(self.texts)

def __getitem__(self, idx):

    text = str(
        self.texts.iloc[idx]
    )

    encoding = self.tokenizer(
        text,
        truncation=True,
        padding="max_length",
        max_length=self.max_length,
        return_tensors="pt"
    )

    return {

        "input_ids":
            encoding["input_ids"].squeeze(0),

        "attention_mask":
            encoding["attention_mask"].squeeze(0),

        # Multi-Label targets
        "category_labels":
            torch.tensor(
                self.category_labels[idx],
                dtype=torch.float
            ),

        # Single-Label target
        "priority_labels":
            torch.tensor(
                self.priority_labels[idx],
                dtype=torch.long
            )
    }

```

```

# =====
# 7) Create Dataset
# =====

train_dataset = EmailDataset(
    train_df["Cleaned_Text"],
    y_train_category,
    y_train_priority,
    tokenizer
)

val_dataset = EmailDataset(
    val_df["Cleaned_Text"],
    y_val_category,
    y_val_priority,
    tokenizer
)

print("Train samples:", len(train_dataset))
print("Validation samples:", len(val_dataset))

# =====
# 8) Build DistilBERT Multi-task Model
# =====
#
# DistilBERT:
# يتعلم تمثيل النص
#
# ثم نضيف رأسين:
#
# Head 1:
# Category prediction
#
# Head 2:
# Priority prediction
#

class DistilBERT_MultiTask(nn.Module):

    def __init__(
        self,
        model_name,
        num_categories,
        num_priorities
    ):

```

```

super().__init__()

self.encoder = AutoModel.from_pretrained(
    model_name
)

hidden_size = (
    self.encoder.config.hidden_size
)

# Category Head
self.category_head = nn.Linear(
    hidden_size,
    num_categories
)

# Priority Head
self.priority_head = nn.Linear(
    hidden_size,
    num_priorities
)

# Loss functions

# للفئات المتعددة
self.category_loss = (
    nn.BCEWithLogitsLoss()
)

# للأولوية الواحدة
self.priority_loss = (
    nn.CrossEntropyLoss()
)

def forward(
    self,
    input_ids,
    attention_mask,
    category_labels=None,
    priority_labels=None
):

    output = self.encoder(
        input_ids=input_ids,
        attention_mask=attention_mask

```

```

)

# تمثيل CLS
pooled = (
    output.last_hidden_state[:,0]
)

category_logits = (
    self.category_head(pooled)
)

priority_logits = (
    self.priority_head(pooled)
)

loss = None

if category_labels is not None:

    loss_category = (
        self.category_loss(
            category_logits,
            category_labels
        )
    )

    loss_priority = (
        self.priority_loss(
            priority_logits,
            priority_labels
        )
    )

    # دمج المهمتين
    loss = (
        loss_category
        +
        loss_priority
    )

return {

    "loss": loss,

    "category_logits":
        category_logits,

```

```

        "priority_logits":
            priority_logits
    }

```

```

# =====
# 9) Create Model
# =====

```

```

model = DistilBERT_MultiTask(
    MODEL_NAME,
    num_categories=len(category_cols),
    num_priorities=len(priority_encoder.classes_)
)

```

```

model.to(device)

```

```

print("Model ready")

```

```

Using device: cuda
Priority classes: ['High' 'Low' 'Normal']

```

```

Warning: You are sending unauthenticated requests to the HF Hub. Please set a
HF_TOKEN to enable higher rate limits and faster downloads.
WARNING:huggingface_hub.utils._http:Warning: You are sending unauthenticated
requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits
and faster downloads.

```

```

{"model_id":"ca7a3ae7a5c54e898eaf104254ddc881","version_major":2,"version_minor":0}

```

```

{"model_id":"f5f32da16bd74d22be4508e5e3210ba9","version_major":2,"version_minor":0}

```

```

{"model_id":"1b1bfb8fc09f41c7b804d5bf3b5e7113","version_major":2,"version_minor":0}

```

```

{"model_id":"e39595eff90140b184f6c3d2b0213ece","version_major":2,"version_minor":0}

```

```

Train samples: 35000
Validation samples: 5000

```

```

{"model_id":"0bf2cb0e32b84d4a9e9355d950bea73e","version_major":2,"version_minor":0}

```

```

{"model_id":"906e00d7bafc43c389d84fa7d89ebf50","version_major":2,"version_minor":0}

```

```

[transformers] DistilBertModel LOAD REPORT from: distilbert-base-uncased

```

Key	Status	
vocab_projector.bias	UNEXPECTED	
vocab_layer_norm.weight	UNEXPECTED	
vocab_transform.weight	UNEXPECTED	
vocab_layer_norm.bias	UNEXPECTED	
vocab_transform.bias	UNEXPECTED	

Notes:

- UNEXPECTED: can be ignored when loading from different task/architecture; not ok if you expect identical arch.

Model ready

DistilBERT Multi-task Training + Evaluation + Visualization

```
# =====  
# DistilBERT Multi-task Training + Evaluation + Visualization  
#  
# Input:  
#     Cleaned_Text فقط  
#  
# Outputs:  
#     1) Multi-Label Category Prediction  
#     2) Single-Label Priority Prediction  
#  
# =====
```

```
# =====  
# 1) Imports  
# =====
```

```
import torch  
import torch.nn as nn  
import numpy as np  
import matplotlib.pyplot as plt  
  
from torch.utils.data import DataLoader  
  
from torch.optim import AdamW  
  
from sklearn.metrics import (  
    f1_score,  
    precision_score,  
    recall_score,  
    accuracy_score,  
    hamming_loss,  
    classification_report,  
    confusion_matrix  
)  
  
import seaborn as sns
```

```
# =====  
# 2) DataLoaders  
# =====  
#  
#     batches تقسيم البيانات إلى  
#     لا يعالج كل البيانات مرة واحدة GPU
```

```

#

BATCH_SIZE = 16

train_loader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True
)

val_loader = DataLoader(
    val_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False
)

# =====
# 3) Training Settings
# =====

EPOCHS = 5

LEARNING_RATE = 2e-5

optimizer = AdamW(
    model.parameters(),
    lr=LEARNING_RATE
)

# Mixed Precision
# GPU لتسريع التدريب على

scaler = torch.cuda.amp.GradScaler()

# تخزين النتائج للرسم

history = {

    "train_loss": [],
    "val_loss": [],

    "category_f1": [],
    "priority_f1": []

}

```

```

best_f1 = 0

# =====
# 4) Training Loop
# =====

for epoch in range(EPOCHS):

    print("\n" + "="*70)

    print(
        f"Epoch {epoch+1}/{EPOCHS}"
    )

    print("="*70)

    # -----
    # Training
    # -----

    model.train()

    train_losses = []

    for batch in train_loader:

        optimizer.zero_grad()

        input_ids = (
            batch["input_ids"]
            .to(device)
        )

        attention_mask = (
            batch["attention_mask"]
            .to(device)
        )

        category_labels = (
            batch["category_labels"]
            .to(device)

```

```

    )

    priority_labels = (
        batch["priority_labels"]
        .to(device)
    )

    # FP16 training

    with torch.cuda.amp.autocast():

        output = model(

            input_ids=input_ids,

            attention_mask=attention_mask,

            category_labels=category_labels,

            priority_labels=priority_labels

        )

        loss = output["loss"]

    scaler.scale(loss).backward()

    scaler.step(optimizer)

    scaler.update()

    train_losses.append(
        loss.item()
    )

    avg_train_loss = np.mean(
        train_losses
    )

    # -----

```

```
# Validation
# -----

model.eval()

val_losses = []

all_category_preds = []
all_category_true = []

all_priority_preds = []
all_priority_true = []

with torch.no_grad():

    for batch in val_loader:

        input_ids = (
            batch["input_ids"]
            .to(device)
        )

        attention_mask = (
            batch["attention_mask"]
            .to(device)
        )

        category_labels = (
            batch["category_labels"]
            .to(device)
        )

        priority_labels = (
            batch["priority_labels"]
            .to(device)
        )

        output = model(

            input_ids=input_ids,
```

```

        attention_mask=attention_mask,

        category_labels=category_labels,

        priority_labels=priority_labels

    )

    val_losses.append(
        output["loss"].item()
    )

# Category prediction
#
# Sigmoid threshold = 0.5

    category_prob = torch.sigmoid(
        output["category_logits"]
    )

    category_pred = (
        category_prob > 0.5
    )

    all_category_preds.extend(
        category_pred.cpu().numpy()
    )

    all_category_true.extend(
        category_labels.cpu().numpy()
    )

# Priority prediction

    priority_pred = torch.argmax(
        output["priority_logits"],
        dim=1
    )

    all_priority_preds.extend(
        priority_pred.cpu().numpy()
    )

```

```

        all_priority_true.extend(
            priority_labels.cpu().numpy()
        )

    avg_val_loss = np.mean(
        val_losses
    )

    # =====
    # Metrics
    # =====

    category_f1 = f1_score(
        all_category_true,
        all_category_preds,
        average="macro"
    )

    priority_f1 = f1_score(
        all_priority_true,
        all_priority_preds,
        average="macro"
    )

    history["train_loss"].append(
        avg_train_loss
    )

    history["val_loss"].append(
        avg_val_loss
    )

    history["category_f1"].append(
        category_f1
    )

```

```

history["priority_f1"].append(
    priority_f1
)

print(
    f"Train Loss      : {avg_train_loss:.4f}"
)

print(
    f"Validation Loss : {avg_val_loss:.4f}"
)

print(
    f"Category Macro F1 : {category_f1:.4f}"
)

print(
    f"Priority Macro F1 : {priority_f1:.4f}"
)

