การศึกษา การผสานอาปาเช่แอร์โฟลว์ และ คลังคุณลักษณะเพื่อส่งเสริมการจัดการข้อมูลสำหรับการเรียนรู้ของเครื่อง

A STUDY OF INTEGRATING APACHE AIRFLOW AND FEATURE STORES TO ENHANCE MACHINE LEARNING DATA MANAGEMENT

วงศกร วงศ์โดชงาม

บัณฑิตวิทยาลัย มหาวิทยาลัยศรีนครินทรวิโรฒ

2567

การศึกษา การผสานอาปาเช่แอร์โฟลว์ และ คลังคุณลักษณะเพื่อส่งเสริมการจัดการข้อมูลสำหรับการเรียนรู้ของเครื่อง

วงศกร วงศ์เดชงาม

สารนิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
วิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาการข้อมูล
คณะวิทยาศาสตร์ มหาวิทยาลัยศรีนครินทรวิโรฒ
ปีการศึกษา 2567

A STUDY OF INTEGRATING APACHE AIRFLOW AND FEATURE STORES TO ENHANCE MACHINE LEARNING DATA MANAGEMENT

VONGSAKORN WONGDECHNGAM

An Master's Project Submitted in Partial Fulfillment of the Requirements

for the Degree of MASTER OF SCIENCE

(Data Science)

Faculty of Science, Srinakharinwirot University

2024

สารนิพนธ์

เรื่อง

การศึกษา การผสานอาปาเช่แอร์โฟลว์ และ คลังคุณลักษณะเพื่อส่งเสริมการจัดการข้อมูลสำหรับการเรียนรู้ของเครื่อง

ของ

วงศกร วงศ์เดชงาม

ได้รับอนุมัติจากบัณฑิตวิทยาลัยให้นับเป็นส่วนหนึ่งของการศึกษาตามหลักสูตร

ปริญญาวิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาการข้อมูล

ของมหาวิทยาลัยศรีนครินทรวิโรฒ

(รองศาสตราจารย์ นายแพทย์ฉัตรชัย เอกปัญญาสกุล)

คณบดีบัณฑิตวิทยาลัย

คณะกรรมการสอบปากเปล่าสารนิพนธ์

..... ที่ปรึกษาหลัก

(ดร.เรืองศักดิ์ ตระกูลพุทธิรักษ์)

..... ประธาน

(ผู้ช่วยศาสตราจารย์จันตรี ผลประเสริฐ)

..... กรรมการ

(ผู้ช่วยศาสตราจารย์ศิริสรรพ เหล่าหะเกียรติ)

ชื่อเรื่อง	การศึกษา การผสานอาปาเช่แอร์โฟลว์ และ คลังคุณลักษณะเพื่อส่งเสริมการจัดการข้อมูลสำหรับการเรียนรู้ของเครื่อง
ผู้วิจัย	วงศกร วงศ์เดชงาม
ปริญญา	วิทยาศาสตรมหาบัณฑิต
ปีการศึกษา	2567
อาจารย์ที่ปรึกษา	ดร. เรืองศักดิ์ ตระกูลพุทธิรักษ์

การศึกษานี้มุ่งเน้นการประยุกต์ใช้เทคโนโลยี Apache Airflow และ Feature Store (FEAST) เพื่อเพิ่มประสิทธิภาพในการจัดการข้อมูลสำหรับระบบการเรียนรู้ของเครื่อง (Machine Learning) โดยเฉพาะในขั้นตอนการจัดเตรียมข้อมูล (Feature Engineering) และการนำคุณลักษณะ (features) กลับมาใช้งานใหม่ในระบบ Data Pipeline ภายใต้สภาพแวดล้อมคลาวด์ด้วย Google Cloud Platform งานวิจัยนี้ได้เปรียบเทียบการพัฒนา Data Pipeline สองรูปแบบ คือรูปแบบที่ใช้ Feature Store (FEAST) และแบบที่ไม่ใช้ Feature Store (Non-FEAST) โดยผลการศึกษาชี้ให้เห็นว่าการใช้ FEAST ช่วยลดความซับซ้อนของกระบวนการสร้างคุณลักษณะได้อย่างชัดเจน และเพิ่มประสิทธิภาพในการจัดเก็บและจัดการคุณลักษณะของข้อมูล ทำให้กระบวนการนำคุณลักษณะกลับมาใช้ซ้ำสะดวกและมีประสิทธิภาพมากขึ้น นอกจากนี้ ผลการทดลองยังแสดงให้เห็นว่าระบบที่ใช้ FEAST สามารถลดอัตราข้อผิดพลาดและเพิ่มประสิทธิภาพในการดำเนินงานโดยรวมได้ดีกว่าระบบที่ไม่มี FEAST อย่างมีนัยสำคัญ ซึ่งแสดงให้เห็นถึงศักยภาพของ FEAST ในการสนับสนุนการจัดการข้อมูลขนาดใหญ่สำหรับองค์กรที่ต้องการความคล่องตัวและประสิทธิภาพในการดำเนินงานระบบ Machine Learning และ Data Engineering

คำสำคัญ : Apache Airflow, FEAST, Feature Store, การจัดการข้อมูลสำหรับการเรียนรู้ของเครื่อง, Feature Engineering

Title	A STUDY OF INTEGRATING APACHE AIRFLOW AND FEATURE STORES TO ENHANCE MACHINE LEARNING DATA MANAGEMENT
Author	VONGSAKORN WONGDECHNGAM
Degree	MASTER OF SCIENCE
Academic Year	2024
Thesis Advisor	Dr. Ruangsak Trakunphutthirak

This research focuses on the application of Apache Airflow and Feature Store (FEAST) technologies to enhance data management efficiency for Machine Learning systems, specifically in the stages of feature engineering and feature reuse within a Data Pipeline deployed on a cloud environment using Google Cloud Platform. The research compares two Data Pipeline implementations: one utilizing FEAST and another without FEAST (Non-FEAST). Findings from the study indicate that FEAST significantly reduces the complexity involved in the feature engineering process and enhances the efficiency of feature storage and management. Additionally, it facilitates easier and more effective reuse of existing features. The experimental results also demonstrate that the FEAST-based system substantially decreases error rates and improves overall operational efficiency compared to the Non-FEAST approach, highlighting FEAST's potential for supporting large-scale data management in organizations aiming for agility and efficiency in Machine Learning and Data Engineering operations.

Keyword : Apache Airflow FEAST Feature Store Machine Learning Data Management Feature Engineering

กิตติกรรมประกาศ

สารนิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยความกรุณา ความอนุเคราะห์ และคำแนะนำอันทรงคุณค่าจาก ดร.เรืองศักดิ์ ตระกูลพุทธิรักษ์ อาจารย์ที่ปรึกษา ซึ่งได้เสียสละเวลาให้คำปรึกษาและชี้แนะอย่างใกล้ชิดตลอดระยะเวลาของการดำเนินงาน ผู้วิจัยขอแสดงความขอบพระคุณท่านอย่างสูงมา ณ โอกาสนี้

นอกจากนี้ ผู้วิจัยขอขอบพระคุณคณะกรรมการสอบปากเปล่าสารนิพนธ์ทุกท่าน ที่ได้กรุณาสละเวลาให้คำแนะนำและข้อเสนอแนะที่มีประโยชน์เป็นอย่างยิ่ง เพื่อให้สารนิพนธ์เล่มนี้มีความสมบูรณ์มากยิ่งขึ้น อีกทั้งผู้วิจัยขอขอบคุณคณาจารย์ทุกท่านจากสาขาวิชาวิทยาการข้อมูล คณะวิทยาศาสตร์ มหาวิทยาลัยศรีนครินทรวิโรฒ ที่ได้ถ่ายทอดวิชาความรู้ตลอดหลักสูตรการศึกษา ทำให้ผู้วิจัยสามารถนำไปใช้ในการศึกษา ค้นคว้า และการทำวิจัยได้อย่างมีประสิทธิภาพ

ผู้วิจัยขอขอบคุณ เพื่อน พี่น้อง และบุคคลรอบข้างทุกท่าน ที่ได้ให้กำลังใจและความช่วยเหลือในหลายๆ ด้าน ตลอดระยะเวลาการศึกษาและการดำเนินงานวิจัยที่ผ่านมา อีกทั้งขอขอบพระคุณครอบครัวที่ให้กำลังใจ เป็นแรงผลักดันสำคัญที่ทำให้ผู้วิจัยสามารถดำเนินงานได้สำเร็จตามเป้าหมายที่ตั้งไว้

ท้ายที่สุดนี้ ผู้วิจัยหวังเป็นอย่างยิ่งว่าสารนิพนธ์ฉบับนี้จะเป็นประโยชน์แก่ผู้ที่สนใจและสามารถนำไปใช้ต่อยอดในการศึกษาค้นคว้าหรือการปฏิบัติงานที่เกี่ยวข้องได้ในอนาคตต่อไป

วงศกร วงศ์เดชงาม

สารบัญ

หน้า

สารบัญตาราง

หน้า

สารบัญรูปภาพ

หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมา Background

ในโลกปัจจุบัน ข้อมูลคือหัวใจสำคัญในการผลักดันในธุรกิจต่างๆ ทั้งองค์กรขนาดใหญ่หรือธุรกิจขนาดย่อม ตั้งแต่ข้อมูลสำหรับการสนับสนุนเพื่อช่วยในเรื่องของความเชื่อมั่น ซึ่งไปจนถึงการยกระดับประสิทธิภาพเทคโนโลยีขับเคลื่อนด้วยข้อมูล เช่น โครงสร้างระบบปัญญาประดิษฐ์และการเรียนรู้เชิงกลไก การจัดการข้อมูลจากแหล่งที่มาหลายแหล่งกลายเป็นความท้าทายที่สำคัญสำหรับองค์กรขนาดใหญ่ เนื่องด้วยการบริหารข้อมูลไม่เพียงแค่ต้องการเก็บข้อมูลในปริมาณสูง แต่ยังต้องสามารถประมวลผลข้อมูลอย่างรวดเร็วและแม่นยำเพื่อตอบสนองความต้องการในเวลาจริง (real-time processing)

ข้อมูลในยุคนี้ไม่เพียงแต่มีปริมาณมหาศาล แต่ยังมีความแตกต่างในด้านลักษณะ ตั้งแต่ข้อมูลซึ่งเป็นโครงสร้าง ข้อมูลกึ่งโครงสร้าง จนถึงข้อมูลที่ไม่มีโครงสร้าง เช่น ข้อความ รูปภาพ และวิดีโอ การนำข้อมูลเหล่านี้มาใช้อย่างมีประสิทธิภาพควรอย่างยิ่งที่ต้องใช้เทคโนโลยีที่สามารถรวบรวม ประมวลผล และแปลงข้อมูลในลักษณะที่สอดรับกับการใช้งาน [1]

หนึ่งในเทคโนโลยีสิ่งนี้เป็นไปตามข้อกำหนดของ Apache Airflow ซึ่งเป็นระบบการจัดการเวิร์กโฟลว์ที่ช่วยในการออกแบบและจัดการรูปแบบการทำงานต่าง ๆ ให้อยู่ในรูปแบบกราฟไม่หมุนเวียน (DAGs: Directed Acyclic Graphs) ซึ่งให้ดำเนินการแปรผลข้อมูลจากหลายแห่งอย่างสม่ำเสมอ มุ่งเน้นที่ในกรณีที่ต้องจัดการข้อมูลขนาดใหญ่ในสภาพแวดล้อมคลาวด์ (cloud environment) นอกจากนี้ การใช้งานคลาวด์แพลตฟอร์ม เช่น Google Cloud Platform หรือ Amazon Web Services (AWS) ช่วยส่งเสริมศักยภาพในการปรับตัว scalability เกี่ยวกับอัตราแสดงผลเนื้อหาให้รองรับข้อมูลในจำนวนที่สูง ที่เอื้อประโยชน์และธุรกิจที่จำเป็นในการแปรผลแบบเรียลไทม์ ซึ่งยังสามารถสร้างการตอบรับได้อย่างรวดเร็วเพื่อปรับตัวตามการเปลี่ยนแปลงของข้อมูล ซึ่งใน Modern Pipeline มักจะแนะนำเทคโนโลยี Cloud-based ตัวอย่าง Google Cloud หรือ AWS เพื่อประมวลผลและเก็บรักษาข้อมูลจำนวนมาก เทคโนโลยีจำพวกนี้ช่วยในการยกระดับประสิทธิภาพทั้งในแง่ของศักยภาพในการประมวลผล (Compute Power) และความคล่องตัว (Flexibility) ซึ่งทำให้การประมวลผลข้อมูลในปริมาณมากสามารถทำได้อย่างราบรื่น [2]

ในกระบวนการพัฒนาและจัดการ Data Pipeline ที่มีความซับซ้อน การใช้งานเครื่องมือ Apache Airflow นั้นแสดงถึงหน้าที่สำคัญ เพื่อช่วยการทำกระบวนการตั้งค่า ลดความยุ่งยาก

ในการนำไปประยุกต์ใช้ด้วยการประเมินสถานะของเวิร์กโฟลว์ รวมถึงระบบยังมีการแจ้งเตือนเมื่อพบปัญหา

เกิดขึ้น ซึ่งระบบ Apache Airflow ยังสามารถพัฒนาให้ตรงกับความต้องการเฉพาะของแต่ละองค์กร เช่น การติดต่อกับแหล่งข้อมูลมากกว่าหนึ่งแหล่ง การจัดการประสิทธิภาพประมวลผลหลายขั้นตอน เพื่อสนับสนุนการทำงานในหลายแพลตฟอร์ม เช่น ฐานข้อมูล บริการเชื่อมต่อข้อมูลและจัดเก็บข้อมูลบน cloud หรือเชื่อมต่อกับ API ต่าง ๆ[3]

นอกจากนี้ การจัดการคุณลักษณะ (features) ในงานด้าน Machine Learning (ML) โดยเฉพาะในกระบวนการ Feature Engineering นั้นมักใช้เวลานานและซับซ้อน ซึ่งอาจส่งผลต่อประสิทธิภาพและความถูกต้องของ Pipeline ทั้งในขั้นตอนการฝึกโมเดล (training) และการนำไปใช้งาน (serving) การนำ FEAST (Feature Store) มาใช้ใน Data Pipeline ช่วยลดความซับซ้อนในการจัดการ features ด้วยการทำให้สามารถเก็บและนำ features กลับมาใช้ซ้ำได้อย่างเป็นระบบ ลดความเสี่ยงในการเกิดข้อผิดพลาด และเพิ่มความสอดคล้องระหว่างการฝึกและใช้งานโมเดล ด้วยศักยภาพของ Apache Airflow และ FEAST ในการลดความซับซ้อนและเพิ่มประสิทธิภาพของ Data Pipeline งานวิจัยนี้จึงมุ่งเน้นศึกษาแนวทางการพัฒนา Data Pipeline ในบริบทของคลาวด์ พร้อมการประยุกต์ใช้ FEAST Feature Store เพื่อสร้างกรอบการทำงานที่มีประสิทธิภาพในการจัดการข้อมูลขนาดใหญ่ ลดความซับซ้อนในขั้นตอน Feature Engineering และเพิ่มความสามารถในการนำคุณลักษณะกลับมาใช้ซ้ำได้ การศึกษาในครั้งนี้จึงมุ่งตอบโจทย์ความต้องการขององค์กรที่ต้องจัดการข้อมูลปริมาณมากในลักษณะงาน Machine Learning และ Data Engineering ผ่านการนำเสนอแนวทางที่ผสมผสานเทคโนโลยีทั้งสอง เพื่อสนับสนุนการพัฒนาระบบจัดการข้อมูลที่ตอบสนองความท้าทายในยุคดิจิทัล [4]

ในช่วงไม่กี่ปีที่ผ่านมา การจัดการข้อมูลในระบบการเรียนรู้ของเครื่อง (Machine Learning: ML) ได้รับการยอมรับว่าเป็นหนึ่งในความท้าทายที่สำคัญที่สุด ระบบ ML จำเป็นต้องอ่านข้อมูลจำนวนมากในระหว่างการฝึกอบรมโมเดล (Model Training) และต้องรองรับการทำนายผล (Inference) ที่หลากหลายทั้งในรูปแบบ Batch และ Online ML การเกิดขึ้นของ Feature Store ในฐานะแพลตฟอร์มข้อมูลเดี่ยวสำหรับจัดการข้อมูล ML ตลอดวงจรชีวิต (ML Lifecycle) เป็นการตอบสนองต่อความต้องการเหล่านี้ ตั้งแต่การแปลงข้อมูล (Feature Engineering) ไปจนถึงการฝึกโมเดลและการใช้งานโมเดล Feature Store ได้รับการพัฒนาในฐานะแพลตฟอร์มที่มีความสามารถสูงสำหรับจัดการข้อมูลคุณลักษณะ (Feature Data)

โดยรองรับการ query ในหลากหลายรูปแบบ เช่น columnar, row-oriented และ similarity search นอกจากนี้ยังช่วยแก้ปัญหาด้านการนำคุณลักษณะกลับมาใช้ซ้ำ การจัดการการแปลงข้อมูล และการรับประกันความถูกต้องของข้อมูลระหว่างการประมวลผลคุณลักษณะ การฝึกโมเดล และการทำนายผล [5]

1.2 วัตถุประสงค์ในการศึกษา

1.2.1 เพื่อศึกษาและออกแบบ Workflow และ DAGs (Directed Acyclic Graphs) ในกระบวนการพัฒนา Data Pipeline ที่เน้นการดึงข้อมูล แปลงข้อมูล และโหลดข้อมูลเข้าสู่ Feature Store โดยมีการกำหนด dataset เป็นตัวกระตุ้น (trigger) ให้เกิดการทำงานที่เป็นระบบและมีประสิทธิภาพ

1.2.2 เพื่อทดลอง FEAST feature store และประเมินประสิทธิภาพของกระบวนการนำคุณลักษณะ (features) ที่มีอยู่กลับมาใช้ใหม่ (reuse) ภายในโครงสร้างข้อมูลของระบบต้นแบบ Data Pipeline ที่พัฒนาอยู่บน Apache Airflow และ Google Cloud Platform

1.2.3 เพื่อทดลอง FEAST feature store ในการช่วยลดขั้นตอนการทำงานของ feature engineer

1.3 สมมติฐาน

1.3.1 การออกแบบ Workflow และ DAGs (Directed Acyclic Graphs) ที่มีการกำหนด dataset เป็นตัวกระตุ้น (trigger) จะช่วยเพิ่มประสิทธิภาพในการจัดการข้อมูล ลดความซับซ้อนของกระบวนการทำงาน และเพิ่มความแม่นยำในการประมวลผลข้อมูลของ Data Pipeline

1.3.2 การนำคุณลักษณะที่มีอยู่กลับมาใช้ใหม่ (feature reuse) ผ่าน FEAST feature store จะช่วยยกระดับคุณภาพของข้อมูลในระบบ Data Pipeline และลดอัตราความผิดพลาดในกระบวนการทำงาน

1.3.3 การใช้ FEAST feature store ในการพัฒนา Data Pipeline จะช่วยลดขั้นตอนและภาระงานของ feature engineer โดยการทำงานอัตโนมัติในกระบวนการจัดการข้อมูลและการเตรียมข้อมูลสำหรับการวิเคราะห์

1.4 ขอบเขต

1.4.1 การพัฒนาและทดสอบต้นแบบ Data Pipeline จะมุ่งเน้นไปที่การใช้งาน Apache Airflow ในการกำหนด workflow สำหรับงาน Big Data บนสภาพแวดล้อมคลาวด์ (Google Cloud Platform)

1.4.2 ระบบ Feature Store ที่เลือกใช้คือ FEAST ซึ่งจะถูกปรับตั้งค่าเพื่อรองรับการจัดเก็บ การอัปเดต และการดึงคุณลักษณะ (features) จากแหล่งข้อมูลขนาดใหญ่

1.4.3 เครื่องมืออื่น ๆ ที่เกี่ยวข้อง (เช่น BigQuery, Cloud Storage, หรือ Dataflow) จะถูกใช้เป็นส่วนหนึ่งของ Data Pipeline เพื่อรับ-ส่งข้อมูลและประมวลผลตาม workflow ที่ออกแบบ

1.4.4 การทดลองจะทำในลักษณะของกรณีศึกษา (Case Study) โดยมีการกำหนดตัวชี้วัด (Key Metrics) ที่เกี่ยวข้องกับการนำคุณลักษณะกลับมาใช้ใหม่ (feature reuse) ระยะเวลาในการพัฒนาคุณลักษณะ (feature engineering time) และประสิทธิภาพของ workflow (workflow performance)

1.5 ประโยชน์ที่จะได้รับ

1.5.1 ช่วยลดความซ้ำซ้อนในการพัฒนาคุณลักษณะ (feature engineering) และลดความผิดพลาดที่เกิดจากการคัดลอกหรือสร้างคุณลักษณะซ้ำ

1.5.2 เป็นกรอบแนวคิด (framework) ในการบริหารจัดการและพัฒนาบุคลากรด้าน Data Engineering ให้สามารถทำงานร่วมกับเครื่องมือ Open Source และระบบคลาวด์ได้อย่างมีประสิทธิภาพ

1.5.3 ได้แนวทางในการออกแบบและปรับใช้ Data Pipeline ที่ซับซ้อนบน Apache Airflow ซึ่งสามารถรองรับการขยายตัวของข้อมูลในระดับ Big Data

1.5.4 ได้องค์ความรู้และตัวอย่างการประยุกต์ใช้ FEAST feature store เพื่อเพิ่มประสิทธิภาพในการจัดเก็บ ดึง และนำคุณลักษณะกลับมาใช้ใหม่ในโครงการด้าน Data Science และ Machine Learning

สรุปเนื้อหาบทนี้ ผู้วิจัยได้นำเสนอแนวคิดและเทคโนโลยีที่เกี่ยวข้องกับการจัดการข้อมูลขนาดใหญ่และการพัฒนาระบบ Data Pipeline ด้วย Apache Airflow และ FEAST โดยชี้ให้เห็นถึงปัญหาและแนวทางแก้ไขที่ชัดเจน เพื่อเป็นแนวทางให้กับผู้อ่านเข้าใจความสำคัญของการศึกษาและการทดลองนี้ โดยในบทต่อไปจะกล่าวถึงทฤษฎีที่เกี่ยวข้องและแนวคิดพื้นฐานในการพัฒนา Data Pipeline และการใช้ Feature Store ในบริบทของงาน Machine Learning อย่างละเอียด

บทที่ 2
ความเป็นมาทางทฤษฎีและการทบทวนวรรณกรรม

2.1. งานวิจัยที่เกี่ยวข้องกับการทำ Data Pipeline with Apache Airflow

การศึกษาครั้งนี้ ผู้วิจัยได้ทำการสำรวจเอกสารและงานวิจัยที่เกี่ยวข้องกับการพัฒนา Data Pipeline และเทคโนโลยีที่ใช้ในการวิจัย โดยเฉพาะ Apache Airflow และ FEAST (Feature Store) เพื่อนำมาประยุกต์ใช้ในการออกแบบและพัฒนาระบบจัดการข้อมูลและฟีเจอร์ (Feature Management) สำหรับงานวิเคราะห์หรือโมเดล Machine Learning เนื้อหาในบทนี้แบ่งออกเป็น 2 ส่วนหลัก ดังนี้

งานวิจัยที่เกี่ยวข้องกับการทำ Data Pipeline ด้วย FEAST และ Apache Airflow ในส่วนนี้ จะกล่าวถึงงานวิจัยที่มุ่งเน้นการพัฒนา Data Pipeline ด้วย Apache Airflow รวมถึงเทคนิคและกรอบการทำงาน (framework) ที่ใช้ในการนำข้อมูลจากต้นทาง (source) ไปยังปลายทาง (sink) ทั้งในรูปแบบสตรีมมิง (streaming) และแบตช์ (batch) นอกจากนี้ ยังมีการกล่าวถึงการเชื่อมต่อ Apache Airflow เข้ากับระบบ Google Cloud รวมถึงการดึงข้อมูล(ingestion) จากแหล่งที่มาหลากหลาย ไม่ว่าจะเป็นข้อมูลที่มีโครงสร้าง (structured),กึ่งโครงสร้าง (semi-structured) หรือไม่มีโครงสร้าง (unstructured) ยิ่งไปกว่านั้น งานวิจัยบางส่วนได้ให้ความสำคัญกับการประเมินและวัดประสิทธิภาพของ Apache Airflow ในแต่ละขั้นตอน เช่น การจัดสรรทรัพยากร การเพิ่มหรือลดขนาดของโหนด และการปรับตั้งค่าการประมวลผล ซึ่งปัจจัยเหล่านี้ล้วนมีผลต่อการยกระดับประสิทธิภาพและลดข้อจำกัดในการประมวลผลข้อมูลขนาดใหญ่

ทฤษฎีและหลักการที่เกี่ยวข้องกับการทำงานในงานวิจัยนี้ ส่วนนี้จะอธิบายทฤษฎีและหลักการที่ใช้รองรับการดำเนินงานวิจัย โดยเฉพาะแนวคิดในการจัดการข้อมูล (Data Management) และการทำงานแบบ Pipeline โดยจะครอบคลุมถึงสถาปัตยกรรมของ Apache Airflow ในการจัดการเวิร์กโฟลว์ (workflow orchestration) ตลอดจนกระบวนการทำ Feature Engineering และความสำคัญของ Feature Store อย่าง FEAST ซึ่งเป็นเครื่องมือที่ช่วยบริหารจัดการฟีเจอร์สำหรับการทำงานของโมเดล Machine Learning ทั้งด้านการจัดเก็บ (storage) และการเรียกใช้งาน (serving) เพื่อให้การพัฒนาและปรับปรุงโมเดลเป็นไปอย่างมีประสิทธิภาพ

