

```
# ربط كولا ب في قولد درايف
from google.colab import drive
drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)

```
# تثبيت المكتبات وتجهيز مجلدات النتائج
#SHAP عشان نفسر النموذج ونعرف ليش النموذج توقع ازدحام
#imbalanced-learn تعالج مشكلة عدم توازن البيانات
#-q عشان ما يطبع تفاصيل كثيرة وقت التثبيت
```

```
!pip -q install shap==0.45.1 imbalanced-learn==0.12.3
```

```
#os عشان نتعامل مع الملفات والمجلدات
#glob من مجلد معين عشان نجيب كل ملفات
#zipfile بالنهاية ZIP عشان نضغط النتائج في ملف
#re مثلًا عشان نلقت الأعمدة Regular Expressions نستخدمه للـ
#warnings نستخدمه لإدارة التحذيرات
```

```
import os
import glob
import zipfile
```

```
# تحديد مجلدات النتائج
#OUT_DIR هذا المجلد الرئيسي للنتائج
#FIG_DIR بنحط فيه الرسومات والمقارنات
#REP_DIR بنحط فيه التقارير وملفات CSV
OUT_DIR = "/content/outputs"
FIG_DIR = os.path.join(OUT_DIR, "figures")
REP_DIR = os.path.join(OUT_DIR, "reports")
```

```
#إنشاء المجلدات
os.makedirs(FIG_DIR, exist_ok=True)
os.makedirs(REP_DIR, exist_ok=True)
```

```
print("Done:", OUT_DIR)
print("Figures:", FIG_DIR)
print("Reports:", REP_DIR)
```

```
===== 544.3/544.3 kB 8.8 MB/s eta 0:00:00
===== 258.3/258.3 kB 12.7 MB/s eta 0:00:00
Done: /content/outputs
Figures: /content/outputs/figures
Reports: /content/outputs/reports
```

```
# قراءة ملفات CSV
```

```
import glob
import os
```

```
# موقع البيانات في قولد درايف
DATA_DIR = "/content/drive/MyDrive/Traffic"
```

```
# البحث عن ملفات CSV
csv_files = sorted(glob.glob(os.path.join(DATA_DIR, "*.csv")))
assert len(csv_files) >= 1, f"I couldn't find any CSV files inside: {DATA_DIR}"
# للتأكد طباعة المسار وعدد الملفات داخل المجلد
print("Path", DATA_DIR)
print("Number of CSV files:", len(csv_files))
```

```
# عرض أول 10 ملفات للتأكد من أسماء الملفات والترتيب
for f in csv_files[:10]:
    print("", os.path.basename(f))
```

```
# اظهار رساله خطأ عند عدم وجود الملفات
assert len(csv_files) >= 1, "I couldn't find the CSV file inside Traffic."
```

```
Path /content/drive/MyDrive/Traffic
Number of CSV files: 31
VSDATA_20250301.csv
VSDATA_20250302.csv
VSDATA_20250303.csv
VSDATA_20250304.csv
VSDATA_20250305.csv
VSDATA_20250306.csv
VSDATA_20250307.csv
VSDATA_20250308.csv
VSDATA_20250309.csv
VSDATA_20250310.csv
```

```
#os للتعامل مع المجلدات وإنشائها
#re للبحث عن الأعمدة V00..V95
#glob للبحث عن الملفات
#pandas لمعالجة البيانات
#numpy للعمليات الرياضية + تحويل أنواع البيانات
import os
import re
import glob
import pandas as pd
import numpy as np

#تحديد المسارات
#DATA_DIR الأصلية CSV هذا مكان ملفات
#PROC_DIR Parquet هذا مجلد جديد بنحفظ فيه ملفات
DATA_DIR = "/content/drive/MyDrive/Traffic"
PROC_DIR = "/content/processed"

#Parquet إنشاء مجلد
os.makedirs(PROC_DIR, exist_ok=True)

#تحديد الملفات مرتبة ابجديا
csv_files = sorted(glob.glob(os.path.join(DATA_DIR, "*.csv")))

assert len(csv_files) > 0, "No CSV files found."

print("CSV files:", len(csv_files))
```

CSV files: 31

```
#قراءة ومعالجة ملف واحد

sample_file = csv_files[0]
df = pd.read_csv(sample_file)

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

Shape before: (113849, 103)

	NB_SCATS_SITE	QT_INTERVAL_COUNT	NB_DETECTOR	V00	V01	V02	V03	V04	V05	V06	...	V90	V91	V92	V93	V94	V95	NM_REG
0	100	2025-03-01	1	4	5	4	3	3	2	3	...	8	2	5	7	8	8	V
1	100	2025-03-01	2	27	27	23	10	15	10	12	...	58	52	45	42	41	40	V
2	100	2025-03-01	3	49	57	36	27	29	24	19	...	72	68	65	50	57	47	V
3	100	2025-03-01	4	9	9	12	14	9	16	17	...	60	47	40	34	44	22	V
4	100	2025-03-01	5	1	4	8	1	3	3	9	...	5	7	4	8	5	4	V

5 rows × 103 columns

```
#معالجة الأعمدة V00 إلى V95

volume_cols = [c for c in df.columns if re.match(r"V\d+", c)]

print("Detected volume columns:", len(volume_cols))
print(volume_cols[:10])

Detected volume columns: 96
['V00', 'V01', 'V02', 'V03', 'V04', 'V05', 'V06', 'V07', 'V08', 'V09']
```

```
#(ملف واحد تجريبي) Parquet تحويل إلى
out_path = os.path.join(PROC_DIR, "test.parquet")
df.to_parquet(out_path, index=False)

print("Saved:", out_path)
```

Saved: /content/processed/test.parquet

```
#تعريفات
#تحديد وقت قراءة الحسابات في الداتا سيت
INTERVAL_MIN = 15
#هذا تحديد للتبنا بعد 15 دقيقة
HORIZON_MIN = 15
#اسم العمود اللي يمثل موقع الإشارة
SITE_COL = "NB_SCATS_SITE"
#رقم الحساس داخل التقاطع
DETECTOR_COL = "NB_DETECTOR"
#اسم المنطقة
REGION_COL = "NM_REGION"
```

```
#بيانات تاريخ القراءة
DATE_COL = "QT_INTERVAL_COUNT"
#Regex لضمان عدم التداخل
V_COL_PATTERN = r"^\d{2}$"

PROC_DIR = "/content/processed"

import os, re
import pandas as pd
import numpy as np

os.makedirs(PROC_DIR, exist_ok=True)
```

```
# Parquet ثم احفظه TIMESTAMP أنشئ + Long إلى Wide اقرأ الملف + حوله من
def process_one_file(csv_path: str):

    # اقرأ ملف CSV
    df = pd.read_csv(csv_path)

    # تأكد الأعمدة الأساسية موجودة في الملف
    must = [SITE_COL, DETECTOR_COL, DATE_COL]
    missing = [c for c in must if c not in df.columns]
    if missing:
        raise ValueError(f"Missing columns {missing} in file: {os.path.basename(csv_path)}")

    # البحث عن أعمدة V00..V95
    vcols = [c for c in df.columns if re.match(r"^\d{2}$", str(c))]
    if len(vcols) == 0:
        # احتياط لو كانت بصيغة V0..V95
        vcols = [c for c in df.columns if re.match(r"^\d{1,3}$", str(c))]
    if len(vcols) == 0:
        raise ValueError(f"No Vxx columns found in: {os.path.basename(csv_path)}")

    # تحويل التاريخ من نص إلى datetime
    df[DATE_COL] = pd.to_datetime(df[DATE_COL], errors="coerce")

    # Wide إلى Long تحويل البيانات من
    # تتحول إلى صف مستقل يمثل قراءة واحدة Vxx كل
    out = df[[SITE_COL, DETECTOR_COL, DATE_COL] + vcols].melt(
        id_vars=[SITE_COL, DETECTOR_COL, DATE_COL],
        value_vars=vcols,
        var_name="INTERVAL_COL",
        value_name="VOLUME"
    )

    # استخراج رقم الفترة من اسم العمود (مثلاً V07 -> 7)
    out["INTERVAL_IDX"] = (
        out["INTERVAL_COL"]
        .str.replace("V", "", regex=False)
        .astype(int)
    )

    # التاريخ + (رقم الفترة x 15 دقيقة) إنشاء TIMESTAMP
    out["TIMESTAMP"] = (
        out[DATE_COL] +
        pd.to_timedelta(out["INTERVAL_IDX"] * INTERVAL_MIN, unit="m")
    )

    # إلى رقم VOLUME تنظيف وتحويل
    out["VOLUME"] = (
        pd.to_numeric(out["VOLUME"], errors="coerce")
        .fillna(0)
        .astype(np.float32)
    )

    # اختيار الأعمدة النهائية وترتيبها زمنياً
    out = (
        out[[SITE_COL, DETECTOR_COL, "TIMESTAMP", "VOLUME", "INTERVAL_IDX"]]
        .sort_values([SITE_COL, DETECTOR_COL, "TIMESTAMP"], kind="mergesort")
    )

    # Parquet تحديد اسم ملف الإخراج بصيغة
    base = os.path.splitext(os.path.basename(csv_path))[0]
    out_path = os.path.join(PROC_DIR, f"{base}.parquet")

    # حفظ الملف
    out.to_parquet(out_path, index=False)

    # إرجاع مسار الملف المحفوظ
    return out_path
```

```

saved = []

for i, f in enumerate(csv_files, 1):
    p = process_one_file(f)

    # حفظ المسار
    saved.append(p)

    # طباعة تقدم المعالجة
    print(f"[{i}/{len(csv_files)}] saved -> {p}")

# يطبع عدد الملفات الكاملة في النهاية
print("Number of Parquet files:", len(saved))