# =====
# Save Best Model
# =====

combined_f1 = (
    category_f1
    +
    priority_f1
) / 2

if combined_f1 > best_f1:

    best_f1 = combined_f1

    torch.save(

        model.state_dict(),

        "best_distilbert_email_model.pt"

    )

```

```

        print(
            "Best model saved!"
        )

# =====
# 5) Final Evaluation
# =====

print("\nTraining Finished")

# -----
# Category Detailed Metrics
# -----

print("\nCATEGORY RESULTS")

print(
    "Hamming Loss:",
    hamming_loss(
        all_category_true,
        all_category_preds
    )
)

print(
    classification_report(
        all_category_true,
        all_category_preds,
        target_names=category_cols
    )
)

# -----
# Priority Detailed Metrics
# -----

print("\nPRIORITY RESULTS")

print(
    classification_report(

```

```

        all_priority_true,

        all_priority_preds,

        target_names=
        priority_encoder.classes_

    )
)

# =====
# 6) Visualization
# =====

# Loss Curve

plt.figure(figsize=(8,5))

plt.plot(
    history["train_loss"],
    label="Train Loss"
)

plt.plot(
    history["val_loss"],
    label="Validation Loss"
)

plt.xlabel("Epoch")

plt.ylabel("Loss")

plt.title(
    "Training and Validation Loss"
)

plt.legend()

plt.show()

# F1 Curve

plt.figure(figsize=(8,5))

plt.plot(
    history["category_f1"],
    label="Category Macro F1"
)

```

```
plt.plot(
    history["priority_f1"],
    label="Priority Macro F1"
)

plt.xlabel("Epoch")

plt.ylabel("F1 Score")

plt.title(
    "Validation F1 Scores"
)

plt.legend()

plt.show()
```

```
# Priority Confusion Matrix
```

```
cm = confusion_matrix(
    all_priority_true,
    all_priority_preds
)
```

```
plt.figure(figsize=(6,5))
```

```
sns.heatmap(
    cm,
    annot=True,
    fmt="d",
    xticklabels=priority_encoder.classes_,
    yticklabels=priority_encoder.classes_
)
```

```
plt.xlabel(
    "Predicted"
)
```

```
plt.ylabel(
    "Actual"
)
```

```
)  
  
plt.title(  
    "Priority Confusion Matrix"  
)  
  
plt.show()
```

```
# Category Performance
```

```
category_f1_scores = f1_score(  
    all_category_true,  
    all_category_preds,  
    average=None  
)
```

```
plt.figure(figsize=(10,5))
```

```
plt.bar(  
    category_cols,  
    category_f1_scores  
)
```

```
plt.xticks(  
    rotation=45,  
    ha="right"  
)
```

```
plt.ylabel(  
    "F1 Score"  
)
```

```
plt.title(  
    "Category-wise F1 Score"  
)
```

```
plt.show()
```

```
/tmp/ipykernel_545/758017852.py:86: FutureWarning:
`torch.cuda.amp.GradScaler(args...)` is deprecated. Please use
`torch.amp.GradScaler('cuda', args...)` instead.
    scaler = torch.cuda.amp.GradScaler()
/tmp/ipykernel_545/758017852.py:169: FutureWarning:
`torch.cuda.amp.autocast(args...)` is deprecated. Please use
`torch.amp.autocast('cuda', args...)` instead.
    with torch.cuda.amp.autocast():
```

```
=====
Epoch 1/5
=====
```

```
Train Loss      : 0.6249
Validation Loss : 0.3340
Category Macro F1 : 0.9315
Priority Macro F1 : 0.8919
Best model saved!
```

```
=====
Epoch 2/5
=====
```

```
/tmp/ipykernel_545/758017852.py:169: FutureWarning:
`torch.cuda.amp.autocast(args...)` is deprecated. Please use
`torch.amp.autocast('cuda', args...)` instead.
    with torch.cuda.amp.autocast():
```

```
Train Loss      : 0.2649
Validation Loss : 0.2317
Category Macro F1 : 0.9578
Priority Macro F1 : 0.9217
Best model saved!
```

```
=====
Epoch 3/5
=====
```

```
/tmp/ipykernel_545/758017852.py:169: FutureWarning:
`torch.cuda.amp.autocast(args...)` is deprecated. Please use
`torch.amp.autocast('cuda', args...)` instead.
    with torch.cuda.amp.autocast():
```

```
Train Loss      : 0.1866
Validation Loss : 0.2195
Category Macro F1 : 0.9655
Priority Macro F1 : 0.9206
Best model saved!
```