จากการรวบรวมข้อมูลและงานวิจัยข้างต้น ผู้วิจัยจะนำมาประยุกต์ใช้ออกแบบและพัฒนา Data Pipeline ซึ่งผสานการทำงานระหว่าง FEAST กับ Apache Airflow เพื่อสร้างกระบวนการส่งต่อข้อมูลที่มีประสิทธิภาพ ตอบสนองต่อปริมาณข้อมูลที่เพิ่มขึ้นอย่างรวดเร็ว และยังคงไว้ซึ่งการปรับตั้งค่าที่ยืดหยุ่นสำหรับการประมวลผลข้อมูลและบริหารจัดการฟีเจอร์ได้อย่างเหมาะสม

2.1.1 Developing Pragmatic Data Pipelines using Apache Airflow on Google Cloud Platform

งานวิจัยนี้ได้กล่าวถึงกระบวนการที่ประกอบด้วยชุดของการดำเนินการที่ย้ายข้อมูลจากแหล่งข้อมูลหนึ่งไปสู่ปลายทาง ความยุ่งยากของ Data Pipeline นั้นแปรผันตามลักษณะการใช้งาน ในรูปแบบดั้งเดิม Data Pipeline จะทำความสะอาดข้อมูล เก็บข้อมูล และย้ายข้อมูลจากจุดหนึ่งไปสู่อีกจุดหนึ่งฟังดูเหมือนง่าย แต่ในความจริงนั้นมันมีความยุ่งยากสูงมาก ด้วยเหตุที่องค์กรต้องบริหารข้อมูลอย่างเป็นระบบขนาดมหึมาและยุ่งยาก อีกทั้งมีการคาดหวังไว้ว่าขั้นตอนการทำงานควรมีความแข็งแรง ทำงานได้รวดเร็ว แจ้งสถานะ และสามารถทำงานซ้ำได้โดยไม่ล้มเหลว

Data Pipeline สมัยใหม่มีลักษณะที่แปลกแยกออกไปเล็กน้อย เนื่องจากต้องสามารถจัดการกับข้อมูลขนาดเพตะไบต์ เก็บข้อมูลในคลาวด์หลากหลายประเภท พร้อมทั้งให้บริการแปรผลข้อมูลแบบเรียลไทม์ได้ Apache Airflow เป็นหนึ่งในอุปกรณ์ที่สนับสนุนให้การสร้าง Data Pipeline เป็นเรื่องง่ายมากขึ้น โดยข้อกำหนดเบื้องต้นที่จำเป็นคือความรู้พื้นฐานด้าน

Python บทความนี้มุ่งเน้นไปที่การสร้าง Data Pipeline สำหรับข้อมูลการแลกเปลี่ยนหุ้นโดยใช้แนวคิด Airflow เช่น DAGs และ Operators[2]

2.1.2 Strategic Data Pipeline Design: Enhancing Operational Efficiency from Oracle to Single Store using Airflow S3 Data Pipelines

บทความนี้ศึกษาการพัฒนา ETL pipeline รูปแบบใหม่ที่ถูกจัดการโดย Apache Airflow ทำการผสานรวมฐานข้อมูล Oracle กับ SingleStore ผ่าน Amazon S3 โครงสร้างสถาปัตยกรรมนี้ช่วยส่งเสริมขีดจำกัดในการขยายตัว และความเชื่อมั่น ด้วยขั้นตอนการผสานข้อมูล ผ่านการแสดงงานซึ่งถูกจัดการตามลำดับ บทความนี้แสดงให้เห็นถึงการปรับปรุงในด้านการประมวลผลข้อมูลและการทำงานอัตโนมัติ ได้พิจารณาเทียบกับเทคนิค ETL ตามแนวทางดั้งเดิม บทความนี้มีความสำคัญ สอดคล้องตามความจำเป็นของกระบวนการ ETL ที่ปรับขนาดได้และมีผลลัพธ์ที่ดีในด้านดูแลข้อมูลระดับหน่วยงาน พร้อมแสดงถึงศักยภาพในการปรับปรุงเมื่อเปรียบเทียบกับวิธีการแบบดั้งเดิม[3]

2.1.3 Modelling Data Pipelines

ข้อมูลถือเป็นทรัพยากรใหม่ซึ่งมีค่าและแสดงถึงกุญแจสู่ความสำเร็จ อย่างไรก็ตามการรวบรวมข้อมูลคุณภาพสูงจากแหล่งข้อมูลที่กระจายอยู่หลายแห่งต้องใช้ความพยายามอย่างมาก นอกจากนี้ยังมีความท้าทายอื่น ๆ ในการเคลื่อนย้ายข้อมูลจากต้นทางไปยังปลายทาง Data pipeline ถูกใช้เพื่อเสริมสร้างประสิทธิภาพโดยรวมในเส้นทางการเคลื่อนที่ของข้อมูล

จากจุดเริ่มต้นถึงจุดหมายปลายทาง เนื่องจากเป็นระบบอัตโนมัติและลดการมีบทบาทในของมนุษย์ซึ่งปกติจะต้องมี

แม้จะมีการวิจัยเกี่ยวกับ ETL และ ELT อยู่บ้าง แต่งานวิจัยในส่วนนี้ยังคงมีจำกัด ETL/ELT pipelines แสดงถึงการแทนภาพเชิงนามธรรมของขั้นตอน data pipeline แบบสมบูรณ์ในทุกขั้นตอน เพื่อใช้คุณค่ามากที่สุดจาก data pipeline เราจำเป็นต้องเข้าใจถึงกิจกรรมภายในแต่ละขั้นตอนและวิธีการเชื่อมโยงกันใน data pipeline แบบครบวงจร

การศึกษานี้มอบภาพรวมออกแบบโมเดลเชิงแนวคิดของ data pipeline ที่สามารถใช้เป็นภาษาเพื่อการการติดต่อระหว่างทีมข้อมูลต่าง ๆ ยิ่งไปกว่านั้น โมเดลนี้สามารถใช้สำหรับการทำงานอัตโนมัติในการพิจารณาตรวจหาข้อผิดพลาด การหาทางออก และการแจ้งเตือนในแต่ละขั้นตอนของ data pipeline[1]

2.1.4 Data Ingestion Frameworks for Data Lakes: An Overview

ในยุคนี้ ข้อมูลถูกมองว่าเป็นทรัพยากรเชิงทุนของบริษัท เนื่องจากข้อมูลถือเป็นหลักการตัดสินใจของคณะกรรมการ หากข้อมูลไม่ถูกต้องหรือไม่ครบถ้วน อาจนำมาซึ่งความเสียหายในด้านระดับการศึกษาสูง การปรากฏของเทคโนโลยีใหม่และการเพิ่มขึ้นของอุปกรณ์และผู้ใช้งานที่เชื่อมต่อกัน ทำให้เกิดแหล่งข้อมูลใหม่ ๆ เช่น ซอฟต์แวร์ แพลตฟอร์ม และโซเชียลเน็ตเวิร์ก เมื่อเผชิญกับปริมาณข้อมูลที่เรียกว่า big data ระบบฐานข้อมูลเริ่มมีการแสดงถึงความไม่สเถียรในการควบคุมและแปรผลข้อมูลนี้ ซึ่งนำไปสู่การปรับปรุงเทคโนโลยีในการรวบรวมข้อมูลใหม่ที่เรียกว่า data warehouse ซึ่งช่วยให้สามารถจัดการข้อมูลที่มีโครงสร้างได้อย่างมีประสิทธิภาพมาหลายปี

ถึงกระนั้น ความสำคัญในการจัดการข้อมูลชนิด กึ่งโครงสร้างและข้อมูลที่ไม่มีโครงสร้าง ทำให้ต้องมีการใช้โซลูชันใหม่ ๆ เช่น data lakes เพื่อตอบสนองความท้าทายใหม่ ๆ ที่เกิดขึ้น ด้วยเหตุผลที่ data lake เป็นเทคโนโลยีสมัยใหม่ จึงยังคงเป็นแนวคิดที่ไม่ชัดเจนเนื่องจากการนำเสนอที่ยังไม่สมบูรณ์หรือซับซ้อน ด้วยเหตุนี้ เราจะนำเสนอการเปรียบเทียบระหว่าง data warehouse และ data lake รวมถึงอภิปรายเกี่ยวกับ big data และ data lakes ในฐานะเทคโนโลยีจัดการข้อมูลใหม่ในแง่การศึกษา ซึ่งนำเสนอปัจจัยสำคัญ โดยเฉพาะ data pipeline ที่ไม่ค่อยถูกกล่าวถึงในวรรณกรรม

ด้วยเหตุนี้ เราจึงได้ทำการศึกษาการเปรียบเทียบเพื่อประเมินโซลูชัน data pipeline ที่มีอยู่และเสนอทางเลือกที่มีคุณค่ามากขึ้น นอกจากนี้ เราจะนำเสนอโครงสร้างระบบนิเวศ data

lake ของมหาวิทยาลัย เพื่อให้เข้าใจแนวคิดของ data lake และองค์ประกอบสำคัญอย่างชัดเจน [6]

2.1.5 Building a Scalable and Robust Data Extraction Pipeline with ApacheAirflow and Cloud Platforms

งานวิจัยนี้นำเสนอความท้าทายในการดึงข้อมูล การแปลงข้อมูล และการโหลดข้อมูล (ETL) ปริมาณมหาศาลจากหลายแหล่ง บทคัดย่อนี้ได้เสนอแนวคิดแก้ไขปัญหาสำหรับการพัฒนา Data Extraction Pipeline ที่สามารถขยายตัวได้และมีความคล่องตัวสูง โดยใช้ Apache Airflow และแพลตฟอร์มคลาวด์ Apache Airflow ซึ่งเป็นแพลตฟอร์มจัดการเวิร์กโฟลว์แบบโอเพนซอร์ส ที่ทำหน้าที่เป็นแกนหลักในการจัดการ Data Pipeline ที่ซับซ้อน ขณะที่แพลตฟอร์มคลาวด์มีศักยภาพในการขยายตัวและมีความน่าเชื่อถือ ที่จำเป็นสำหรับการดำเนินงานแปรผลข้อมูลขนาดใหญ่ ซึ่งอธิบายถึงองค์ประกอบสำคัญและสถาปัตยกรรมของ Data Extraction Pipeline ที่เสนอ

เริ่มต้นด้วยภาพรวมของความสามารถในการจัดการเวิร์กโฟลว์ของ Apache Airflow โดยเน้นถึงความสามารถในการกำหนดเวลา ตรวจสอบ และจัดการเวิร์กโฟลว์ได้อย่างง่ายดาย การใช้ประโยชน์จากความยืดหยุ่นของ Airflow ผ่านตัวดำเนินการ (Operators) และฮุค (Hooks) ที่สามารถกำหนดเองได้ ทำให้ Pipeline นี้สามารถผสานเข้ากับแหล่งข้อมูลและปลายทางต่าง ๆ ได้อย่างไร้รอยต่อ เช่น ฐานข้อมูล, API และบริการจัดเก็บข้อมูลบนคลาวด์และนอกจากนี้ ยังเน้นถึงข้อดีของการใช้แพลตฟอร์มคลาวด์ เช่น (AWS), (GCP) หรือ MS Azure ในการโฮสต์ Data Extraction Pipeline โดยสภาพแวดล้อมบนคลาวด์เหล่านี้มีทรัพยากรการประมวลผลที่สามารถยืดหยุ่นได้ ทำให้ Pipeline สามารถปรับขนาดการประมวลผลได้แบบไดนามิกตามความผันผวนของปริมาณงาน นอกจากนี้ ยังสามารถใช้บริการแบบ Serverless เช่น AWS Lambda, Google Cloud Functions หรือ Azure Functions ในการแปรผล ไม่ต้องใช้เซิร์ฟเวอร์ ซึ่งเพิ่มความสามารถในการขยายตัวและประหยัดต้นทุนและยังกล่าวถึงกลยุทธ์ในการเพิ่มความทนทานและความน่าเชื่อถือให้กับ Data Extraction Pipeline เทคนิคต่าง ๆ เช่น การกำหนดตารางที่ทนทานต่อข้อบกพร่อง การจัดการข้อผิดพลาด และการประเมินคุณภาพ ป้องกันการสูญเสียข้อมูลและลดการหยุดนิ่ง นอกจากนี้ยังกล่าวถึงการใช้เฟรมเวิร์กการประมวลผลข้อมูลแบบกระจาย เช่น Apache Spark หรือ Apache Flink เพื่อเร่งขั้นตอนดึงและแปลงข้อมูล โดยเฉพาะอย่างยิ่งสำหรับชุดข้อมูลขนาดใหญ่ [7]

2.1.6 The Hopsworks Feature Store for Machine Learning

งานวิจัยนี้นำเสนอ Hopsworks Feature Store ซึ่งเป็นแพลตฟอร์มการจัดการข้อมูลที่ออกแบบมาเพื่อรองรับการพัฒนาระบบ Machine Learning (ML) อย่างมีประสิทธิภาพ โดยเน้นแก้ปัญหาที่เกี่ยวข้องกับการจัดการข้อมูลฟีเจอร์ในทุกขั้นตอนของวงจรชีวิต ML ตั้งแต่กระบวนการสร้างฟีเจอร์ (Feature Engineering) การฝึกโมเดล (Model Training) ไปจนถึงการคาดการณ์ผลลัพธ์ (Inference) งานวิจัยนี้มุ่งตอบโจทย์ความท้าทายที่เกิดขึ้นจากความซับซ้อนของข้อมูลฟีเจอร์ ซึ่งรวมถึงการนำฟีเจอร์กลับมาใช้ซ้ำ (Feature Reuse) การจัดการทรานส์ฟอร์เมชันข้อมูล (Data Transformation) และการบำรุงรักษาความสอดคล้องของข้อมูลระหว่างขั้นตอนต่าง ๆ ในระบบ ML Hopsworks ได้รับการพัฒนาโดยใช้สถาปัตยกรรม Feature, Training, and Inference Pipelines (FTI) ที่แบ่งงานออกเป็น 3 ส่วนหลัก ซึ่งแต่ละส่วนสามารถพัฒนาและดำเนินการได้อย่างอิสระ สถาปัตยกรรมนี้ช่วยลดความซับซ้อนในการจัดการข้อมูลและส่งเสริมการพัฒนาระบบ ML อย่างยั่งยืน โดยรองรับการทำงานทั้งแบบ Batch และ Streaming รวมถึงการจัดการฟีเจอร์ในแบบ Realtime ซึ่งช่วยให้ระบบ ML สามารถตอบสนองความต้องการที่หลากหลายขององค์กร

หนึ่งในความก้าวหน้าที่สำคัญของ Hopsworks คือการผสานรวมเทคโนโลยีที่ทันสมัย เช่น Arrow Flight, DuckDB, Apache Hudi และ RonDB เพื่อเพิ่มประสิทธิภาพในการจัดการข้อมูล โดย Hopsworks รองรับการประมวลผลข้อมูลในรูปแบบที่หลากหลาย เช่น การสืบค้นข้อมูลแบบ Columnar, Row-Oriented และ Vector-Based โดยเฉพาะอย่างยิ่งการรองรับการ Query ข้อมูลฟีเจอร์แบบ Point-in-Time Join ซึ่งช่วยลดปัญหาการรั่วไหลของข้อมูลในชุดการฝึกโมเดล

นอกจากนี้ งานวิจัยยังได้เน้นย้ำถึงความสามารถของ Hopsworks ในการสนับสนุนงานที่ต้องการปรับขนาดได้สูง (Scalability) ด้วยการผสานแพลตฟอร์มคลาวด์ เช่น AWS, GCP และ Azure เพื่อจัดการ Pipeline ข้อมูลฟีเจอร์ที่ซับซ้อนและสามารถประมวลผลข้อมูลได้อย่างมีประสิทธิภาพ การใช้งาน Serverless Services เช่น AWS Lambda และ Google Cloud Functions ยังช่วยเพิ่มความคล่องตัวและลดต้นทุนในการดำเนินงาน

นอกจากการจัดการข้อมูลฟีเจอร์ Hopsworks ยังมอบเครื่องมือสำหรับการตรวจสอบคุณภาพข้อมูลโดยใช้ Great Expectations และการสร้าง Metadata เพื่อกำกับดูแลข้อมูลอย่างเหมาะสม ฟังก์ชันการตรวจสอบคุณภาพข้อมูลเหล่านี้ช่วยป้องกันการเกิดข้อผิดพลาดในข้อมูลและส่งเสริมความน่าเชื่อถือของระบบ ML

ผลการทดลองเปรียบเทียบกับแพลตฟอร์มอื่น เช่น AWS Sagemaker และ Databricks แสดงให้เห็นว่า Hopsworks มีความเร็วและประสิทธิภาพที่เหนือกว่าในด้านการอ่านข้อมูล การจัดการ Query แบบ Point-in-Time Join และการลด Latency ในงาน Online Feature Store ทั้งนี้ งานวิจัยยังได้เปิดประเด็นสำหรับการพัฒนาฟีเจอร์ใหม่ ๆ เช่น การสนับสนุนระบบ ML แบบเรียลไทม์ และการรวมการทำงานระหว่าง Python และ Data Warehouses

ด้วยเหตุนี้ Hopsworks จึงเป็นแพลตฟอร์มที่มีศักยภาพสูงสำหรับการจัดการข้อมูลในระบบ ML โดยเฉพาะในองค์กรที่ต้องการระบบการจัดการข้อมูลที่ครบวงจรและมีประสิทธิภาพสูงในทุกขั้นตอนของวงจรชีวิต ML [5]

2.1.7 Managed Geo-Distributed Feature Store: Architecture and System Design

งานวิจัยนี้นำเสนอแนวทางการออกแบบและสถาปัตยกรรมของ ฟีเจอร์สโตร์ที่มีการจัดการแบบกระจายศูนย์ (Managed Geo-Distributed Feature Store) ซึ่งเป็นระบบที่พัฒนาขึ้นเพื่อสนับสนุนการจัดการข้อมูลฟีเจอร์ในระบบ Machine Learning (ML) โดยมุ่งเน้นไปที่การจัดการข้อมูลขนาดใหญ่ การรักษาความสอดคล้องของข้อมูลระหว่างขั้นตอนการฝึกและการพยากรณ์ และการป้องกันปัญหาการรั่วไหลของข้อมูล งานวิจัยนี้ตอบสนองต่อความท้าทายที่เกี่ยวข้องกับการจัดการข้อมูลฟีเจอร์ในกระบวนการ MLOps ทั้งในระดับองค์กรและระดับภูมิภาค พร้อมทั้งเสนอแนวทางการแก้ไขปัญหาที่เหมาะสมสำหรับองค์กรที่ต้องการปรับปรุงประสิทธิภาพและความน่าเชื่อถือของระบบ ML

ฟีเจอร์สโตร์ที่พัฒนาขึ้นมีจุดมุ่งหมายเพื่อสร้างระบบที่รวมศูนย์สำหรับการจัดเก็บ การบริหารจัดการ และการค้นคืนข้อมูลฟีเจอร์ โดยสนับสนุนการค้นหาและนำฟีเจอร์กลับมาใช้ซ้ำ (Feature Reuse) ลดความซ้ำซ้อนของการทำงานระหว่างทีมงาน และเพิ่มความแม่นยำของข้อมูลที่ใช้ในกระบวนการ ML ระบบนี้ช่วยให้สามารถจัดการข้อมูลฟีเจอร์ได้อย่างมีประสิทธิภาพในทุกขั้นตอน ตั้งแต่การสร้างฟีเจอร์ การจัดการเวอร์ชัน การตรวจสอบคุณภาพ ไปจนถึงการใช้งานในกระบวนการฝึกและการพยากรณ์ผล

หนึ่งในฟีเจอร์สำคัญที่ระบบนี้นำเสนอคือ การป้องกันข้อมูลรั่วไหล (Data Leakage Prevention) ซึ่งช่วยป้องกันไม่ให้โมเดล ML ใช้ข้อมูลในอนาคตระหว่างการฝึก ซึ่งอาจทำให้ผลลัพธ์เกิดการประเมินที่เกินจริงในสภาพแวดล้อมจริง นอกจากนี้ ระบบยังรองรับการดึงข้อมูลฟีเจอร์แบบ Point-in-Time เพื่อให้มั่นใจว่าฟีเจอร์ที่ใช้จะเป็นข้อมูลที่ถูกต้องตามเวลาที่กำหนด งานวิจัยยังได้นำเสนอ สถาปัตยกรรมแบบฮับและสโปก (Hub and Spoke Architecture) เพื่อสนับสนุนการแบ่งปันและการใช้งานฟีเจอร์ข้ามภูมิภาคหรือระบบงานที่แตกต่างกัน

โดยฟีเจอร์สโตร์ถูกกำหนดให้เป็นศูนย์กลาง (Hub) และระบบ ML ที่ใช้งานฟีเจอร์จะเป็นปลายทาง (Spoke) ช่วยลดการพัฒนาฟีเจอร์ซ้ำซ้อนในทีมงานต่าง ๆ และส่งเสริมการทำงานร่วมกันอย่างมีประสิทธิภาพ

ในด้านการปรับปรุงประสิทธิภาพ ฟีเจอร์สโตร์ที่พัฒนาขึ้นใช้เทคโนโลยีล่าสุด เช่น Apache Spark และทรัพยากรประมวลผลแบบเซิร์ฟเวอร์เลส (Serverless Compute) เพื่อจัดการข้อมูลขนาดใหญ่และลดต้นทุนในกระบวนการ นอกจากนี้ยังมีการเพิ่มประสิทธิภาพในกระบวนการ Query ผ่านการใช้ DSL เพื่อคำนวณฟีเจอร์แบบ Rolling Window Aggregation และเพิ่มประสิทธิภาพการประมวลผลโดยรวมของระบบ

ฟีเจอร์สโตร์ยังรองรับการจัดการข้อมูลในลักษณะ Geo-Distributed ซึ่งช่วยให้องค์กรสามารถใช้ข้อมูลฟีเจอร์ที่สร้างในภูมิภาคหนึ่งไปใช้งานในอีกภูมิภาคได้ โดยไม่กระทบต่อความเร็วในการประมวลผลหรือความน่าเชื่อถือของข้อมูล ระบบนี้ได้รับการออกแบบให้สามารถขยายขีดความสามารถได้สูง (Scalable) และสามารถรักษาความเสถียรของระบบ (High Availability) ภายใต้เงื่อนไข SLA ที่เข้มงวด

ผลการศึกษาแสดงให้เห็นว่าฟีเจอร์สโตร์ที่มีการจัดการนี้สามารถช่วยลดปัญหาการจัดการข้อมูลในระบบ ML ได้อย่างมีประสิทธิภาพ และเพิ่มความคล่องตัวในการดำเนินงาน MLOps ในองค์กร งานวิจัยนี้ยังเปิดประเด็นสำหรับการพัฒนาระบบในอนาคต เช่น การผสานการทำงานกับ LLMs และการรองรับระบบ ML แบบเรียลไทม์ เพื่อให้เหมาะสมกับความต้องการที่หลากหลายขององค์กรในยุคดิจิทัล [8]

2.1.8 Integrating AI and Data Engineering Building Robust Pipelines for Real-Time Data

การบูรณาการปัญญาประดิษฐ์ (AI) และวิศวกรรมข้อมูล (Data Engineering) นับเป็นปัจจัยสำคัญที่ช่วยเพิ่มขีดความสามารถของการวิเคราะห์ข้อมูลแบบเรียลไทม์ในปัจจุบัน งานวิจัยนี้มุ่งเน้นการสำรวจองค์ประกอบสำคัญที่เกี่ยวข้องกับการพัฒนา Data Pipeline เพื่อสนับสนุนการทำงานของระบบวิเคราะห์ข้อมูลที่ขับเคลื่อนด้วย AI โดยครอบคลุมถึงการออกแบบสถาปัตยกรรม การเลือกใช้เทคโนโลยี และการกำหนดระเบียบวิธีเชิงปฏิบัติที่เหมาะสมสำหรับการสร้าง Pipeline ที่สามารถขยายตัวได้ (Scalable) มีประสิทธิภาพ (Efficient) และมีความเสถียรสูง (Resilient)

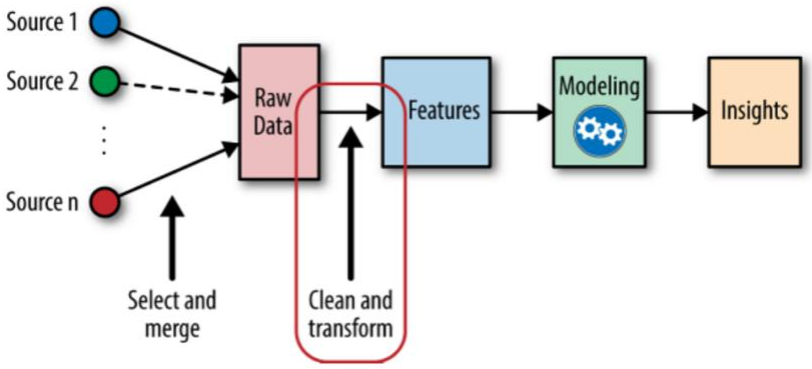
งานวิจัยนี้ให้ความสำคัญกับกระบวนการที่สำคัญในระบบ Pipeline ได้แก่ การดึงข้อมูล (Data Ingestion) การแปลงข้อมูล (Data Transformation) และการจัดเก็บข้อมูล (Data Storage) พร้อมทั้งนำเสนอการประยุกต์ใช้เทคนิค AI เช่น การเรียนรู้ของเครื่อง (Machine Learning)

และการเรียนรู้เชิงลึก (Deep Learning) เพื่อสนับสนุนการตัดสินใจแบบเรียลไทม์ นอกจากนี้ยังได้ศึกษาแนวทางปฏิบัติที่ดีที่สุดสำหรับการรักษาคุณภาพข้อมูล (Data Quality) การกำกับดูแลข้อมูล (Data Governance) และการจัดการการใช้งานโมเดล AI ภายในสภาพแวดล้อมแบบเรียลไทม์

