# เอกสารประกอบการบรรยาย

# Data Science with Python

บริษัท แลนด์ แอนด์ เฮ้าส์ จำกัด (มหาชน) | Land & Houses

โดย ผู้ช่วยศาสตราจารย์ ดร.วสิศ ลิ้มประเสริฐ

# บทที่ 4: การนำเสนอผลและการทำระบบอัตโนมัติ (Visualization & Automation)

**Main Story (บทที่ 4):** "ตอนนี้เรามี 'ตาราง Excel' สรุปผลแล้ว แต่ผู้บริหารต้องการ 'Dashboard' ที่ดูง่ายและเข้าใจได้ทันที ในบทนี้ เราจะแปลงตารางตัวเลข (จากบทที่ 3) ให้เป็นกราฟที่สวยงามด้วย Seaborn และสุดท้าย... เราจะ 'ห่อ' กระบวนการทั้งหมด (ตั้งแต่บทที่ 2) ให้กลายเป็น 'Pipeline อัตโนมัติ' ด้วย Prefect เพื่อให้รายงานนี้ทำงานเองทุกเดือน"

## 4.1 การนำเสนอผลด้วยภาพ (Data Visualization) (3.5 ชม.)

* **บรรยาย:** แนะนำ Matplotlib (โครง) และ Seaborn (สีสันและสถิติ)
* **Demo:**  
  import matplotlib.pyplot as plt  
  import seaborn as sns  
    
  # ตั้งค่า Theme ให้สวยงาม  
  sns.set\_theme(style="whitegrid", font="Tahoma") # (สมมติมีฟอนต์ Tahoma)
* **Main Story Link:** "สร้าง Dashboard สรุปสุขภาพพอร์ตสินเชื่อ"

### 4.1.1 กราฟเปรียบเทียบและสัดส่วน (Comparison & Proportion)

* **Workshop 4.1 (Bar Plot):**
  + (ข้อมูลจาก report\_by\_stage ในบทที่ 3)
  + plt.figure(figsize=(10, 6))
  + sns.barplot(data=report\_by\_stage.reset\_index(), x='Stage\_Name', y='sum')
  + plt.title('Total Principal by Stage')
  + plt.ylabel('Total Principal (Baht)')
  + plt.xlabel('Stage')
  + plt.show()
* **Workshop 4.2 (Pie Chart):**
  + (ข้อมูลจาก report\_by\_stage['count'])
  + plt.figure(figsize=(8, 8))
  + plt.pie(report\_by\_stage['count'], labels=report\_by\_stage.index, autopct='%1.1f%%', startangle=90)
  + plt.title('Account Count by Stage')
  + plt.show()

### 4.1.2 กราฟการกระจายตัว (Distribution)

* **Workshop 4.3 (Histogram):**
  + "ดูการกระจายตัวของ DPD เฉพาะกลุ่มที่ค้างชำระ (NPL)"
  + df\_npl = df\_merged[df\_merged['Stage\_Name'] == '3. NPL']
  + sns.histplot(data=df\_npl, x='FDPDUE01', bins=30, kde=True)
  + plt.title('DPD Distribution (NPL Accounts Only)')
  + plt.show()
* **Workshop 4.4 (Box Plot):**
  + "ดูการกระจายตัวของ 'ยอดหนี้' (FPRINCAM) เทียบตาม 'ประเภท Product' (FPRODTY)"
  + plt.figure(figsize=(12, 7))
  + sns.boxplot(data=df\_merged, x='FPRODTY', y='FPRINCAM')
  + plt.title('Principal Distribution by Product Type')
  + plt.ylim(0, 20000000) # (สมมติ) ตัด outlier เพื่อให้ดูกราฟง่ายขึ้น
  + plt.show()

### 4.1.3 กราฟความสัมพันธ์และตาราง (Relationship & Matrix)

* **Workshop 4.5 (Count Plot):**
  + "วิธิง่ายๆ ในการนับจำนวนบัญชีในแต่ละ Product"
  + sns.countplot(data=df\_merged, x='FPRODTY', order=df\_merged['FPRODTY'].value\_counts().index)
  + plt.title('Account Count by Product Type')
  + plt.show()
* **Workshop 4.6 (Scatter Plot):**
  + "ดูความสัมพันธ์ระหว่าง 'อายุสินเชื่อ' (Loan\_Age\_Days) กับ 'ยอดหนี้' (FPRINCAM)"
  + sns.scatterplot(data=df\_merged.sample(2000), x='Loan\_Age\_Days', y='FPRINCAM', hue='Stage\_Name', alpha=0.6)
  + plt.title('Loan Age vs. Principal')
  + plt.show()
* **Workshop 4.7 (Heatmap):**
  + **Main Story Link:** "แปลงตาราง Pivot (จาก Lab 3.8) ให้เป็นภาพ Heatmap"
  + plt.figure(figsize=(12, 8))
  + sns.heatmap(pivot\_report, annot=True, fmt=',.0f', cmap='Reds', linewidths=.5)
  + plt.title('Heatmap of Principal by DPD Bucket and Product')
  + plt.show()