```
=====
Epoch 4/5
=====
```

```
/tmp/ipykernel_545/758017852.py:169: FutureWarning:
`torch.cuda.amp.autocast(args...)` is deprecated. Please use
`torch.amp.autocast('cuda', args...)` instead.
    with torch.cuda.amp.autocast():
```

Train Loss : 0.1340
Validation Loss : 0.2051
Category Macro F1 : 0.9677
Priority Macro F1 : 0.9350
Best model saved!

=====
Epoch 5/5
=====

/tmp/ipykernel_545/758017852.py:169: FutureWarning:
`torch.cuda.amp.autocast(args...)` is deprecated. Please use
`torch.amp.autocast('cuda', args...)` instead.
with torch.cuda.amp.autocast():

Train Loss : 0.0944
Validation Loss : 0.2458
Category Macro F1 : 0.9677
Priority Macro F1 : 0.9314

Training Finished

CATEGORY RESULTS

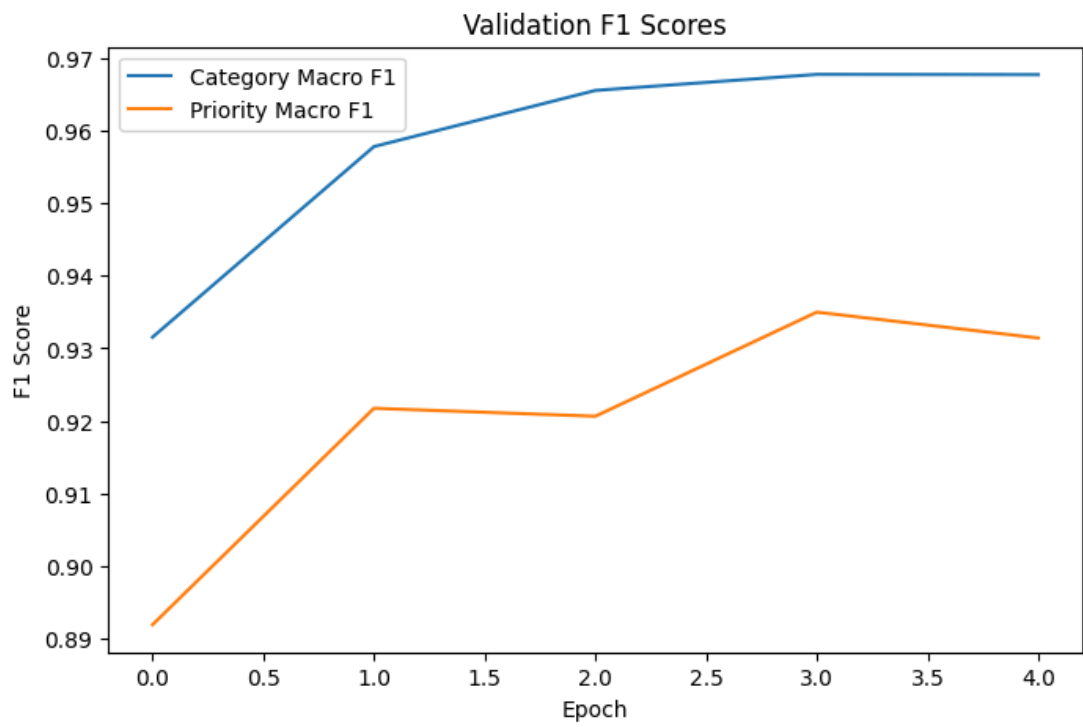
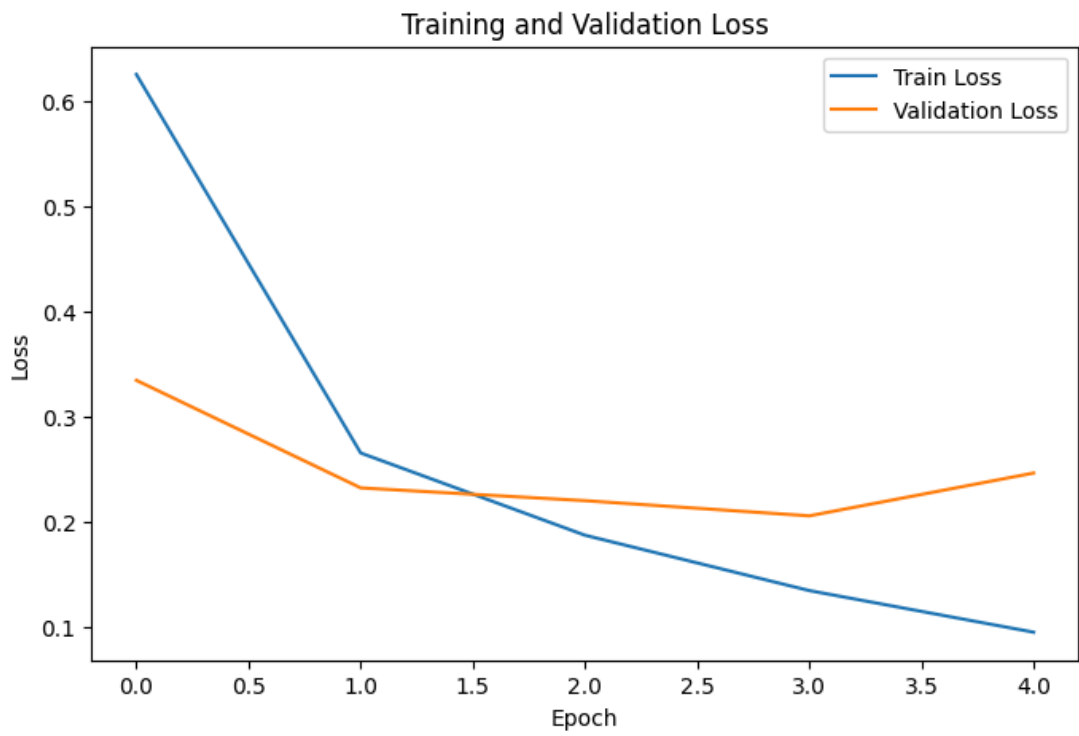
Hamming Loss: 0.017133333333333334

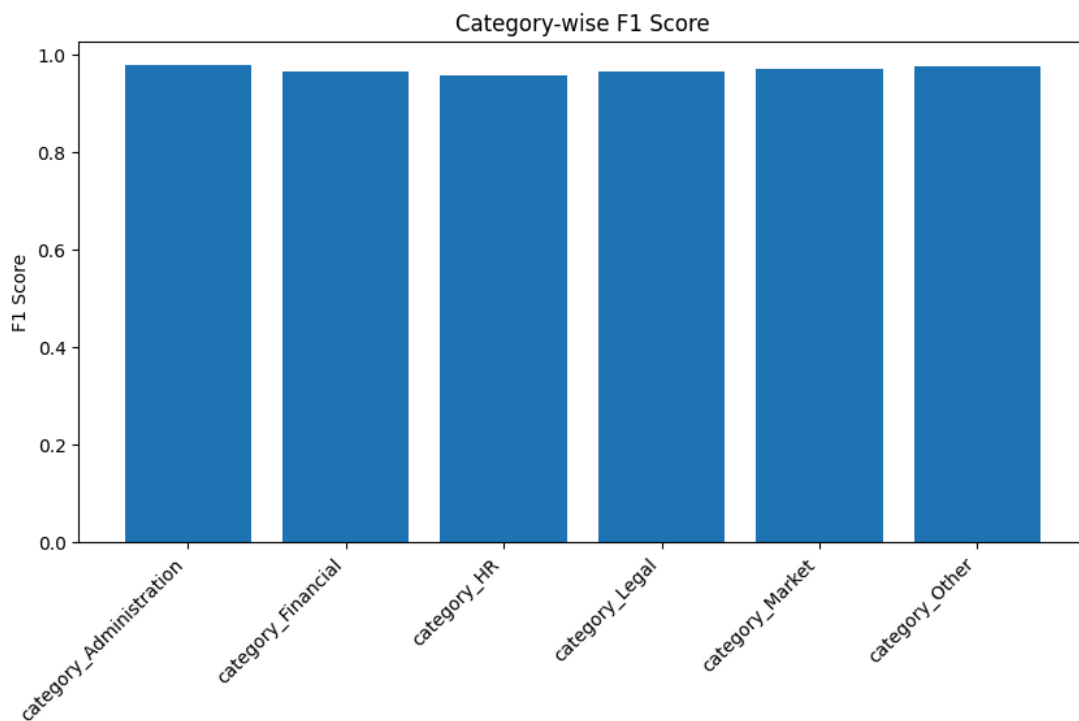
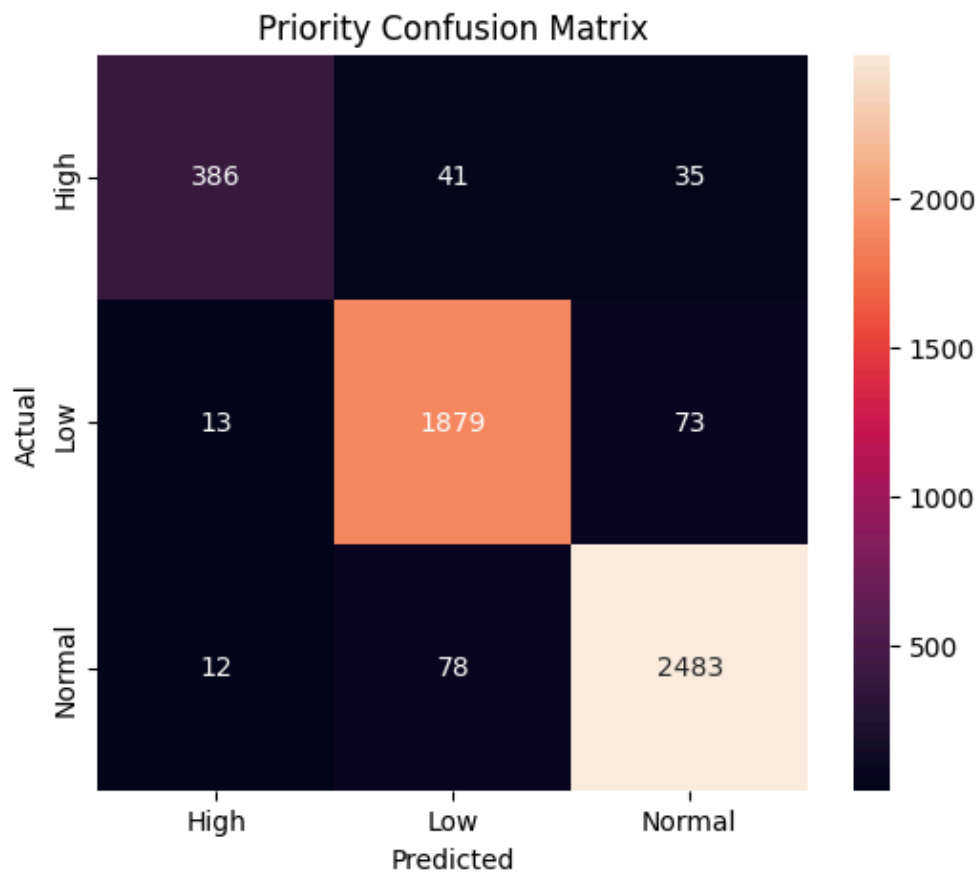
	precision	recall	f1-score	support
category_Administration	0.98	0.97	0.98	2713
category_Financial	0.99	0.94	0.96	1278
category_HR	0.99	0.93	0.96	998
category_Legal	0.99	0.94	0.96	826
category_Market	0.98	0.96	0.97	1785
category_Other	0.97	0.98	0.98	1049
micro avg	0.98	0.96	0.97	8649
macro avg	0.98	0.95	0.97	8649
weighted avg	0.98	0.96	0.97	8649
samples avg	0.98	0.96	0.97	8649

PRIORITY RESULTS

	precision	recall	f1-score	support
High	0.94	0.84	0.88	462
Low	0.94	0.96	0.95	1965
Normal	0.96	0.97	0.96	2573
accuracy			0.95	5000
macro avg	0.95	0.92	0.93	5000
weighted avg	0.95	0.95	0.95	5000

/usr/local/lib/python3.12/dist-
packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in samples with no predicted
labels. Use `zero\_division` parameter to control this behavior.
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))





DistilBERT لنموذج Threshold Tuning ضبط حدود القرار

```
# =====
# DistilBERT لنموذج Threshold Tuning ضبط حدود القرار
#
# الهدف:
# Category (Multi-label Classification) تحسين قرار 1)
```

```

# High Priority وخاصة Priority تحسين قرار 2)
#
# ملاحظات:
# لا يوجد تدريب هنا -
# أوزان النموذج لا تتغير -
# لاتخاذ القرار Threshold فقط نبحث عن أفضل -
#
# لأن المتوفر لدينا Validation Loader نستخدم
# val_loader
# =====

import numpy as np
import torch

from sklearn.metrics import (
    f1_score,
    classification_report,
    hamming_loss
)

# =====
# تحميل أفضل نموذج محفوظ 1)
# =====

model.load_state_dict(
    torch.load(
        "best_distilbert_email_model.pt",
        map_location=device
    )
)

model.eval()

print("بنجاح DistilBERT تم تحميل أفضل نموذج")

# =====
# Validation Set استخراج احتمالات النموذج من 2)
#
# Category:
# لأن الرسالة يمكن أن تحتوي أكثر من تصنيف Sigmoid
#
# Priority:
# لأن الرسالة لها أولوية واحدة فقط Softmax
#
# =====

category_probs = []
priority_probs = []

```

```
category_true = []  
priority_true = []
```

```
with torch.no_grad():
```

```
    # تم التعديل:  
    # test_loader بدلاً من val_loader استخدام
```

```
    for batch in val_loader:
```

```
        input_ids = batch["input_ids"].to(device)
```

```
        attention_mask = batch["attention_mask"].to(device)
```

```
        category_labels = batch["category_labels"]
```

```
        priority_labels = batch["priority_labels"]
```

```
        outputs = model(  
            input_ids=input_ids,  
            attention_mask=attention_mask  
        )
```

```
        # Category احتمالات
```

```
        cat_probability = torch.sigmoid(  
            outputs["category_logits"]  
        )
```