งานวิจัยยังวิเคราะห์ประเด็นความท้าทายที่สำคัญ เช่น การจัดการความล่าช้า (Latency) การเพิ่มความสามารถในการขยายระบบ (Scalability) และการออกแบบการสื่อสารที่มีความหน่วงต่ำ (Low-Latency Communication) ระหว่างองค์ประกอบต่าง ๆ ของระบบ Pipeline พร้อมทั้งเสนอวิธีการแก้ไขปัญหาเชิงเทคนิคเพื่อเพิ่มประสิทธิภาพของระบบโดยรวม งานวิจัยนี้จึงเป็นรากฐานที่สำคัญสำหรับการพัฒนา Data Pipeline ที่ตอบสนองความต้องการในยุคที่ AI และข้อมูลแบบเรียลไทม์มีบทบาทสำคัญต่อการตัดสินใจในองค์กร [9]

2.1.9 Feature Engineering for Machine Learning

ก่อนที่จะเข้าสู่การสร้างฟีเจอร์ (Feature Engineering) ทำความเข้าใจภาพรวมของขั้นตอนการทำงานของระบบ Machine Learning เพื่อช่วยให้เห็นภาพว่าเนื้อหานี้มีส่วนเกี่ยวข้องอย่างไรในกระบวนการโดยรวม เราจะเริ่มต้นด้วยแนวคิดพื้นฐานเกี่ยวกับ ข้อมูล และ โมเดล ส่วนฟีเจอร์ คือ การแปลงข้อมูลดิบให้อยู่ในรูปตัวเลข เพื่อให้โมเดลนำไปใช้ได้ วิธีแปลงข้อมูลมีได้หลายแบบ จึงทำให้ฟีเจอร์มีหลากหลาย ซึ่งต้องมาจากข้อมูลที่มีอยู่ และควรสอดคล้องกับโมเดลด้วย[10]



ภาพประกอบ 1 The place of feature engineering in the machine learning workflow ที่มา [10]

2.2. ทฤษฎีเกี่ยวกับหลักการทำงานที่ใช้ในงานวิจัยนี้

2.2.1 Airflow Directed Acyclic Graph (DAG)

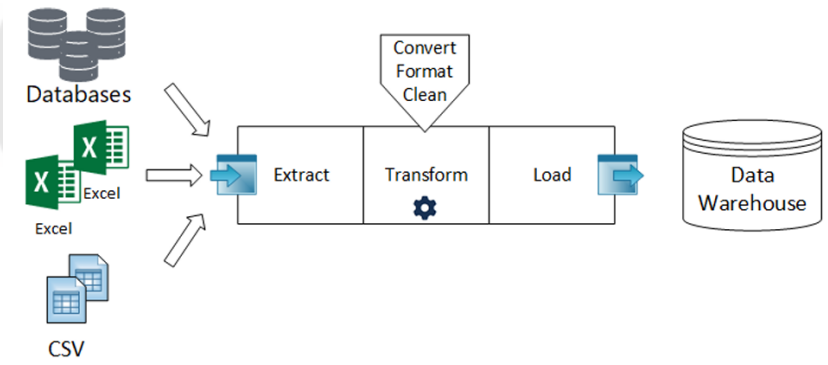
(DAG) เป็นรากฐานที่สำคัญ เพื่อจัดการเวิร์กโฟลว์ที่ซับซ้อนในระบบต่าง ๆ ใช้โครงสร้างกราฟที่ไม่มีวงจร เพื่อให้งานสามารถจัดเรียงตามลำดับและถูกประมวลผลตามเงื่อนไขที่กำหนด การทำงานของ Airflow DAG ช่วยในการจัดการงานต่าง ๆ อย่างมีประสิทธิภาพ โดยทำให้แน่ใจว่าขั้นตอนแต่ละขั้นตอนจะถูกดำเนินการในลำดับที่ถูกต้อง การใช้ Airflow ยังช่วยยกระดับประสิทธิภาพทำอัตโนมัติและลดการใช้แรงงานมนุษย์ในกระบวนการที่มีความซับซ้อนซ้ำซ้อน

หลักการพื้นฐานของ Directed Acyclic Graph (DAG) ใน Airflow เช่นเดียวกับการใช้เส้นทางที่ไม่มีการย้อนกลับ ทำให้เชื่อมั่นได้ว่าดำเนินงานทุกอย่างจะไปข้างหน้า DAG จะทำงานโดยจัดเรียงและประเมินลำดับของงานหรือ "tasks" แต่ละตัว และการประมวลผลแบบนี้เหมาะกับการดำเนินการในระบบที่ต้องการการจัดการงานขนาดใหญ่หรือหลายขั้นตอน

การสร้าง DAG จะช่วยให้ผู้พัฒนาสามารถกำหนดเส้นทางการทำงานที่ชัดเจนและสามารถตรวจสอบความสัมพันธ์ระหว่างงานต่าง ๆ ที่ดำเนินการ การใช้งาน Airflow ยังมีประโยชน์ในด้านการประเมินสถานการณ์ดำเนินงานและระบุจุดเกิดความล้มเหลวได้อย่างรวดเร็ว เพื่อให้สามารถจัดการปัญหาได้ทันเวลา [11]

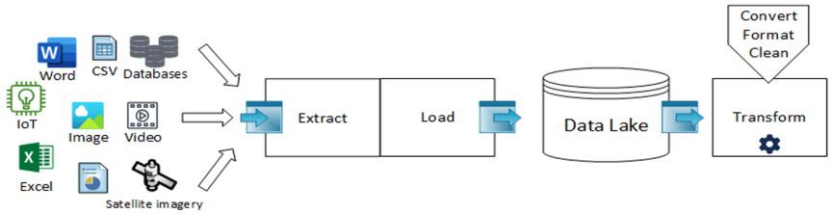
2.2.2 Data Ingestion

การจัดการข้อมูลขนาดใหญ่ (Big Data) คือข้อที่จำเป็นต่อยุคปัจจุบัน การพัฒนาระบบที่สามารถรองรับข้อมูลจากหลายแหล่งข้อมูลและจัดเก็บอย่างมีประสิทธิภาพในแนวทางของ Data Lake มีบทบาทสำคัญต่าง ๆ การทำความเข้าใจทฤษฎีและหลักการปฏิบัติของ Data Ingestion สำหรับ Data Lake จะช่วยให้นักวิจัยและผู้พัฒนาระบบสามารถออกแบบขั้นตอนที่มีคุณภาพสูงในการควบคุมข้อมูลที่ซับซ้อนและหลากหลายรูปแบบ งานวิจัยนี้จะอธิบายถึงแนวคิดที่ใช้ในขั้นตอน Data Ingestion ซึ่งรวมถึงการเปรียบเทียบกระบวนการ ETL และ ELT การจัดการ Data Pipeline และหัวใจสำคัญการจัดการ Metadata ขั้นตอน ETL และ ELT เป็นหลักการพื้นฐานในการดึงข้อมูลและจัดเก็บในระบบข้อมูลขนาดใหญ่ โดยทั้งสองกระบวนการมีจุดเด่นที่แตกต่างกัน



ภาพประกอบ 2 ETL Process for data warehouse ที่มา [6]

ETL คือขั้นตอนดำเนินงานแบบดั้งเดิมที่นิยมใช้ใน Data Warehouse โดยข้อมูลจะถูกดึงออกจากแหล่งข้อมูล (Extract) จากนั้นจะถูกแปลงโครงสร้างและปรับรูปแบบให้ตรงกับโครงสร้างที่ต้องการ (Transform) ก่อนที่จะถูกโหลดเข้าสู่ระบบ Data Warehouse (Load) ข้อได้เปรียบของกระบวนการนี้คือการที่ข้อมูลถูกจัดเก็บในรูปแบบที่พร้อมใช้งานและมีคุณภาพสูง แต่ข้อเสียคือใช้เวลานานและซับซ้อนในการเตรียมข้อมูล



ภาพประกอบ *3* ELT process for Data Lake ที่มา [6]

ELT (Extract, Load, Transform) ใช้ใน Data Lake โดยข้อมูลจะถูกดึงและโหลดเข้าสู่ Data Lake ในรูปแบบดิบ ๆ โดยไม่ต้องผ่านการแปลงก่อน เมื่อใช้งานข้อมูล การแปลงจะเกิดขึ้นในขั้นตอนภายหลังที่โหลดเสร็จ การดำเนินการนี้ช่วยเพิ่มความยืดหยุ่นและความเร็วในการจัดเก็บข้อมูล อีกทั้งยังรองรับข้อมูลทุกประเภท[6]

2.2.3 Operators and Tasks

ใน Apache Airflow งานหรือ Tasks ภายใน Pipeline จะถูกสร้างและจัดการผ่านสิ่งที่เรียกว่า Operators ซึ่งเป็นตัวแทนของแต่ละหน่วยการทำงาน Operators เป็นโมดูลที่มีฟังก์ชันเฉพาะในการประมวลผลงาน เช่น การดึงแหล่งข้อมูลต่าง ๆ ฐานข้อมูล, API หรือบริการการเก็บข้อมูลบนระบบคลาวด์ ขึ้นกับความจำเป็นของระบบงานนั้นๆ ทำให้ Pipeline มีความเสถียร

PythonOperator เป็น Operator ที่ใช้เรียกใช้งานฟังก์ชัน Python ผู้ใช้สร้างฟังก์ชันที่กำหนดเองได้ เช่น การดึงข้อมูล การดำเนินการข้อมูล หรือการบันทึกข้อมูล

BashOperator เป็น Operator ที่ใช้สำหรับเรียกคำสั่ง Bash เหมาะสำหรับการเรียกใช้งานสคริปต์หรือคำสั่งระบบในระบบ Linux

SQLOperator ใช้ในการรันคำสั่ง SQL เพื่อดึงข้อมูลจากฐานข้อมูล เช่น MySQL, PostgreSQL หรือ SQL Server

GCSToGCSOperator เป็น Operator ที่ใช้การจัดกับข้อมูลใน Google Cloud Storage (GCS) โดยตรง ผู้ใช้สามารถใช้ GCSToGCSOperator เพื่อทำงานกับไฟล์หรือข้อมูลที่เก็บอยู่ใน GCS ได้ เช่น การอัปโหลดไฟล์ไปยัง GCS การดึงข้อมูลจาก GCS หรือการลบไฟล์จาก GCS ทำให้การเชื่อมต่อและประมวลผลข้อมูลที่อยู่บน Google Cloud มีประสิทธิภาพ ทั้งนี้ยังมีข้อบ่งชี้การทำงานเฉพาะที่เกี่ยวกับ GCS ได้ตามการของผู้ใช้

นอกเหนือจาก Operators ที่มีให้ใช้ในระบบ Airflow ผู้ใช้ยังสามารถสร้าง Operators ที่กำหนดเองเพื่อสนับสนุนในการใช้งานเฉพาะในระบบงานของตนเองได้ ตัวอย่างเช่น การสร้าง API Operator ที่ออกแบบมาเพื่อเชื่อมต่อกับ API เฉพาะเพื่อดึงข้อมูล และ Custom Operator ที่ใช้กับการประมวลผลข้อมูลที่ซับซ้อนหรือโครงสร้างข้อมูลพิเศษ นอกจากนี้ Airflow ยังสนับสนุนการทำงานร่วมกับแหล่งข้อมูลหลากหลายรูปแบบ เช่น การเชื่อมต่อไปยัง S3, Google Cloud Storage หรือ Azure Blob Storage ซึ่งช่วยให้สามารถดึงข้อมูลจากแหล่งต่าง ๆ ได้อย่างคล่องตัว

2.2.4 Scheduling and Execution

สิ่งที่สำคัญของ Apache Airflow คือความสามารถในการกำหนดตารางเวลาและควบคุมการทำงานของงานต่าง ๆ ใน Pipeline โดยผู้ใช้สามารถกำหนดเวลาได้หลากหลายรูปแบบ เช่น การกำหนดให้ Pipeline ทำงานตามเวลาที่กำหนดไว้ล่วงหน้า เช่น ทุกวัน ทุกชั่วโมง หรือทุกสัปดาห์ นอกจากนี้ Airflow ยังตั้งค่าดำเนินการให้ทำงานตามเหตุการณ์ที่เกิดขึ้นได้ เช่น กระตุ้นงานเมื่อมีไฟล์ใหม่ถูกอัปโหลดมายังระบบ หรือเมื่อมีการปรับเปลี่ยนในแหล่งข้อมูลบางอย่าง Scheduler ใน Airflow ทำหน้าที่เป็นตัวจัดการลำดับของงานตามที่กำหนดไว้ใน DAG (Directed Acyclic Graph) ซึ่งจะตรวจสอบว่าเงื่อนไขที่กำหนดไว้สำหรับการประมวลผลงานแต่ละงานได้รับการตอบสนองแล้วหรือไม่ เมื่อเงื่อนไขถูกตอบสนอง Scheduler จะเรียกใช้งานเหล่านั้น ตามลำดับที่ระบุไว้ อีกทั้ง Airflow ยังดูแลการพึ่งพาของงานให้ถูกต้อง ซึ่งช่วยลดปัญหาการทำงานที่ซ้ำซ้อนหรือการทำงานที่ไม่เป็นไปตามลำดับ อาจส่งผลกระทบต่อความถูกต้องของกระบวนการได้

2.2.5 Designing for Robustness

ใน Data Pipeline ที่ซับซ้อนและมีการเชื่อมต่อกับแหล่งข้อมูลหลายประเภท การจัดการข้อผิดพลาดเป็นเรื่องสำคัญมากเพื่อให้แน่ใจว่าระบบทำงานได้อย่างต่อเนื่อง Apache Airflow มีกลไกในการจัดการข้อผิดพลาดที่มีประสิทธิภาพ เช่น การตั้งค่าการเรียกซ้ำหากงานบางงานล้มเหลว การตั้งค่าให้มีการหยุดหรือแจ้งเตือนเมื่อเกิดข้อผิดพลาดที่ร้ายแรง เป็นต้น นอกจากนี้ การตรวจสอบคุณภาพข้อมูลเป็นอีกหนึ่งหัวใจหลักของการจัดการ Pipeline ที่เชื่อถือได้ ในงานวิจัยนี้ ได้กล่าวถึงการใช้เทคนิคการประเมินข้อมูลเพื่อให้แน่ใจว่าข้อมูลที่ถูกดึงเข้ามานั้นมีความถูกต้องตามข้อกำหนด เช่น การตรวจสอบว่าข้อมูลไม่ขาดหายหรือมีความคลาดเคลื่อน การประเมินคุณภาพของข้อมูลก่อนจะโหลดเข้าสู่ระบบช่วยป้องกันความท้าทายที่อาจพบในขั้นตอนประมวลผลขั้นตอนต่อไป โดยรวมแล้ว ข้อผิดพลาดและการประเมินข้อมูลใน Airflow

ช่วยเพิ่มความเชื่อมั่นของระบบ Data Pipeline ส่งผลให้ระบบทำงานได้อย่างต่อเนื่อง แม้จะมีความผิดพลาดในบางขั้นตอนก็ตาม[7]

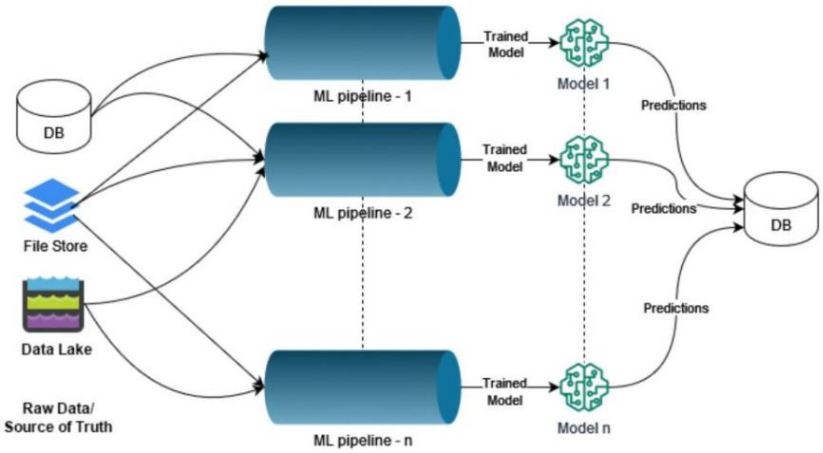
2.2.6 FEAST Feature Store

ในงานด้าน Machine Learning (ML) การจัดการและบริหารข้อมูลคุณลักษณะ (Features) ถือเป็นกระบวนการสำคัญที่ส่งผลโดยตรงต่อประสิทธิภาพและความถูกต้องของโมเดล ML การสร้างระบบที่สามารถจัดเก็บและเรียกใช้คุณลักษณะได้อย่างมีประสิทธิภาพ เป็นสิ่งจำเป็นในกระบวนการพัฒนาและปรับใช้ ML Pipelines โดยเฉพาะในระบบที่ต้องรองรับข้อมูลปริมาณมากและการใช้งานแบบเรียลไทม์ (Real-Time Applications)

FEAST (Feature Store for Machine Learning) เป็นแพลตฟอร์มที่ถูกพัฒนาขึ้นเพื่อแก้ไขปัญหาเหล่านี้ โดยทำหน้าที่เป็นระบบจัดการคุณลักษณะ (Feature Management System) ที่รองรับทั้งการเก็บข้อมูล การแปลงข้อมูล และการให้บริการข้อมูลคุณลักษณะ (Feature Serving) FEAST ช่วยลดความซับซ้อนในการจัดการข้อมูลและเพิ่มความสามารถในการใช้งานซ้ำของคุณลักษณะใน ML Pipelines [4]

2.2.7 Batch Model Pipeline

Batch Model คือรูปแบบหนึ่งของการพัฒนาระบบการทำนาย (prediction systems) ในงานวิจัยที่เน้นการคำนวณข้อมูลในรูปแบบที่มีการประมวลผลตามรอบเวลา (Scheduled Processing) เช่น รายชั่วโมง รายวัน หรือรายสัปดาห์ ขึ้นอยู่กับความต้องการของงานวิจัยและทรัพยากรที่ใช้ โมเดลประเภทนี้เหมาะสำหรับงานที่ไม่ได้ต้องการการทำนายแบบเรียลไทม์ (Real-Time Prediction) ซึ่งสามารถจัดการกับข้อมูลขนาดใหญ่ได้โดยใช้โครงสร้างพื้นฐานที่เรียบง่าย เช่น เฟรมเวิร์กสำหรับการประมวลผลข้อมูลแบบกระจาย (Distributed Data Processing Frameworks) เช่น Apache Spark, Hadoop หรือ Python



ภาพประกอบ 4 Batch model pipeline ที่มา [12]

การประมวลผลใน batch model pipeline มีความล่าช้าในการทำงาน (latency) ตั้งแต่นาทีจนถึงชั่วโมง และทำงานตามตารางเวลา โดยมักใช้งานร่วมกับเครื่องมือจัดการงาน เช่น Apache Airflow

กระบวนการ Batch Model Pipeline อ่านข้อมูลดิบจากหลายแหล่ง เช่น Database, Data Lake หรือ File Store ทำการทำความสะอาดข้อมูล (data cleaning) และประมวลผลคุณลักษณะ (feature engineering) ส่งข้อมูลผ่านขั้นตอนต่าง ๆ ใน pipeline เพื่อฝึกอบรมและใช้งานโมเดล ผลลัพธ์จากการทำนายจะถูกบันทึกในพื้นที่จัดเก็บแบบถาวร เช่น Database หรือ S3 Bucket เพื่อการเข้าถึงในอนาคต หาก pipeline ล้มเหลว (เช่น เข้าถึงข้อมูลไม่ได้ หรือโค้ดมีปัญหา) ระบบจะส่งสัญญาณเตือนและหยุดการทำงาน [12]

2.2.8 Data Correction

การปรับปรุงคุณภาพข้อมูลเป็นขั้นตอนสำคัญก่อนการนำข้อมูลไปวิเคราะห์ ซึ่งปัญหาที่พบบ่อยได้แก่ ข้อมูลที่ไม่สะอาด ข้อมูลซ้ำซ้อน และความยากลำบากในการรวมข้อมูลจากแหล่งที่มาต่างกัน ข้อมูลที่ไม่สะอาดมักเกิดจากค่าที่ขาดหาย ค่าที่ผิดพลาด หรือรูปแบบข้อมูลที่ไม่เป็นไปตามมาตรฐาน ส่วนข้อมูลซ้ำซ้อนเกิดขึ้นเมื่อมีการบันทึกข้อมูลของสิ่งเดียวกันในหลายที่ โดยอาจมีรูปแบบที่แตกต่างกัน เช่น การสะกดชื่อที่หลากหลาย การรวมข้อมูล

จากแหล่งต่างๆ ก็เป็นอีกหนึ่งความท้าทาย เนื่องจากขาดตัวระบุที่เป็นมาตรฐาน ทำให้การจับคู่ข้อมูลไม่สามารถทำได้อย่างแม่นยำ

ข้อผิดพลาดของข้อมูลสามารถเกิดขึ้นได้ในทุกขั้นตอน ตั้งแต่การป้อนข้อมูล การวัดผล การประมวลผล ไปจนถึงการรวมข้อมูล การแก้ไขปัญหาข้อมูลที่ผิดหรือไม่สอดคล้องกันมักต้องอาศัยความรู้เฉพาะด้าน ซึ่งทำให้กระบวนการอัตโนมัติเป็นไปได้ยาก เว้นแต่จะมีเครื่องมือที่ออกแบบมาเฉพาะ การพัฒนาเครื่องมือที่ช่วยให้ผู้ใช้สามารถมีส่วนร่วมในกระบวนการทำความสะอาดข้อมูล เช่น การแสดงตัวอย่างการเปลี่ยนแปลงที่เกิดขึ้น และการเรียนรู้จากตัวอย่าง สามารถช่วยลดความซับซ้อนของกระบวนการนี้ได้

การจัดการข้อมูลซ้ำซ้อนข้อมูลซ้ำซ้อนมักเป็นปัญหาในกรณีที่ต้องรวมชุดข้อมูลจากหลายแหล่ง หรือเมื่อระบบไม่มีฟังก์ชันสำหรับการอัปเดตบันทึกข้อมูล ตัวอย่างเช่น ระบบที่บันทึกข้อมูลเป็นฟอร์มแยกแต่ละรายการ อาจทำให้มีการสร้างข้อมูลซ้ำโดยไม่ตั้งใจ วิธีแก้ปัญหานี้คือการพัฒนาเครื่องมือที่สามารถค้นหาความคล้ายคลึงกันของข้อมูล เพื่อระบุข้อมูลที่ซ้ำซ้อนในรูปแบบต่างๆ อย่างไรก็ตาม การจัดการข้อมูลซ้ำซ้อนยังคงเป็นความท้าทาย เนื่องจากข้อมูลเดียวกันอาจถูกแทนด้วยรูปแบบที่หลากหลาย

ความท้าทายในการรวมข้อมูลการรวมข้อมูลจากแหล่งต่างๆ มักประสบปัญหาการขาดตัวระบุที่เป็นมาตรฐาน ซึ่งทำให้การจับคู่ข้อมูลจากหลายแหล่งทำได้ยาก วิธีที่นิยมใช้คือการเลือกแอตทริบิวต์ร่วมที่สามารถเชื่อมโยงข้อมูลได้ เช่น ชื่อสถานที่หรือบุคคล อย่างไรก็ตาม การสะกดผิดและการแปลงชื่อท้องถิ่นเป็นตัวอักษรอื่นอาจก่อให้เกิดความคลาดเคลื่อนอีกทางหนึ่ง อีกวิธีที่ช่วยเพิ่มความแม่นยำคือการใช้การเชื่อมโยงระเบียนแบบความน่าจะเป็น ซึ่งจะพิจารณาตัวระบุที่เป็นไปได้หลายรูปแบบ และคำนวณความน่าจะเป็นของการจับคู่ข้อมูลเดียวกัน

การพัฒนากระบวนการและเครื่องมือที่ช่วยแก้ไขข้อมูลเหล่านี้จะช่วยให้ผู้ใช้งานข้อมูลทั่วไปสามารถปรับปรุงคุณภาพข้อมูลได้ง่ายขึ้น และเพิ่มประสิทธิภาพของการวิเคราะห์ข้อมูลในระยะยาว [13]

2.2.9 Data Collection and Structuring

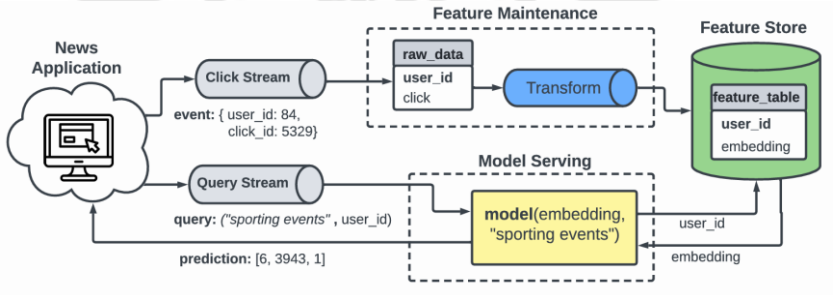
ในปัจจุบัน มีการพัฒนาเครื่องมือหลายชนิดเพื่อลดข้อผิดพลาดในกระบวนการเก็บรวบรวมข้อมูล ตัวอย่างเช่น เครื่องมือที่สามารถแปลงภาพของแบบฟอร์มกระดาษให้เป็นข้อมูลดิจิทัล โดยใช้กลุ่มคนช่วยกันตรวจสอบและบันทึกข้อมูลเพื่อลดข้อผิดพลาดในกระบวนการถอดความ อีกตัวอย่างหนึ่งคือเครื่องมือสำหรับการเก็บข้อมูลในอุปกรณ์มือถือ

ซึ่งมีการออกแบบให้ผู้ใช้งานสามารถเลือกข้อมูลที่เคยกรอกมาก่อนหน้า แทนการกรอกข้อมูลใหม่ทุกครั้ง วิธีนี้ช่วยลดข้อผิดพลาดจากการสะกดคำที่อาจเกิดขึ้นได้