```

```

[1/31] saved -> /content/processed/VSDATA_20250301.parquet
[2/31] saved -> /content/processed/VSDATA_20250302.parquet
[3/31] saved -> /content/processed/VSDATA_20250303.parquet
[4/31] saved -> /content/processed/VSDATA_20250304.parquet
[5/31] saved -> /content/processed/VSDATA_20250305.parquet
[6/31] saved -> /content/processed/VSDATA_20250306.parquet
[7/31] saved -> /content/processed/VSDATA_20250307.parquet
[8/31] saved -> /content/processed/VSDATA_20250308.parquet
[9/31] saved -> /content/processed/VSDATA_20250309.parquet
[10/31] saved -> /content/processed/VSDATA_20250310.parquet
[11/31] saved -> /content/processed/VSDATA_20250311.parquet
[12/31] saved -> /content/processed/VSDATA_20250312.parquet
[13/31] saved -> /content/processed/VSDATA_20250313.parquet
[14/31] saved -> /content/processed/VSDATA_20250314.parquet
[15/31] saved -> /content/processed/VSDATA_20250315.parquet
[16/31] saved -> /content/processed/VSDATA_20250316.parquet
[17/31] saved -> /content/processed/VSDATA_20250317.parquet
[18/31] saved -> /content/processed/VSDATA_20250318.parquet
[19/31] saved -> /content/processed/VSDATA_20250319.parquet
[20/31] saved -> /content/processed/VSDATA_20250320.parquet
[21/31] saved -> /content/processed/VSDATA_20250321.parquet
[22/31] saved -> /content/processed/VSDATA_20250322.parquet
[23/31] saved -> /content/processed/VSDATA_20250323.parquet
[24/31] saved -> /content/processed/VSDATA_20250324.parquet
[25/31] saved -> /content/processed/VSDATA_20250325.parquet
[26/31] saved -> /content/processed/VSDATA_20250326.parquet
[27/31] saved -> /content/processed/VSDATA_20250327.parquet
[28/31] saved -> /content/processed/VSDATA_20250328.parquet
[29/31] saved -> /content/processed/VSDATA_20250329.parquet
[30/31] saved -> /content/processed/VSDATA_20250330.parquet
[31/31] saved -> /content/processed/VSDATA_20250331.parquet
Number of Parquet files: 31

```

```

# تعريف مسار مجلد الملفات المعالجة

import os

PROC_DIR = "/content/processed"

# نتأكد أن المجلد موجود
assert os.path.exists(PROC_DIR), "Processed folder does not exist."

print("Processed folder found:", PROC_DIR)

```

```
Processed folder found: /content/processed
```

```

# إعدادات

import os
import pandas as pd
import numpy as np

INTERVAL_MIN = 15          # كل قراءة تمثل 15 دقيقة
HORIZON_MIN = 15           # الهدف التنبؤ بعد 15 دقيقة

SITE_COL = "NB_SCATS_SITE"  # اسم العمود الذي يمثل موقع الإشارة
DETECTOR_COL = "NB_DETECTOR" # اسم العمود الذي يمثل رقم الحساس داخل التقاطع

PROC_DIR = "/content/processed" # مجلد ملفات Parquet الناتجة من process_one_file
FEATURED_DIR = "/content/featured" # مجلد جديد لحفظ الملفات بعد إضافة الميزات

THRESHOLD_QUANTILE = 0.85 # أعلى 15% من قراءات تعتبر ازدحام

os.makedirs(FEATURED_DIR, exist_ok=True) # إذا ما كان موجود featured إنشاء مجلد

print("PROC_DIR:", PROC_DIR) # للتأكد طباعة مسار
print("FEATURED_DIR:", FEATURED_DIR) # للتأكد featured طباعة مسار

```

```
PROC_DIR: /content/processed
FEATURED_DIR: /content/featured
```

```
# جلب ملفات Parquet من مجلد processed

# PROC_DIR داخل parquet. نجيب كل الملفات التي تنتهي بـ
pfiles = sorted([
    f for f in os.listdir("processed")
    if f.endswith(".parquet") and f != "test.parquet"
])

print(" Parquet files found:", len(pfiles)) # عدد الملفات
print(" First 5 files:", pfiles[:5])        # عرض أول 5 أسماء للتأكد
```

```
Parquet files found: 31
First 5 files: ['VSDATA_20250301.parquet', 'VSDATA_20250302.parquet', 'VSDATA_20250303.parquet', 'VSDATA_20250304.parquet',
```

```
# واحد Parquet لملف Feature Engineering دالة تعمل
# الهدف: نضيف HOUR/DAY_OF_WEEK/IS_WEEKEND + LAGs + FUTURE_CONGESTION
# ثم نحفظ الناتج بشكل خفيف لتقليل RAM

def featurize_one_parquet(parquet_path: str) -> pd.DataFrame:

    df = pd.read_parquet(parquet_path) # قراءة الملف

    # processed تأكد من الأعمدة الموجودة فعلاً في
    must = [SITE_COL, DETECTOR_COL, "TIMESTAMP", "VOLUME"]
    missing = [c for c in must if c not in df.columns]
    if missing:
        raise ValueError(f"Missing columns {missing} in {parquet_path}")

    # تأكد من نوع الوقت
    df["TIMESTAMP"] = pd.to_datetime(df["TIMESTAMP"], errors="coerce")

    # shift ترتيب مهم قبل
    df = df.sort_values(["NB_SCATS_SITE", "NB_DETECTOR", "TIMESTAMP"], kind="mergesort")

    # عدد الخطوات للمستقبل
    shift_steps = HORIZON_MIN // INTERVAL_MIN # = 1

    # إنشاء future_volume
    g = df.groupby(["NB_SCATS_SITE", "NB_DETECTOR"])["VOLUME"]
    df["future_volume"] = g.shift(-1)

    # خصائص الوقت
    df["HOUR"] = df["TIMESTAMP"].dt.hour
    df["DAY_OF_WEEK"] = df["TIMESTAMP"].dt.dayofweek
    df["IS_WEEKEND"] = df["DAY_OF_WEEK"].isin([4, 5]).astype(np.int8)

    # LAG Features
    df["LAG_1"] = g.shift(1)
    df["LAG_2"] = g.shift(2)
    df["LAG_4"] = g.shift(4)

    # shift حذف الصفوف المتأثرة بالـ
    df = df.dropna(subset=["LAG_1", "LAG_2", "LAG_4", "future_volume"]).copy()

    # إنشاء الهدف
    df["TARGET"] = (df["future_volume"] >= THRESH).astype(np.int8)

    # تقليل استهلاك الذاكرة
    float_cols = ["VOLUME", "LAG_1", "LAG_2", "LAG_4", "future_volume"]
    for c in float_cols:
        df[c] = df[c].astype(np.float32)

    df["HOUR"] = df["HOUR"].astype(np.int8)
    df["DAY_OF_WEEK"] = df["DAY_OF_WEEK"].astype(np.int8)

    return df
```

```
# processed من عينة من أول ملف THRESH حساب
sample_file = os.path.join(PROC_DIR, pfiles[0])
df_sample = pd.read_parquet(sample_file)

df_sample = df_sample.sort_values(["NB_SCATS_SITE", "NB_DETECTOR", "TIMESTAMP"])
g = df_sample.groupby(["NB_SCATS_SITE", "NB_DETECTOR"])["VOLUME"]
df_sample["future_volume"] = g.shift(-1)

THRESH = df_sample["future_volume"].quantile(0.85)
```

```
print("THRESH =", THRESH)
```

```
THRESH = 24.0
```

```
# بدون دمج كامل Parquet على كل ملفات Feature Engineering تشغيل
# ثم حفظ ناتج كل ملف في FEATURED_DIR

featured_paths = [] # قائمة لحفظ مسارات الملفات الناتجة (featured)
failed = [] # قائمة لحفظ الملفات التي فشلت (لو فيه ملف فيه مشكلة)

os.makedirs(FEATURED_DIR, exist_ok=True) # إذا غير موجود FEATURED_DIR إنشاء مجلد

for i, f in enumerate(pfiles, 1): # مع عداد يبدأ من 1 parquet ملف على كل ملفات
    in_path = os.path.join(PROC_DIR, f) # مسار ملف الإدخال (processed)
    out_path = os.path.join(FEATURED_DIR, f) # مسار ملف الإخراج (featured)

    try:
        df_feat = featurize_one_parquet(in_path) # على الملف (بدون دمج) Feature Engineering تشغيل
        df_feat.to_parquet(out_path, index=False) # داخل FEATURED_DIR parquet حفظ الناتج كـ
        featured_paths.append(out_path) # حفظ المسار في القائمة

        # طباعة تقدم التنفيذ كل 5 ملفات + أول ملف + آخر ملف
        if i % 5 == 0 or i == 1 or i == len(pfiles):
            print(f"[{i}/{len(pfiles)}] OK -> {f} | shape={df_feat.shape}")

    except Exception as e:
        failed.append((f, str(e))) # تخزين اسم الملف ورسالة الخطأ
        print(f"[{i}/{len(pfiles)}] FAILED -> {f} | {e}") # عرض الخطأ مباشرة للتشخيص

print("\n=====")
print(" Total featured files:", len(featured_paths)) # عدد الملفات التي نجحت
print(" Failed files:", len(failed)) # عدد الملفات التي فشلت
if failed:
    print("First 3 failures:", failed[:3]) # عرض أول 3 أخطاء (للتشخيص)
```

```
[1/31] OK -> VSDATA_20250301.parquet | shape=(10360259, 13)
[5/31] OK -> VSDATA_20250305.parquet | shape=(10360259, 13)
[10/31] OK -> VSDATA_20250310.parquet | shape=(10366083, 13)
[15/31] OK -> VSDATA_20250315.parquet | shape=(10368995, 13)
[20/31] OK -> VSDATA_20250320.parquet | shape=(10369723, 13)
[25/31] OK -> VSDATA_20250325.parquet | shape=(10374091, 13)
[30/31] OK -> VSDATA_20250330.parquet | shape=(10374091, 13)
[31/31] OK -> VSDATA_20250331.parquet | shape=(10374091, 13)
```

```
=====
Total featured files: 31
Failed files: 0
```

```
# بدون دمج كل البيانات (Preview) عينة للعرض فقط

SAMPLE_PER_FILE = 5000 # عدد صفوف نأخذها من كل ملف
MAX_FILES = 5 # نأخذ عينة من أول 5 ملفات فقط

parts = [] # قائمة لتجميع العينات

for fp in featured_paths[:min(MAX_FILES, len(featured_paths))]:
    d = pd.read_parquet(fp)
    if len(d) > SAMPLE_PER_FILE:
        d = d.sample(n=SAMPLE_PER_FILE, random_state=42)
    parts.append(d)

df_all_sample = pd.concat(parts, ignore_index=True) if parts else pd.DataFrame()

print(" df_all_sample shape:", df_all_sample.shape) # حجم العينة
df_all_sample.head() # عرض أول الصفوف
```

```
df_all_sample shape: (25000, 13)
```