```
        # Priority احتمالات
```

```
        pri_probability = torch.softmax(  
            outputs["priority_logits"],  
            dim=1  
        )
```

```
        category_probs.append(  
            cat_probability.cpu().numpy()  
        )
```

```

        priority_probs.append(
            pri_probability.cpu().numpy()
        )

        category_true.append(
            category_labels.cpu().numpy()
        )

        priority_true.append(
            priority_labels.cpu().numpy()
        )

category_probs = np.vstack(category_probs)
priority_probs = np.vstack(priority_probs)

category_true = np.vstack(category_true)
priority_true = np.concatenate(priority_true)

print("تم استخراج احتمالات التنبؤ بنجاح")

# =====
# 3) Category Threshold Tuning
#
# القيمة الحالية غالباً:
#
# Threshold = 0.5
#
# Macro F1 نجرب عدة قيم ونختار الأفضل حسب
#
# =====

threshold_results = []

for threshold in np.arange(0.1,0.91,0.05):

    category_prediction = (
        category_probs >= threshold
    ).astype(int)

```

```

macro_f1 = f1_score(
    category_true,
    category_prediction,
    average="macro"
)

ham_loss = hamming_loss(
    category_true,
    category_prediction
)

threshold_results.append(
    (
        threshold,
        macro_f1,
        ham_loss
    )
)

best_category_threshold = max(
    threshold_results,
    key=lambda x:x[1]
)

print("\n=====")
print("أفضل Threshold للـ Category")
print("=====")

print(
    f"Threshold : {best_category_threshold[0]:.2f}"
)

print(
    f"Macro F1 : {best_category_threshold[1]:.4f}"
)

print(
    f"Hamming Loss : {best_category_threshold[2]:.4f}"
)

# تطبيق أفضل Threshold

category_prediction_best = (

```

```
category_probs >= best_category_threshold[0]
).astype(int)
```

```
print("\nCATEGORY REPORT بعد الضبط")
```

```
print(
    classification_report(
        category_true,
        category_prediction_best,
        zero_division=0
    )
)
```

```
# =====
# 4) Priority Threshold Tuning
#
# الهدف:
# رفع Recall لفئة High
#
# أهم فئة في النظام High Priority لأن
#
# =====
```

```
# Labels التأكد من ترتيب الـ
# إذا كان ترتيبك:
# ['High', 'Low', 'Normal']
# يبقى 0 صحيح
```

```
# =====
# 4) Priority Threshold Tuning
#
# الهدف:
# رفع Recall لفئة High Priority
# =====
```

```
# عندك غالباً Labels ترتيب الـ:
# 0 = High
# 1 = Low
# 2 = Normal
#
# إذا كان مختلفاً غيره هنا
```

```
high_index = 0
```

```
priority_results = []
```

```
for threshold in np.arange(0.2,0.71,0.05):
```

```
# القرار الطبيعي:  
# اختيار أعلى احتمال
```

```
priority_prediction = np.argmax(  
    priority_probs,  
    axis=1  
)
```

```
# Threshold أعلى من High تعديل القرار إذا كانت
```

```
high_prediction = (  
    priority_probs[:,high_index]  
    >= threshold  
)
```

```
priority_prediction[high_prediction] = high_index
```

```
report = classification_report(  
    priority_true,  
    priority_prediction,  
    output_dict=True,  
    zero_division=0  
)
```

```
# استخدام رقم الفئة بدل الاسم
```

```
high_recall = report[  
    str(high_index)  
]["recall"]
```

```
macro_f1 = report[  
    "macro avg"  
]["f1-score"]
```

```
priority_results.append(  
    (  
        threshold,  
        high_recall,  
        macro_f1  
    )  
)
```

```
best_priority_threshold = max(
```

```

priority_results,
key=lambda x:x[2]
)

print("\n=====")
print("أفضل Threshold لل High Priority")
print("=====")

print(
    f"Threshold : {best_priority_threshold[0]:.2f}"
)

print(
    f"High Recall : {best_priority_threshold[1]:.4f}"
)

print(
    f"Macro F1 : {best_priority_threshold[2]:.4f}"
)

print("\n=====")
print("أفضل Threshold لل High Priority")
print("=====")

print("\n=====")
print("بنجاح Threshold Tuning اكتمل")
print("=====")

```

بنجاح DistilBERT تم تحميل أفضل نموذج
تم استخراج احتمالات التنبؤ بنجاح

```

=====
أفضل Threshold لل Category
=====

```

```

Threshold : 0.60
Macro F1 : 0.9681
Hamming Loss : 0.0171

```

CATEGORY REPORT	بعد الضبط				
	precision	recall	f1-score	support	
0	0.99	0.96	0.97	2713	
1	1.00	0.93	0.96	1278	
2	0.99	0.92	0.96	998	
3	1.00	0.94	0.97	826	
4	0.99	0.95	0.97	1785	
5	0.97	0.98	0.98	1049	
micro avg	0.99	0.95	0.97	8649	
macro avg	0.99	0.95	0.97	8649	
weighted avg	0.99	0.95	0.97	8649	
samples avg	0.98	0.96	0.96	8649	

```
=====
High Priority لـ Threshold أفضل
=====
Threshold : 0.50
High Recall : 0.8571
Macro F1 : 0.9350
```

```
=====
High Priority لـ Threshold أفضل
=====
```

```
=====
بنجاح Threshold Tuning اكتمل
=====
```

Holdout Test Set على DistilBERT التقييم النهائي لنموذج

```
# =====
# Holdout Test Set على DistilBERT التقييم النهائي لنموذج
#
# الملف:
# holdout_test_set.csv
#
# الهدف:
# قياس الأداء النهائي على بيانات لم يستخدمها النموذج سابقاً -
# لا يوجد تدريب هنا -
# لا يوجد تعديل على أوزان النموذج -
#
# Validation Set: مسبقاً من Thresholds تم اختيار
#
# Category Threshold = 0.60
# Priority Threshold = 0.50
#
# =====
```

```
import pandas as pd
import numpy as np
import torch

from torch.utils.data import Dataset, DataLoader

from sklearn.metrics import (
    classification_report,
    f1_score,
    hamming_loss,
    accuracy_score,
    confusion_matrix
)
```

```
# =====
# Holdout Test Set تحديد مسار ملف الاختبار (1)
```

```

#
# Validation هذه العينة مستقلة عن التدريب والـ
# وتستخدم للحكم النهائي على النموذج
#
# =====

holdout_path = (
    "/content/drive/MyDrive/Thesis_Project/"
    "holdout_test_set.csv"
)

holdout_df = pd.read_csv(
    holdout_path
)

print("Holdout Dataset Shape:")
print(holdout_df.shape)

# =====
# 2) تعريف أعمدة Category
#
# Multi-label عبارة عن Category
#
# أي أن الإيميل الواحد يمكن أن ينتمي لأكثر من تصنيف
#
# مثال:
# Financial + Administration
#
# =====

category_columns = [

    "category_Administration",
    "category_Financial",
    "category_HR",
    "category_Legal",
    "category_Market",
    "category_Other"

]

# =====
# إلى أرقام Priority تحويل (3)
#
# تصنيف واحد فقط Priority:
#
# High

```

```
# Low
# Normal
#
# =====
```

```
priority_mapping = {

    "High":0,
    "Low":1,
    "Normal":2

}
```

```
# =====
# 4) لاختبار إنشاء Dataset
#
# يجب أن تكون نفس طريقة تجهيز البيانات المستخدمة أثناء التدريب
#
# نفس:
# - Cleaned_Text
# - Tokenizer
# - max_length
#
# =====
```

```
class HoldoutDataset(Dataset):

    def __init__(
        self,
        dataframe,
        tokenizer,
        max_length=256
    ):

        self.texts = dataframe[
            "Cleaned_Text"
        ].astype(str).tolist()

        self.category_labels = dataframe[
            category_columns
        ].values.astype(float)

        self.priority_labels = dataframe[
            "priority"
        ].map(priority_mapping).values
```

```

        self.tokenizer = tokenizer

        self.max_length = max_length

def __len__(self):

    return len(self.texts)

def __getitem__(self, index):

    encoding = self.tokenizer(

        self.texts[index],

        padding="max_length",

        truncation=True,

        max_length=self.max_length,

        return_tensors="pt"

    )

    return {

        "input_ids":
            encoding["input_ids"].squeeze(0),

        "attention_mask":
            encoding["attention_mask"].squeeze(0),

        "category_labels":
            torch.tensor(
                self.category_labels[index],
                dtype=torch.float
            ),

        "priority_labels":
            torch.tensor(
                self.priority_labels[index],

```

```

        dtype=torch.long
    )

}

```

```

# =====
# 5) لاختبار DataLoader إنشاء
#
# لأننا نريد ترتيب ثابت للتقييم shuffle=False
#
# =====

```

```

holdout_dataset = HoldoutDataset(

    holdout_df,

    tokenizer

)

```

```

holdout_loader = DataLoader(

    holdout_dataset,

    batch_size=32,

    shuffle=False

)

```

```

print("تم تجهيز Holdout DataLoader")

```

```

# =====
# تحميل أفضل نموذج محفوظ (6)
#
# هذا هو النموذج الذي حقق أفضل أداء أثناء التدريب
#
# =====

```

```

model.load_state_dict(

    torch.load(

        "best_distilbert_email_model.pt",


```

```

        map_location=device

    )

)

model.eval()

print("DistilBERT تم تحميل أفضل نموذج")