กระบวนการประมวลผลข้อมูลใน Pipeline จำเป็นต้องอาศัยเครื่องมือที่สามารถจัดการข้อมูลในรูปแบบและโครงสร้างที่เหมาะสม การจัดระเบียบข้อมูลอย่างถูกต้องจึงเป็นสิ่งสำคัญสำหรับการทำงานของ Pipeline อย่างราบรื่น ซึ่งในบริบทของข้อมูลเพื่อการพัฒนา การจัดการข้อมูลในรูปแบบมาตรฐานมักใช้ไฟล์ในรูปแบบค่าที่คั่นด้วยเครื่องหมายจุลภาค (CSV) โดยให้แต่ละคอลัมน์แทนตัวชี้วัด และบรรทัดแรกแสดงชื่อของตัวชี้วัด อย่างไรก็ตาม ความต้องการรูปแบบใหม่ของข้อมูลมักเกิดขึ้นตามการเปลี่ยนแปลง ส่งผลให้เครื่องมือที่รองรับรูปแบบใหม่ต้องพัฒนาตามไปด้วยในภายหลัง ในกรณีที่เครื่องมือยังไม่รองรับ ผู้ใช้งานมักแก้ไขโดยการเขียนสคริปต์เฉพาะสำหรับการแปลงข้อมูลเพื่อให้ใช้งานได้ตรงตามความต้องการ[13]

2.2.10 **การจัดเก็บฟีเจอร์** (Feature Storage)

โดยทั่วไปแล้ว ฟีเจอร์สโตร์มีหน้าที่ให้บริการฟีเจอร์แก่โมเดลแบบออนไลน์ (online model serving pipelines) ด้วยค่าหน่วงต่ำ (low latency) และจัดเก็บข้อมูลประวัติรวมถึงฟีเจอร์ปริมาณมากสำหรับกระบวนการฝึกโมเดล (model training pipelines) ด้วย ดังนั้น ฟีเจอร์สโตร์มักมีดาต้าสโตร์ (datastore) สองประเภทหลัก ๆ ได้แก่ Offline Store สำหรับใช้ในการฝึกโมเดลแบบออฟไลน์ Online Store สำหรับให้บริการฟีเจอร์แบบออนไลน์



ภาพประกอบ 5 serving pipeline with a feature store ที่มา [14]

สำหรับ Offline Store มักจะเป็นระบบจัดเก็บข้อมูลที่รองรับอัตราการรับ-ส่งข้อมูลสูง (high throughput) แต่ค่าหน่วงสูง (high latency) เช่น Cloud Object Store หรือ Data Lake

ในขณะที่ Online Store ต้องให้บริการฟีเจอร์ด้วยค่าหน่วงต่ำอย่างเข้มงวด (ในระดับหลักร้อยมิลลิวินาที) จึงมักต้องใช้การเก็บข้อมูลใน K/V Store ที่อยู่ในหน่วยความจำ (in-memory) เช่น Redis และเป็นเพียงชุดย่อยของฟีเจอร์ทั้งหมด

ความท้าทายประการหนึ่งของการแบ่งการจัดเก็บฟีเจอร์ระหว่าง Online Store และ Offline Store คือการรักษาความสอดคล้อง (consistency) ระหว่างข้อมูลออฟไลน์และออนไลน์ ความแตกต่างในรูปแบบการนำเข้าฟีเจอร์ (ingestion), การทำ Materialization หรือแม้แต่การนิยามฟีเจอร์ ระหว่าง Offline Store กับ Online Store ส่งผลให้ฟีเจอร์ชุดหนึ่งที่เสิร์ฟให้กระบวนการฝึกโมเดลและกระบวนการทำนายไม่เหมือนกันโดยสิ้นเชิง ซึ่งความแตกต่างเล็กน้อยเหล่านี้อาจนำไปสู่การเกิด Data Drift ในโมเดล และก่อให้เกิดความเสื่อมถอย (degradation) อย่างมีนัยสำคัญต่อความแม่นยำของการทำนาย

ด้วยเหตุนี้ฟีเจอร์สโตร์บางแห่งจึงอนุญาตให้ทำการอัปเดตโดยตรงได้เฉพาะใน Offline Store หรือ Online Store อย่างใดอย่างหนึ่งเท่านั้น จากนั้นจึงทำการซิงก์ (sync) ข้อมูลจากสโตร์หนึ่งไปยังอีกสโตร์หนึ่ง วิธีนี้แม้จะช่วยลดความเสี่ยงจากการเกิด Data Drift ได้ แต่ก็อาจเพิ่มภาระและค่าหน่วงในการอัปเดตฟีเจอร์ขึ้นมาอีก [14]

2.2.11 **การเปรียบเทียบฟีเจอร์สโตล์** (Comparison Feature Store)

Databricks Feature Store Databricks Feature Store ทำงานบนฐาน Delta Lake ซึ่งเป็น open-source table format ขยายมาจาก Parquet พร้อม transaction log และ metadata สำหรับ ACID guarantees และ time travel

ข้อดี ACID & Time travel Delta Lake บันทึก commit log ทุกครั้งที่เขียนข้อมูล ทำให้สามารถย้อนดูสถานะของ table ณ จุดเวลาที่ต้องการ (time travel) และ rollback ได้ง่าย และIntegration กับ Spark ecosystem ถ้า pipeline ของคุณอยู่บน Spark อยู่แล้ว (เช่น MLflow, Delta Live Tables) การใช้ Feature Store คือตัวเลือกที่เชื่อมต่อได้ทันที

ข้อเสีย ภาษาและ DSL สองแบบ แม้จะมี Python API ให้ แต่เบื้องหลังยังใช้ Spark DSL และ SQL ทำให้ data scientist อาจเจอ friction ในการสลับ context ระหว่าง Python code กับ Spark SQL

Amazon SageMaker Feature Store SageMaker Feature Store เก็บข้อมูลฟีเจอร์บน S3 แล้วจัดการ metadata ผ่าน Glue Data Catalog พร้อม table format เป็น Apache Iceberg (Parquet-backed)

ข้อดี Scale บน S3 ใช้ object store ที่ scale ได้ไม่จำกัดของ AWS S3 จึงเหมาะกับข้อมูลขนาดใหญ่และ high-throughput workload และ Iceberg table format: รองรับ atomic operations, schema evolution, และ time travel ซึ่งช่วยให้การเปลี่ยน schema หรือย้อนดู history ทำได้สะดวกในตัว Iceberg

ข้อเสีย Partitioning จำกัดเฉพาะ hourly buckets Iceberg บน SageMaker กำหนด partitioning เป็นรายชั่วโมงล่วงหน้า ทำให้ไม่สามารถปรับ custom partition ที่ตอบโจทย์ pattern ของข้อมูลจริง เช่น partition ตาม user_id หรือ event type ได้เต็มที่

Hopsworks Feature Store ระบบนี้ผสาน HopsFS (fork ของ HDFS) กับ Apache Hudi ในการจัดเก็บ Parquet + commit log และใช้ Arrow Flight ร่วมกับ DuckDB เพื่อ query แบบ vectorized zero-copy

ข้อดี Vectorized execution ด้วย DuckDB: ทำ query ได้เร็วสูงสุดในงานวิจัยนี้ โดยไม่ต้องพึ่ง cluster ภายนอก (embedded in Python) และ API Python-first Data scientist เข้าถึง feature ผ่าน Python API โดยตรง และไม่ต้องสลับ context ไปเขียน SQL หรือ Spark DSL

ข้อเสีย การติดตั้งและดูแลระบบที่ซับซ้อน ต้องจัดการ HopsFS/Hudi cluster เพิ่มเติม ซึ่งไม่ง่ายเท่า object store บน cloud และ Community ใหม่: ชุมชนและการสนับสนุนระดับ enterprise ยังน้อยกว่าเจ้าใหญ่[15, 16]

ในบทนี้ผู้วิจัยได้ทำการสำรวจและทบทวนงานวิจัยที่เกี่ยวข้องกับการพัฒนา Data Pipeline โดยเฉพาะการใช้เครื่องมือ Apache Airflow และ FEAST (Feature Store) เพื่อประยุกต์ในการจัดการข้อมูลและคุณลักษณะ (Feature Management) สำหรับการวิเคราะห์และสร้างโมเดล Machine Learning โดยได้แบ่งเนื้อหาออกเป็น 2 ส่วนหลักคือ งานวิจัยที่เกี่ยวข้องกับการทำ Data Pipeline ด้วย FEAST และ Apache Airflow และ ทฤษฎีและหลักการที่เกี่ยวข้องในการทำงานวิจัยนี้ ในส่วนของงานวิจัยที่เกี่ยวข้อง ได้กล่าวถึงงานวิจัยต่าง ๆ ที่เน้นการใช้ Apache Airflow ในการออกแบบและจัดการ Data Pipeline บนสภาพแวดล้อมคลาวด์ รวมถึงการพัฒนาโครงสร้างสถาปัตยกรรมที่มีประสิทธิภาพทั้งในรูปแบบ Batch นอกจากนี้ยังมีการกล่าวถึงการจัดการและประเมินประสิทธิภาพในการใช้งาน Airflow ในสถานการณ์จริง และการเชื่อมต่อกับบริการต่าง ๆ บนแพลตฟอร์มคลาวด์ เช่น Google Cloud Platform และ ทฤษฎีที่เกี่ยวข้อง ผู้วิจัยได้อธิบายหลักการทำงานของ Apache Airflow โดยเน้นที่โครงสร้าง

Directed Acyclic Graph (DAG) ซึ่งเป็นหัวใจสำคัญในการจัดการงานและการดำเนินการของ Data Pipeline รวมถึงขั้นตอนและองค์ประกอบต่าง ๆ เช่น Operators, Tasks, Scheduling, การออกแบบเพื่อเพิ่มความทนทาน (Robustness) และการจัดการข้อผิดพลาด นอกจากนี้ยังได้กล่าวถึงทฤษฎีที่เกี่ยวข้องกับการจัดการข้อมูล เช่น การนำเข้าข้อมูล (Data Ingestion) การจัดเก็บและการจัดโครงสร้างข้อมูล (Data Storage and Structuring), การจัดเก็บฟีเจอร์ (Feature Storage) และความสำคัญของ FEAST Feature Store ซึ่งช่วยให้การจัดการข้อมูลฟีเจอร์ในระบบ Machine Learning ทำได้ง่ายและมีประสิทธิภาพยิ่งขึ้น

และในบทต่อไป ผู้วิจัยจะกล่าวถึงขั้นตอนการออกแบบและพัฒนาระบบ Data Pipeline ที่ผสานการใช้งาน Apache Airflow และ FEAST Feature Store อย่างละเอียด รวมทั้งนำเสนอขั้นตอนการทดลองของระบบที่พัฒนาขึ้นตามกรอบแนวคิดที่กำหนดไว้ในบทนี้

บทที่ 3
ระเบียบวิธีวิจัย

การวิจัยครั้งนี้มีเป้าหมายเพื่อศึกษาการทำงานของ Feature Store (FEAST) โดยเน้นการบูรณาการกับ Apache Airflow ในการจัดการเวิร์กโฟลว์และพัฒนา Data Pipeline ที่มีประสิทธิภาพสำหรับงานด้าน Machine Learning (ML) ภายใต้สภาพแวดล้อมคลาวด์ การวิจัยมุ่งเน้นไปที่การวิเคราะห์และประเมินประสิทธิภาพของ FEAST ในด้านการจัดเก็บ การแปลงข้อมูล และการให้บริการข้อมูลแบบเรียลไทม์ เพื่อรองรับการใช้งานในองค์กรขนาดใหญ่

3.1 วางแผนการดำเนินงานวิจัย

ผู้วิจัยทำการวางแผนงานการดำเนินงานวิจัย ระหว่าง เดือนสิงหาคม พ.ศ. 2567 ถึง เดือน เมษายน พ.ศ. 2568 และดำเนินการตามขั้นตอนต่าง ๆ ดังแสดงในตาราง 1

ตาราง 1 แผนการดำเนินงานวิจัย

ขั้นตอนดำเนินงาน	พ.ศ. 2567					พ.ศ. 2568			
	ส.ค.	ก.ย.	ต.ค.	พ.ย.	ธ.ค.	ม.ค.	ก.พ.	มี.ค.	เม.ษ.
1. วางแผนการทำงาน									
2. ศึกษาทฤษฎี									
3. เก็บรวบรวมข้อมูล									
4. เตรียมข้อมูล									
5. ออกแบบขั้นตอน									
6. ปรับปรุงประสิทธิภาพ									
7. ประเมินผล									
8. สรุปผลและรายงานผล									

3.2 ขั้นตอนการดำเนินการวิจัย

ขั้นตอนการดำเนินการวิจัยในงานศึกษานี้ประกอบไปด้วยพัฒนาและวิเคราะห์การทำงานของระบบที่เกี่ยวข้อง เพื่อให้เข้าใจข้อกำหนดและปัญหาที่ต้องแก้ไข จากนั้นจึงทำการออกแบบ

ระบบและเวิร์กโฟลว์ที่ใช้สำหรับ Data Pipeline ทั้งในรูปแบบที่ใช้ Feature Store (FEAST) และรูปแบบที่ไม่ใช้ Feature Store (Non-FEAST) เพื่อเปรียบเทียบประสิทธิภาพ ต่อจากนั้นเป็นขั้นตอนการพัฒนาและทดสอบระบบตามแผนที่ได้ออกแบบไว้ โดยการดำเนินงานทั้งสองรูปแบบและถูกนำไปทดสอบภายใต้สภาพแวดล้อมบนคลาวด์ (Google Cloud Platform) และขั้นตอนสุดท้ายเป็นการประเมินผลและวิเคราะห์ข้อมูลที่ได้จากการทดสอบ เพื่อสรุปวัดผลประสิทธิภาพและความเหมาะสมของการใช้งาน FEAST ในกระบวนการจัดการข้อมูลและคุณลักษณะสำหรับงาน Machine Learning

ภาพประกอบ 5 Flow Chart วิธีดำเนินการพัฒนา Data Pipeline ร่วมกับ FEAST

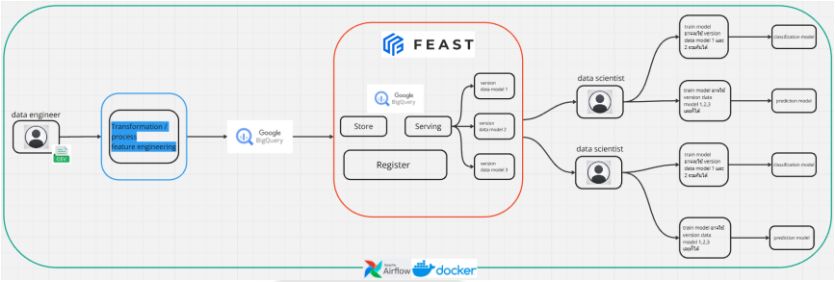
3.2.1 การศึกษาและวิเคราะห์การทำงาน

ศึกษาและทำความเข้าใจเกี่ยวกับแนวคิดและเทคโนโลยีที่เกี่ยวข้องกับการพัฒนา Feature Store โดยเฉพาะ FEAST และการจัดการเวิร์กโฟลว์ด้วย Apache Airflow รวมถึงการสร้าง Data Pipeline เพื่อรองรับการทำงานที่มีความซับซ้อนในระบบ Machine Learning (ML) Pipelines โดยรายละเอียดของการดำเนินการมีดังนี้:

ศึกษาทฤษฎีและเอกสารที่เกี่ยวข้อง ทำการวิเคราะห์งานวิจัยที่เกี่ยวข้องกับการจัดการข้อมูลคุณลักษณะ (Feature Management) และการพัฒนา Feature Store

ศึกษาบทบาทของ Apache Airflow ในการจัดการและอำนวยความสะดวกใน Workflow สำหรับ ML Pipelines ทำความเข้าใจเทคโนโลยีที่เกี่ยวข้อง และวิเคราะห์การใช้งานฐานข้อมูล เช่น BigQuery (สำหรับ Offline Store)

3.2.2 การออกแบบระบบและเวิร์กโฟลว์ของ FEAST



ภาพประกอบ 6 Workflow Model FEAST

การออกแบบ Data Pipeline สร้าง Data Pipeline โดยใช้ Apache Airflow เพื่อจัดการกระบวนการแปลงข้อมูลตั้งแต่การโหลดข้อมูล (Data Ingestion) การสร้างคุณลักษณะ (Feature Engineering) ไปจนถึงการให้บริการข้อมูลคุณลักษณะ (Feature Serving)

ขั้นตอนใน Workflow Model FEAST ภาพรวมของ Data Pipeline และบทบาทของผู้มีส่วนร่วมในภาพ แสดงถึง Data Pipeline ซึ่งมีการประสานงานระหว่างสามส่วนหลัก ได้แก่ Data Engineer ผู้ดูแลกระบวนการเตรียมข้อมูลและสร้าง Feature ต่าง ๆ Data Scientist ผู้วิจัยและพัฒนาโมเดลต่าง ๆ โดยอาศัยคุณลักษณะ(Features) ที่ได้จากกระบวนการข้างต้น ระบบ Serving หรือ Feature Store (FEAST): สำหรับจัดเก็บและให้บริการข้อมูลคุณลักษณะ (Features) รวมถึงการจัดการเวอร์ชันของชุดข้อมูล (Data Model Versions)

ภายใน Pipeline มีการใช้ Apache Airflow เพื่อออร์เคสเทรต (orchestrate) ขั้นตอนต่าง ๆ ตั้งแต่การโหลดข้อมูลเข้าสู่ Data Warehouse จนถึงการให้บริการข้อมูลเพื่อทำ Machine Learning (ML) และการใช้ Docker เพื่อควบคุมสภาพแวดล้อมการทำงานให้มีความยืดหยุ่นและทำซ้ำได้ (reproducible)

กระบวนการ Data Ingestion และการเก็บข้อมูลใน BigQuery มีการนำข้อมูลจาก Local files เข้าสู่ Google BigQuery โดยขั้นตอนในภาพแสดงถึงการที่ Data Engineer ทำการรวบรวมไฟล์ข้อมูลจากแหล่งต่าง ๆ นำเข้าสู่ BigQuery ผ่านกระบวนการที่ Apache Airflow ควบคุม (Task หรือ DAG ขั้นตอน “Data Ingestion”) BigQuery ทำหน้าที่เป็น Data Warehouse

ซึ่งออกแบบมาเพื่อรองรับการจัดเก็บข้อมูลขนาดใหญ่และรองรับการวิเคราะห์ได้อย่างรวดเร็ว ขั้นตอนนี้จะช่วยให้ข้อมูลพร้อมที่จะถูกนำไปประมวลผลต่อ ไม่ว่าจะเป็นการทำ Feature Engineering หรือการฝึกโมเดล

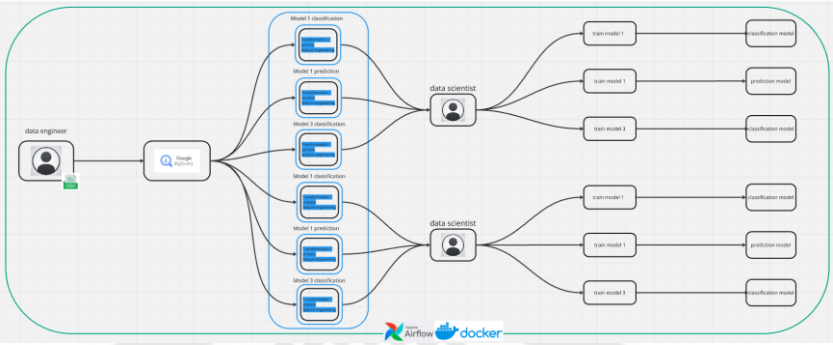
การแปลงข้อมูลใน Virtual Environment ของ FEAST (Feature Store) ข้อมูลอยู่ใน BigQuery จะถูกนำเข้าสู่ FEAST ซึ่งทำหน้าที่เป็น Feature Store หรือระบบบริหารจัดการ “คุณลักษณะ” โดยในภาพอธิบายเป็นขั้นตอน Process Feature Engineering: เป็นการนำข้อมูลจาก BigQuery มาผ่านงานแปลง (transformation) เช่น การสร้าง Feature เชิงเวลา (Temporal Features) หรือการทำ Aggregation เพื่อให้ได้ Feature ที่พร้อมใช้ในการเทรนโมเดล ระบบจะเก็บ Feature ไว้ในรูปแบบ Version (ในภาพปรากฏเป็น version data model 1, 2, 3 ซึ่งสามารถต่างกันตามจำนวน Feature หรือวิธีการเลือกคุณลักษณะ) กระบวนการนี้สอดคล้องกับแนวคิดในบทความที่ระบุว่า “การแปลงข้อมูลใน Virtual Environment ของ FEAST” ช่วยให้การปรับปรุง Feature ทำได้อย่างยืดหยุ่นและเป็นมาตรฐาน รวมถึงการจัดเก็บเป็นเวอร์ชันเพื่อรองรับการทดลอง (experimentation) ของ Data Scientist การให้บริการข้อมูลคุณลักษณะ (Feature Serving)

จากขั้นตอน Feature Engineering ข้อมูล “คุณลักษณะ” จะพร้อมนำไปใช้ในลักษณะผ่าน Google BigQuery (Offline Serving) สำหรับงานวิเคราะห์เชิงลึก (analytical workloads) หรือการสร้างโมเดลแบบ batch ที่ต้องการชุดข้อมูลใหญ่ Data Scientist อาจโหลดชุดข้อมูลเวอร์ชันต่าง ๆ จาก BigQuery เพื่อทำการเทรนโมเดล เช่น การเทรน Classification Model หรือ Prediction Model การฝึกและทดสอบโมเดลโดย Data Scientist

Data Scientist ทั้งสองคนสามารถเลือกรับ Feature Model เวอร์ชันต่าง ๆ จากระบบ Serving ไปใช้ได้อย่างยืดหยุ่น เพื่อฝึกโมเดลเช่น Classification Model อาจต้องใช้ Feature เฉพาะบางชุด หรือหลายชุดรวมกันเพื่อเพิ่มความแม่นยำ Prediction Model อาจต้องใช้ Feature ที่แตกต่าง เช่น ข้อมูลเชิงเวลา หรือชุด Feature ที่ได้จากการ Aggregate

การแบ่งเวอร์ชันของ Data Model (feature sets) และการเลือกใช้เวอร์ชันมากกว่าหนึ่งร่วมกัน จึงเป็นกระบวนการทดสอบเพื่อหาเวอร์ชันที่ให้ผลลัพธ์ดีที่สุด รวมถึงสามารถรองรับการปรับจูน (tuning) อย่างต่อเนื่อง

3.2.2 การออกแบบระบบและเวิร์กโฟลว์ของ NON FEAST (Non Feature store)



ภาพประกอบ 7 Workflow Model NON FEAST

ในภาพแสดงการทำงานโดยมี Data Engineer ที่นำข้อมูลจากแหล่งต่าง ๆ (แสดงเป็น Local files หรือแหล่งอื่น ๆ) เข้า Google BigQuery จากนั้น Data Scientist แต่ละคนจะดึงข้อมูลจาก BigQuery มาทำ Feature Engineering (แสดงเป็น “process feature engineering”) ก่อนฝึก (train) โมเดลของตนเอง เช่น train model 1, train model 3 จะไม่มีการเก็บเวอร์ชันของ Feature” หมายความว่า ขั้นตอน Feature Engineering นี้เกิดขึ้นใหม่แทบทุกครั้งที่มีการสร้างโมเดลใหม่ หรือปรับปรุงโมเดลเดิม (retrain) เพราะไม่มีแพลตฟอร์มกลาง (Feature Store) สำหรับจัดเก็บ Feature ที่ผ่านการแปลง (transformed features) หรือ mapping ระหว่าง Feature กับเวอร์ชันของข้อมูล Data Ingestion นำข้อมูลจาก Local หรือแหล่งอื่นเข้าสู่ BigQuery ซึ่ง Data Engineer ทำการโหลดข้อมูลจากไฟล์ในเครื่อง (Local Files) หรือระบบอื่น ๆ (เช่น CSV, JSON, ฐานข้อมูล) เข้า Google BigQuery ใช้เครื่องมืออย่าง Apache Airflow เพื่อ Orchestrate ให้กระบวนการนำข้อมูล (upload) เป็นอัตโนมัติ Data Transformation (Pre-Processing / Cleaning) หลังจากข้อมูลอยู่ใน BigQuery แล้ว อาจมีการทำ Data Cleaning หรือ Transformation เบื้องต้นผ่าน BigQuery SQL, Python Scripts หรือ Airflow Operator ต่าง ๆ ตัวอย่างเช่น การจัดการ Missing Value, การจัดรูปแบบคอลัมน์, หรือการกรองข้อมูลที่ไม่จำเป็น Feature Engineering กระบวนการสร้างฟีเจอร์ Data Scientist ดึงข้อมูลจาก BigQuery มายัง Jupyter Notebook หรือ Python Script

ทำการสร้าง Feature (เช่น การสร้าง Feature เชิงเวลา, Aggregation, Encoding) ด้วยโค้ดเฉพาะกิจ บางครั้งอาจเก็บฟีเจอร์ที่ได้เป็นตารางใหม่ใน BigQuery การเก็บฟีเจอร์ใน BigQuery

หากต้องการใช้งานฟีเจอร์ร่วมกัน ทีมงานอาจตกลงกันว่า “เมื่อสร้างฟีเจอร์แล้ว ให้ push กลับไปเก็บที่ตาราง BigQuery ชื่อ features_v1 หรือ features_v2” หรือเก็บเป็นไฟล์ CSV ใน Cloud Storage/Local ใช้ Airflow เพื่อ Automate ได้บางส่วน เช่น หลังจากวิ่งสคริปต์ Feature Engineering เสร็จแล้วให้เขียนข้อมูลกลับ BigQuery Offline Serving / การนำฟีเจอร์ไปใช้เทรนโมเดล (Training)