	NB_SCATS_SITE	NB_DETECTOR	TIMESTAMP	VOLUME	INTERVAL_IDX	future_volume	HOUR	DAY_OF_WEEK	IS_WEEKEND	LAG_1	LAG_2	L
0	6681	20	2025-03-01 17:45:00	0.0	71	0.0	17	5	1	0.0	0.0	
1	5455	11	2025-03-01 08:45:00	0.0	35	0.0	8	5	1	0.0	0.0	
			2025-03-									

```
# تثبيت XGBoost
!pip install -q xgboost
```

```
# تم تثبيته XGBoost التأكد ان
import xgboost
print("XGBoost version:", xgboost.__version__)
```

```
XGBoost version: 3.2.0
```

```
# تقسيم زمني للملفات

# featured عدد ملفات
split_f = int(len(featured_paths) * 0.8)

# أول 80% تريب
train_files = featured_paths[:split_f]

# آخر 20% اختبار
test_files = featured_paths[split_f:]

print("Train files:", len(train_files))
print("Test files:", len(test_files))
```

```
Train files: 24
Test files: 7
```

```
# تعريف قائمة ملفات parquet
PROC_DIR = "processed"

pq_files = sorted([
    os.path.join(PROC_DIR, f)
    for f in os.listdir(PROC_DIR)
    if f.endswith(".parquet") and f != "test.parquet"
])
```

```
print("عدد الملفات:", len(pq_files))
print("3 أول:", [os.path.basename(x) for x in pq_files[:3]])
```

```
عدد الملفات: 31
3 أول: ['VSDATA_20250301.parquet', 'VSDATA_20250302.parquet', 'VSDATA_20250303.parquet']
```

```
# إنشاء FEATURED
import os

FEATURED_DIR = "featured_parquet"
os.makedirs(FEATURED_DIR, exist_ok=True)

print("FEATURED بدء إنشاء ملفات...")

featured_paths = []
failed = []

for i, fp in enumerate(pq_files, 1):
    try:
        df_feat = featurize_one_parquet(fp)

        out_path = os.path.join(FEATURED_DIR, os.path.basename(fp))
        df_feat.to_parquet(out_path, index=False)
        featured_paths.append(out_path)

        if i % 5 == 0 or i == 1 or i == len(pq_files):
            print(f"[{i}/{len(pq_files)}] OK -> {os.path.basename(out_path)} | shape={df_feat.shape}")

    except Exception as e:
        failed.append((os.path.basename(fp), str(e)))
        print(f"[{i}/{len(pq_files)}] FAILED -> {os.path.basename(fp)} | {e}")

print("\n=====")
print("Total featured files:", len(featured_paths))
print("Failed files:", len(failed))
if failed:
    print("First 3 failures:", failed[:3])
```

```
FEATURED بدء إنشاء ملفات...
[1/31] OK -> VSDATA_20250301.parquet | shape=(10360259, 13)
[5/31] OK -> VSDATA_20250305.parquet | shape=(10360259, 13)
[10/31] OK -> VSDATA_20250310.parquet | shape=(10366083, 13)
[15/31] OK -> VSDATA_20250315.parquet | shape=(10368995, 13)
[20/31] OK -> VSDATA_20250320.parquet | shape=(10369723, 13)
[25/31] OK -> VSDATA_20250325.parquet | shape=(10374091, 13)
[30/31] OK -> VSDATA_20250330.parquet | shape=(10374091, 13)
```

```
[31/31] OK -> VSDATA_20250331.parquet | shape=(10374091, 13)
```

```
=====
Total featured files: 31
Failed files: 0
```

```
import os
import pandas as pd

CANDIDATE_DIRS = ["featured", "processed_featured", "featured_parquet", "featurized", "features", "processed"]

def list_parquets(d):
    if not os.path.isdir(d):
        return []
    return sorted([os.path.join(d, f) for f in os.listdir(d) if f.endswith(".parquet")])

pq_files = []

for d in CANDIDATE_DIRS:
    files = list_parquets(d)
    if not files:
        continue

    # فحص أول ملف لتحديد الأعمدة
    cols = pd.read_parquet(files[0]).head(2).columns.tolist()

    # خذ واعتمد Features إذا فيه
    if ("LAG_1" in cols) and ("TARGET" in cols):
        pq_files = files
        print("تم اختيار المجلد :", d)
        break

if not pq_files:
    print("ما لقيت ملفات LAG_1 و TARGET فيها Featured")
else:
    print("عدد الملفات:", len(pq_files))
    print("مثال ملف:", pq_files[0])
    print("أعمدة عينة:", pd.read_parquet(pq_files[0]).head(2).columns.tolist())
```

```
تم اختيار المجلد: featured
عدد الملفات: 31
مثال ملف: featured/VSDATA_20250301.parquet
أعمدة عينة: ['NB_SCATS_SITE', 'NB_DETECTOR', 'TIMESTAMP', 'VOLUME', 'INTERVAL_IDX', 'future_volume', 'HOUR', 'DAY_OF_WEEK', 'IS
```

```
# تحميل عينة للتدريب
FEATURES = ["VOLUME", "LAG_1", "LAG_2", "LAG_4",
            "HOUR", "DAY_OF_WEEK", "IS_WEEKEND"]

TARGET = "TARGET"

def load_sample(files, n_per_file=50000):
    parts = []
    for fp in files:
        df = pd.read_parquet(fp, columns=FEATURES + [TARGET])
        if len(df) > n_per_file:
            df = df.sample(n=n_per_file, random_state=42)
        parts.append(df)
    return pd.concat(parts, ignore_index=True)

train_df = load_sample(train_files, n_per_file=50000)
test_df = load_sample(test_files, n_per_file=50000)

X_train = train_df[FEATURES]
y_train = train_df[TARGET].astype(int)

X_test = test_df[FEATURES]
y_test = test_df[TARGET].astype(int)

print("X_train:", X_train.shape, "y_train:", y_train.shape, "Positive%:", y_train.mean().round(3))
print("X_test :", X_test.shape, "y_test :", y_test.shape, "Positive%:", y_test.mean().round(3))

assert set(np.unique(y_train)).issubset({0,1})
assert set(np.unique(y_test)).issubset({0,1})
```

```
X_train: (1200000, 7) y_train: (1200000,) Positive%: 0.159
X_test : (350000, 7) y_test : (350000,) Positive%: 0.16
```

```
# X و y تجهيز

# فصل الميزات عن الهدف (Train)
X_train = train_df[FEATURES]
```

```

y_train = train_df[TARGET]

# فصل الميزات عن الهدف (Test)
X_test = test_df[FEATURES]
y_test = test_df[TARGET]

# فحص NaN
nan_train = X_train.isna().sum().sum() + y_train.isna().sum()
nan_test = X_test.isna().sum().sum() + y_test.isna().sum()

print("NaN in Train:", int(nan_train))
print("NaN in Test:", int(nan_test))

# نحذف الصفوف المتأثرة: NaN: إذا فيه
if nan_train > 0:
    tmp = train_df.dropna(subset=FEATURES + [TARGET]).copy()
    X_train = tmp[FEATURES]
    y_train = tmp[TARGET]

if nan_test > 0:
    tmp = test_df.dropna(subset=FEATURES + [TARGET]).copy()
    X_test = tmp[FEATURES]
    y_test = tmp[TARGET]

print(" X_train shape:", X_train.shape)
print(" X_test shape:", X_test.shape)