# =====
# استخراج تنبؤات النموذج 7)
#
# Category:
# Sigmoid نستخدم
# لها احتمال مستقل لأن كل
#
#
# Priority:
# Softmax نستخدم
# لأن لدينا اختيار واحد فقط
#
# =====

category_probs = []

priority_probs = []

category_true = []

priority_true = []

with torch.no_grad():

    for batch in holdout_loader:

        input_ids = batch[
            "input_ids"
        ].to(device)

```

```
attention_mask = batch[
    "attention_mask"
].to(device)
```

```
outputs = model(
    input_ids=input_ids,
    attention_mask=attention_mask
)
```

```
# احتمالات Category
```

```
cat_probability = torch.sigmoid(
    outputs["category_logits"]
)
```

```
# احتمالات Priority
```

```
pri_probability = torch.softmax(
    outputs["priority_logits"],
    dim=1
)
```

```
category_probs.append(
    cat_probability.cpu().numpy()
)
```

```
priority_probs.append(
    pri_probability.cpu().numpy()
)
```

```
category_true.append(
```

```

        batch["category_labels"].numpy()

    )

    priority_true.append(

        batch["priority_labels"].numpy()

    )

category_probs = np.vstack(category_probs)
priority_probs = np.vstack(priority_probs)

category_true = np.vstack(category_true)
priority_true = np.concatenate(priority_true)

print("تم استخراج التنبؤات بنجاح")

# =====
# النهائي Threshold تطبيق (8)
#
# Validation هذه القيم تم تحديدها مسبقاً من
#
# =====

category_threshold = 0.60
priority_threshold = 0.50

# Category Prediction

category_pred = (

    category_probs >= category_threshold

).astype(int)

# Priority Prediction

```

```

priority_pred = np.argmax(

    priority_probs,

    axis=1

)

# تعديل High Priority
#
# القيمة المحددة High إذا تجاوز احتمال
# High نعطيه أولوية
#

high_index = 0

high_mask = (

    priority_probs[:, high_index]

    >= priority_threshold

)

priority_pred[high_mask] = high_index

# =====
# Category النتائج النهائية (9)
# =====

print("\n=====")
print("FINAL CATEGORY RESULTS - HOLDOUT TEST")
print("=====")

print(

    "Hamming Loss:",

    hamming_loss(

        category_true,

        category_pred

```

```

    )
)

print(
    "Macro F1:",
    f1_score(
        category_true,
        category_pred,
        average="macro"
    )
)

print(
    classification_report(
        category_true,
        category_pred,
        target_names=category_columns,
        zero_division=0
    )
)

# =====
# 10) النتائج النهائية Priority
# =====

print("\n=====")
print("FINAL PRIORITY RESULTS - HOLDOUT TEST")
print("=====")

print(

```

```
        "Accuracy:",

        accuracy_score(

            priority_true,

            priority_pred

        )

    )

    print(

        "Macro F1:",

        f1_score(

            priority_true,

            priority_pred,

            average="macro"

        )

    )

    print(

        classification_report(

            priority_true,

            priority_pred,

            target_names=[

                "High",

                "Low",

                "Normal"

            ],

            zero_division=0

        )

    )
```

```

print("\nConfusion Matrix:")

print(
    confusion_matrix(
        priority_true,
        priority_pred
    )
)

```

```

print("\n=====")
print("Holdout Test Set اكتمل التقييم النهائي على")
print("=====")

```

Holdout Dataset Shape:
(10000, 9)
Holdout DataLoader تم تجهيز
DistilBERT تم تحميل أفضل نموذج
تم استخراج التنبؤات بنجاح

```

=====
FINAL CATEGORY RESULTS - HOLDOUT TEST
=====

```

Hamming Loss: 0.01755

Macro F1: 0.9672121917636307

	precision	recall	f1-score	support
category_Administration	0.99	0.95	0.97	5435
category_Financial	0.99	0.93	0.96	2456
category_HR	0.99	0.93	0.96	1889
category_Legal	1.00	0.93	0.96	1810
category_Market	0.99	0.95	0.97	3614
category_Other	0.97	0.99	0.98	2095
micro avg	0.99	0.95	0.97	17299
macro avg	0.99	0.95	0.97	17299
weighted avg	0.99	0.95	0.97	17299
samples avg	0.98	0.96	0.97	17299

```

=====
FINAL PRIORITY RESULTS - HOLDOUT TEST
=====

```

Accuracy: 0.9529

Macro F1: 0.9326535828034164

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

High	0.89	0.86	0.88	925
Low	0.95	0.96	0.96	3929
Normal	0.96	0.97	0.96	5146
accuracy			0.95	10000
macro avg	0.94	0.93	0.93	10000
weighted avg	0.95	0.95	0.95	10000

Confusion Matrix:

```
[[ 798   63   64]
 [  36 3762  131]
 [  58  119 4969]]
```

```
=====
Holdout Test Set اكتمل التقييم النهائي على
=====
```

النهائية DistilBERT حفظ نتائج

```
# =====
# النهائية DistilBERT حفظ نتائج
#
# النتائج المستخدمة للمقارنة النهائية
# Holdout Test Set
# =====
```

```
import pandas as pd
import json
```

نتائج عامة

```
distilbert_results = {

    "Model": "DistilBERT Multi-task",

    "Dataset": "Holdout Test Set",

    "Samples": 10000,

    "Category_Hamming_Loss": 0.01755,

    "Category_Macro_F1": 0.9672121918,

    "Priority_Accuracy": 0.9529,

    "Priority_Macro_F1": 0.9326535828,

    "High_Recall": 0.8627,

    "Category_Threshold": 0.60,

    "Priority_Threshold": 0.50

}
```

```
# حفظ CSV
```

```
pd.DataFrame(  
    [distilbert_results]  
).to_csv(  
    "/content/drive/MyDrive/Thesis_Project/distilbert_final_results.csv",  
    index=False  
)
```

```
# حفظ JSON
```

```
with open(  
    "/content/drive/MyDrive/Thesis_Project/distilbert_final_results.json",  
    "w"  
) as f:  
  
    json.dump(  
        distilbert_results,  
        f,  
        indent=4  
    )
```

```
print("النهائية بنجاح DistilBERT تم حفظ نتائج")
```

النهائية بنجاح DistilBERT تم حفظ نتائج

المقارنات بين النماذج بعد تشغيلها على العينة النهائية للاختبار

```
# =====  
# PHASE 3B  
# PRIORITY CLASSIFICATION  
#  
# BASELINE MODEL #2  
# TF-IDF + LINEAR SVM  
#  
# Final Version:  
# - Training  
# - Validation Evaluation  
# - Holdout Test Evaluation  
# - Save Model + Results  
#  
# =====
```

```

import pandas as pd
import numpy as np
import json
import joblib
import os

from sklearn.preprocessing import LabelEncoder

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.svm import LinearSVC

from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

# =====
# 1) Paths
# =====

base_path = "/content/drive/MyDrive/Thesis_Project/"

train_path = base_path + "train_set.csv"

val_path = base_path + "validation_set.csv"

holdout_path = base_path + "holdout_test_set.csv"

save_path = base_path + "SVM_Priority_Results"

os.makedirs(
    save_path,
    exist_ok=True
)

# =====
# 2) Load Data
# =====

```

```

train_df = pd.read_csv(train_path)

val_df = pd.read_csv(val_path)

holdout_df = pd.read_csv(holdout_path)


print("="*70)
print("DATASETS LOADED")
print("="*70)


print("Train      :", train_df.shape)

print("Validation:", val_df.shape)

print("Holdout    :", holdout_df.shape)


# =====
# 3) Prepare Text Data
# =====


X_train = train_df["Cleaned_Text"]

X_val = val_df["Cleaned_Text"]

X_holdout = holdout_df["Cleaned_Text"]


# =====
# 4) Encode Priority Labels
#
# نفس الترميز يستخدم لكل المجموعات
#
# =====


encoder = LabelEncoder()

y_train = encoder.fit_transform(
    train_df["priority"]
)

y_val = encoder.transform(
    val_df["priority"]
)

```

```
y_holdout = encoder.transform(  
    holdout_df["priority"]  
)
```

```
print("\nPriority Classes:")
```

```
print(  
    list(encoder.classes_)  
)
```

```
# =====  
# 5) TF-IDF Feature Extraction  
#  
# على التدريب فقط fit يتم  
# للباقي transform ثم  
#  
# =====
```

```
vectorizer = TfidfVectorizer(  
  
    max_features=20000,  
  
    ngram_range=(1,2),  
  
    min_df=5,  
  
    max_df=0.95,  
  
    sublinear_tf=True  
)
```

```
X_train_tfidf = vectorizer.fit_transform(  
    X_train  
)
```

```
X_val_tfidf = vectorizer.transform(  
    X_val  
)
```

```
X_holdout_tfidf = vectorizer.transform(  
    X_holdout  
)
```

```

print("\nTF-IDF Completed")

print(
    "Vocabulary Size:",
    len(vectorizer.vocabulary_)
)

# =====
# 6) Train Linear SVM
# =====

svm_model = LinearSVC(

    C=1.0,

    random_state=42

)

print("\nTraining Linear SVM...")

svm_model.fit(

    X_train_tfidf,

    y_train

)

print("Training Completed")

# =====
# 7) Function For Evaluation
# =====

def evaluate_model(
    y_true,
    y_pred,
    dataset_name
):

    results = {

```

```

        "Dataset": dataset_name,

        "Accuracy":
accuracy_score(
    y_true,
    y_pred
),

        "Macro_F1":
f1_score(
    y_true,
    y_pred,
    average="macro"
),

        "Weighted_F1":
f1_score(
    y_true,
    y_pred,
    average="weighted"
),

        "Precision":
precision_score(
    y_true,
    y_pred,
    average="macro"
),

        "Recall":
recall_score(
    y_true,
    y_pred,
    average="macro"
)