### 4.1.4 การสร้าง Dashboard

* **Demo:** plt.subplots()
* **Workshop 4.8 (Dashboard 2x2):**
  + (รวม Workshop 4.2, 4.1, 4.4, 4.3 ไว้ในภาพเดียว)
  + fig, axes = plt.subplots(2, 2, figsize=(20, 16))
  + fig.suptitle('Portfolio Health Dashboard (July 2024)', fontsize=20)
  + **Top-Left (Pie):**
  + axes[0, 0].pie(report\_by\_stage['count'], labels=report\_by\_stage.index, autopct='%1.1f%%', startangle=90)
  + axes[0, 0].set\_title('Account Count by Stage')
  + **Top-Right (Bar):**
  + sns.barplot(ax=axes[0, 1], data=report\_by\_stage.reset\_index(), x='Stage\_Name', y='sum')
  + axes[0, 1].set\_title('Total Principal by Stage')
  + **Bottom-Left (Box):**
  + sns.boxplot(ax=axes[1, 0], data=df\_merged, x='FPRODTY', y='FPRINCAM')
  + axes[1, 0].set\_title('Principal Distribution by Product')
  + axes[1, 0].set\_ylim(0, 20000000)
  + **Bottom-Right (Histogram):**
  + sns.histplot(ax=axes[1, 1], data=df\_merged[df\_merged['FDPDUE01'] > 90], x='FDPDUE01', bins=30, kde=True)
  + axes[1, 1].set\_title('DPD Distribution (NPL Accounts Only)')
  + plt.tight\_layout(rect=[0, 0.03, 1, 0.95]) # ปรับ layout ให้พอดี
  + plt.savefig('monthly\_dashboard.png')
  + plt.show()

## 4.2 การทำระบบอัตโนมัติ (Automation) (2.5 ชม.)

### 4.2.1 การเปลี่ยน Notebook เป็น Script

* **Main Story Link:** "Notebook เหมาะสำหรับ 'ทดลอง' แต่ Script (.py) เหมาะสำหรับ 'ทำงานจริงอัตโนมัติ'"
* **Workshop 4.9 (Scripting with argparse):**
  + สร้างไฟล์ run\_report.py
  + (ย้ายโค้ดฟังก์ชัน load\_data, clean\_data, feature\_engineer, create\_report จากบทที่ 2-4 มาใส่)
  + เพิ่ม argparse เพื่อรับค่าวันที่จากภายนอก

import argparse  
  
def main(report\_date\_str):  
 # (โค้ดหลักในการเรียกใช้ฟังก์ชันต่างๆ)  
 # df\_tran, df\_perf = load\_data(report\_date\_str)  
 # df\_merged = clean\_data(df\_tran, df\_perf)  
 # ...  
 # create\_report(df\_final, report\_date\_str)  
 print(f"Report for {report\_date\_str} created successfully.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 parser = argparse.ArgumentParser(description='Run NPL Report Pipeline.')  
 parser.add\_argument('--date', required=True, help='Report date in YYYYMMDD format')  
 args = parser.parse\_args()  
  
 main(args.date)  
  
# รันใน Terminal: python run\_report.py --date 20240731

### 4.2.2 การเชื่อมต่อฐานข้อมูล (Database Connection)

* **Demo:** sqlalchemy
* **Demo (Read):** (ใช้ Northwind/Chinook)  
  from sqlalchemy import create\_engine  
  # (ตัวอย่าง PostgreSQL) engine = create\_engine('postgresql://user:pass@host:port/dbname')  
  engine = create\_engine('sqlite:///chinook.db')  
  df\_invoice = pd.read\_sql('SELECT \* FROM invoices LIMIT 10', engine)
* **Workshop 4.10 (Write to DB):**
  + (ปรับแก้ฟังก์ชัน create\_report ใน run\_report.py)
  + "บันทึกตารางสรุป pivot\_report ลงใน Database"

def create\_report(df\_final, report\_date\_str):  
 # (โค้ดสร้าง pivot\_report)  
  
 # บันทึกลง DB  
 db\_engine = create\_engine('sqlite:///report\_db.sqlite')  
 report\_table\_name = f"npl\_report\_{report\_date\_str}"  
 pivot\_report.to\_sql(report\_table\_name, con=db\_engine, if\_exists='replace')  
 print(f"Report saved to DB table: {report\_table\_name}")

### 4.2.3 การทำ Workflow อัตโนมัติ (Prefect)

* **บรรยาย:** ทำไมดีกว่า Task Scheduler/Cron? (Dependencies, Retries, Monitoring, UI)
* **Demo:** @task, @flow
* **Workshop 4.11 (Advanced Prefect Flow):**
  + (ในไฟล์ run\_report.py)

from prefect import task, flow, get\_run\_logger  
import pandas as pd  
from sqlalchemy import create\_engine  
import sys  
  