```

```

NaN in Train: 0
NaN in Test: 0
X_train shape: (1200000, 7)
X_test shape: (350000, 7)

```

```

# مقارنة النماذج (Accuracy / Precision / Recall / F1 / ROC AUC)

import pandas as pd
import numpy as np

# استدعاء النماذج للمقارنة
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

# استدعاء مقاييس التقييم
from sklearn.metrics import (
    accuracy_score,          # الدقة العامة
    precision_score,         # Precision
    recall_score,            # Recall
    f1_score,                # F1
    roc_auc_score            # ROC AUC
)

def safe_roc_auc(model, X, y):
    if hasattr(model, "predict_proba"):
        proba = model.predict_proba(X)[: , 1]
        return roc_auc_score(y, proba)
    elif hasattr(model, "decision_function"):
        scores = model.decision_function(X)
        return roc_auc_score(y, scores)
    else:
        return np.nan

```

```

# تعريف النماذج التي بنقارنها

models = {
    #Logistic Regression
    "Logistic Regression": LogisticRegression(
        max_iter=500,
        class_weight="balanced"
    ),
    #Random Forest
    "Random Forest": RandomForestClassifier(
        n_estimators=200,          # عدد الأشجار
        max_depth=14,             # عمق الشجرة
        random_state=42,          # تثبيت النتائج
        n_jobs=-1,                # تسريع باستخدام كل الأنوية
        class_weight="balanced"   # يعالج عدم التوازن
    ),
    #Gradient Boosting
    "Gradient Boosting": GradientBoostingClassifier(
        random_state=42           # تثبيت النتائج
    )
}

```

```
print("Models to compare:", list(models.keys()))
```

```
Models to compare: ['Logistic Regression', 'Random Forest', 'Gradient Boosting']
```

```
# تدريب كل نموذج + حساب المقاييس + تجميع النتائج
```

```
results = []
```

```
# قائمة لتجميع نتائج كل نموذج
```

```
for name, model in models.items():
    model.fit(X_train, y_train)
```

```
# تدريب النموذج على بيانات التدريب
```

```
y_pred = model.predict(X_test)
```

```
# مقارنة النماذج (Accuracy / Precision / Recall / F1 / ROC AUC)
```

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

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, ExtraTreesClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
```

```
# XGBoost
try:
    from xgboost import XGBClassifier
    HAS_XGB = True
except Exception:
    HAS_XGB = False
```

```
# تحقق أن بيانات التدريب جاهزة
assert all(v in globals() for v in ["X_train", "X_test", "y_train", "y_test"]), \
    "Run the X_train/X_test/y_train/y_test preparation cell before comparison."
```

```
if "X_train_small" in globals() and "y_train_small" in globals() and "X_test_small" in globals() and "y_test_small" in glo
    Xtr, ytr = X_train_small, y_train_small
    Xte, yte = X_test_small, y_test_small
    print("Using SMALL dataset:", Xtr.shape, Xte.shape)
else:
    assert "X_train" in globals() and "y_train" in globals() and "X_test" in globals() and "y_test" in globals(), \
        "I couldn't find X_train/X_test.. Run the X_train and X_test preparation cell first."
    Xtr, ytr = X_train, y_train
    Xte, yte = X_test, y_test
    print("Using FULL dataset:", Xtr.shape, Xte.shape)
```

```
# تعريف النماذج
```

```
models = {
    "LogReg": LogisticRegression(max_iter=2000, n_jobs=-1),
    "RandomForest": RandomForestClassifier(n_estimators=300, max_depth=14, random_state=42, n_jobs=-1),
    "GradBoost": GradientBoostingClassifier(random_state=42),
    "ExtraTrees": ExtraTreesClassifier(n_estimators=500, random_state=42, n_jobs=-1),
}
```

```
if HAS_XGB:
    models["XGBoost"] = XGBClassifier(
        n_estimators=500,
        max_depth=6,
        learning_rate=0.05,
        subsample=0.8,
        colsample_bytree=0.8,
        reg_lambda=1.0,
        eval_metric="logloss",
        random_state=42,
        n_jobs=-1
    )
```

```
rows = []
for name, model in models.items():
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    # ROC-AUC
    if hasattr(model, "predict_proba"):
        y_score = model.predict_proba(X_test)[:, 1]
    else:
        y_score = model.decision_function(X_test)

    rows.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
```

```
"Precision": precision_score(y_test, y_pred, zero_division=0),
"Recall": recall_score(y_test, y_pred, zero_division=0),
"F1": f1_score(y_test, y_pred, zero_division=0),
"ROC_AUC": roc_auc_score(y_test, y_score),
})

results = pd.DataFrame(rows).sort_values("ROC_AUC", ascending=False)
display(results)
```

Using FULL dataset: (1200000, 7) (350000, 7)

	Model	Accuracy	Precision	Recall	F1	ROC_AUC
4	XGBoost	0.981166	0.942781	0.939487	0.941131	0.997679
1	RandomForest	0.980909	0.942880	0.937668	0.940267	0.997630
2	GradBoost	0.980780	0.942500	0.937240	0.939863	0.997466
0	LogReg	0.979363	0.949085	0.920605	0.934628	0.997043
3	ExtraTrees	0.979883	0.939370	0.934798	0.937078	0.996762

```
# مقارنة زمن التدريب بين النماذج

import time

time_results = []

for name, model in models.items():
    start = time.time()
    model.fit(X_train, y_train)
    train_time = time.time() - start

    time_results.append({
        "Model": name,
        "Training Time (sec)": train_time
    })

time_df = pd.DataFrame(time_results)
print(time_df)
```

	Model	Training Time (sec)
0	LogReg	3.420531
1	RandomForest	82.132786
2	GradBoost	100.880697
3	ExtraTrees	86.099509
4	XGBoost	13.622698

```
# بدون ضغط كل الداتا بالرام Parquet تدريب وتقييم من ملفات

import os, glob, re
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# استدعاء النموذج
from sklearn.ensemble import RandomForestClassifier

# استدعاء أدوات التقييم
from sklearn.metrics import (
    classification_report, confusion_matrix,
    roc_auc_score, roc_curve,
    precision_recall_curve
)

# تحديد المسارات (عند حسب مجلدك)
OUT_PQ_DIR = "/content/drive/MyDrive/Traffic_parquet" # parquet مكان ملفات
OUT_DIR = "/content/outputs" # مجلد المخرجات
FIG_DIR = os.path.join(OUT_DIR, "figures") # مجلد الصور
REP_DIR = os.path.join(OUT_DIR, "reports") # مجلد التقارير