    }

    print("\n")
    print("="*70)

    print(dataset_name)

    print("="*70)

    for key,value in results.items():

        if key != "Dataset":

```

```

        print(
            f"{key}: {value:.4f}"
        )

    print("\nClassification Report")

    print(
        classification_report(
            y_true,
            y_pred,
            target_names=encoder.classes_
        )
    )

    print("\nConfusion Matrix")

    print(
        confusion_matrix(
            y_true,
            y_pred
        )
    )

    return results

# =====
# 8) Validation Evaluation
# =====

val_pred = svm_model.predict(
    X_val_tfidf
)

val_results = evaluate_model(

```

```

        y_val,

        val_pred,

        "Validation"
    )

# =====
# 9) Final Holdout Evaluation
# =====

holdout_pred = svm_model.predict(
    X_holdout_tfidf
)

holdout_results = evaluate_model(

    y_holdout,

    holdout_pred,

    "Holdout_Test"
)

# =====
# 10) Save Results
# =====

all_results = pd.DataFrame(

    [
        val_results,

        holdout_results

    ]
)

all_results.to_csv(

    save_path + "/SVM_priority_results.csv",

```

```

        index=False
    )

    with open(
        save_path + "/SVM_priority_results.json",
        "w"
    ) as f:

        json.dump(
            all_results.to_dict(
                orient="records"
            ),
            f,
            indent=4
        )

# =====
# 11) Save Model Components
# =====

joblib.dump(
    svm_model,
    save_path + "/linear_svm_priority.pkl"
)

joblib.dump(
    vectorizer,
    save_path + "/tfidf_vectorizer.pkl"
)

joblib.dump(
    encoder,

```

```

    save_path + "/priority_encoder.pkl"

)

print("\n")
print("="*70)
print("SVM PRIORITY FINAL PIPELINE COMPLETED")
print("="*70)

print(
    "Results saved in:"
)

print(save_path)

=====
DATASETS LOADED
=====
Train      : (35000, 9)
Validation: (5000, 9)
Holdout    : (10000, 9)

Priority Classes:
['High', 'Low', 'Normal']

TF-IDF Completed
Vocabulary Size: 20000

Training Linear SVM...
Training Completed

=====
Validation
=====
Accuracy: 0.8986
Macro_F1: 0.8565
Weighted_F1: 0.8966
Precision: 0.8905
Recall: 0.8327

Classification Report

```

	precision	recall	f1-score	support
High	0.87	0.66	0.75	462
Low	0.90	0.90	0.90	1965
Normal	0.90	0.94	0.92	2573
accuracy			0.90	5000
macro avg	0.89	0.83	0.86	5000
weighted avg	0.90	0.90	0.90	5000

```
Confusion Matrix
[[ 304   73   85]
 [  27 1763  175]
 [   18  129 2426]]
```

```
=====
Holdout_Test
=====
```

```
Accuracy: 0.9079
Macro_F1: 0.8695
Weighted_F1: 0.9064
Precision: 0.8982
Recall: 0.8483
```

```
Classification Report
```

	precision	recall	f1-score	support
High	0.88	0.69	0.77	925
Low	0.91	0.91	0.91	3929
Normal	0.91	0.94	0.93	5146
accuracy			0.91	10000
macro avg	0.90	0.85	0.87	10000
weighted avg	0.91	0.91	0.91	10000

```
Confusion Matrix
[[ 637  129  159]
 [  48 3586  295]
 [   43  247 4856]]
```

```
=====
SVM PRIORITY FINAL PIPELINE COMPLETED
=====
```

```
Results saved in:
/content/drive/MyDrive/Thesis_Project/SVM_Priority_Results
```

```
# =====
# PHASE 3A
# CATEGORY CLASSIFICATION
#
# BASELINE MODEL #2
# TF-IDF + Linear SVM
#
# Final Pipeline:
# - Multi-Label Classification
# - Validation Evaluation
# - Holdout Test Evaluation
# - Save Results
# =====
```

```
import pandas as pd
```

```

import json
import joblib
import os

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.svm import LinearSVC

from sklearn.multiclass import OneVsRestClassifier

from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    hamming_loss,
    jaccard_score
)

# =====
# 1) Paths
# =====

base_path = "/content/drive/MyDrive/Thesis_Project/"

train_path = base_path + "train_set.csv"

val_path = base_path + "validation_set.csv"

holdout_path = base_path + "holdout_test_set.csv"

save_path = base_path + "SVM_Category_Results"

os.makedirs(
    save_path,
    exist_ok=True
)

# =====
# 2) Load Data
# =====

train_df = pd.read_csv(train_path)

```

```
val_df = pd.read_csv(val_path)
```

```
holdout_df = pd.read_csv(holdout_path)
```

```
print("="*70)
```

```
print("DATASETS LOADED")
```

```
print("="*70)
```

```
print("Train      :", train_df.shape)
```

```
print("Validation:", val_df.shape)
```

```
print("Holdout   :", holdout_df.shape)
```

```
# =====
```

```
# 3) Category Labels
```

```
# =====
```

```
category_columns = [
```

```
    "category_Administration",
```

```
    "category_Financial",
```

```
    "category_HR",
```

```
    "category_Legal",
```

```
    "category_Market",
```

```
    "category_Other"
```

```
]
```

```
# =====
```

```
# 4) Prepare Data
```

```
# =====
```

```
X_train = train_df["Cleaned_Text"]
```

```
X_val = val_df["Cleaned_Text"]
```

```
X_holdout = holdout_df["Cleaned_Text"]
```

```
y_train = train_df[
    category_columns
].values
```

```
y_val = val_df[
    category_columns
].values
```

```
y_holdout = holdout_df[
    category_columns
].values
```

```
# =====
# 5) TF-IDF
#
# فقط على التدريب fit
# للباقي transform
#
# =====
```

```
vectorizer = TfidfVectorizer(

    max_features=20000,

    ngram_range=(1,2),

    min_df=5,

    max_df=0.95,

    sublinear_tf=True

)

X_train_tfidf = vectorizer.fit_transform(
    X_train
)

X_val_tfidf = vectorizer.transform(
    X_val
)
```

```

X_holdout_tfidf = vectorizer.transform(
    X_holdout
)

print("\nTF-IDF Completed")

print(
    "Vocabulary:",
    len(vectorizer.vocabulary_)
)

# =====
# 6) Train Multi-label Linear SVM
#
# OneVsRest:
# يحول المشكلة إلى عدة Binary Classifiers
#
# مثال:
# Financial yes/no
# HR yes/no
# Legal yes/no
#
# =====

svm_model = OneVsRestClassifier(
    LinearSVC(
        random_state=42
    )
)

print("\nTraining Linear SVM...")

svm_model.fit(
    X_train_tfidf,
    y_train
)

print("Training Completed")

```

```

# =====
# 7) Evaluation Function
# =====

def evaluate_category(
    y_true,
    y_pred,
    name
):

    results = {

        "Dataset": name,

        "Subset_Accuracy":
            accuracy_score(
                y_true,
                y_pred
            ),

        "Micro_F1":
            f1_score(
                y_true,
                y_pred,
                average="micro",
                zero_division=0
            ),

        "Macro_F1":
            f1_score(
                y_true,
                y_pred,
                average="macro",
                zero_division=0
            ),

        "Hamming_Loss":
            hamming_loss(
                y_true,
                y_pred
            ),

        "Jaccard":
            jaccard_score(
                y_true,
                y_pred,
                average="samples",

```

```

        zero_division=0
    )

}

print("\n")
print("="*70)

print(name)

print("="*70)

for k,v in results.items():

    if k != "Dataset":

        print(
            f"{k}: {v:.4f}"
        )

return results

```

```

# =====
# 8) Validation Evaluation
# =====

```

```

val_pred = svm_model.predict(
    X_val_tfidf
)

```

```

val_results = evaluate_category(

    y_val,

    val_pred,

    "Validation"

)

```

```

# =====
# 9) Final Holdout Evaluation
# =====

```

```
holdout_pred = svm_model.predict(  
    X_holdout_tfidf  
)
```

```
holdout_results = evaluate_category(  
    y_holdout,  
    holdout_pred,  
    "Holdout_Test"  
)
```

```
# =====  
# 10) Save Results  
# =====
```

```
results_df = pd.DataFrame(  
    [  
        val_results,  
        holdout_results  
    ]  
)
```

```
results_df.to_csv(  
    save_path + "/SVM_category_results.csv",  
    index=False  
)
```

```
with open(  
    save_path + "/SVM_category_results.json",  
    "w"  
) as f:
```

```

json.dump(
    results_df.to_dict(
        orient="records"
    ),
    f,
    indent=4
)

```

```

# =====
# 11) Save Model Components
# =====

```

```

joblib.dump(
    svm_model,
    save_path + "/linear_svm_category.pkl"
)

```

```

joblib.dump(
    vectorizer,
    save_path + "/tfidf_vectorizer.pkl"
)

```

```

print("\n")
print("="*70)
print("SVM CATEGORY PIPELINE COMPLETED")
print("="*70)

```

```

print(
    "Saved in:"
)

```

```

print(save_path)

```

```

=====
DATASETS LOADED

```

```
=====
Train      : (35000, 9)
Validation: (5000, 9)
Holdout    : (10000, 9)
```

TF-IDF Completed
Vocabulary: 20000

Training Linear SVM...
Training Completed