Data Scientist จะทำการ “Train Model” โดยดึงฟีเจอร์จากตาราง BigQuery ที่สร้างไว้หรือโหลดไฟล์ CSV/Parquet ที่มีฟีเจอร์

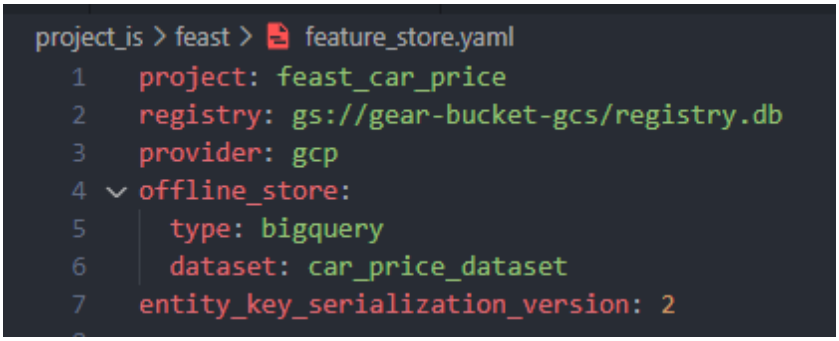
เมื่ออยากเปลี่ยนฟีเจอร์ใหม่ ก็ต้องรันกระบวนการสร้างฟีเจอร์ใหม่อีกครั้ง และสร้างไฟล์/ตารางใหม่เอง

3.2.3 การพัฒนาและทดสอบระบบ

3.2.3.1 FEAST

ติดตั้งและกำหนดค่า FEAST ให้เชื่อมต่อกับ Offline Store (BigQuery) เพื่อใช้งานร่วมกันในกระบวนการจัดการข้อมูลคุณลักษณะ (Features) อย่างเป็นระบบ โดยติดตั้ง FEAST จาก docker images ในเวอร์ชั่น 0.41และ Python เวอร์ชั่น 3.10

กำหนดค่าการเชื่อมต่อกับ BigQuery ในไฟล์ feature_store.yaml หรือไฟล์กำหนดค่าต่าง ๆ ของ Feast ให้ตั้งค่าการเชื่อมต่อ (credentials) กับ BigQuery ดังภาพประกอบ 8



ภาพประกอบ 8 Config Feature Store

ระบุชื่อโครงการ (project) ชื่อ dataset ชื่อตาราง เพื่อให้ Feast สามารถอ่าน–เขียนข้อมูลฟีเจอร์ได้ การเตรียมข้อมูลและแปลงข้อมูล (Feature Engineering)

ในขั้นตอนนี้ ผู้วิจัยได้เขียนสคริปต์ Python เพื่ออ่านข้อมูลดิบที่จัดเก็บในรูปแบบไฟล์ CSV จากนั้นทำการแปลงข้อมูลเบื้องต้น โดยปรับแต่งชื่อคอลัมน์ให้เป็นมาตรฐาน และเพิ่มฟีเจอร์ที่สำคัญ เช่น price_per_mile, car_age, mileage_per_year, และ remaining_useful_life เพื่อให้ข้อมูลมีความสมบูรณ์พร้อมใช้งานในขั้นตอนต่อไป นอกจากนี้ยังมีการเพิ่มคอลัมน์ event_timestamp ซึ่งจำเป็นต่อการจัดการฟีเจอร์ของ Feast หลังจากแปลงข้อมูลเสร็จเรียบร้อยแล้ว ข้อมูลที่ได้จะถูกบันทึกกลับไปเป็น CSV ไฟล์ใหม่ในโฟลเดอร์ที่กำหนดไว้ดังภาพประกอบ 9

```python
def feature_engineering():
    if not os.path.exists(transformed_folder):
        os.makedirs(transformed_folder)

    # กำหนด path สำหรับไฟล์ต้นทาง transformed1
    transformed1_file = os.path.join(transformed_folder, "transformed1_car_price_dataset.csv")

    try:
        # อ่านข้อมูลจากไฟล์ transformed1
        df = pd.read_csv(transformed1_file)
        logger.info(f"อ่านไฟล์ {transformed1_file} สำเร็จ จำนวนแถว: {len(df)}")
    except Exception as e:
        logger.error(f"เกิดข้อผิดพลาดในการอ่านไฟล์ {transformed1_file}: {e}")
        return

    # สร้างคอลัมน์ car_id แบบ sequential ใหม่สำหรับทุกแถว
    num_rows = len(df)
    df["car_id"] = range(global_car_id, global_car_id + num_rows)
    global_car_id += num_rows

    # เพิ่มคอลัมน์ event_timestamp ด้วยเวลาปัจจุบัน
    df['event_timestamp'] = pd.Timestamp.now()

    # ---- Transformation code ที่ต้องการ ----
    df["car_age"] = 2025 - df["year"]
    df["mileage_per_year"] = (df["mileage"] / (df["car_age"] + 1)).round(2)
    df["remaining_useful_life"] = 20 - df["car_age"]
    df["remaining_useful_life"] = df["remaining_useful_life"].clip(lower=0)

    logger.info(f"Columns after adding car_id: {df.columns.tolist()}")
    logger.info(f"Before saving CSV, columns: {df.columns}")

    # บันทึกผลลัพธ์เป็นไฟล์ transformed2
    transformed2_file = os.path.join(transformed_folder, "transformed2_car_price_dataset.csv")
    try:
        df.to_csv(transformed2_file, index=False)
    except Exception as e:
        return
```

ภาพประกอบ 9 Config Feature Engineering

การโหลดข้อมูลเข้าสู่ BigQuery ขั้นตอนนี้จะใช้ Google BigQuery เป็น Offline Store สำหรับ Feast โดยผู้วิจัยเขียนสคริปต์ Python ที่ดึงข้อมูล CSV ที่ผ่านการแปลงเรียบร้อยแล้ว นำเข้าไปจัดเก็บใน BigQuery พร้อมกำหนด Schema ของข้อมูลให้สอดคล้องกับแต่ละฟีเจอร์ที่กำหนดขึ้น ได้แก่ brand, model, year, engine_size, fuel_type, transmission, mileage, car_doors,owner_count,price,price_per_mile,mileage_per_year,car_age,remaining_useful_life และ event_timestamp เพื่อความถูกต้องในการจัดเก็บและเรียกใช้งานข้อมูลดังรูปภาพ 10

```python
def load_local_csv_to_bigquery():
    if not csv_files:
        logger.info("No CSV files found to load into BigQuery.")
        return

    for filename in csv_files:
        try:
            file_path = os.path.join(transformed_folder, filename)
            df = pd.read_csv(file_path)
            logger.info(f"File {filename} read successfully with shape: {df.shape}")

            # ตรวจสอบว่ามีคอลัมน์ event_timestamp หรือไม่
            if 'event_timestamp' not in df.columns:
                logger.error(f"File {filename} missing 'event_timestamp' column. Skipping this file.")
                continue

            # แปลงคอลัมน์ event_timestamp ให้เป็น datetime
            df['event_timestamp'] = pd.to_datetime(df['event_timestamp'])

            table_ref = f"{project_id}.{dataset_id}.{table_id}"
            job_config = bigquery.LoadJobConfig(
                schema=[
                    bigquery.SchemaField("car_id", "INTEGER"),
                    bigquery.SchemaField("brand", "STRING"),
                    bigquery.SchemaField("model", "STRING"),
                    bigquery.SchemaField("year", "INTEGER"),
                    bigquery.SchemaField("engine_size", "FLOAT"),
                    bigquery.SchemaField("fuel_type", "STRING"),
                    bigquery.SchemaField("transmission", "STRING"),
                    bigquery.SchemaField("mileage", "INTEGER"),
                    bigquery.SchemaField("car_doors", "INTEGER"),
                    bigquery.SchemaField("owner_count", "INTEGER"),
                    bigquery.SchemaField("price", "INTEGER"),
                    bigquery.SchemaField("price_per_mile", "FLOAT"),
                    bigquery.SchemaField("mileage_per_year", "FLOAT"),
                    bigquery.SchemaField("car_age", "FLOAT"),
                    bigquery.SchemaField("remaining_useful_life", "FLOAT"),
                    bigquery.SchemaField("event_timestamp", "TIMESTAMP"),
                ],
                write_disposition="WRITE_TRUNCATE"  # เขียนทับข้อมูลเดิม
            )
            job = client.load_table_from_dataframe(df, table_ref, job_config=job_config)
            job.result()

            # logger.info(f"Loaded {filename} to BigQuery table {table_ref}")
        except Exception as e:
            logger.error(f"Error loading {filename} to BigQuery: {e}")
```

ภาพประกอบ 10 Config Load Data To Bigquery

การลงทะเบียนฟีเจอร์ใน Feast Feature Store ในขั้นตอนนี้เป็นการนำข้อมูลที่ถูกจัดเก็บใน BigQuery มาลงทะเบียนเป็น Feature View ผ่าน Feast Feature Store โดยมีการกำหนด Entity, Schema และ Feature Definitions ไว้ในไฟล์ feature_view_car_price.py ซึ่งจะทำให้ Feast สามารถบริหารจัดการและควบคุมเวอร์ชันของฟีเจอร์ที่สร้างขึ้นได้อย่างมีประสิทธิภาพ ดังภาพประกอบ 11

```python
def register_feature_in_feast():
    """
    ลงทะเบียนหรืออัปเดต Feature Store ของ Feast
    """
    fs = FeatureStore(repo_path="/opt/airflow/feast")

    try:
        fs.apply([car_price_feature_view])
    except Exception as e:
        logger.error(f"Error applying feature views: {e}")

    logger.info(f"Feature Views: {fs.list_feature_views()}")
    logger.info(f"Entities: {fs.list_entities()}")
```

ภาพประกอบ 11 Config Registry Feast

นอกจากนี้ยังสามารถตรวจสอบความถูกต้องได้ผ่านการแสดงผลลัพธ์ของฟังก์ชัน fs.feature_views() และ fs.entities()

การฝึกอบรมโมเดล (Model Training) ในการวิจัยนี้ ขั้นตอนการฝึกอบรมโมเดลเพื่อพยากรณ์ราคาและอายุการใช้งานที่เหลือของรถยนต์ได้นำเทคนิคการดึงข้อมูลฟีเจอร์ (feature extraction) ผ่านระบบ FEAST feature store มาใช้เป็นส่วนสำคัญ โดยมีรายละเอียดดังนี้

การสร้าง FeatureStore Object เริ่มต้นด้วยการสร้างออบเจ็กต์ของ FEAST feature store ด้วยการระบุพาธของ repository ที่เก็บไฟล์คอนฟิกูเรชันและคำจำกัดความของฟีเจอร์ ซึ่งในที่นี้ได้กำหนดไว้ที่ /opt/airflow/feast เพื่อใช้เป็นฐานข้อมูลสำหรับการดึงฟีเจอร์ที่ได้มีการกำหนดไว้ล่วงหน้าในระบบ

การกำหนด Entity DataFrame ในขั้นตอนถัดมา จำเป็นต้องระบุ entity ซึ่งจะเป็นตัวกำหนดว่าข้อมูลฟีเจอร์ที่ดึงมานั้นสอดคล้องกับเหตุการณ์ (event) และเวลาที่เกี่ยวข้องอย่างไร โดยใช้ SQL query เพื่อดึงข้อมูล car_id และ event_timestamp จาก BigQuery Entity DataFrame นี้มีความสำคัญในการจับคู่ข้อมูลฟีเจอร์กับตัวระบุเฉพาะ (identifier) ของแต่ละเรคคอร์ด เพื่อให้การดึงข้อมูลมีความถูกต้องและครบถ้วนตามช่วงเวลาที่ระบุไว้

การเรียกใช้ get_historical_features ด้วย Entity DataFrame ที่เตรียมไว้แล้ว ระบบจะเรียกใช้เมธอด get_historical_features ของ FEAST เพื่อดึงข้อมูลฟีเจอร์ที่ต้องการ โดยระบุชื่อฟีเจอร์ในรูปแบบ <feature_view_name>:<field_name> ซึ่งฟีเจอร์ที่ดึงมานั้นครอบคลุมข้อมูลที่เกี่ยวข้องกับยี่ห้อ รุ่น ปีที่ผลิต ขนาดเครื่องยนต์ และคุณลักษณะอื่น ๆ ที่มีผลต่อการพยากรณ์ราคาและอายุการใช้งานของรถยนต์ การดึงข้อมูลด้วย get_historical_features นี้เป็นขั้นตอนที่สำคัญ เนื่องจากช่วยให้ได้ข้อมูลฟีเจอร์ที่ถูกต้องตามช่วงเวลาและสอดคล้องกับ entity ทำให้ข้อมูลที่นำมาฝึกอบรมโมเดลมีความสมบูรณ์ หลังจากได้ข้อมูลฟีเจอร์ในรูปแบบ pandas DataFrame แล้ว จึงมีการปรับเปลี่ยนชื่อคอลัมน์ให้มีความกระชับและสื่อความหมายได้ชัดเจนมากยิ่งขึ้น ซึ่งจะช่วยให้กระบวนการประมวลผลข้อมูลต่อไป เช่น การแปลงข้อมูล (Label Encoding สำหรับฟีเจอร์ประเภท Categorical และการทำ StandardScaler สำหรับฟีเจอร์ประเภท Numeric) สามารถนำไปสร้างโมเดลแบบ Random Forest Regressor ได้อย่างมีประสิทธิภาพ

ซึ่งขั้นตอนนี้ใช้ข้อมูลฟีเจอร์ที่จัดเก็บไว้ใน BigQuery เพื่อสร้างโมเดลพยากรณ์ราคาและอายุการใช้งานที่เหลือของรถยนต์ โดยใช้โมเดลประเภท Random Forest Regressor ก่อนดำเนินการสร้างโมเดล จะทำการเตรียมข้อมูลด้วยวิธี Label Encoding สำหรับฟีเจอร์ที่เป็นข้อมูลประเภท Categorical และ StandardScaler สำหรับฟีเจอร์ที่เป็น Numeric หลังจากการฝึกโมเดลแล้วจะทำการบันทึกโมเดลที่ได้ด้วยไลบรารี joblib เป็นไฟล์ .pkl เก็บไว้ที่โฟลเดอร์ /opt/airflow/models เพื่อใช้งานในขั้นตอนถัดไปดังภาพประกอบ 12 และ 13

```python
def train_model_feast():
    # 1) สร้าง FeatureStore object
    fs = FeatureStore(repo_path="/opt/airflow/feast")

    entity_df = f"""
    SELECT
        car_id,
        event_timestamp
    FROM `is-my-project-428015.car_price_dataset.car_price_feast` WHERE event_timestamp IS NOT NULL
    """
    # 3) เรียก get_historical_features เพื่อดึงฟีเจอร์ตามที่ต้องการ

    training_df = fs.get_historical_features(
        entity_df=entity_df,
        features=[
            # ระบุชื่อฟีเจอร์ในรูปแบบ <feature_view_name>:<field_name>
            "car_price_feature_view:brand",
            "car_price_feature_view:model",
            "car_price_feature_view:year",
            "car_price_feature_view:engine_size",
            "car_price_feature_view:fuel_type",
            "car_price_feature_view:transmission",
            "car_price_feature_view:mileage",
            "car_price_feature_view:car_doors",
            "car_price_feature_view:owner_count",
            "car_price_feature_view:price",
            "car_price_feature_view:price_per_mile",
            "car_price_feature_view:mileage_per_year",
            "car_price_feature_view:car_age",
            "car_price_feature_view:remaining_useful_life"
        ]
    ).to_df()  # ได้เป็น pandas DataFrame พร้อมฟีเจอร์ครบ

    # ตัวอย่าง: เลือก columns ที่จะใช้เป็นฟีเจอร์ (X) และ target (y)
    categorical_features = ["car_price_feature_view:brand",
                            "car_price_feature_view:model",
                            "car_price_feature_view:fuel_type",
                            "car_price_feature_view:transmission"]
```

ภาพประกอบ 12 Config Extract Feast Feature Store

การทำนายผลด้วยโมเดล (Prediction) ขั้นตอนนี้ทำการทดสอบโมเดลที่สร้างขึ้นโดยโหลดโมเดลที่บันทึกไว้ และนำข้อมูลชุดใหม่จาก BigQuery มาใช้ทำนายราคาและอายุการใช้งานที่เหลือของรถยนต์ ในขั้นตอนนี้ข้อมูลจะถูกเตรียมให้ตรงกับขั้นตอนที่ใช้ในการเทรน เช่น การแปลงข้อมูลหมวดหมู่ (categorical features) และการปรับสเกลข้อมูล (scaling) ก่อนที่ผลลัพธ์จากการทำนายจะถูกบันทึกกลับไปยัง BigQuery เพื่อให้ทีม Data Scientist หรือระบบอื่น ๆ สามารถนำผลลัพธ์ไปใช้งานต่อได้ทันที

```python
# Label Encoding
label_encoders = {}
for col in ["brand", "model", "year", "fuel_type", "engine_size",
            "transmission", "mileage", "car_doors", "owner_count", "price", "price_per_mile"]:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
    # เพิ่มการนับการ reuse สำหรับฟีเจอร์นี้ (ในที่นี้ถือว่า task นี้ใช้งานฟีเจอร์ 1 ครั้ง)
    feature_reuse_counts[col] += 1

# ✅ สะสมลงไฟล์
update_feature_reuse_counts(feature_reuse_counts)

# เลือกฟีเจอร์กับเป้าหมาย
features = ["brand", "model", "car_age", "engine_size", "fuel_type",
            "transmission", "mileage", "mileage_per_year",
            "car_doors", "owner_count"]
target = "price"

X = df[features]
y = df[target]

# Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_scaled, y)

# Save
joblib.dump(model, "/opt/airflow/models/car_price_model_feast.pkl")
joblib.dump(scaler, "/opt/airflow/models/scaler_car_price_feast.pkl")
joblib.dump(label_encoders, "/opt/airflow/models/label_encoders_car_price_feast.pkl")

logger.info("✅ Model trained (using Feast) and saved!")
```

ภาพประกอบ 13 Config Train Model_Feast

3.2.3.2 NON FEAST

เพื่อเปรียบเทียบกับการใช้ FEAST จะมีการพัฒนาระบบด้วยกระบวนการสร้างฟีเจอร์แบบทั่วไปที่ไม่มีการใช้ Feature Store ที่ได้นำเสนอโค้ด DAG (Directed Acyclic Graph) ของ Apache Airflow เพื่อแสดงกระบวนการดึงข้อมูล แปลงข้อมูล สร้างโมเดล และทำนายผลนั้น

มีรายละเอียดเชิงกระบวนการวิจัย (Methodology) ดังต่อไปนี้ การประยุกต์ใช้ Non-Feast Pipeline

Task Feature_Engineering ในขั้นตอนนี้ เริ่มจากการดึงข้อมูลจากไฟล์ CSV ที่ชื่อว่า transformed1_car_price_dataset.csv ซึ่งจัดเก็บไว้ในโฟลเดอร์ transformed โดยขั้นตอนนี้จะมีการตรวจสอบและเตรียมข้อมูลให้พร้อมสำหรับการแปลงข้อมูลเพิ่มเติม จากนั้นจึงเริ่มกระบวนการ Feature Engineering

สร้าง car_id เป็นหมายเลขลำดับต่อเนื่อง (Sequential) เพื่อระบุรถยนต์แต่ละคันอย่างชัดเจนและไม่ซ้ำกัน

เพิ่มคอลัมน์ event_timestamp เพื่อบันทึกเวลา ณ ปัจจุบันที่ข้อมูลถูกประมวลผล เป็นประโยชน์ในการติดตามและตรวจสอบความถูกต้องของข้อมูลย้อนหลังดังภาพประกอบ 14

```python
# สร้างคอลัมน์ car_id แบบ sequential
num_rows = len(df)
df["car_id"] = range(global_car_id, global_car_id + num_rows)
global_car_id += num_rows

# เพิ่มคอลัมน์ event_timestamp ด้วยเวลาปัจจุบัน
df['event_timestamp'] = pd.Timestamp.now()
```

ภาพประกอบ 14 Config Created Sequential

แปลงข้อมูลผ่านฟังก์ชันที่เตรียมไว้ดังภาพประกอบ 15

```python
# เรียกใช้ transformation functions ที่ถูกตกแต่งด้วย decorator
df = compute_car_age(df)
df = compute_mileage_per_year(df)
df = compute_remaining_useful_life(df)
df = encode_categorical(df, "brand", "model", "fuel_type", "transmission")

# Feature Engineering car_ages
df["car_age"] = 2025 - df["year"]
df["mileage_per_year"] = (df["mileage"] / (df["car_age"] + 1)).round(2)
df["remaining_useful_life"] = 20 - df["car_age"]
df["remaining_useful_life"] = df["remaining_useful_life"].clip(lower=0)

logger.info(f"งาน feature_engineering เสร็จสิ้น, ตัวอย่างข้อมูล (5 แถว):\n{df.head()}")

# บันทึกผลลัพธ์เป็นไฟล์ transformed2
transformed2_file = os.path.join(transformed_folder, "transformed2_car_price_dataset.csv")
try:
    df.to_csv(transformed2_file, index=False)
    logger.info(f"บันทึกไฟล์ที่แปลงแล้วสำเร็จ: {transformed2_file}")
except Exception as e:
    logger.error(f"เกิดข้อผิดพลาดในการบันทึกไฟล์ {transformed2_file}: {e}")
    return

return df
```

ภาพประกอบ 15 Config Transformation Data

จากภาพประกอบ compute_car_age คำนวณอายุของรถยนต์จากปีปัจจุบัน (2025) compute_mileage_per_year คำนวณระยะทางเฉลี่ยที่รถยนต์วิ่งต่อปี เพื่อวิเคราะห์การ

ใช้งานจริง compute_remaining_useful_life: คำนวณอายุการใช้งานที่เหลือของรถยนต์ โดยตั้งค่าการใช้งานสูงสุดไว้ที่ 20 ปี และจำกัดค่าไม่ให้ต่ำกว่า 0 เพื่อให้ข้อมูลที่ได้มีความสมเหตุสมผล encode_categorical แปลงข้อมูลประเภทตัวอักษร เช่น brand, model, fuel_type, transmission ให้อยู่ในรูปแบบตัวเลขด้วยเทคนิค Label Encoding เพื่อให้สามารถนำไปใช้ในการสร้างโมเดลได้อย่างมีประสิทธิภาพ หลังจากนั้น ผลลัพธ์ที่ได้จะถูกจัดเก็บในไฟล์ transformed2_car_price_dataset.csv ดังภาพประกอบ 15

ขั้นตอน Data Loading ไปยัง BigQuery ในขั้นตอนนี้ จะนำข้อมูลที่ได้จาก Feature Engineering ซึ่งถูกบันทึกไว้ในไฟล์ transformed2_car_price_dataset.csv มาดำเนินการต่อ

```
    # แปลงคอลัมน์ event_timestamp ให้เป็น datetime
    df['event_timestamp'] = pd.to_datetime(df['event_timestamp'])

    table_ref = f"{project_id}.{dataset_id}.{table_id}"
    job_config = bigquery.LoadJobConfig(
        schema=[
            bigquery.SchemaField("car_id", "INTEGER"),
            bigquery.SchemaField("brand", "STRING"),
            bigquery.SchemaField("model", "STRING"),
            bigquery.SchemaField("year", "INTEGER"),
            bigquery.SchemaField("engine_size", "FLOAT"),
            bigquery.SchemaField("fuel_type", "STRING"),
            bigquery.SchemaField("transmission", "STRING"),
            bigquery.SchemaField("mileage", "INTEGER"),
            bigquery.SchemaField("car_doors", "INTEGER"),
            bigquery.SchemaField("owner_count", "INTEGER"),
            bigquery.SchemaField("price", "INTEGER"),
            bigquery.SchemaField("price_per_mile", "FLOAT"),
            bigquery.SchemaField("mileage_per_year", "FLOAT"),
            bigquery.SchemaField("car_age", "FLOAT"),
            bigquery.SchemaField("remaining_useful_life", "FLOAT"),
            bigquery.SchemaField("event_timestamp", "TIMESTAMP"),
        ],
        write_disposition="WRITE_TRUNCATE"  # เขียนทับข้อมูลเดิม
    )
    job = client.load_table_from_dataframe(df, table_ref, job_config=job_config)
    job.result()

    logger.info(f"Loaded {filename} to BigQuery table {table_ref}")
except Exception as e:
    logger.error(f"Error loading {filename} to BigQuery: {e}")
```

ภาพประกอบ 16 Config Load Data To Bigquery

โดยจะตรวจสอบข้อมูลให้ถูกต้องก่อนนำไปใช้จริง จากนั้นกำหนด Schema ให้สอดคล้องกับข้อมูลที่มีอยู่ใน DataFrame หลังจากนั้นข้อมูลทั้งหมดจะถูกนำเข้าสู่ BigQuery โดยการตั้งค่า

ให้เขียนทับข้อมูลเดิมทุกครั้ง (write_disposition="WRITE_TRUNCATE") ซึ่งช่วยให้มั่นใจได้ว่าข้อมูลในระบบจะมีความทันสมัยและถูกต้องที่สุด

ขั้นตอนการสร้างโมเดลและฝึกโมเดล (Model Training) ขั้นตอนนี้จะดำเนินงานแยกออกเป็นสองโมเดลที่แตกต่างกัน เพื่อให้ครอบคลุมเป้าหมายในการวิเคราะห์ ได้แก่ model car price และ model car age ดังภาพประกอบ 16