# إنشاء المجلدات لو غير موجودة
os.makedirs(FIG_DIR, exist_ok=True)
os.makedirs(REP_DIR, exist_ok=True)

# حجم العينات لكل ملف (لتخفيف ضغط الرام)
TRAIN_ROWS_PER_FILE = 120_000 # عينة من كل ملف تدريب
TEST_ROWS_PER_FILE = 120_000 # عينة من كل ملف اختبار

# (من ملفات التدريب) عينة لحساب
THR_SAMPLE_PER_FILE = 50_000

# تعريف الازدحام (أعلى 15% ازدحام)
```

```
THRESHOLD_QUANTILE = 0.85
```

```
# تثبيت العشوائية
```

```
RANDOM_STATE = 42
```

```
print("OUT_PQ_DIR:", OUT_PQ_DIR)
```

```
print("FIG_DIR:", FIG_DIR)
```

```
print("REP_DIR:", REP_DIR)
```

```
OUT_PQ_DIR: /content/drive/MyDrive/Traffic_parquet
```

```
FIG_DIR: /content/outputs/figures
```

```
REP_DIR: /content/outputs/reports
```

```
# دالة حفظ الصور
```

```
def savefig(name):
```

```
    # name: اسم الصورة داخل figures
```

```
    path = os.path.join(FIG_DIR, name) # بناء مسار الحفظ
```

```
    plt.tight_layout() # ترتيب الشكل
```

```
    plt.savefig(path, dpi=170) # حفظ الصورة بجودة مناسبة
```

```
    plt.close()
```

```
    return path
```

```
# وترتيبها زمنياً Parquet جلب ملفات
```

```
# استدعاء ملفات Parquet
```

```
pq_files = sorted([
    os.path.join("featured_parquet", f)
    for f in os.listdir("featured_parquet")
    if f.endswith(".parquet")
])
```

```
print("Number of Parquet files:", len(pq_files))
```

```
assert len(pq_files) >= 2, "If you didn't find Parquet, make sure the conversion step worked."
```

```
# دالة استخراج التاريخ من اسم الملف (مثل 20250301)
```

```
def extract_date_from_name(path):
```

```
    base = os.path.basename(path)
```

```
    m = re.search(r"(\d{8})", base)
```

```
    return m.group(1) if m else base
```

```
# ترتيب الملفات حسب التاريخ
```

```
pq_files = sorted(pq_files, key=extract_date_from_name)
```

```
print("First file:", os.path.basename(pq_files[0]))
```

```
print("Last file:", os.path.basename(pq_files[-1]))
```

```
Number of Parquet files: 31
```

```
First file: VSDATA_20250301.parquet
```

```
Last file: VSDATA_20250331.parquet
```

```
# على مستوى الملفات Train/Test تقسيم زمني
```

```
split_ratio = 0.80 # تدريب 80%
```

```
split_idx = int(len(pq_files) * split_ratio)
```

```
train_files = pq_files[:split_idx] # ملفات التدريب (أيام أقدم)
```

```
test_files = pq_files[split_idx:] # ملفات الاختبار (أيام أحدث)
```

```
print("Train days:", len(train_files), "Test days:", len(test_files))
```

```
print("First Train:", os.path.basename(train_files[0]))
```

```
print("First Test :", os.path.basename(test_files[0]))
```

```
Train days: 24 Test days: 7
```

```
First Train: VSDATA_20250301.parquet
```

```
First Test : VSDATA_20250325.parquet
```

```
# future_volume بناءً على threshold حساب
```

```
future_samples = [] # من الترتيب future_volume نخزن عينات
```

```
for f in train_files:
```

```
    df = pd.read_parquet(f, columns=["future_volume"])
```

```
    if len(df) > THR_SAMPLE_PER_FILE:
```

```
        df = df.sample(THR_SAMPLE_PER_FILE, random_state=RANDOM_STATE) # عينة لتخفيف الرام
```

```
        future_samples.append(df["future_volume"].to_numpy()) # تحويل لمصفوفة
```

```
# دمج كل العينات في مصفوفة واحدة
```

```
future_samples = np.concatenate(future_samples)
```

```
# حساب threshold حسب quantile
THRESH = float(np.quantile(future_samples, THRESHOLD_QUANTILE))

print("congestion threshold (future_volume) =", THRESH)
```

```
congestion threshold (future_volume) = 26.0
```

```
# تدريب

# الميزات المستخدمة
feature_cols = ["VOLUME", "LAG_1", "LAG_2", "LAG_4", "HOUR", "DAY_OF_WEEK", "IS_WEEKEND"]

X_list, y_list = [], [] # X و y قوائم لتجميع

for i, f in enumerate(train_files, 1):
    # فقط قراءة الميزات + future_volume
    df = pd.read_parquet(f, columns=feature_cols + ["future_volume"])

    # أخذ عينة إذا الملف كبير
    if len(df) > TRAIN_ROWS_PER_FILE:
        df = df.sample(TRAIN_ROWS_PER_FILE, random_state=RANDOM_STATE)

    X_list.append(df[feature_cols])
    y_list.append(df["future_volume"])

X_train = pd.concat(X_list, ignore_index=True)
y_train = pd.concat(y_list, ignore_index=True)

print(f"[Train {i}/{len(train_files)}] rows={len(df):,}")

# دمج كل أيام التدريب في مصفوفة واحدة
X_train = np.vstack(X_list)
y_train = np.concatenate(y_list)

print("Train matrix:", X_train.shape, "labels:", y_train.shape)
print("نسبة الاندحام (train sample):", round(float(y_train.mean()), 4))
```

```
[Train 24/24] rows=120,000
Train matrix: (2880000, 7) labels: (2880000,)
نسبة الاندحام (train sample): 13.2779
```

```
# Random Forest تدريب النموذج

rf = RandomForestClassifier(
    n_estimators=150,          # عدد الأشجار
    max_depth=18,              # تحديد عمق الشجر
    max_features="sqrt",      # يقلل التعقيد
    min_samples_leaf=5,
    min_samples_split=10,
    bootstrap=True,
    oob_score=False,
    random_state=RANDOM_STATE,
    n_jobs=-1,
    class_weight="balanced",
)
rf.fit(X_train, y_train)
print("The model was trained")
```

```
The model was trained
```

```
# اختبار
all_proba = [] # لتجميع احتمالات الفئة 1
all_true = [] # لتجميع القيم الحقيقية

for i, f in enumerate(test_files, 1):
    df = pd.read_parquet(f, columns=feature_cols + ["future_volume"])

    # عينة من ملف الاختبار
    if len(df) > TEST_ROWS_PER_FILE:
        df = df.sample(TEST_ROWS_PER_FILE, random_state=RANDOM_STATE)

    # القيم الحقيقية
    y_true = (df["future_volume"].to_numpy() >= THRESH).astype(np.int8)

    # الميزات
    X = df[feature_cols].fillna(0).to_numpy(dtype=np.float32)

    # احتمالات التنبؤ
    proba = rf.predict_proba(X)[: , 1]
```

```

all_proba.append(proba)
all_true.append(y_true)

print(f"[Test {i}/{len(test_files)}] rows={len(df):,}")

# دمج كل أيام الاختبار
y_test = np.concatenate(all_true)
proba = np.concatenate(all_proba)

# باستخدام 0/1 إلى تحويل الاحتمالات إلى 0.5 threshold
pred = (proba >= 0.5).astype(np.int8)

print(" Test merged:", y_test.shape, proba.shape)

```

```

[Test 1/7] rows=120,000
[Test 2/7] rows=120,000
[Test 3/7] rows=120,000
[Test 4/7] rows=120,000
[Test 5/7] rows=120,000
[Test 6/7] rows=120,000
[Test 7/7] rows=120,000
Test merged: (840000,) (840000,)

```

```

# التقييم + حفظ التقرير + الرسومات

# تقرير التصنيف
rep = classification_report(y_test, pred, digits=4)
print(rep)

# حفظ التقرير في ملف
report_path = os.path.join(REP_DIR, "classification_report.txt")
with open(report_path, "w", encoding="utf-8") as f:
    f.write(rep)

# AUC + مصفوفة الالتباس
cm = confusion_matrix(y_test, pred)
auc = roc_auc_score(y_test, proba)
print("AUC:", auc)

# Confusion Matrix
plt.figure()
plt.imshow(cm)
plt.title("Confusion Matrix")
plt.xlabel("Predicted"); plt.ylabel("True")
for r in range(cm.shape[0]):
    for c in range(cm.shape[1]):
        plt.text(c, r, str(cm[r, c]), ha="center", va="center")
p_cm = savefig("cm.png")