```
=====
Validation
=====
Subset_Accuracy: 0.8092
Micro_F1: 0.9328
Macro_F1: 0.9266
Hamming_Loss: 0.0374
Jaccard: 0.8869
```

```
=====
Holdout_Test
=====
Subset_Accuracy: 0.8117
Micro_F1: 0.9333
Macro_F1: 0.9269
Hamming_Loss: 0.0370
Jaccard: 0.8857
```

```
=====
SVM CATEGORY PIPELINE COMPLETED
=====
Saved in:
/content/drive/MyDrive/Thesis_Project/SVM_Category_Results
```

The final comparison

```
# =====
# FINAL MODEL COMPARISON
#
# الهدف:
# Holdout Test Set مقارنة النماذج النهائية على
#
# Models:
# 1) Linear SVM + TF-IDF
#   - Category Multi-label
#   - Priority Multi-class
#
# 2) DistilBERT Multi-task
#   - Category Multi-label
#   - Priority Multi-class
#
# يتم استخدام النتائج المحفوظة مسبقاً
```

```

# بدون إعادة تدريب أي نموذج
#
# =====

import pandas as pd
import matplotlib.pyplot as plt
import os

# =====
# تحديد المسارات 1)
# =====

base_path = "/content/drive/MyDrive/Thesis_Project/"

distilbert_file = (
    base_path +
    "distilbert_final_results.csv"
)

svm_category_file = (
    base_path +
    "SVM_Category_Results/"
    "SVM_category_results.csv"
)

svm_priority_file = (
    base_path +
    "SVM_Priority_Results/"
    "SVM_priority_results.csv"
)

output_path = (
    base_path +
    "Final_Model_Comparison/"
)

os.makedirs(
    output_path,
    exist_ok=True
)

# =====
# تحميل النتائج 2)

```

```

# =====

distilbert = pd.read_csv(
    distilbert_file
)

svm_category = pd.read_csv(
    svm_category_file
)

svm_priority = pd.read_csv(
    svm_priority_file
)

# Holdout تأخذ فقط نتائج
svm_category_test = svm_category[
    svm_category["Dataset"]=="Holdout_Test"
].iloc[0]

svm_priority_test = svm_priority[
    svm_priority["Dataset"]=="Holdout_Test"
].iloc[0]

distilbert_test = distilbert.iloc[0]

# =====
# إنشاء جدول المقارنة النهائي (3)
# =====

comparison = pd.DataFrame({

    "Model":[

        "Linear SVM",

        "DistilBERT Multi-task"

    ],

    "Category Macro F1":[

        svm_category_test["Macro_F1"],


```

```

        distilbert_test["Category_Macro_F1"]
    ],

    "Category Hamming Loss":[

        svm_category_test["Hamming_Loss"],

        distilbert_test["Category_Hamming_Loss"]

    ],

    "Priority Accuracy":[

        svm_priority_test["Accuracy"],

        distilbert_test["Priority_Accuracy"]

    ],

    "Priority Macro F1":[

        svm_priority_test["Macro_F1"],

        distilbert_test["Priority_Macro_F1"]

    ],

    "High Recall":[

        0.69,

        distilbert_test["High_Recall"]

    ]

})

print("="*80)

print("FINAL MODEL COMPARISON")

print("="*80)

display(comparison)

```

حفظ الجدول

```
comparison.to_csv(  
  
    output_path +  
    "final_model_comparison.csv",  
  
    index=False  
)
```

```
# =====  
# Macro F1 صورة مقارنة (4)  
# =====  
#  
# الوصف:  
#  
# يوضح هذا الشكل قدرة كل نموذج على تحقيق توازن  
# عبر جميع الفئات Precision و Recall بين.  
#  
# ارتفاع العمود يعني أداء أفضل.  
#  
# =====
```

```
plt.figure(  
    figsize=(8,5)  
)
```

```
plt.bar(  
  
    comparison["Model"],  
  
    comparison["Category Macro F1"]  
)
```

```
plt.title(  
    "Category Classification - Macro F1 Comparison"  
)
```

```
plt.ylabel(  
    "Macro F1 Score"  
)
```

```
plt.xticks(  
    rotation=20  
)
```

```
plt.tight_layout()
```

```
plt.savefig(
```

```
    output_path +  
    "category_macro_f1.png",
```

```
    dpi=300
```

```
)
```

```
plt.show()
```

```
print(
```

```
"""
```

```
وصف الصورة:
```

```
Category. لمهمة تصنيف Macro F1 الشكل يوضح مقارنة  
بسبب قدرته على فهم DistilBERT Multi-task نلاحظ تفوق  
على Linear SVM السياق الدلالي للنص، بينما يعتمد  
TF-IDF. الأنماط الإحصائية الناتجة من
```

```
"""
```

```
)
```

```
# =====  
# صورة مقارنة (5) Priority Macro F1  
# =====
```

```
plt.figure(
```

```
    figsize=(8,5)
```

```
)
```

```
plt.bar(
```

```
    comparison["Model"],
```

```
    comparison["Priority Macro F1"]
```

```
)
```

```
plt.title(
```

```
    "Priority Classification - Macro F1 Comparison"
```

```
)
```

```
plt.ylabel(  
    "Macro F1 Score"  
)
```

```
plt.xticks(  
    rotation=20  
)
```

```
plt.tight_layout()
```

```
plt.savefig(  
  
    output_path +  
    "priority_macro_f1.png",  
  
    dpi=300  
)
```

```
plt.show()
```

```
print(  
    """  
    وصف الصورة:  
  
    الشكل يوضح أداء النماذج في تحديد أولوية البريد الإلكتروني  
    أعلى، مما يدل على Macro F1 حقق DistilBERT يظهر أن  
    قدرة أفضل في التعامل مع الفئات غير المتوازنة وخاصة  
    فئة High Priority.  
    """  
)
```

```
# =====  
# Hamming Loss مقارنة (6)  
# =====  
#  
# الأقل أفضل Hamming Loss  
#  
# =====
```

```
plt.figure(  
    figsize=(8,5)  
)
```

```
plt.bar(

    comparison["Model"],

    comparison["Category Hamming Loss"]

)

plt.title(
    "Category Classification - Hamming Loss Comparison"
)

plt.ylabel(
    "Hamming Loss (Lower is Better)"
)

plt.xticks(
    rotation=20
)

plt.tight_layout()

plt.savefig(

    output_path +
    "hamming_loss_comparison.png",

    dpi=300

)

plt.show()

print(
    """
وصف الصورة:

الشكل يوضح نسبة أخطاء التصنيف المتعدد
القيمة الأقل تعني أن النموذج ارتكب أخطاء أقل
في تحديد الفئات المرتبطة بالبريد الإلكتروني
    """
)
```

```
# =====  
# 7) مقارنة High Priority Recall  
# =====  
  
plt.figure(  
    figsize=(8,5)  
)  
  
plt.bar(  
    comparison["Model"],  
    comparison["High Recall"]  
)  
  
plt.title(  
    "High Priority Recall Comparison"  
)  
  
plt.ylabel(  
    "Recall"  
)  
  
plt.xticks(  
    rotation=20  
)  
  
plt.tight_layout()  
  
plt.savefig(  
    output_path +  
    "high_priority_recall.png",  
    dpi=300  
)  
  
plt.show()  
  
print(  

```

```
"""
```

وصف الصورة:

High Priority. الشكل يوضح قدرة النماذج على اكتشاف رسائل مهم لأن فقدان رسالة عالية الأولوية Recall ارتفاع. قد يؤدي إلى تأخير التعامل معها. بسبب قدرته الأعلى على التقاط DistilBERT يتفوق المعنى والسياق.

```
"""
```

```
)
```

```
print("\n")
```

```
print("="*80)
```

```
print("تم إنشاء وحفظ جميع المقارنات والرسومات بنجاح")
```

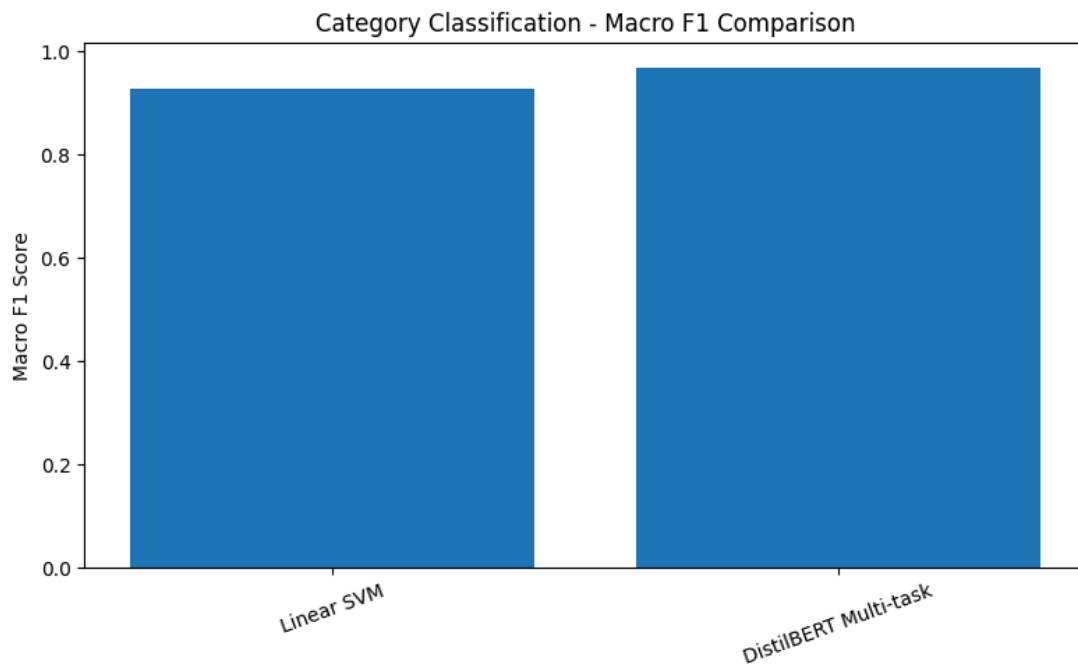
```
print("="*80)
```