โมเดลพยากรณ์ราคา (Price Prediction) ดึงข้อมูลที่ได้โหลดไว้ใน Google BigQuery แล้วทำ Feature Engineering อีกครั้งด้วยการคำนวณคอลัมน์ที่จำเป็น ทำการแปลงข้อมูล เช่น Label Encoding และ Scaling เพื่อให้เหมาะสมต่อการนำไปสร้างโมเดล ดังภาพประกอบ 17

```python
def train_model():

    client = bigquery.Client(project=project_id)
    query = f"SELECT * FROM `{project_id}.{dataset_id}.{table_id}`"
    df = client.query(query).to_dataframe()
    # ปริ้นข้อมูล DataFrame ที่มีขนาดใหญ่
    logger.info(f"DataFrame head:\n{df.head(100)}")


    # ✅ ตรวจสอบว่ามีโฟลเดอร์ models หรือไม่ ถ้าไม่มีให้สร้าง
    models_folder = "/opt/airflow/models"
    if not os.path.exists(models_folder):
        os.makedirs(models_folder)

    # ใช้ transformation functions สำหรับ train_model (จะถูกนับใน task 'train_model')
    df = compute_car_age_train(df)
    df = compute_mileage_per_year_train(df)
    df = compute_remaining_useful_life_train(df)

    categorical_features = ["brand", "model", "fuel_type", "transmission"]
    label_encoders = {}
    for col in categorical_features:
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
        label_encoders[col] = le

    features = ["brand", "model", "car_age", "engine_size", "fuel_type", "transmission",
                "mileage", "mileage_per_year", "car_doors", "owner_count"]
    target = "price"

    X = df[features]
    y = df[target]
```

ภาพประกอบ 17 Config Train Model Car Price

ต่อมาแบ่งข้อมูลเป็น Training และ Test Set โดยกำหนดสัดส่วนที่เหมาะสม ฝึกโมเดลด้วยอัลกอริทึม Random Forest Regressor ซึ่งเป็นโมเดลที่ได้รับความนิยมในงานด้านการพยากรณ์ที่มีความแม่นยำสูงดังภาพประกอบ 18

```python
# ✅ แบ่งข้อมูลเป็น Training และ Test Set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# ✅ ทำ Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# ✅ Train โมเดล
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train_scaled, y_train)

# ✅ ทำนายผลบน Training และ Test Set
y_train_pred = model.predict(X_train_scaled)
y_test_pred = model.predict(X_test_scaled)

# ✅ คำนวณค่าประเมินผลลัพธ์
train_mse = mean_squared_error(y_train, y_train_pred)
test_mse = mean_squared_error(y_test, y_test_pred)
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)

# ✅ แสดงผลลัพธ์
logger.info("📊 Model Overfitting Test Results:")
logger.info(f"✅ Training MSE: {train_mse}")
logger.info(f"✅ Test MSE: {test_mse}")
logger.info(f"✅ Training R² Score: {train_r2}")
logger.info(f"✅ Test R² Score: {test_r2}")

print("📊 Model Overfitting Test Results:")
print(f"✅ Training MSE: {train_mse}")
print(f"✅ Test MSE: {test_mse}")
print(f"✅ Training R² Score: {train_r2}")
print(f"✅ Test R² Score: {test_r2}")

joblib.dump(model, "/opt/airflow/models/car_price_model.pkl")
joblib.dump(scaler, "/opt/airflow/models/scaler_car_price.pkl")
joblib.dump(label_encoders, "/opt/airflow/models/label_encoders_car_price.pkl")

logger.info("✅ Model trained, test and saved!")
```

ภาพประกอบ 18 Config Random Forest Car Price

ทำการประเมินประสิทธิภาพของโมเดลด้วยการวัดค่าต่างๆ เช่น MSE และ R^2 Score และบันทึกโมเดลที่ได้ รวมทั้ง scaler และ label encoders เพื่อนำไปใช้งานในขั้นตอนถัดไป

โมเดลพยากรณ์อายุการใช้งานที่เหลือของรถยนต์ (Remaining Useful Life Prediction) ดังภาพประกอบ 19 และ 20

```python
def train_model_car_age():

    client = bigquery.Client(project=project_id)
    query = f"SELECT * FROM `{project_id}.{dataset_id}.{table_id}`"
    df = client.query(query).to_dataframe()
    # ปริ้นข้อมูล DataFrame ที่มีขนาดใหญ่
    logger.info(f"DataFrame head:\n{df.head(100)}")

    # ใช้ transformation functions สำหรับ train_model_car_age (นับใน task 'train_model_car_age')
    df = compute_car_age_train_age(df)
    df = compute_mileage_per_year_train_age(df)
    df = compute_remaining_useful_life_train_age(df)

    models_folder = "/opt/airflow/models"
    if not os.path.exists(models_folder):
        os.makedirs(models_folder)

    # Feature Engineering car_ages
    df["car_age"] = 2025 - df["year"]
    df["mileage_per_year"] = (df["mileage"] / (df["car_age"] + 1)).round(2)
    df["remaining_useful_life"] = 20 - df["car_age"]
    df["remaining_useful_life"] = df["remaining_useful_life"].clip(lower=0)

    categorical_features = ["brand", "model", "fuel_type", "transmission"]
    label_encoders = {}
    for col in categorical_features:
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
        label_encoders[col] = le

    features = ["brand", "car_age", "model", "engine_size",
                "fuel_type", "transmission", "mileage",
                "mileage_per_year", "car_doors", "owner_count"]
    target = "remaining_useful_life"

    X = df[features]
    y = df[target]
```

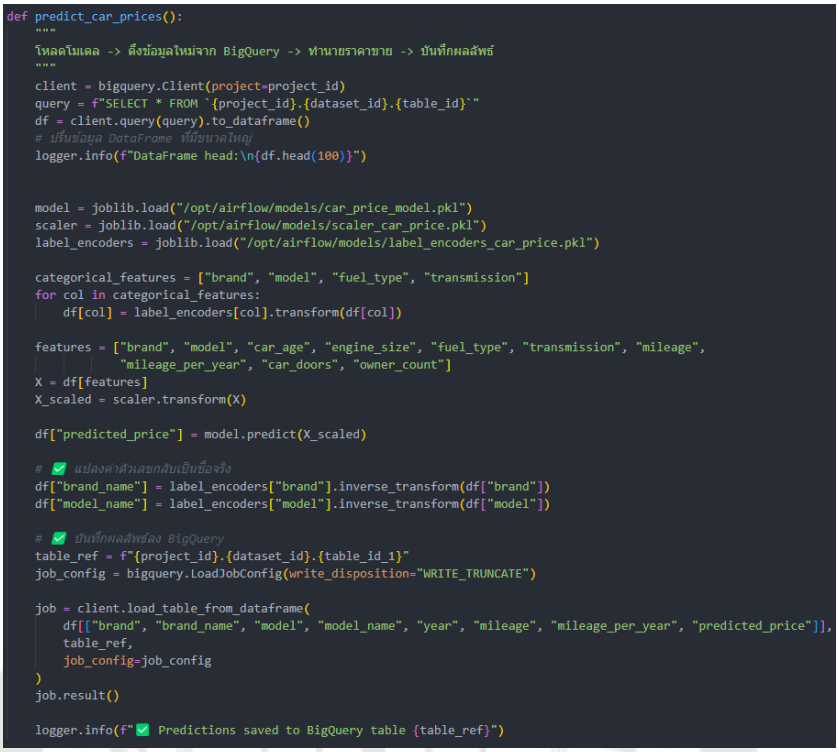
ภาพประกอบ 19 Config Train Model Car Age

ดึงข้อมูลจาก BigQuery และดำเนินการแปลงข้อมูลแบบเดียวกับโมเดลแรก ทำ Label Encoding, Scaling, แบ่งข้อมูล และฝึกโมเดลด้วย Random Forest Regressor เพื่อประเมินประสิทธิภาพของโมเดลผ่านการวัดค่าความผิดพลาดและการทำนายที่ถูกต้อง บันทึกโมเดลไว้เพื่อนำไปใช้งานต่อไป

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

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train_scaled, y_train)

# ✅ ทำนายผลบน Training และ Test Set
y_train_pred = model.predict(X_train_scaled)
y_test_pred = model.predict(X_test_scaled)

# ✅ คำนวณค่าประเมินผลลัพธ์
train_mse = mean_squared_error(y_train, y_train_pred)
test_mse = mean_squared_error(y_test, y_test_pred)
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)

# ✅ แสดงผลลัพธ์
logger.info("📊 Model Overfitting Test Results:")
logger.info(f"✅ Training MSE: {train_mse}")
logger.info(f"✅ Test MSE: {test_mse}")
logger.info(f"✅ Training R² Score: {train_r2}")
logger.info(f"✅ Test R² Score: {test_r2}")

print("📊 Model Overfitting Test Results:")
print(f"✅ Training MSE: {train_mse}")
print(f"✅ Test MSE: {test_mse}")
print(f"✅ Training R² Score: {train_r2}")
print(f"✅ Test R² Score: {test_r2}")


joblib.dump(model, "/opt/airflow/models/car_ages_model.pkl")
joblib.dump(scaler, "/opt/airflow/models/scaler_car_age.pkl")
joblib.dump(label_encoders, "/opt/airflow/models/label_encoders_car_age.pkl")

logger.info("✅ car_ages Model trained and saved!")
```

ภาพประกอบ 20 Config Random Forest Car Age

ทำนายราคาขายรถยนต์ โหลดโมเดลและข้อมูลใหม่จาก BigQuery ทำการ Label Encoding และ Scaling ตามที่เคยฝึกไว้ ทำนายราคาด้วยโมเดลและนำผลลัพธ์ที่ได้ไปจัดเก็บกลับไว้ที่ Google BigQuery



```python
def predict_car_prices():
    """
    โหลดโมเดล -> ดึงข้อมูลใหม่จาก BigQuery -> ทำนายราคาขาย -> บันทึกผลลัพธ์
    """
    client = bigquery.Client(project=project_id)
    query = f"SELECT * FROM `{project_id}.{dataset_id}.{table_id}`"
    df = client.query(query).to_dataframe()
    # ปริ้นข้อมูล DataFrame ที่มีขนาดใหญ่
    logger.info(f"DataFrame head:\n{df.head(100)}")


    model = joblib.load("/opt/airflow/models/car_price_model.pkl")
    scaler = joblib.load("/opt/airflow/models/scaler_car_price.pkl")
    label_encoders = joblib.load("/opt/airflow/models/label_encoders_car_price.pkl")

    categorical_features = ["brand", "model", "fuel_type", "transmission"]
    for col in categorical_features:
        df[col] = label_encoders[col].transform(df[col])

    features = ["brand", "model", "car_age", "engine_size", "fuel_type", "transmission", "mileage",
                "mileage_per_year", "car_doors", "owner_count"]
    X = df[features]
    X_scaled = scaler.transform(X)

    df["predicted_price"] = model.predict(X_scaled)

    # ✅ แปลงค่าตัวเลขกลับเป็นชื่อจริง
    df["brand_name"] = label_encoders["brand"].inverse_transform(df["brand"])
    df["model_name"] = label_encoders["model"].inverse_transform(df["model"])

    # ✅ บันทึกผลลัพธ์ลง BigQuery
    table_ref = f"{project_id}.{dataset_id}.{table_id_1}"
    job_config = bigquery.LoadJobConfig(write_disposition="WRITE_TRUNCATE")

    job = client.load_table_from_dataframe(
        df[["brand", "brand_name", "model", "model_name", "year", "mileage", "mileage_per_year", "predicted_price"]],
        table_ref,
        job_config=job_config
    )
    job.result()

    logger.info(f"✅ Predictions saved to BigQuery table {table_ref}")
```

ภาพประกอบ 21 Config Prediction Car Price

ทำนายอายุการใช้งานที่เหลือของรถยนต์ โหลดโมเดลที่ฝึกไว้และดำเนินการแปลงข้อมูลเพื่อการทำนาย ทำนายอายุการใช้งานที่เหลือ และนำผลลัพธ์ไปจัดเก็บไว้ที่ Google BigQuery

```python
def predict_car_age():

    models_folder = "/opt/airflow/models"
    client = bigquery.Client(project=project_id)
    query = f"SELECT * FROM `{project_id}.{dataset_id}.{table_id}`"
    df = client.query(query).to_dataframe()
    # ปริ้นข้อมูล DataFrame ที่มีขนาดใหญ่
    logger.info(f"DataFrame head:\n{df.head(100)}")


    # ตรวจสอบคอลัมน์ใน DataFrame
    print("Columns in DataFrame:", df.columns.tolist())

    # โหลดโมเดลที่ Train แล้ว
    model = joblib.load(os.path.join(models_folder, "car_ages_model.pkl"))
    scaler = joblib.load(os.path.join(models_folder, "scaler_car_age.pkl"))
    label_encoders = joblib.load(os.path.join(models_folder, "label_encoders_car_age.pkl"))

    categorical_features = ["brand", "model", "fuel_type", "transmission"]
    for col in categorical_features:
        df[col] = label_encoders[col].transform(df[col])

    features = ["brand", "car_age", "model", "engine_size",
                "fuel_type", "transmission", "mileage",
                "mileage_per_year", "car_doors", "owner_count"]
    X = df[features]
    print(f"%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% dataFrame (top 5 rows):\n{X.head()}")
    X_scaled = scaler.transform(X)

    df["predicted_car_age"] = model.predict(X_scaled)

    # ✅ แปลงค่าตัวเลขกลับเป็นชื่อจริง
    df["brand_name"] = label_encoders["brand"].inverse_transform(df["brand"])
    df["model_name"] = label_encoders["model"].inverse_transform(df["model"])

    # บันทึกผลลัพธ์ลง BigQuery
    table_ref = f"{project_id}.{dataset_id}.{table_id_2}"
    job_config = bigquery.LoadJobConfig(write_disposition="WRITE_TRUNCATE")

    job = client.load_table_from_dataframe(
        df[["brand", "brand_name", "model", "model_name", "year", "mileage", "predicted_car_age"]],
        table_ref,
        job_config=job_config
    )
    job.result()

    logger.info(f"✅ RUL Predictions saved to BigQuery table {table_ref}")
```

ภาพประกอบ 22 Config Prediction Car Age

กระบวนการทั้งหมดนี้จะถูกจัดการอย่างเป็นระบบและต่อเนื่องกันผ่าน Apache Airflow ซึ่งใช้ Directed Acyclic Graph (DAG) ในการกำหนดลำดับขั้นตอนการดำเนินงาน เพื่อให้การวิจัยและพัฒนา Data Pipeline แบบ Non-Feast มีประสิทธิภาพและชัดเจนสำหรับการนำไปเปรียบเทียบกับ FEAST Feature Store ดังภาพประกอบ 22

3.2.4 การประเมินผลและวิเคราะห์ข้อมูล

ประเมินเวลาที่ใช้ในการดำเนินการแต่ละขั้นตอนใน Pipeline เช่น การโหลดข้อมูล (Data Ingestion), การแปลงข้อมูล (Feature Engineering) [10] และการสร้างชุดข้อมูลสำหรับการฝึก

โมเดล (Training Data Generation) วิเคราะห์กระบวนการในการสร้างและจัดเก็บคุณลักษณะ (Features) โดยเปรียบเทียบระหว่างการใช้ FEAST กับการจัดการคุณลักษณะแบบดั้งเดิมสามารถอธิบายได้ดังนี้

FEAST เป็นระบบที่ถูกออกแบบมาเพื่อช่วยบริหารจัดการฟีเจอร์อย่างเป็นระบบ สามารถจัดการเวอร์ชันของข้อมูลได้โดยอัตโนมัติ และป้องกันการรั่วไหลของข้อมูล (Data Leakage) ด้วยคุณสมบัติการทำ Point-in-Time Joins [17]ในขั้นตอนนี้ได้มีการเขียน Feature Definition และกำหนด Entity ไว้ที่ไฟล์ feature_view_car_price.py และแนะนำให้ใช้คำสั่ง fs.apply() ในการลงทะเบียนฟีเจอร์ลงใน Feast โดยการทำงานนี้จะช่วยให้ทีม Data Scientist สามารถเรียกดูและใช้งานฟีเจอร์ย้อนหลังได้ง่าย และลดความผิดพลาดในการเรียกข้อมูลเวอร์ชันต่าง ๆ อย่างเป็นระบบ หลังจากลงทะเบียนฟีเจอร์ใน Feast แล้ว จึงทำการดึงข้อมูลจาก Feature Store มายังสภาพแวดล้อม Python เพื่อทำการฝึกสอนโมเดล โดยใช้โมเดลประเภท Random Forest Regression สำหรับการพยากรณ์ราคาขายรถยนต์ และโมเดล Random Forest Regression เช่นเดียวกันในการทำนายอายุการใช้งานที่เหลือของรถยนต์ โดยข้อมูลที่นำมานั้นได้ผ่านขั้นตอนการแปลงข้อมูลเพิ่มเติม เช่น การ Encoding

ข้อมูลประเภท Category ด้วย LabelEncoder และ Scaling ข้อมูลด้วย StandardScaler เพื่อให้ข้อมูลอยู่ในมาตรฐานเดียวกัน หลังจากโมเดลถูกสร้างและประเมินประสิทธิภาพแล้ว จะถูกบันทึกเป็นไฟล์โมเดล (pickle) เพื่อรอใช้งานในขั้นตอนถัดไป และขั้นตอนสุดท้ายของกระบวนการนี้คือการนำโมเดลที่ผ่านการฝึกสอนไปใช้งานจริง (Prediction) โดยจะมีการดึงข้อมูลฟีเจอร์ล่าสุดจาก BigQuery ผ่าน Feast จากนั้นทำการแปลงและปรับแต่งข้อมูลให้สอดคล้องกับรูปแบบของฟีเจอร์ที่โมเดลถูกฝึกไว้ เช่น การเข้ารหัส (Encoding) การปรับสเกลข้อมูล (Scaling) แล้วจึงทำการพยากรณ์ผลลัพธ์ซึ่งอาจเป็นราคาขายที่คาดการณ์ได้ (predicted_price) หรืออายุการใช้งานรถที่เหลือ (predicted_car_age) และผลลัพธ์ที่ได้จะถูกนำกลับไปจัดเก็บไว้ในตารางผลลัพธ์ที่ Google BigQuery อีกครั้ง เพื่อให้สามารถนำไปใช้งานหรืออ้างอิงในขั้นตอนต่อไป

จากขั้นตอนที่กล่าวมาข้างต้น พบว่าการนำ Feast มาใช้งานร่วมกับ Airflow ช่วยให้ผู้วิจัยสามารถจัดการฟีเจอร์ได้อย่างเป็นระบบ โดยมีการติดตามและบริหารเวอร์ชันของฟีเจอร์โดยอัตโนมัติ สามารถป้องกันการรั่วไหลของข้อมูล (Data Leakage) ผ่านการทำ Point-in-Time Join อย่างมีประสิทธิภาพ ลดความซับซ้อนในการจัดการข้อมูล และเพิ่มความรวดเร็วในการพัฒนาและการนำไปใช้จริง ข้อดีเหล่านี้แตกต่างจากการจัดการข้อมูลแบบ Non-Feast

อย่างมีนัยสำคัญ ซึ่งต้องจัดการเวอร์ชันฟีเจอร์แบบ manual และขาดระบบควบคุมที่ชัดเจน ส่งผลให้กระบวนการจัดการข้อมูลอาจมีความผิดพลาดและซับซ้อนมากขึ้นในระยะยาว

NON-FEAST โครงสร้าง Pipeline ใช้งาน Airflow เพื่อกำหนดลำดับการทำงานอัตโนมัติ ตั้งแต่การแปลงข้อมูล (feature engineering) ไปจนถึงการเทรนโมเดลและทำนายผล การจัดการเวอร์ชันข้อมูล ใช้วิธี Manual ผ่านการตั้งชื่อไฟล์และการเก็บรุ่นโมเดลด้วย joblib ในโฟลเดอร์ ในขณะที่ Feast จะจัดการเวอร์ชันฟีเจอร์และเวลา (timestamp) ให้โดยอัตโนมัติ

Feature Engineering เขียนสคริปต์ Python ทั้งหมด (เช่น โค้ดfeature_engineering() train_model()) ทำให้อิสระต่อการปรับเปลี่ยน แต่ยากในการควบคุมมาตรฐานในทีมใหญ่

Serving กระบวนการทำนาย (prediction) อาศัยโมเดลที่เก็บแบบไฟล์ pkl ไม่ได้มี Online Store หรือระบบบริการฟีเจอร์แบบเรียลไทม์ ต้องพึ่งสคริปต์ predict_car_prices() หรือ predict_car_age() เพื่ออัปโหลดผลลัพธ์กลับสู่ Google BigQuery

ดังนั้นในการวิจัยครั้งนี้ ส่วน Non-Feast จึงเป็นตัวแทนของกระบวนการดั้งเดิม (traditional pipeline) ที่ทีม Data Scientist และ Data Engineer สามารถทำเองได้ด้วยการใช้เครื่องมือ Open Source ทั่วไป (Airflow + Python + BigQuery) แต่ขาดความสะดวกในด้านการจัดการเวอร์ชันและกระบวนการ Point-in-Time Join ซึ่งเป็นจุดเด่นของ Feast ในการป้องกัน Data Leakage และจัดระเบียบข้อมูลฟีเจอร์สำหรับโมเดลได้อย่างเป็นระบบมากกว่า

ผลการเปรียบเทียบที่ได้นำเสนอในบทถัดไป จะช่วยตอกย้ำข้อดีและข้อจำกัดของแนวทาง Non-Feast เมื่อเทียบกับ Feast ทั้งในด้านประสิทธิภาพ ความง่ายในการพัฒนา และการบำรุงรักษาระบบในระยะยาวต่อไป

3.3 การรวบรวมข้อมูล

การดึงข้อมูลจากแหล่งข้อมูลเปิด (Open Data Sources) ในขั้นตอนนี้จะทำการดึงข้อมูลราคารถยนต์ (Car Price) จาก Kaggle ซึ่งเป็นแหล่งข้อมูลเปิดที่ได้รับความนิยมและมีข้อมูลที่หลากหลาย สามารถดาวน์โหลดข้อมูลได้ในรูปแบบไฟล์ CSV เพื่อใช้ในการวิเคราะห์ความเหมาะสมของ Pipeline

ขั้นตอนการรวบรวมข้อมูล การดึงข้อมูลจาก Kaggle ข้อมูล Car Price จาก Kaggle จะถูกดาวน์โหลดผ่านเว็บ Kaggle หรือผ่าน Kaggle API (หากต้องการอัตโนมัติ) ในรูปแบบ CSV โดยอาจตั้งค่ากระบวนการให้อัปเดตข้อมูลสม่ำเสมอตามระยะเวลาที่ต้องการ (เช่น รายวัน หรือรายชั่วโมง) เพื่อให้ข้อมูลมีความทันสมัย การจัดเก็บข้อมูลใน Data Lake หลังจากดาวน์โหลดข้อมูล CSV เรียบร้อยแล้ว ไฟล์ CSV จะถูกนำไปจัดเก็บใน Data Lake บน Google Cloud

Storage (GCS) ซึ่งสามารถปรับขยาย (Scalability) ได้ดีและรองรับปริมาณข้อมูลขนาดใหญ่มีการกำหนด

โครงสร้างการจัดเก็บตามวันที่และเวลาที่เก็บข้อมูล เพื่อให้ง่ายต่อการค้นหาและประมวลผลในภายหลังการส่งข้อมูลไปยัง Google BigQuery เมื่อข้อมูลถูกจัดเก็บใน GCS ระบบจะทำการประมวลผลและส่งข้อมูลเข้าสู่ Google BigQuery ซึ่งเป็น Data Warehouse บนคลาวด์ข้อมูลใน BigQuery จะถูกเตรียมไว้สำหรับการวิเคราะห์เชิงลึก เช่น การสร้างรายงานและการพัฒนาโมเดล Machine Learning เพื่อทำนายราคาหรือปัจจัยอื่น ๆ ที่เกี่ยวข้องกับตลาดรถยนต์การตรวจสอบและปรับปรุงคุณภาพข้อมูล

ก่อนการนำข้อมูลเข้าสู่ Google BigQuery จะมีการตรวจสอบความสมบูรณ์ของข้อมูล (Data Integrity) เช่น การตรวจสอบค่าที่หายไป (Missing Values) และการปรับโครงสร้างข้อมูลให้อยู่ในรูปแบบมาตรฐานเดียวกัน มีการบันทึก Metadata เพื่อรองรับการบริหารจัดการข้อมูลในระยะยาวและช่วยในการติดตามการเปลี่ยนแปลงของข้อมูลแต่ละเวอร์ชันได้อย่างมีประสิทธิภาพ

3.4 เครื่องมือที่ใช้ในการวิจัย

3.4.1 FEAST

ใช้จัดเก็บข้อมูลคุณลักษณะในที่นี้จะใช้แบบ Offline Store บทบาทของ FEAST ในงานวิจัยนี้คือช่วยลดความซับซ้อนในการจัดการคุณลักษณะ และเพิ่มความแม่นยำในการดึงข้อมูลคุณลักษณะมาใช้ในกระบวนการฝึกสอนโมเดล (Training) และการนำไปใช้งาน (Inference) นอกจากนี้ FEAST ยังช่วยบันทึก Metadata ที่เกี่ยวข้องกับคุณลักษณะเพื่อให้สามารถติดตามและจัดการได้ง่ายในระยะยาว[17]

3.4.2 Apache Airflow

เป็นเครื่องมือที่ใช้สำหรับจัดการ Workflow และควบคุมกระบวนการทำงานในรูปแบบของ Directed Acyclic Graphs (DAGs) ซึ่งเป็นโครงสร้างที่แสดงลำดับการทำงานของ Task อย่างชัดเจนในงานวิจัยนี้ Airflow ถูกนำมาใช้ในหลายด้าน เช่น:

การตั้งเวลาการดึงข้อมูล (Scheduling Data Extraction) จากแหล่งข้อมูลต้นทาง เช่น car price

การจัดการกระบวนการ ETL (Extract, Transform, Load) เพื่อเตรียมข้อมูลให้พร้อมสำหรับการจัดเก็บใน Data Lake และ Data Warehouse[18]

การควบคุมการทำงานของโมดูล FEAST เพื่อดึงข้อมูลคุณลักษณะเข้าสู่ระบบ

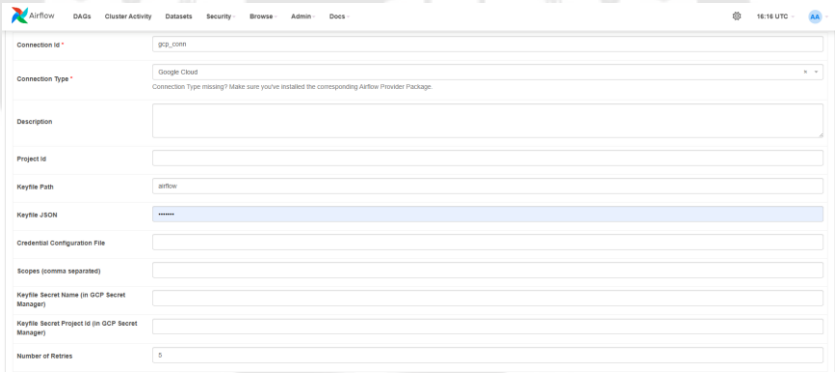
การตรวจสอบและบันทึกผลการทำงาน (Logging and Monitoring) เพื่อลดข้อผิดพลาดในกระบวนการ Pipeline

3.4.3 Google Cloud Platform (GCP)

เป็นโครงสร้างพื้นฐานคลาวด์ที่ถูกเลือกใช้เป็นแพลตฟอร์มหลักสำหรับจัดเก็บและประมวลผลข้อมูลในงานวิจัยนี้ โดยบริการสำคัญที่ใช้ ได้แก่:

Google BigQuery: ทำหน้าที่เป็น Offline Store สำหรับเก็บข้อมูลคุณลักษณะในรูปแบบตาราง (Tabular Data) ที่สามารถเรียกใช้งานได้ง่ายด้วย SQL Query

ข้อได้เปรียบของ GCP คือความสามารถในการปรับขยาย (Scalability) รองรับข้อมูลปริมาณมาก และความยืดหยุ่นในการเชื่อมต่อกับเครื่องมือภายนอก[17, 18] เช่น Apache Airflow และ FEAST โดยขั้นตอนการทำงานในการเชื่อมต่อกับ apache airflow ดังภาพประกอบ 23



ภาพประกอบ 23 Google Cloud Platform Connect to Apache Airflow

สร้าง Service Account บน Google Cloud Platform (GCP) เพื่อเชื่อมต่อกับ Apache Airflow อย่างเป็นระบบ โดยมีการกำหนดเครื่องมือ อุปกรณ์ และวิธีการที่ใช้ ตลอดจนแนวทางการทดสอบการทำงาน เพื่อให้มั่นใจว่า Apache Airflow สามารถเข้าถึงทรัพยากรบน GCP ได้อย่างถูกต้อง

3.4.4 Python และ Jupyter Notebook

เป็นภาษาโปรแกรมหลักที่ใช้พัฒนากระบวนการต่างๆ ใน Pipeline โดยใช้ร่วมกับไลบรารี [19] ที่เกี่ยวข้อง เช่น Pandas, NumPy, และ scikit-learn การเขียนสคริปต์สำหรับดึงและแปลงข้อมูลที่ใช้ในงาน Data Processing (RandomForestRegressor RandomForestClassifier) Feature Engineering, รวมถึงบันทึกโมเดลด้วย joblib การพัฒนาและทดสอบโมเดล Machine Learning การทดลองใช้ FEAST และตรวจสอบการทำงานของข้อมูลคุณลักษณะ ความยืดหยุ่นและความง่ายในการพัฒนาของ Python และ Jupyter Notebook ทำให้สามารถทดลองและปรับแต่งระบบได้อย่างรวดเร็ว ซึ่งช่วยลดเวลาในการพัฒนาและเพิ่มประสิทธิภาพในการแก้ไขข้อผิดพลาด

3.4.5 Docker

ใช้ในการรัน Airflow และ Python environment เพื่อควบคุมเวอร์ชันของไลบรารีให้คงที่ไม่เกิดปัญหา dependency conflicts

3.5 การวัดผลและการประเมิน

3.5.1 ความสะดวกในการนำคุณลักษณะกลับมาใช้ซ้ำ (Feature Reusability)

เพื่อประเมินความสามารถของระบบในการนำคุณลักษณะ (features) ที่เคยถูกสร้างขึ้นแล้วกลับมาใช้งานซ้ำได้อย่างสะดวกและมีประสิทธิภาพ ผู้วิจัยได้ออกแบบตัวแปร feature_reuse_counts ซึ่งเป็น Global Dictionary สำหรับใช้ในการติดตามจำนวนครั้งที่เรียกใช้คุณลักษณะแต่ละตัวในระบบที่พัฒนาโดยใช้ FEAST Feature Store ดังภาพประกอบ 24