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, proba)
plt.figure()
plt.plot(fpr, tpr)
plt.title(f"ROC Curve (AUC={auc:.4f})")
plt.xlabel("FPR"); plt.ylabel("TPR")
plt.plot([0, 1], [0, 1], "--")
p_roc = savefig("roc.png")

# Precision-Recall Curve
prec, rec, _ = precision_recall_curve(y_test, proba)
plt.figure()
plt.plot(rec, prec)
plt.title("Precision-Recall Curve")
plt.xlabel("Recall"); plt.ylabel("Precision")
p_pr = savefig("pr.png")

print(" Images saved:", p_cm, p_roc, p_pr)
print(" Report saved:", report_path)

```

	precision	recall	f1-score	support
0	0.6221	0.2966	0.4017	711742
1	0.0000	0.0000	0.0000	128258
accuracy			0.2513	840000
macro avg	0.3110	0.1483	0.2008	840000
weighted avg	0.5271	0.2513	0.3404	840000

```

AUC: 0.03766613112162038
Images saved: /content/outputs/figures/cm.png /content/outputs/figures/roc.png /content/outputs/figures/pr.png
Report saved: /content/outputs/reports/classification_report.txt

```

```
# حفظ نموذج Random Forest في rf

from sklearn.ensemble import RandomForestClassifier

if "Random Forest" in models:
    rf = models["Random Forest"]
else:
    rf = RandomForestClassifier(
        n_estimators=150,
        max_depth=18,
        min_samples_leaf=5,
        random_state=RANDOM_STATE,
        n_jobs=-1,
        class_weight="balanced"
    )

# تأكد إنه متدرب (لو ما كان متدرب ندره الآن)
try:
    _ = rf.feature_importances_
    print("Random Forest is present and training")
except Exception:
    rf.fit(X_train, y_train)
    print("Random Forest was trained and kept in rf")
```

Random Forest was trained and kept in rf

```
import pandas as pd
import os

# تأكد المجلد موجود
REP_DIR = "reports"
os.makedirs(REP_DIR, exist_ok=True)

# تأكد النموذج مدرب
assert hasattr(rf, "feature_importances_"), "Random Forest is not trained."

feature_importances = pd.Series(
    rf.feature_importances_,
    index=feature_cols
).sort_values(ascending=False)

top_features = feature_importances.head(10)

top_features_path = os.path.join(REP_DIR, "top_features_random_forest.csv")
top_features.to_csv(top_features_path, header=["importance"])

print("The top 10 features of Random Forest have been saved:", top_features_path)
display(top_features)
```

The top 10 features of Random Forest have been saved: reports/top_features_random_forest.csv

	0
VOLUME	0.199274
LAG_1	0.187723
LAG_4	0.186755
LAG_2	0.184979
HOURL	0.137318
DAY_OF_WEEK	0.081751
IS_WEEKEND	0.022200

dtype: float64

```
# ضغط النتائج + إنشاء تقرير Markdown

import os
import pandas as pd
import zipfile
from datetime import datetime
import matplotlib.pyplot as plt

OUT_DIR = "/content/outputs"
FIG_DIR = os.path.join(OUT_DIR, "figures")
REP_DIR = os.path.join(OUT_DIR, "reports")

# إنشاء المجلدات
os.makedirs(OUT_DIR, exist_ok=True)
```

```

os.makedirs(FIG_DIR, exist_ok=True)
os.makedirs(REP_DIR, exist_ok=True)

# توحيد العتبة
if "thr" not in globals():
    if "THRESH" in globals():
        thr = THRESH
    else:
        raise ValueError("لازم تحسب العتبة قبل التقرير. ولا THRESH لا يوجد.")

# متغيرات اختيارية
if "THRESHOLD_QUANTILE" not in globals():
    THRESHOLD_QUANTILE = float("nan")

if "auc" not in globals():
    auc = float("nan")

if "rep" not in globals():
    rep = "N/A (classification report not available)"

# استخراج Feature Importance من RF
assert "rf" in globals(), "rf غير معرف"
assert hasattr(rf, "feature_importances_"), "Random Forest مدرب"
assert "feature_cols" in globals(), "feature_cols غير معرف"

feature_importances = pd.Series(rf.feature_importances_, index=feature_cols).sort_values(ascending=False)
top_features = feature_importances.head(10)

# حفظ أعلى 10 ميزات
top_features_path = os.path.join(REP_DIR, "top_features.csv")
top_features.to_csv(top_features_path, header=["importance"])

# رسم Feature Importance (Top 20)
plt.figure(figsize=(10, 4))
feature_importances.head(20).sort_values().plot(kind="barh")
plt.title("Feature Importance (Top 20)")
plt.xlabel("Importance")
plt.tight_layout()

fi_path = os.path.join(FIG_DIR, "feature_importance.png")
plt.savefig(fi_path, dpi=170)
plt.close()

# --- بناء التقرير ---
report_lines = []
report_lines.append("# Traffic Congestion Prediction Report")
report_lines.append(f"Generated on: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
report_lines.append("")

report_lines.append("### Dataset Overview")
report_lines.append(f"- Number of daily files: {len(pq_files) if 'pq_files' in globals() else 'N/A'}")
report_lines.append(f"- Training days: {len(train_files) if 'train_files' in globals() else 'N/A'}")
report_lines.append(f"- Testing days: {len(test_files) if 'test_files' in globals() else 'N/A'}")
report_lines.append(f"- Train sample size: {len(y_train):,} if 'y_train' in globals() else '- Train sample size: N/A')")
report_lines.append(f"- Test sample size: {len(y_test):,} if 'y_test' in globals() else '- Test sample size: N/A')")
report_lines.append("")

report_lines.append("### Congestion Definition")
report_lines.append(f"- Threshold percentile: {THRESHOLD_QUANTILE*100:.0f}%" if pd.notna(THRESHOLD_QUANTILE) else "- Thresh")
report_lines.append(f"- Threshold value (future_volume): {thr:.4f}")
report_lines.append("")

report_lines.append("### Model")
report_lines.append("RandomForestClassifier")
report_lines.append("")
report_lines.append("### Evaluation Results (Test Set)")
report_lines.append(f"- ROC AUC: {auc:.4f}" if pd.notna(auc) else "- ROC AUC: N/A")
report_lines.append("")

report_lines.append("### Classification Report")
report_lines.append("```")
report_lines.append(rep)
report_lines.append("```")
report_lines.append("")

report_lines.append("### Top 10 Most Important Features")
for feat, val in top_features.items():
    report_lines.append(f"- {feat} : {val:.4f}")
report_lines.append("")

report_lines.append("### Generated Figures")
report_lines.append("")

```

```

report_lines.append("### Confusion Matrix")
report_lines.append("![Confusion Matrix](../figures/cm.png)")
report_lines.append("")
report_lines.append("### ROC Curve")
report_lines.append("![ROC Curve](../figures/roc.png)")
report_lines.append("")
report_lines.append("### Precision-Recall Curve")
report_lines.append("![Precision-Recall Curve](../figures/pr.png)")
report_lines.append("")
report_lines.append("### Feature Importance")
report_lines.append("![Feature Importance](../figures/feature_importance.png)")
report_lines.append("")
report_lines.append("---")
report_lines.append("End of Report")

report_md = "\n".join(report_lines)
report_path = os.path.join(REP_DIR, "final_report.md")

with open(report_path, "w", encoding="utf-8") as f:
    f.write(report_md)

print("The report was created:", report_path)
print("Top features CSV:", top_features_path)
print("Feature importance figure:", fi_path)

# --- ضغط كل النتائج ---
zip_path = "/content/Traffic_Project_Final_Outputs.zip"

with zipfile.ZipFile(zip_path, "w", zipfile.ZIP_DEFLATED) as z:
    for root, _, files in os.walk(OUT_DIR):
        for file in files:
            full_path = os.path.join(root, file)
            arc_name = os.path.relpath(full_path, OUT_DIR)
            z.write(full_path, arcname=arc_name)

print("ZIP file has been created:")
print(zip_path)

```

```

The report was created: /content/outputs/reports/final_report.md
Top features CSV: /content/outputs/reports/top_features.csv
Feature importance figure: /content/outputs/figures/feature_importance.png
ZIP file has been created:
/content/Traffic_Project_Final_Outputs.zip

```

```

# عرض التقرير

import os
from IPython.display import Markdown, display

REPORT_PATH = "/content/outputs/reports/final_report.md"
FIG_DIR = "/content/outputs/figures"
COLAB_REPORT_PATH = "/content/outputs/reports/final_report_colab.md"

# تأكد أن التقرير موجود
if not os.path.exists(REPORT_PATH):
    raise FileNotFoundError(f"Report not found: {REPORT_PATH}")