@task(retries=3, retry\_delay\_seconds=10)  
def load\_data(date\_str: str):  
 logger = get\_run\_logger()  
 logger.info(f"Loading data for {date\_str}...")  
 # (สมมติ) ดึงจาก DB  
 # engine = create\_engine("postgresql://...")  
 # df\_tran = pd.read\_sql(f"SELECT \* FROM tran WHERE frpdate='{date\_str}'", engine)  
 # ...  
 # (สำหรับ Test)  
 df\_tran = pd.read\_csv('Transection\_20240731.csv')  
 df\_perf = pd.read\_csv('Performance.csv')  
 if len(df\_tran) == 0:  
 raise ValueError("No transaction data found!")  
 return df\_tran, df\_perf  
  
@task  
def clean\_data(df\_tran, df\_perf):  
 logger = get\_run\_logger()  
 logger.info("Cleaning data...")  
 # (โค้ด Clean จากบทที่ 2)  
 df\_merged = pd.merge(df\_tran, df\_perf, left\_on='FCUSNO', right\_on='CIF', how='left')  
 df\_merged['FDPDUE01'].fillna(0, inplace=True)  
 return df\_merged  
  
@task  
def feature\_engineer(df\_merged):  
 logger = get\_run\_logger()  
 logger.info("Engineering features...")  
 # (โค้ดสร้าง DPD\_Bucket, Stage\_Name จากบทที่ 3)  
 bins = [-1, 0, 30, 60, 90, float('inf')]  
 labels = ['0. No DPD', '1. 1-30 Days', '2. 31-60 Days', '3. 61-90 Days', '4. 90+ Days']  
 df\_merged['DPD\_Bucket'] = pd.cut(df\_merged['FDPDUE01'], bins=bins, labels=labels)  
 return df\_merged  
  
@task  
def create\_report(df\_final, date\_str):  
 logger = get\_run\_logger()  
 logger.info("Creating and saving report...")  
 # (โค้ดสร้าง pivot\_report จากบทที่ 3)  
 pivot\_report = df\_final.pivot\_table(index='DPD\_Bucket', ...)  
  
 # (โค้ดสร้าง Dashboard จาก Workshop 4.8)  
 # ...  
 # plt.savefig(f'dashboard\_{date\_str}.png')  
 logger.info(f"Dashboard saved to dashboard\_{date\_str}.png")  
 return True  
  
@flow(log\_prints=True)  
def monthly\_npl\_pipeline(report\_date: str = "20240731"):  
 print(f"--- Starting NPL Pipeline for {report\_date} ---")  
 df\_tran, df\_perf = load\_data(date\_str=report\_date)  
 df\_merged = clean\_data(df\_tran, df\_perf)  
 df\_final = feature\_engineer(df\_merged)  
 create\_report(df\_final, report\_date)  
 print(f"--- Pipeline for {report\_date} finished successfully ---")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # รัน Flow โดยตรง: python run\_report.py  
 # (Prefect จะใช้ค่า default "20240731")  
 # หรือ: python run\_report.py --date 20240731  
 report\_date = sys.argv[1].split('=')[1] if len(sys.argv) > 1 else "20240731"  
 monthly\_npl\_pipeline(report\_date=report\_date)  
  
 # (สำหรับการ Deploy)  
 # monthly\_npl\_pipeline.serve(  
 # name="npl-report-deployment",   
 # cron="0 8 1 \* \*") # รันทุก 8 โมงเช้า วันที่ 1 ของเดือน

## 4.3 Mini-Game: ทบทวนความรู้ (Review Game)

**ภารกิจ:** จับคู่ "เครื่องมือ (Tool)" (ข้อ A-H) กับ "หน้าที่ (Job)" (ข้อ 1-8) ให้ถูกต้อง

เครื่องมือ (Tools):

A. sns.heatmap()

B. plt.subplots()

C. Prefect @flow

D. SQLAlchemy create\_engine()

E. Prefect @task(retries=3)

F. plt.savefig()

G. argparse

H. sns.boxplot()

**หน้าที่ (Jobs):**

1. ฟังก์ชันที่กำหนด "ขั้นตอนย่อย" ที่สามารถ "ลองใหม่" (Retry) ได้ 3 ครั้ง
2. ใช้สำหรับ "บันทึก" Dashboard ที่เราสร้างให้เป็นไฟล์รูปภาพ
3. ใช้สร้าง "Engine" เพื่อเชื่อมต่อ Python กับฐานข้อมูล
4. ใช้สร้างกราฟ "ตารางสี" ที่สรุปยอดหนี้ตาม DPD และ Product
5. ฟังก์ชันหลักที่ "รวบรวม" Task ต่างๆ มาเรียงต่อกันเป็น Pipeline
6. คำสั่งที่ใช้สร้าง "กรอบ" สำหรับวางกราฟหลายๆ อัน (Dashboard)
7. ใช้สร้างกราฟ "กล่อง" เพื่อดูการกระจายตัวของยอดหนี้
8. Library ที่ช่วยให้ Script (.py) สามารถ "รับค่า" จากภายนอกได้ (เช่น --date)

*(เฉลย: A=4, B=6, C=5, D=3, E=1, F=2, G=8, H=7)*

## 4.4 ถาม–ตอบ (Q&A)