```python
feature_counter = {}

REUSE_LOG_PATH = "/opt/airflow/logs/feature_reuse_counts.csv"

# กำหนด global dictionary สำหรับนับ reuse ของฟีเจอร์
feature_reuse_counts = {"brand": 0, "model": 0, "year": 0, "engine_size": 0,
                        "fuel_type": 0, "transmission": 0, "mileage": 0,
                        "car_doors": 0, "owner_count": 0, "price": 0, "price_per_mile": 0}
```

ภาพประกอบ 24 Function Feature Reuse Count

ภายในฟังก์ชันแปลงข้อมูล (เช่น การแปลงค่าประเภทข้อมูลแบบ Category เป็น Label Encoding) ผู้วิจัยได้วางกลไกนับจำนวนการใช้ฟีเจอร์ซ้ำ โดยใช้การวนลูปฟีเจอร์ที่กำหนดไว้ล่วงหน้า และทำการบันทึกค่าลงใน feature_reuse_counts[col] เมื่อพบว่ามีการเรียกใช้คุณลักษณะนั้นใน Task หนึ่ง ๆ ซึ่งหมายถึงการนำฟีเจอร์กลับมาใช้โดยไม่ต้องสร้างใหม่

```python
# Label Encoding
label_encoders = {}
for col in ["brand", "model", "year", "fuel_type", "engine_size",
            "transmission", "mileage", "car_doors", "owner_count", "price", "price_per_mile"]:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
    # เพิ่มการนับการ reuse สำหรับฟีเจอร์นี้ (ในที่นี้ถือว่า task นี้ใช้งานฟีเจอร์ 1 ครั้ง)
    feature_reuse_counts[col] += 1

# ✅ สะสมลงไฟล์
update_feature_reuse_counts(feature_reuse_counts)
```

ภาพประกอบ 25 Config Feature Reused Counts

การนับดังกล่าวครอบคลุมฟีเจอร์ที่สำคัญในระบบทั้งหมด เช่น brand, model, year, engine_size, mileage, fuel_type, transmission, car_doors, owner_count, price และ price_per_mile ซึ่งล้วนเป็นคุณลักษณะพื้นฐานที่ใช้ในการฝึกหรือทำนายโมเดลหลายแบบ การที่ระบบสามารถเรียกใช้คุณลักษณะเหล่านี้ซ้ำได้ในหลาย DAG หรือหลายรอบการทำงาน

แสดงให้เห็นถึงความสามารถของ Feature Store ในการสนับสนุนการ reuse ได้อย่างมีประสิทธิภาพ

หลังจากที่ตัวนับการใช้งาน (feature_reuse_counts) ได้รับการอัปเดตตามการรันของ pipeline แล้ว ข้อมูลเหล่านี้จะถูกบันทึกลงไปไว้ในไฟล์ CSV โดยที่ใช้ฟังก์ชัน update_feature_reuse_counts() เพื่อใช้เป็นข้อมูลในการประเมินผลเชิงปริมาณในบทที่ 4 ต่อไป ซึ่งมีเกณฑ์ในการประเมินความสามารถในการ reuse ฟีเจอร์จะประกอบด้วย จำนวนครั้งที่มีการเรียกใช้แต่ละฟีเจอร์ซ้ำ

3.5.2 **ลดการใช้งานคุณลักษณะวิศวกรรม** (Feature Engineering)

การประเมินความสามารถของระบบ FEAST ในการลดการทำงานซ้ำซ้อนของ Feature Engineering จะวัดจาก “จำนวนครั้งที่มีการรันฟังก์ชัน Feature Engineering” ในแต่ละ DAG โดยเฉพาะอย่างยิ่งในขั้นตอนการเตรียมข้อมูลก่อนการฝึกหรือทำนายโมเดล

ผู้วิจัยได้ออกแบบตัวเก็บข้อมูลการรันของ Task feature_engineering() เพื่อติดตามว่าในแต่ละ DAG มีการเรียกใช้ฟังก์ชันดังกล่าวกี่ครั้ง และนำข้อมูลมาเปรียบเทียบระหว่างระบบที่ใช้ FEAST กับระบบ Non-FEAST ซึ่งจะสะท้อนให้เห็นว่าระบบใดสามารถลดการทำงานในขั้นตอน Feature Engineering ดังภาพประกอบ 26

จำนวนการรันฟังก์ชัน Feature Engineering (ครั้ง) ต่อ DAG

```python
feature_counter = {}

def count_feature_step(dag_name, task_name):
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            key = (dag_name, task_name)
            feature_counter[key] = feature_counter.get(key, 0) + 1
            logger.info(f"DAG '{dag_name}', Task '{task_name}': Function '{func.__name__}' called {feature_counter[key]} times")

            # เรียกฟังก์ชันจริง
            result = func(*args, **kwargs)

            csv_path = "/opt/airflow/logs/feature_counters.csv"
```

ภาพประกอบ 26 Function Feature Count Step

ผู้วิจัยได้ออกแบบกลไกสำหรับติดตามและนับจำนวนการเรียกใช้งานฟังก์ชันที่เกี่ยวข้องกับการทำ Feature Engineering ภายในระบบ Feast Data Pipeline และ Non-Feast Data Pipeline โดยใช้ Python Decorator เพื่อเก็บสถิติการใช้งานของแต่ละ Task และ DAG ที่ระบุ

โดย Decorator นี้ทำหน้าที่นับจำนวนครั้งที่ฟังก์ชันถูกเรียกใช้ และบันทึกผลลัพธ์ลงในตัวแปร feature_counter ซึ่งถูกกำหนดให้เป็น dictionary สำหรับเก็บคู่ค่า (dag_name, task_name) กับจำนวนครั้งที่ถูกเรียก

Decorator ถูกนำไปใช้กับฟังก์ชัน Feature Engineering หลายรายการ เพื่อเพิ่มประสิทธิภาพในการนับจำนวนการใช้งานคุณลักษณะทางวิศวกรรม เช่น compute_car_age, compute_mileage_per_year, compute_remaining_useful_life และได้ encode_categorical โดยทุกฟังก์ชันทุกตัวเหล่านี้จะอยู่ในภายใต้กระบวนการขั้นตอน data pipeline (DAG) feast_transform_load_car_price_data,non_feast_transform_load_car_price_data Task ชื่อ feature_engineering ซึ่งสะท้อนถึงการทำงานทั้ง 2 แบบ โดยแบบที่ใช่ Feast จะทำในส่วนของ Feature Engineer ทั้งหมด ดังภาพประกอบ 27

```python
@count_feature_step("feast_transform_and_load_car_price_data", "feature_engineering")
def compute_car_age(df):
    df["car_age"] = 2025 - df["year"]
    return df

@count_feature_step("feast_transform_and_load_car_price_data", "feature_engineering")
def compute_mileage_per_year(df):
    df["mileage_per_year"] = (df["mileage"] / (df["car_age"] + 1)).round(2)
    return df

@count_feature_step("feast_transform_and_load_car_price_data", "feature_engineering")
def compute_remaining_useful_life(df):
    df["remaining_useful_life"] = 20 - df["car_age"]
    df["remaining_useful_life"] = df["remaining_useful_life"].clip(lower=0)
    return df

@count_feature_step("feast_transform_and_load_car_price_data", "feature_engineering")
def encode_categorical(df, *cols):
    # หากมีการส่งค่าเพียงหนึ่งอันและเป็น tuple หรือ list ให้ unpack ค่าออกมาเก็บในตัวแปรใหม่
    if len(cols) == 1 and isinstance(cols[0], (tuple, list)):
        col_list = cols[0]
    else:
        col_list = cols
    from sklearn.preprocessing import LabelEncoder
    for col in col_list:
        if col not in df.columns:
            logger.error(f"Column {col} not found in dataframe")
            continue
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
    return df
```

ภาพประกอบ 27 Config Count Feature Step

แต่ในส่วนของที่ไม่มีการใช้ Feature Store (Non-Feast) จะทำในขั้นตอนที่มีการ Transformation ข้อมูล ดังนั้นผลที่ได้การใช้ Feast มีการทำขั้นตอนการทำ Feature Engineer ทั้งหมด 4 ครั้ง ส่วนที่ไม่ใช้ NON Feast มีการทำขั้นตอนการทำ Feature Engineer ทั้งหมด 9 ครั้ง

ในบทนี้ ผู้วิจัยได้อธิบายกระบวนการดำเนินงานวิจัยทั้งหมด ตั้งแต่การวางแผนการดำเนินงาน การออกแบบระบบและเวิร์กโฟลว์ของ Data Pipeline ทั้งในรูปแบบที่ใช้ FEAST Feature Store และแบบดั้งเดิม (Non-Feast) รวมถึงการพัฒนาและทดสอบระบบในแต่ละขั้นตอน การรวบรวมข้อมูลจากแหล่งข้อมูลเปิด การเลือกใช้เครื่องมือและเทคโนโลยีที่เกี่ยวข้อง ตลอดจนการกำหนดเกณฑ์และวิธีการวัดผลเพื่อใช้ในการประเมินประสิทธิภาพของระบบ ในบทถัดไป ผู้อ่านจะได้พบกับการแสดงผลการออกแบบ Workflow และพัฒนาระบบ Data Pipeline ผลการทดลองที่แสดงการเปรียบเทียบเชิงประสิทธิภาพระหว่างระบบที่ใช้ FEAST กับระบบ Non-Feast ทั้งในด้านความสามารถในการ reuse ฟีเจอร์ และลดความซับซ้อนของขั้นตอนการทำ Feature Engineering เพื่อสะท้อนให้เห็นถึงข้อดีและข้อจำกัดของแต่ละแนวทางอย่างเป็นรูปธรรม

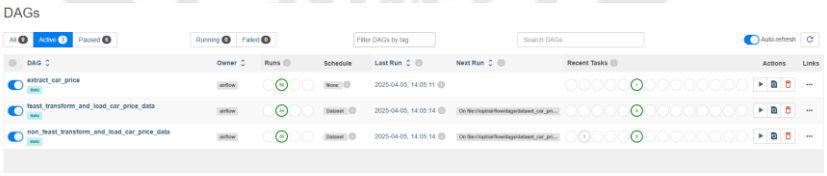
บทที่ 4
ผลการวิจัยและการวิเคราะห์ข้อมูล

4.1 ผลการพัฒนาระบบ Data Pipeline

ในส่วนนี้จะแสดงผลการออกแบบและพัฒนาระบบ Data Pipeline โดยการใช้ Apache Airflow และ FEAST Feature Store เข้าด้วยกันอย่างเป็นระบบเพื่อรองรับการประมวลผลข้อมูลในรูปแบบ Batch แบบอัตโนมัติผ่าน DAGs (Directed Acyclic Graphs) ซึ่งมีขั้นตอนดังนี้

4.1.1 การออกแบบ Workflow และ DAGs

DAGs (Directed Acyclic Graphs) ถูกออกแบบมาเพื่อให้การประมวลผลข้อมูลดำเนินไปอย่างมีลำดับ โดยมีการกำหนด Task หลัก ได้แก่ การดึงข้อมูล การแปลงข้อมูล และการโหลดข้อมูลเข้าสู่ Feature Store เพื่อปรับเปลี่ยนรูปแบบข้อมูลให้พร้อมสำหรับการสร้างคุณลักษณะ (Feature Engineering) จากนั้นจึงมีขั้นตอนการโหลดข้อมูลเข้าสู่ Feature Store โดยใช้ FEAST เป็นตัวกลางในการจัดเก็บและให้บริการข้อมูลคุณลักษณะในรูปแบบที่เป็นมาตรฐานและสามารถจัดการเวอร์ชันได้อย่างมีประสิทธิภาพ กระบวนการดังกล่าวถูกออกแบบให้สามารถทำงานแบบอัตโนมัติ โดยที่ dataset ที่ถูกจัดเก็บใน Feature Store โดยการรันแบบใช้ dataset เป็นตัว trigger ไปยัง DAGs ต่อไป ดังภาพประกอบ 28



ภาพประกอบ 28 Scheduler Dataset

Apache Airflow ช่วยให้ผู้ใช้งานสามารถกำหนดการทำงานของระบบ Workflow ได้อย่างยืดหยุ่นและปรับแต่งได้ตามความต้องการของแต่ละกระบวนการงาน เนื่องจาก Airflow ใช้โครงสร้างแบบ Directed Acyclic Graphs (DAGs) ทำให้การระบุลำดับขั้นตอนของงานแต่ละขั้นเป็นไปอย่างชัดเจนและสามารถปรับเปลี่ยนได้ง่ายตามความซับซ้อนของกระบวนการ

นอกจากนี้ Airflow ยังรองรับการเชื่อมต่อกับแหล่งข้อมูลหลากหลายรูปแบบ ทั้งการดึงข้อมูลจากฐานข้อมูลเชิงสัมพันธ์ การเชื่อมต่อกับ API ของแหล่งข้อมูลภายนอก และการรับข้อมูลจากไฟล์ในรูปแบบต่าง ๆ เช่น CSV, JSON หรือข้อมูลบนคลาวด์

4.1.2 การพัฒนา Feature Store ด้วย FEAST

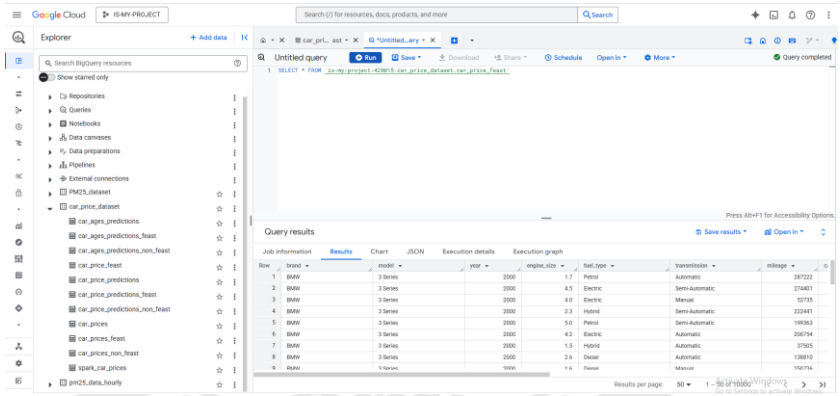
```python
car_entity = Entity(
    name="car_id",
    value_type=ValueType.INT64,
    description="Brand of the car (used as entity key).",
)