# قراءة التقرير الأصلي
with open(REPORT_PATH, "r", encoding="utf-8") as f:
    md = f.read()

# تحويل مسارات الصور إلى مسارات مطلقة تتناسب كولا ب
def make_images_absolute(md_text: str, fig_dir: str) -> str:
    md_text = md_text.replace("../figures/", fig_dir.rstrip("/") + "/")
    md_text = md_text.replace("figures/", fig_dir.rstrip("/") + "/")
    return md_text

md_abs = make_images_absolute(md, FIG_DIR)

# حفظ نسخة خاصة بكولا ب
with open(COLAB_REPORT_PATH, "w", encoding="utf-8") as f:
    f.write(md_abs)

print("Colab-ready report saved to:", COLAB_REPORT_PATH)

# تحذير إذا صورة غير موجودة
missing_imgs = []
for name in ["cm.png", "roc.png", "pr.png", "feature_importance.png"]:
    p = os.path.join(FIG_DIR, name)
    if not os.path.exists(p):
        missing_imgs.append(p)

```

```
if missing_imgs:
    print(" Warning: Some images are missing:")
    for p in missing_imgs:
        print("-", p)

# عرض التقرير داخل كولا ب
display(Markdown(md_abs))
```

Colab-ready report saved to: /content/outputs/reports/final_report_colab.md

Traffic Congestion Prediction Report

Generated on: 2026-03-01 14:23:56

Dataset Overview

- Number of daily files: 31
- Training days: 24
- Testing days: 7
- Train sample size: 2,880,000
- Test sample size: 840,000

Congestion Definition

- Threshold percentile: 85%
- Threshold value (future_volume): 26.0000

Model

RandomForestClassifier

Evaluation Results (Test Set)

- ROC AUC: 0.0377

Classification Report

	precision	recall	f1-score	support
0	0.6221	0.2966	0.4017	711742
1	0.0000	0.0000	0.0000	128258
accuracy			0.2513	840000
macro avg	0.3110	0.1483	0.2008	840000
weighted avg	0.5271	0.2513	0.3404	840000

Top 10 Most Important Features

- VOLUME : 0.1993
- LAG_1 : 0.1877
- LAG_4 : 0.1868
- LAG_2 : 0.1850
- HOUR : 0.1373
- DAY_OF_WEEK : 0.0818
- IS_WEEKEND : 0.0222

Generated Figures

Confusion Matrix



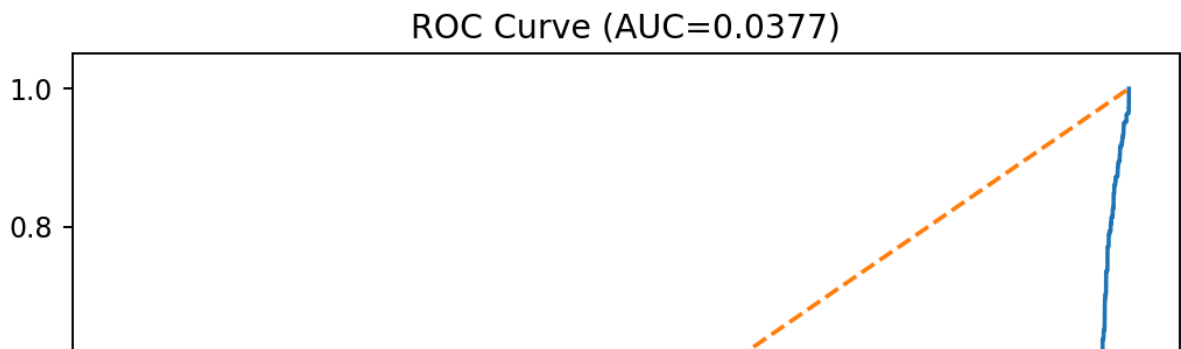
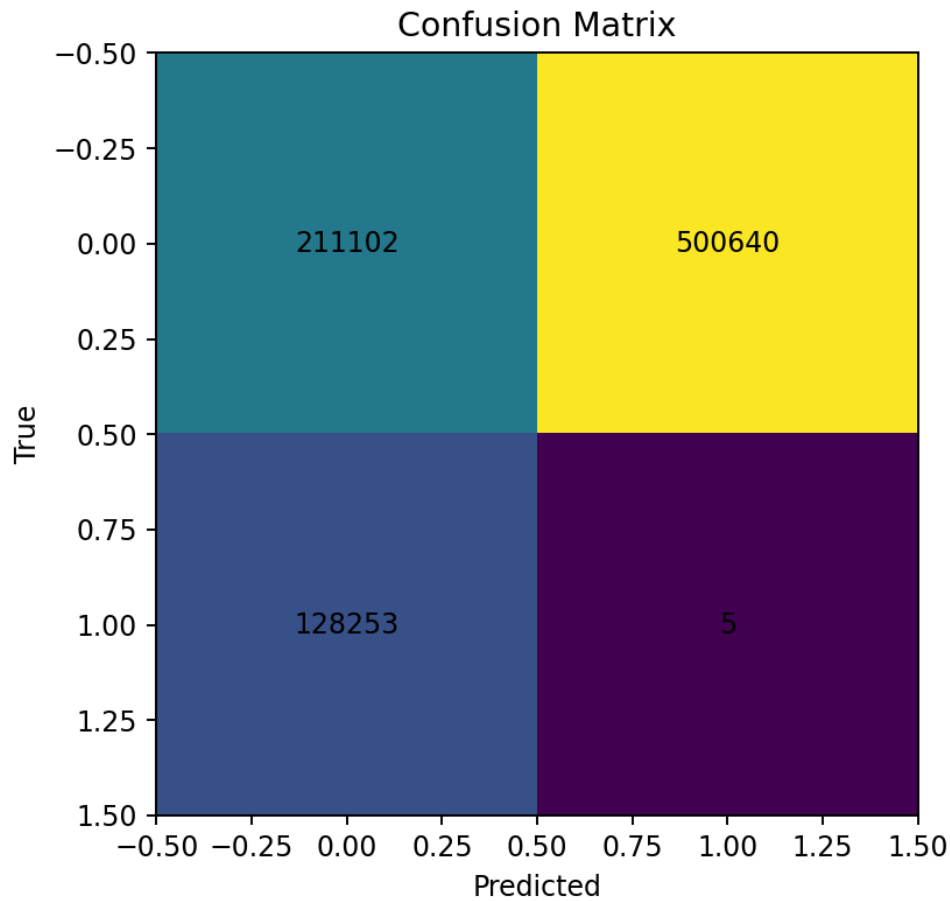
ROC Curve



Precision-Recall Curve

```
from IPython.display import Image, display

display(Image("/content/outputs/figures/cm.png"))
display(Image("/content/outputs/figures/roc.png"))
display(Image("/content/outputs/figures/pr.png"))
display(Image("/content/outputs/figures/feature_importance.png"))
```



```
import numpy as np
import pandas as pd

#تعريف feature_cols
feature_cols = [
    "VOLUME",
    "LAG_1",
    "LAG_2",
    "LAG_4",
    "HOUR",
    "DAY_OF_WEEK",
    "IS_WEEKEND"
]

#تجهيز قوائم جديدة
X_test_list = []
y_test_list = []

for f in test_files:
    df = pd.read_parquet(f, columns=feature_cols + ["TARGET"])

    X = df[feature_cols].to_numpy(dtype=np.float32)
    y = df["TARGET"].to_numpy(dtype=np.int8)

    X_test_list.append(X)
    y_test_list.append(y)
```

```
#دمج كل الأيام في مصفوفة واحدة
X_test = np.vstack(X_test_list)
y_test = np.concatenate(y_test_list)
```

```
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_test shape: (72618637, 7)
y_test shape: (72618637,)
```