```
print(
    "المسار:"
)
```

```
print(output_path)
```

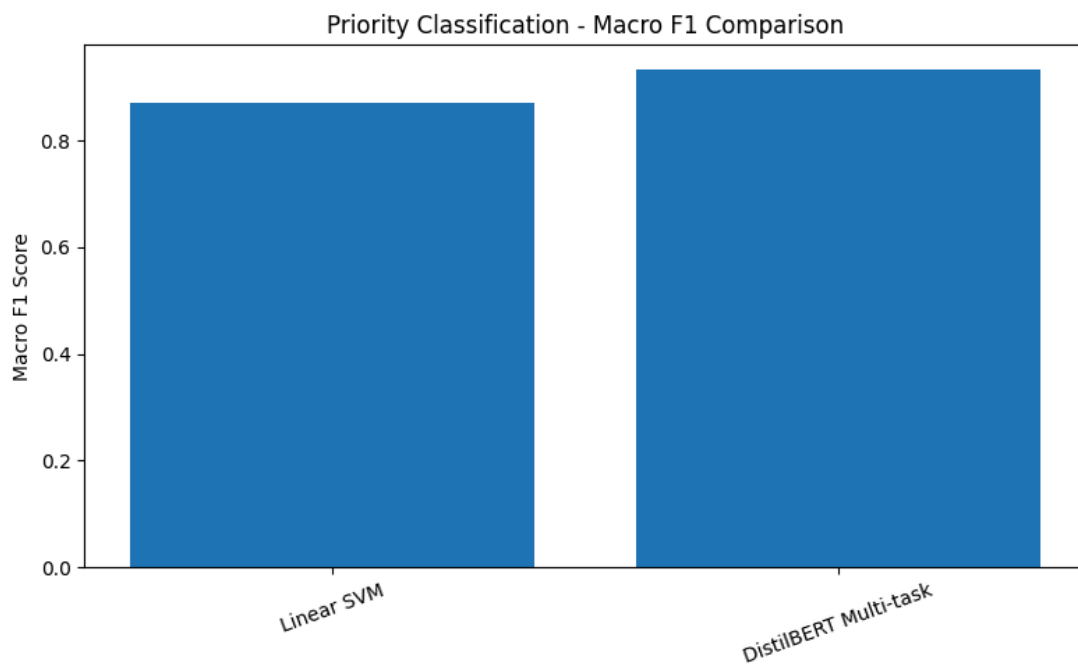
```
=====
FINAL MODEL COMPARISON
=====
```

	Model	Category Macro F1	Category Hamming Loss	Priority Accuracy	Priority Macro F1	High Recall
0	Linear SVM	0.926879	0.03700	0.9079	0.869483	0.6900
1	DistilBERT Multi-task	0.967212	0.01755	0.9529	0.932654	0.8627



:وصف الصورة

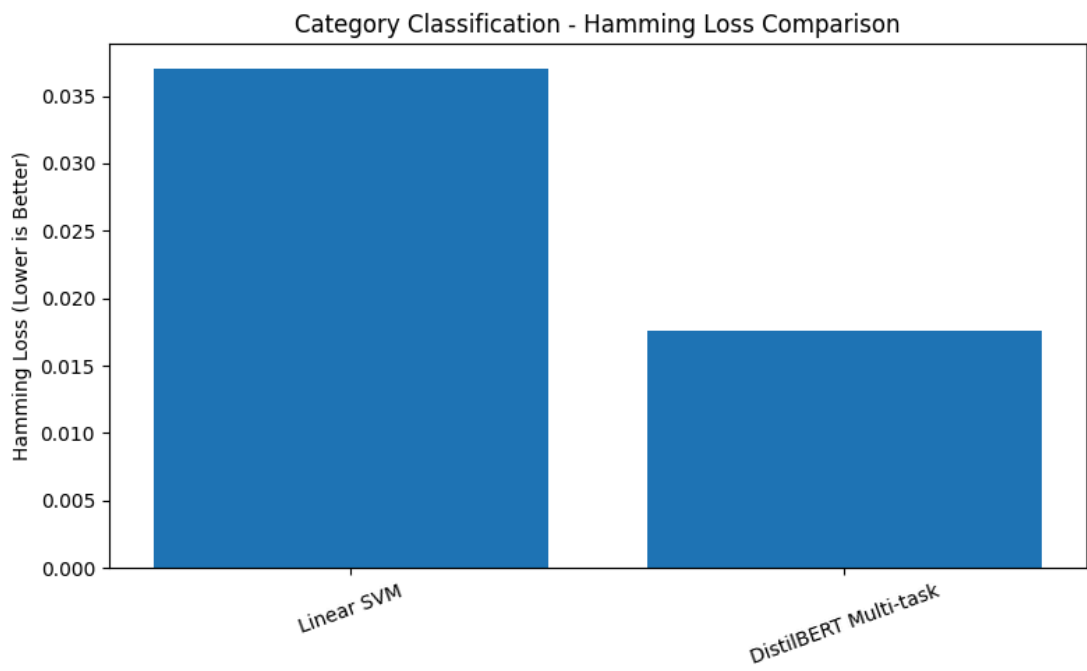
Category. لمهمة تصنيف Macro F1 الشكل يوضح مقارنة بسبب قدرته على فهم DistilBERT Multi-task نلاحظ تفوق على Linear SVM السياق الدلالي للنص، بينما يعتمد TF-IDF. الأنماط الإحصائية الناتجة من



:وصف الصورة

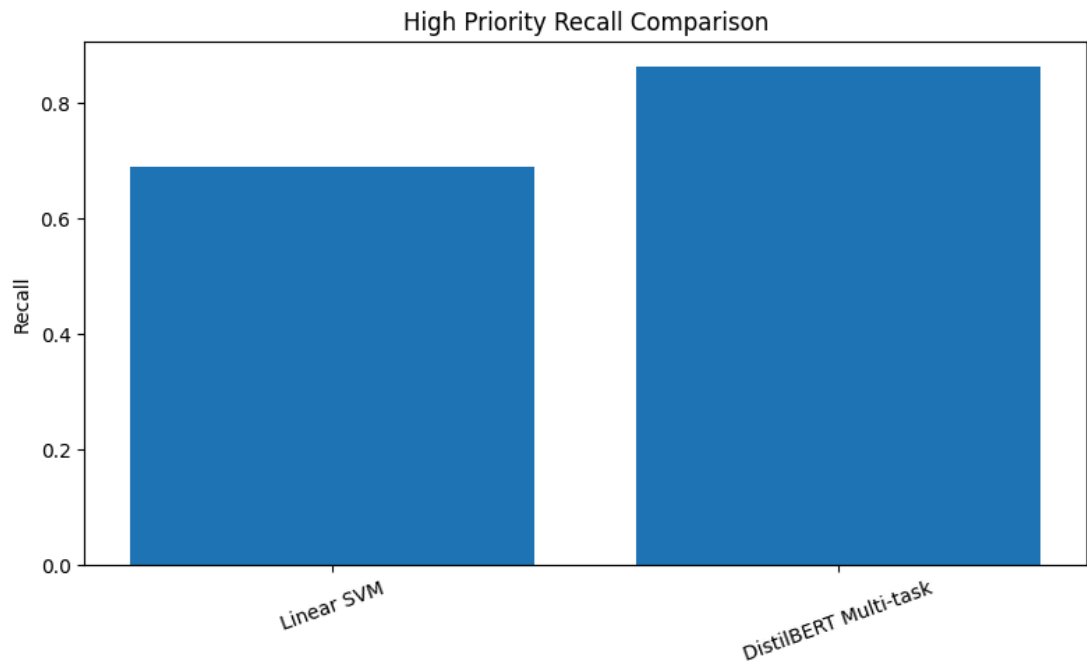
. الشكل يوضح أداء النماذج في تحديد أولوية البريد الإلكتروني أعلى، مما يدل على Macro F1 حقق DistilBERT يظهر أن قدرة أفضل في التعامل مع الفئات غير المتوازنة وخاصة

High Priority. فئة



وصف الصورة:

الشكل يوضح نسبة أخطاء التصنيف المتعدد القيمة الأقل تعني أن النموذج ارتكب أخطاء أقل في تحديد الفئات المرتبطة بالبريد الإلكتروني.



وصف الصورة:

High Priority. الشكل يوضح قدرة النماذج على اكتشاف رسائل

مهم لأن فقدان رسالة عالية الأولوية Recall ارتفاع
قد يؤدي إلى تأخير التعامل معها.
بسبب قدرته الأعلى على التقاط DistilBERT يتفوق
المعنى والسياق.

=====
تم إنشاء وحفظ جميع المقارنات والرسومات بنجاح
=====
المسار:
/content/drive/MyDrive/Thesis_Project/Final_Model_Comparison/