# ---------------------------------------------
# Feature View: ระบุ Fields (Features) ที่เราต้องการ
# ---------------------------------------------
car_price_feature_view = FeatureView(
    name="car_price_feature_view",
    entities=[car_entity],
    schema=[
        Field(name="car_id", dtype=Int64),
        Field(name="brand", dtype=String),
        Field(name="model", dtype=String),
        Field(name="year", dtype=Int64),
        Field(name="engine_size", dtype=Float32),
        Field(name="fuel_type", dtype=String),
        Field(name="transmission", dtype=String),
        Field(name="mileage", dtype=Int64),
        Field(name="car_doors", dtype=Int64),
        Field(name="owner_count", dtype=Int64),
        Field(name="price", dtype=Int64),
        Field(name="price_per_mile", dtype=Float32),
        Field(name="car_age", dtype=Float32),
        Field(name="mileage_per_year", dtype=Float32),
        Field(name="remaining_useful_life", dtype=Float32),
    ],
    source=car_price_source,
    online=False,  # ตั้งค่าเป็น offline-only
)
```

ภาพประกอบ 29 Registry FEAST Feature Store

ใช้ FEAST เพื่อจัดการคุณลักษณะ (features) ในกระบวนการ Machine Learning ซึ่งช่วยลดความซับซ้อนในการจัดการข้อมูล

การนำ Feature Store มาใช้ช่วยเพิ่มความสะดวกในการเรียกใช้งานคุณลักษณะซ้ำ (Feature Reusability) และลดเวลาในการทำ Feature Engineering ดังภาพประกอบ 29

4.1.3 การเชื่อมต่อกับ Google Cloud Platform



ภาพประกอบ 30 Data Warehouse Offline Store

ระบบถูกตั้งค่าบน Google Cloud Platform (GCP) โดยใช้บริการ Google BigQuery สำหรับเก็บข้อมูลในรูปแบบ Offline Store และใช้ Cloud Functions เพื่อการประมวลผลที่รวดเร็ว

การใช้งาน GCP ช่วยเสริมความสามารถในการประมวลผลข้อมูลขนาดใหญ่ และเพิ่มความยืดหยุ่นของระบบ ดังภาพประกอบ 30

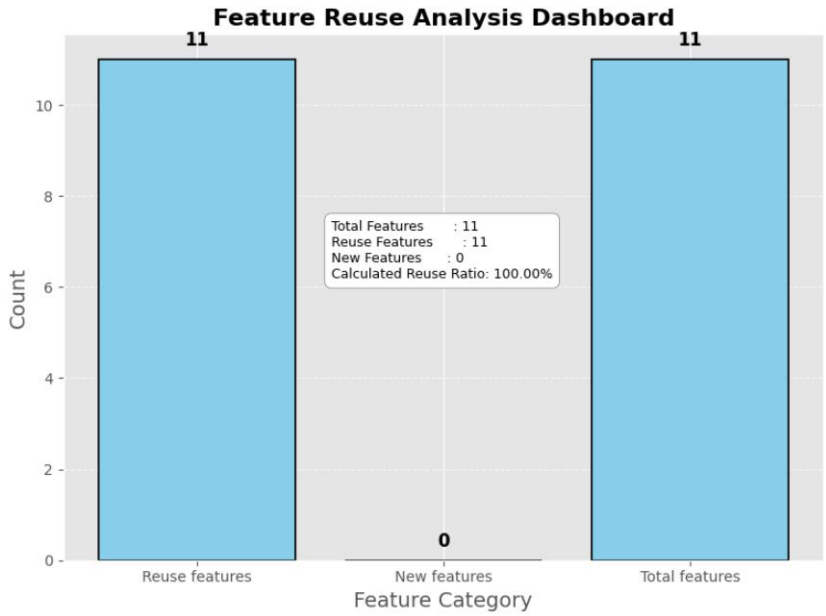
4.2 การวิเคราะห์ผลการทดลอง

4.2.1 การประเมินผลด้านการใช้งานคุณลักษณะกลับมาใช้ใหม่

ระบบที่พัฒนาสามารถนำคุณลักษณะกลับมาใช้ซ้ำได้อย่างมีประสิทธิภาพ โดยลดการสร้างคุณลักษณะใหม่ซ้ำในแต่ละกระบวนการ

จะเห็นได้ว่า dataset car price เป็นข้อมูลหลังจากการ transformation บางส่วนแล้ว ซึ่งจะเก็บไว้เป็น feature store และเมื่อใดที่ผู้ใช้งานร้องขอให้สร้าง feature store ใหม่ สามารถเรียกใช้งานจาก dataset car price นี้ได้อย่างง่ายดาย เพื่อลดความซ้ำซ้อนในการสร้าง feature engineering หลายๆครั้งและการทำ feature store ยังช่วยลดการ train และ test models โดยการไม่สร้าง feature engineer ใหม่ซ้ำๆอีกด้วย

ผลการทดลองเรียกใช้งานคุณลักษณะกลับมาใช้ใหม่



ภาพประกอบ 31 Feature Reuse Analysis Dashboard

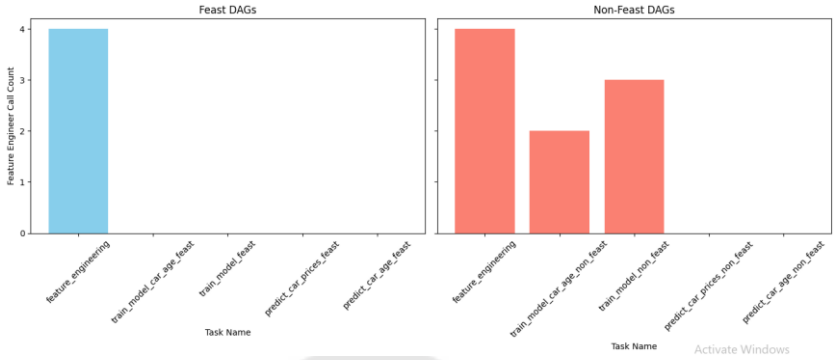
อัตราการนำกลับมาใช้ (Feature Reuse Ratio) = 100 % แสดงให้เห็นว่าทุกคุณลักษณะที่จำเป็นต่อกระบวนการเรียนรู้ถูกดึงจาก feature store โดยตรง ไม่จำเป็นต้องสร้างขึ้นใหม่

ไม่มีการสร้างคุณลักษณะใหม่ ลดภาระการทำ feature engineering ที่ซ้ำซ้อน และลดความเสี่ยงต่อ data drift ที่อาจเกิดขึ้นหากนิยามคุณลักษณะไม่สอดคล้องกันระหว่างรอบการทดลอง

ผลลัพธ์ด้านประสิทธิภาพ เวลารวมของกระบวนการ train-test สั้นลง เนื่องจากตัดขั้นตอนคำนวณคุณลักษณะออกไปดังภาพประกอบ 31

4.2.2 **การประเมินผลด้านลดการใช้งานคุณลักษณะวิศวกรรม (Feature Engineering)**

การประเมินในส่วนนี้มีเป้าหมายเพื่อวิเคราะห์ว่าการนำ FEAST Feature Store มาใช้สามารถลดขั้นตอนการทำงานของ Feature Engineer ได้จริงหรือไม่ โดยเปรียบเทียบจำนวนครั้งของการเรียกใช้งานขั้นตอนการทำ Feature Engineering ระหว่างเวิร์กโฟลว์ที่ใช้ FEAST (Feast DAGs) และเวิร์กโฟลว์ที่ไม่ใช้ FEAST (Non-Feast DAGs)



ภาพประกอบ 32 Feature Engineering Call Comparison between Feast and Non-Feast DAGs

จากผลการทดลองตามภาพประกอบ 32 พบว่าการใช้ FEAST ในการพัฒนา Data Pipeline สามารถลดจำนวนครั้งของการเรียกใช้งานขั้นตอน Feature Engineering ได้อย่างมีนัยสำคัญ โดยในกลุ่มที่ใช้ FEAST พบว่ามีการเรียกใช้งานขั้นตอน Feature Engineering เพียงแค่ 4 ครั้ง ในขณะที่กลุ่มที่ไม่ใช้ FEAST มีการเรียกใช้งานขั้นตอน Feature Engineering มากถึง 9 ครั้ง ซึ่งสะท้อนให้เห็นว่าการใช้ FEAST Feature Store สามารถช่วยลดภาระงานและขั้นตอนการทำงานของ Feature Engineer ได้อย่างชัดเจน ส่งผลให้กระบวนการพัฒนาและบริหารจัดการคุณลักษณะมีประสิทธิภาพและความสะดวกมากขึ้น

4.3 ประเด็นที่ท้าทายและข้อจำกัด

4.3.1 ความซับซ้อนของการเชื่อมต่อกับหลายแหล่งข้อมูล

การตั้งค่า Apache Airflow เพื่อเชื่อมต่อกับแหล่งข้อมูลหลายรูปแบบต้องการการปรับแต่งอย่างละเอียด และอาจเกิดข้อผิดพลาดในกระบวนการเชื่อมต่อ

4.3.2 การจัดการ Metadata

แม้ว่า FEAST จะช่วยลดความซับซ้อนของกระบวนการจัดการคุณลักษณะ แต่ยังคงต้องการการจัดการ Metadata อย่างเป็นระบบเพื่อให้ข้อมูลที่จัดเก็บสอดคล้องกับความต้องการของระบบ

ในบทนี้ได้กล่าวถึงผลการวิจัยการใช้ Apache Airflow ร่วมกับ FEAST บนสภาพแวดล้อม Cloud Platform พบว่าการใช้งานร่วมกันสามารถเพิ่มประสิทธิภาพในการจัดการข้อมูลแบบครบวงจรได้อย่างมีประสิทธิภาพ โดย Apache Airflow ช่วยบริหารจัดการเวิร์กโฟลว์ข้อมูลที่มีความซับซ้อนและมีการดำเนินงานเป็นขั้นตอนอย่างเป็นระบบ และ FEAST ช่วยให้สามารถจัดเก็บและนำกลับมาใช้ซ้ำของคุณลักษณะที่สำคัญในการเรียนรู้ของเครื่อง (Machine Learning) ได้อย่างมีประสิทธิภาพและมีประสิทธิผล นอกจากนี้ การวิจัยยังพบว่าการจัดเก็บข้อมูลแบบ Feature Store ด้วย FEAST บน Google BigQuery ช่วยเพิ่มความน่าเชื่อถือและความถูกต้องของข้อมูล เนื่องจาก FEAST มีระบบการจัดการคุณลักษณะที่มีมาตรฐานและรองรับการทำงานแบบเวอร์ชันข้อมูล (Data Versioning) ซึ่งทำให้การนำข้อมูลไปใช้งานในการฝึกโมเดลมีความแม่นยำและสามารถตรวจสอบย้อนหลังได้ง่าย อย่างไรก็ตาม แม้ว่า FEAST จะช่วยเพิ่มประสิทธิภาพและความสะดวกในการนำคุณลักษณะกลับมาใช้ซ้ำได้เป็นอย่างดี แต่การจัดการ Metadata และการเชื่อมต่อกับหลายแหล่งข้อมูลยังคงเป็นข้อจำกัดที่ควรต้องศึกษาเพิ่มเติม การจัดการ Metadata ที่เป็นระบบและมีมาตรฐานที่ชัดเจนยิ่งขึ้นจะช่วยลดความผิดพลาดและเพิ่มประสิทธิภาพในการบริหารจัดการข้อมูลขนาดใหญ่และซับซ้อนต่อไป ดังนั้น การวิจัยนี้แสดงให้เห็นว่าการผสานรวม Apache Airflow และ FEAST บนแพลตฟอร์มคลาวด์มีศักยภาพในการพัฒนากระบวนการ Data Pipeline ที่มีประสิทธิภาพและสามารถขยายผลไปสู่การใช้งานในเชิงพาณิชย์หรือองค์กรที่มีข้อมูลจำนวนมากและซับซ้อนในอนาคตได้อย่างมีประสิทธิผล

บทที่ 5
สรุปผลการวิจวิจัย อภิปรายผลการวิจัย และข้อเสนอแนะ

ในการวิจัยนี้ได้ศึกษาแนวทางการพัฒนาและการจัดการเวิร์กโฟลว์โดยใช้ FEAST และ Apache Airflow เพื่อเพิ่มประสิทธิภาพในการจัดการข้อมูลสำหรับกระบวนการ Feature Engineering โดยสามารถแบ่งหัวข้อในการสรุปผลได้ดังต่อไปนี้

1.สรุปผลการวิจัย

2.อภิปรายผลการวิจัย

3.ข้อเสนอแนะ

5.1 สรุปผลการวิจัย

จากผลการทดลองเปรียบเทียบกระบวนการจัดการข้อมูลระหว่างการใช้งาน FEAST และกระบวนการแบบ Non-FEAST พบว่า การใช้งาน FEAST ช่วยลดเวลาในการดำเนินงานด้าน Feature Engineering อย่างชัดเจน เนื่องจาก FEAST มีการจัดเก็บคุณลักษณะที่ผ่านกระบวนการแปลงข้อมูลไว้ใน Feature Store ซึ่งสามารถเรียกใช้งานได้โดยตรง ลดขั้นตอนการดำเนินการซ้ำซ้อน และลดเวลาการทำงานโดยรวมลงได้อย่างมีประสิทธิภาพ

นอกจากนี้ FEAST ยังเพิ่มศักยภาพในการนำคุณลักษณะกลับมาใช้งานซ้ำ (Feature Reusability) ได้สูงกว่าแบบ Non-FEAST เนื่องจากมีการจัดเก็บข้อมูลคุณลักษณะในลักษณะที่เป็นระบบ มีการรองรับเวอร์ชันข้อมูลอย่างชัดเจน ส่งผลให้การจัดการข้อมูลสำหรับโมเดล Machine Learning สามารถดำเนินการได้อย่างสะดวก รวดเร็ว และแม่นยำมากขึ้น รวมทั้งยังช่วยลดความซับซ้อนของเวิร์กโฟลว์ในการประมวลผลข้อมูล ส่งผลต่อประสิทธิภาพการบริหารจัดการข้อมูลโดยภาพรวมดีขึ้น

ตาราง 2 Reuse Count Feature Store

Feature	brand	model	year	Engine_ size	fuel_ type	transmission	car_ door	owner_ count	price	price_per _mile	mileage
FEAST	2	2	2	2	2	2	2	2	2	2	2
NON FEAST	0	0	0	0	0	0	0	0	0	0	0

5.2 อภิปรายผลการวิจัย

จากการศึกษาพบว่าการจัดการข้อมูลและการรองรับข้อมูลแบบ Realtime ถือเป็นประเด็นที่สำคัญอย่างยิ่งในการพัฒนาระบบ Machine Learning และ Data Engineering เนื่องจากข้อมูลในยุคปัจจุบันมีการอัปเดตและเปลี่ยนแปลงอย่างรวดเร็วและต่อเนื่อง จึงจำเป็นต้องมีการออกแบบระบบ Data Pipeline ที่สามารถรองรับการจัดการข้อมูลแบบ Realtime เพื่อให้สามารถตอบสนองการเปลี่ยนแปลงของข้อมูลได้ทันที ซึ่ง Apache Airflow มีความเหมาะสมในการจัดการ Workflow เนื่องจากสามารถกำหนดเวลาการทำงานและการเรียกใช้งานงาน (tasks) ได้อย่างยืดหยุ่น อีกทั้งยังมีความสามารถในการเชื่อมต่อกับระบบต่าง ๆ เช่น API และฐานข้อมูลที่รองรับข้อมูลแบบเรียลไทม์ ซึ่งช่วยให้การประมวลผลข้อมูลสามารถดำเนินการได้อย่างมีประสิทธิภาพ

ในขณะที่การใช้ Feature Store อย่าง FEAST ช่วยเสริมการรองรับข้อมูลแบบ Realtime ได้เป็นอย่างดี โดย FEAST มีระบบการจัดเก็บข้อมูลฟีเจอร์ที่แยกส่วนชัดเจนระหว่าง Offline Store และ Online Store ทำให้สามารถตอบสนองการใช้งานที่แตกต่างกันได้อย่างมีประสิทธิภาพ โดยเฉพาะในส่วนของ Online Store ซึ่งถูกออกแบบมาให้รองรับการให้บริการข้อมูลฟีเจอร์ด้วยค่าหน่วงเวลาที่ต่ำมาก ทำให้สามารถสนับสนุนระบบ Machine Learning ที่ต้องการการตอบสนองอย่างทันทีทันใด (low latency)

อย่างไรก็ตาม จากการทดลองพบว่าข้อมูลที่นำมาใช้ในงานวิจัยมีจำนวนฟีเจอร์และขนาดของข้อมูลค่อนข้างจำกัด ทำให้ไม่สามารถแสดงประสิทธิภาพสูงสุดของระบบ Feature Store ได้อย่างเต็มที่ ซึ่งโดยปกติแล้ว ระบบ Feature Store จะให้ประโยชน์สูงสุดเมื่อมีการใช้งานฟีเจอร์จำนวนมาก และมีข้อมูลที่มีความซับซ้อนและขนาดใหญ่ ดังนั้นในการใช้งานจริง ควรพิจารณาปริมาณและความหลากหลายของข้อมูลให้สอดคล้องกับความสามารถของระบบ Feature Store เพื่อให้ได้รับประโยชน์สูงสุดในการจัดการข้อมูลและเพิ่มประสิทธิภาพในกระบวนการ Feature Engineering และการนำฟีเจอร์กลับมาใช้ซ้ำได้อย่างมีประสิทธิภาพมากที่สุด

5.3 ข้อเสนอแนะ

จากวัตถุประสงค์ที่กำหนดไว้ในงานวิจัย สามารถเสนอแนะแนวทางในการดำเนินงานต่อยอดได้ดังนี้

5.3.1 **แนวทางในการพัฒนาและจัดการเวิร์กโฟลว์ด้วย** Apache Airflow **และ** FEAST

แนะนำการพัฒนา Custom Operators หรือ Hooks เพิ่มเติมใน Apache Airflow เพื่อเพิ่มความสามารถในการจัดการข้อมูลจากแหล่งข้อมูลที่หลากหลายมากขึ้น

ศึกษาและพัฒนากลไกการตรวจสอบคุณภาพข้อมูลแบบครบวงจร เพื่อเพิ่มความถูกต้องและเชื่อถือได้ของข้อมูลภายในเวิร์กโฟลว์

5.3.2 **แนวทางการพัฒนา** FEAST

เพิ่มประสิทธิภาพ FEAST ในการรองรับข้อมูลแบบ Streaming และ Real-Time เพื่อเพิ่มประสิทธิภาพในการให้บริการข้อมูลที่ต้องการใช้งานทันที (Real-time Inference)

ศึกษาและขยายความสามารถของ FEAST ด้านการทำ Point-in-Time Joins ให้มีความถูกต้องแม่นยำและรองรับข้อมูลที่มีความซับซ้อนมากขึ้น

5.3.3 **แนวทางในการลดขั้นตอน** Feature Engineering

ศึกษาและนำเครื่องมือ Automated Feature Selection มาใช้เพื่อลดการดำเนินงานที่ต้องทำด้วยตนเอง และเพิ่มความแม่นยำในการเลือกคุณลักษณะที่มีประสิทธิภาพ

พัฒนาแนวทางการปรับแต่งกระบวนการ Feature Engineering แบบอัตโนมัติร่วมกับ FEAST และ Machine Learning Pipelines เพื่อให้การจัดการข้อมูลมีความยืดหยุ่นและรวดเร็วยิ่งขึ้น

บรรณานุกรม

[1] J. B. Aiswarya Raj, Helena Holmstr¨ om Olsson, Tian J. Wang,, " Modelling Data Pipelines," presented at the 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2020.

[2] Sameer Shukla, "Developing Pragmatic Data Pipelines using Apache Airflow on Google Cloud Platform," International Journal of Computer Sciences and Engineering, 2022.

[3] Pankaj Dureja, "Strategic Data Pipeline Design: Enhancing Operational Efficiency from Oracle to Single Store using Airflow S3 Data Pipelines," International Journal of Science and Research (IJSR), *2022*, doi: https://dx.doi.org/10.21275/SR24731115303.

[4] a. Alexandru A. Ormenis, Mahmoud Ismail, Kim Hammar, Robin Andersson, Ermias Gebremeskel, and A. K. Theofilos Kakantousis, Fabio Buso, Gautier Berthou, Jim Dowling, Seif Haridi,, "HORIZONTALLY SCALABLE ML PIPELINES WITH A FEATURE STORE," presented at the IEEE transactions on software engineering, 2019.

[5] F. B. Javier de la Rúa Martínez, Antonios Kouzoupis, Alexandru A. Ormenisan, Salman Niazi, "The Hopsworks Feature Store for Machine Learning," presented at the SIGMOD-Companion '24, 2024.

[6] T. Z. Hamza Elkina, "Data Ingestion Frameworks for Data Lakes: An Overview," Journal for Re Attach Therapy and Developmental Diversities 2023.

[7] Dylan Stilinski, "Building a Scalable and Robust Data Extraction Pipeline with ApacheAirflow and Cloud Platforms," Computing and Information Technology · April *2024*, *2024.*

[8] B. R. Anya Li, Feng Pan, Mickey Zhang, Qianjun Xu, Runhan Li, Sethu Raman, Shail Paragbhai Shah, Vivienne Tang, , "Managed Geo-Distributed Feature Store: Architecture and System Design," presented at the arXiv, 2023.

[9] Alladi Deekshith, "Integrating AI and Data Engineering: Building Robust Pipelines

for Real-Time Data Analytics," International Journal of Sustainable Development in Computing Science.

[10] A. C. Alice Zheng, R. Roumeliotis, Ed. Feature Engineering for Machine Learning. O'Reilly Media, 2018.

[11] A. M. P. Shubha B G, "Airflow Directed Acyclic Graph," Journal of Signal Processing, 2019, doi: http://doi.org/10.5281/zenodo.3247274

[12] P. P. Ltd, Feature Store for Machine Learning. https://books.google.co.th/books?hl=th&lr=&id=rgpyEAAAQBAJ&oi=fnd&pg=PP1&dq=feature+store&ots=6-rG35bour&sig=IAIeFMVVlfmOtUQZ25ejJCDCVC0&redir_esc=y#v=onepage&q&f=false, 2022.

[13] A. V. Fahad Pervaiz, Richard Anderson, "Examining the Challenges in Development Data Pipeline," presented at the ACM SIGCAS Conference on Computing and Sustainable Societies (COMPASS 2019), 2019.

[14] X. M. Sarah Wooders, Amit Narang, Kevin Lin, Ion Stoica, Joseph M. Hellerstein, Natacha Crooks, Joseph E. Gonzalez, "RALF: Accuracy-Aware Scheduling for Feature Store Maintenance," in *VLDB Endowment*, 2023, vol. 17, no. 3, doi: 10.14778/3632093.3632116.

[15] A. Khazanchi, "Faster Reading with DuckDB and Arrow Flight on Hopsworks: Benchmark and Performance Evaluation of Offline Feature Stores," Project in Software Engineering of Distributed Systems, School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, 2023. [Online]. Available: https://www.diva portal.org/smash/get/diva2:1801362/FULLTEXT01.pdf

[16] F. I. Kiet Hoang Nguyen, Arman Markar, Shaho Ahmed, Beni Mahato, "Develop Machine Learning Models using AWS," Cybersecurity Engineering Department, George Mason University, Fairfax, VA George Mason University, Fairfax, VA, 2023.

[17] D. J. Kataria, R. Limbachiya, Ed. feature store for machine learing. 2022.

[18] M. P. Co., T. Louvar, Ed. Data Pipeline with Apache Airflow. 2021.

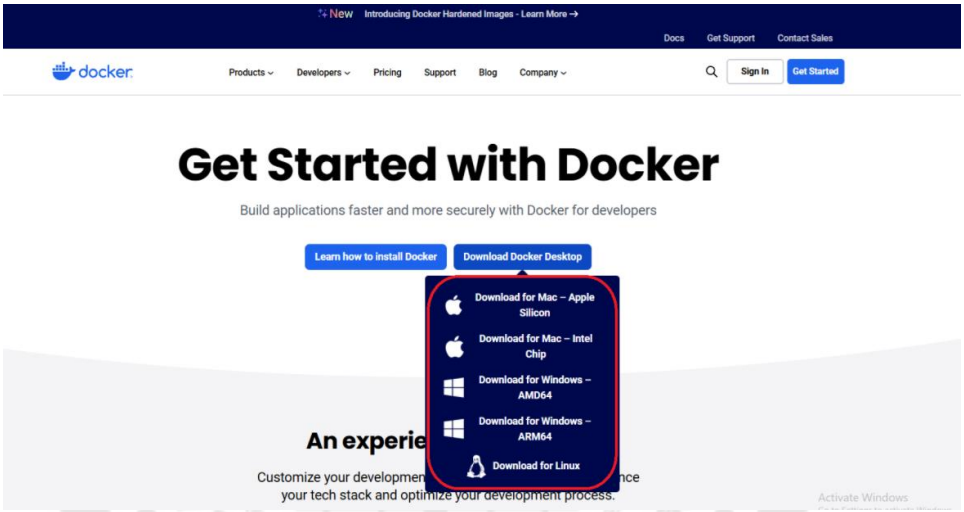
[19] M. Lutz, L. L. a. F. Willison, Ed. Programming Python. 2001.

ภาคผนวก

วิธีการติดตั้ง Apache Airflow บน Localhost

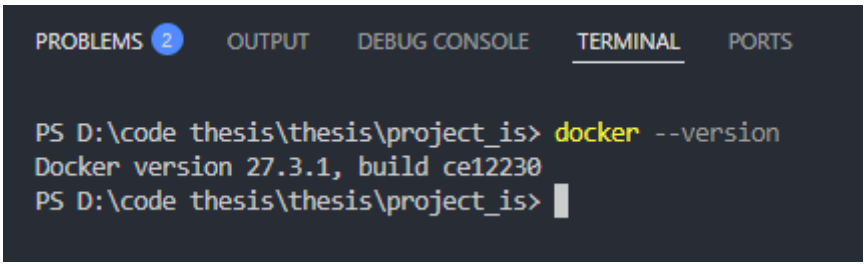
ขั้นตอนที่ 1 ติดตั้ง docker ผ่าน website https://www.docker.com/get-started/

1.1 เลือกตามความเหมาะสมสำหรับระบบปฏิบัติการ windows / mac os



ภาพประกอบ 33 Install Docker System

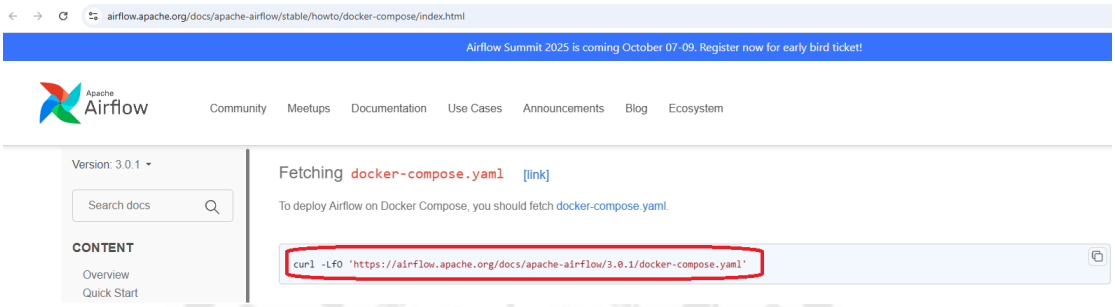
1.2 เมื่อ download เสร็จเรียบร้อย ให้เปิดเทอร์มินัล (Terminal หรือ PowerShell บน Windows)และรันคำสั่งตรวจสอบเวอร์ชัน: docker –version



ภาพประกอบ 34 Check Version Docker

ขั้นตอนที่ 2 วิธีการติดตั้ง apache airflow ผ่านเว็บไซต์ทางการ โดยลิ้งค์จะอยู่ด้านล่างนี้ https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html

2.1 เลือกชุดคำสั่งและคัดลอก: curl -LfO 'https://airflow.apache.org/docs/apache-airflow/3.0.1/docker-compose.yaml'



ภาพประกอบ 35 Copy Script Curl

2.2 หลังจากคัดลอก curl ให้นำมาวางไว้ที่ path ที่เราต้องการจะติดตั้ง docker-compose

2.3 Enter เพื่อรันคำสั่งดังกล่าว ซึ่งการติดตั้งครั้งแรกจะใช้เวลาประมาณ 30 นาที

2.4 หลังจากที่รันเสร็จจะได้ Files docker compose มา และซึ่งใน files จะประกอบไปด้วย service ต่างๆ และแนะนำให้เปลี่ยน config จาก AIRFLOW__CORE__EXECUTOR: CelerylExecutor ให้กลายเป็น LocalExecutor เพื่อป้องกันไม่ให้ airflow ทำการกระจายโหลดโดยworker หลายตัว จะได้ไม่หนักทรัพยากรบนเครื่องเกินไป

```yaml
x-airflow-common:
  &airflow-common
  # In order to add custom dependencies or upgrade provider packages you can use your extended image.
  # Comment the image line, place your Dockerfile in the directory where you placed the docker-compose.yaml
  # and uncomment the "build" line below, Then run `docker-compose build` to build the images.
  # image: ${AIRFLOW_IMAGE_NAME:-airflow:2.10.3lastversion}
  build: .
  environment:
    &airflow-common-env
    AIRFLOW__CORE__EXECUTOR: LocalExecutor
    AIRFLOW__DATABASE__SQL_ALCHEMY_CONN: postgresql+psycopg2://airflow:airflow@postgres/airflow
    # AIRFLOW__CELERY__RESULT_BACKEND: db+postgresql://airflow:airflow@postgres/airflow
    # AIRFLOW__CELERY__BROKER_URL: redis://:@redis:6379/0
    AIRFLOW__CORE__FERNET_KEY: ''
    AIRFLOW__CORE__DAGS_ARE_PAUSED_AT_CREATION: 'true'
    AIRFLOW__CORE__LOAD_EXAMPLES: 'false'
    AIRFLOW__API__AUTH_BACKENDS: 'airflow.api.auth.backend.basic_auth,airflow.api.auth.backend.session'
    # yamllint disable rule:line-length
```

ภาพประกอบ 36 Changes Executor

2.5 แนะนำให้ comment redis

```yaml
# redis:
#   # Redis is limited to 7.2-bookworm due to licencing change
#   # https://redis.io/blog/redis-adopts-dual-source-available-licensing/
#   image: redis:7.2-bookworm
#   expose:
#     - 6379
#   healthcheck:
#     test: ["CMD", "redis-cli", "ping"]
#     interval: 10s
#     timeout: 30s
#     retries: 50
#     start_period: 30s
#   restart: always
```

ภาพประกอบ 37 Comment Config Redis

2.6 แนะนำให้ comment airflow-worker

```
# airflow-worker:
#   <<: *airflow-common
#   command: celery worker
#   healthcheck:
#     # yamllint disable rule:line-length
#     test:
#       - "CMD-SHELL"
#       - 'celery --app airflow.providers.celery.executors.celery_executor.app inspect ping -d "celery@$${HOSTNAME}" ||
#     interval: 30s
#     timeout: 10s
#     retries: 5
#     start_period: 30s
#   environment:
#     <<: *airflow-common-env
#     # Required to handle warm shutdown of the celery workers properly
#     # See https://airflow.apache.org/docs/docker-stack/entrypoint.html#signal-propagation
#     DUMB_INIT_SETSID: "0"
#   restart: always
#   depends_on:
#     <<: *airflow-common-depends-on
#     airflow-init:
#       condition: service_completed_successfully
```

ภาพประกอบ 38 Comment Config Airflow Worker

2.9 แนะนำให้ comment flower

```
# flower:
#   <<: *airflow-common
#   command: celery flower
#   profiles:
#     - flower
#   ports:
#     - "5555:5555"
#   healthcheck:
#     test: ["CMD", "curl", "--fail", "http://localhost:5555/"]
#     interval: 30s
#     timeout: 10s
#     retries: 5
#     start_period: 30s
#   restart: always
#   depends_on:
#     <<: *airflow-common-depends-on
#     airflow-init:
#       condition: service_completed_successfully
```

ภาพประกอบ 39 Comment Config Airflow Flower

จากนั้นสามารถใช้คำสั่งรัน docker compose up -d เพื่อเปิดการใช้งาน docker ร่วมกับการทำงานของ apache aiflow บน localhost การใช้คำสั่งนี้จะช่วยให้ผู้ใช้งานสามารถเปิดใช้งาน Apache Airflow บนเครื่องในรูปแบบของ localhost ได้อย่างสมบูรณ์

ประวัติผู้เขียน