```
# لمقارنة النماذج إنشاء results_df
```

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

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
```

```
rows = []
```

```
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(
        n_estimators=80,
        max_depth=18,
        min_samples_leaf=5,
        random_state=42,
        n_jobs=-1,
        class_weight="balanced"
    ),
    "Extra Trees": ExtraTreesClassifier(
        n_estimators=120,
        max_depth=18,
        min_samples_leaf=5,
        random_state=42,
        n_jobs=-1,
        class_weight="balanced"
    )
}
```

```
for name, model in models.items():
    print("Training:", name)
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    if hasattr(model, "predict_proba"):
        y_score = model.predict_proba(X_test)[:,-1]
    else:
        y_score = model.decision_function(X_test)

    rows.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1": f1_score(y_test, y_pred),
        "ROC_AUC": roc_auc_score(y_test, y_score)
    })
```

```
results_df = pd.DataFrame(rows).sort_values("ROC_AUC", ascending=False)
```

```
display(results_df)
```

```
Training: Logistic Regression
```

```
# رسم ROC_AUC مقارنة
```

```
import matplotlib.pyplot as plt
import os
```

```
OUT_DIR = "/content/outputs"
FIG_DIR = os.path.join(OUT_DIR, "figures")
os.makedirs(FIG_DIR, exist_ok=True)
```

```
# غير موجود، يعطينا تنبيه إذا
if "results_df" not in globals():
    print(" results_df غير موجود. شغل خلية مقارنة النماذج أولاً عشان يتولد results_df.")
```

```

else:
    # ترتيب النتائج حسب ROC_AUC
    results_sorted = results_df.sort_values("ROC_AUC", ascending=False)

    plt.figure(figsize=(8,5))
    plt.bar(results_sorted["Model"], results_sorted["ROC_AUC"])

    plt.title("Model Comparison (ROC AUC)")
    plt.ylabel("ROC AUC")
    plt.xticks(rotation=30)
    plt.ylim(0.0, 1.0)
    plt.tight_layout()

    cmp_path = os.path.join(FIG_DIR, "model_comparison_roc_auc.png")
    plt.savefig(cmp_path, dpi=150)
    plt.close()

    print("The drawing was saved:", cmp_path)

    # عرض الرسم
    plt.figure(figsize=(8,5))
    plt.bar(results_sorted["Model"], results_sorted["ROC_AUC"])
    plt.title("Model Comparison (ROC AUC)")
    plt.ylabel("ROC AUC")
    plt.xticks(rotation=30)
    plt.ylim(0.0, 1.0)
    plt.tight_layout()
    plt.show()

```

```

# حفظ جدول المقارنة كملف CSV داخل outputs/reports

import os

OUT_DIR = "/content/outputs"
REP_DIR = os.path.join(OUT_DIR, "reports")

# إنشاء المجلد إذا غير موجود
os.makedirs(REP_DIR, exist_ok=True)

# تحديد مكان الحفظ
cmp_csv = os.path.join(REP_DIR, "model_comparison.csv")

# موجود results_df تأكد أن
assert "results_df" in globals(), "results_df معرف غير"

# حفظ DataFrame
results_df.to_csv(cmp_csv, index=False)

print("The comparison table has been saved:", cmp_csv)

```

```

# Average Precision + PR Curve رسم

from sklearn.metrics import average_precision_score, precision_recall_curve
import matplotlib.pyplot as plt
import os

# موجودين y_test و proba تأكد أن
assert "y_test" in globals(), "y_test معرف غير"
assert "proba" in globals(), "proba معرف غير"

OUT_DIR = "/content/outputs"
FIG_DIR = os.path.join(OUT_DIR, "figures")
os.makedirs(FIG_DIR, exist_ok=True)

# حساب Average Precision
ap = average_precision_score(y_test, proba)

# حساب Precision-Recall Curve
prec, rec, _ = precision_recall_curve(y_test, proba)

# رسم المنحنى
plt.figure(figsize=(7,5))
plt.plot(rec, prec, label=f"AP = {ap:.4f}")
baseline = y_test.mean()
plt.hlines(baseline, 0, 1, linestyle="dashed", label="Baseline")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve")
plt.legend()
plt.grid(True)
plt.tight_layout()

```

```
# حفظ الصورة
ap_path = os.path.join(FIG_DIR, "pr.png")
plt.savefig(ap_path, dpi=150)
plt.close()

print("Average Precision:", round(ap, 4))
print("The drawing was saved:", ap_path)
```

```
# توزيع حجم البيانات المستقبلي

import os
import matplotlib.pyplot as plt
import numpy as np

OUT_DIR = "/content/outputs"
FIG_DIR = os.path.join(OUT_DIR, "figures")
os.makedirs(FIG_DIR, exist_ok=True)

# موجود future_samples تأكد أن
assert "future_samples" in globals(), "future_samples معرف غير"

if "thr" not in globals():
    if "THRESH" in globals():
        thr = THRESH
    else:
        raise ValueError("معرفين THRESH ولا thr لا")

future_samples = np.asarray(future_samples)

plt.figure(figsize=(7,5))

plt.hist(future_samples, bins=60, alpha=0.75)
plt.axvline(thr, color="red", linestyle="--", linewidth=2, label=f"Threshold = {thr:.2f}")

plt.title("Distribution of Future Volume with Threshold")
plt.xlabel("Future Volume")
plt.ylabel("Frequency")
plt.legend()
plt.tight_layout()

dist_path = os.path.join(FIG_DIR, "future_volume_distribution.png")
plt.savefig(dist_path, dpi=170)
plt.close()

print("The drawing was saved:", dist_path)
```

```
# رسم أهمية الميزات

import os
import matplotlib.pyplot as plt

OUT_DIR = "/content/outputs"
FIG_DIR = os.path.join(OUT_DIR, "figures")
os.makedirs(FIG_DIR, exist_ok=True)

# موجود feature_importances تأكد أن
assert "feature_importances" in globals(), "feature_importances معرف غير"

plt.figure(figsize=(8,5))

# نعرض فقط أفضل 15
feature_importances.head(15).sort_values().plot(kind="barh")

plt.title("Feature Importance (Top 15)")
plt.xlabel("Importance Score")
plt.tight_layout()

fi_path = os.path.join(FIG_DIR, "feature_importance.png")
plt.savefig(fi_path, dpi=170)
plt.close()

print("The drawing was saved:", fi_path)
```

```
# Confusion Matrix كنسب مئوية

import numpy as np
import matplotlib.pyplot as plt
import os
```

```

# موجود cm تأكد أن
assert "cm" in globals(), "cm غير معرف (لازم تحسب confusion_matrix)"

OUT_DIR = "/content/outputs"
FIG_DIR = os.path.join(OUT_DIR, "figures")
os.makedirs(FIG_DIR, exist_ok=True)

# تحويل إلى نسب مئوية
cm_norm = cm.astype(float) / cm.sum(axis=1, keepdims=True)

plt.figure(figsize=(6,5))
plt.imshow(cm_norm, cmap="Blues")
plt.colorbar()

plt.title("Normalized Confusion Matrix (%)")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")

classes = ["No Congestion", "Congestion"]
plt.xticks([0,1], classes)
plt.yticks([0,1], classes)

# كتابة القيم داخل المربعات
for i in range(cm_norm.shape[0]):
    for j in range(cm_norm.shape[1]):
        plt.text(j, i, f"{cm_norm[i, j]*100:.1f}%",
                 ha="center", va="center",
                 color="white" if cm_norm[i, j] > 0.5 else "black")

plt.tight_layout()

cm_norm_path = os.path.join(FIG_DIR, "cm.png")
plt.savefig(cm_norm_path, dpi=170)
plt.close()

print("The drawing was saved:", cm_norm_path)

```

```

# رسم يوم واحد كسلسلة زمنية

import os
import matplotlib.pyplot as plt
import pandas as pd

OUT_DIR = "/content/outputs"
FIG_DIR = os.path.join(OUT_DIR, "figures")
os.makedirs(FIG_DIR, exist_ok=True)

# قراءة يوم واحد فقط بالأعمدة المطلوبة
sample_day = pd.read_parquet(
    test_files[0],
    columns=["TIMESTAMP", "VOLUME"]
)

sample_day = sample_day.sort_values("TIMESTAMP")

plt.figure(figsize=(11,4))

plt.plot(
    sample_day["TIMESTAMP"],
    sample_day["VOLUME"],
    linewidth=1
)

plt.axhline(THRESH, color="red", linestyle="--", label="Congestion Threshold")
plt.title("Traffic Volume Over One Day")
plt.xlabel("Time")
plt.ylabel("Volume")
plt.xticks(rotation=45)
plt.grid(alpha=0.3)
plt.legend()
plt.tight_layout()

ts_path = os.path.join(FIG_DIR, "sample_day_timeseries.png")
plt.savefig(ts_path, dpi=170)
plt.close()

print("The drawing was saved:", ts_path)

```

```

# تحليل الأخطاء

```

```
import numpy as np

# تأكد أن المتغيرات موجودة
assert "pred" in globals(), "pred معرف"
assert "y_test" in globals(), "y_test معرف"

pred = np.asarray(pred)
y_test = np.asarray(y_test)

false_pos = np.sum((pred == 1) & (y_test == 0))
false_neg = np.sum((pred == 0) & (y_test == 1))

true_pos = np.sum((pred == 1) & (y_test == 1))
true_neg = np.sum((pred == 0) & (y_test == 0))

print("False Positives:", false_pos)
print("False Negatives:", false_neg)
print("True Positives:", true_pos)
print("True Negatives:", true_neg)
print("FP Rate:", round(false_pos / len(y_test), 4))
```