



โรงเรียนบดินทรเดชา (สิงห์ สิงหเสนี)
 โครงการสาขาคอมพิวเตอร์และหุ่นยนต์

เรื่อง Machine Learning for ship disaster

คณะผู้จัดทำ

- [illegible]

ชั้นมัธยมศึกษาปีที่ 5/7

ครูที่ปรึกษา

ครูเอกอนงค์ แก้วดี
ครูสิทธิพงษ์ ศุภณัฐ

โครงการนี้เป็นส่วนหนึ่งของกิจกรรมการเรียนรู้สู่โครงงาน
(Project Based Learning)
ปีการศึกษา 2565



ใบอนุญาตโครงการ
โรงเรียนบดินทรเดชา (สิงห์ สิงหเสนี)

โครงการสาขาคอมพิวเตอร์และหุ่นยนต์
เรื่อง Machine Learning for ship disaster

รายนามผู้จัดทำ

1. นายภาค

หล่อวิจิตร

ชั้นมัธยมศึกษาปีที่ 5/7

โครงการนี้ได้รับการพิจารณาอนุมัติให้นับเป็นส่วนหนึ่งของการศึกษาภายใต้
กิจกรรมการเรียนรู้สู่โครงการ ปีการศึกษา 2565

ครูเอกอนงค์ แก้วดี ครูที่ปรึกษาคนที่ 1 ครูสิทธิพงษ์ ศุนย์ปั่ง ครูที่ปรึกษาคนที่ 2

กิตติกรรมประกาศ

โครงการนี้สำเร็จขึ้นได้ด้วยการสนับสนุน การให้คำปรึกษา รวมถึงการให้ความอนุเคราะห์ด้านวัสดุอุปกรณ์ และสถานที่ปฏิบัติงานจาก ผู้ปกครอง โดยเฉพาะอย่างยิ่งทีมงานบริษัท OpenAI ที่พัฒนา ChatGPT ขึ้นมา ซึ่ง AI ตัวนี้ได้ให้คำแนะนำ แนวคิด ตลอดจนแก้ไขข้อบกพร่องต่างๆ มาโดยตลอด จนโครงการเล่มนี้เสร็จสมบูรณ์ ผู้จัดทำจึงขอกราบขอบพระคุณเป็นอย่างสูง

ท้ายสุดนี้ผู้จัดทำหวังเป็นอย่างยิ่งว่า โครงการนี้จะเป็นประโยชน์ต่อการศึกษาเกี่ยวกับการพัฒนา Machine Learning Model ในด้านการ classification ต่อไป

คณะผู้จัดทำ

ชื่อเรื่องโครงการ : Machine Learning for ship disaster

สาขาโครงการ : คอมพิวเตอร์และหุ่นยนต์

ผู้จัดทำโครงการ : 1. นายภาค หล่อวิจิตร เลขที่ 7

ชั้น/ห้อง : ม. 5/7

ครูที่ปรึกษาคนที่ 1 : ครูเอกอนงค์ แก้วดี

ครูที่ปรึกษาคนที่ 2 : ครูสิทธิพงษ์ ศุูนย์บั้ง

ปีการศึกษา : 2565

บทคัดย่อ

การจัดทำโครงการในครั้งนี้มีวัตถุประสงค์ 1.เพื่อสร้างโปรแกรมตัวอย่างที่บอกความน่าจะเป็นในการรอดชีวิตในอุบัติเหตุทางเรือได้ 2.เพื่อศึกษากระบวนการ Preprocessing ข้อมูล 3.เพื่อศึกษากระบวนการสร้าง Machine Learning Model ที่มีประสิทธิภาพ 4.เพื่อศึกษาวิธีการ Deployment ของ Machine Learning Model โดยดำเนินการด้วยแนวคิดเชิงคำนวณและกระบวนการออกแบบเชิงวิศวกรรม ซึ่งได้มีการนำทั้งสองแนวคิดมาประยุกต์ในทุกขั้นตอนการทำงาน ตั้งแต่การพัฒนาในระดับฟังก์ชัน ไปจนถึงการวางแผนการทำงาน

ผลการศึกษาและจัดทำโครงการพบว่า โปรแกรมสามารถทำนายค่าความน่าจะเป็นในการรอดชีวิตของคนบนเรือไททานิคได้อย่างแม่นยำ โดยในขั้นตอนการ preprocessing ทางผู้จัดทำได้แปลงข้อมูลใน dataset ให้อยู่ในรูป Numpy array เพื่อให้สามารถนำไปใช้ในการฝึก ML ต่อได้ ซึ่งระหว่างการฝึกทางผู้จัดทำได้พบว่าการปรับ Hyperparameter บางตัวสามารถทำให้ประสิทธิภาพของ Model ดีขึ้นได้ และเมื่อปรับปรุง Model จนพบ Model ที่ดีที่สุดแล้ว ทางผู้จัดทำก็ได้นำ Model ดังกล่าวไปใช้ในโปรแกรม Prototype ก็สามารถบรรลุจุดประสงค์ของโครงการที่ว่าค่าความน่าจะเป็นจะเปลี่ยนต่อเมื่อ feature มีการเปลี่ยน (ตามที่ ML เห็นสมควร) แม้การพัฒนาโปรแกรมจะผ่านพ้นไปทุกวัตถุประสงค์ แต่ทั้งหมดนี้ก็จะเกิดขึ้นไม่ได้เลยหากเราไม่ได้ยึดแนวคิดที่ว่า ข้อมูลที่ดีจะนำมาสู่ Model ที่ได้ เป็นพื้นฐานตลอดการทำงานในโครงการนี้

สารบัญ

	หน้า
กิตติกรรมประกาศ	ก
บทคัดย่อ	ข
สารบัญ	ค
สารบัญตาราง	ง
สารบัญภาพ	ฉ
บทที่ 1 บทนำ	1
1.1 ที่มาและความสำคัญ	1
1.2 วัตถุประสงค์	2
1.3 ขอบเขตของการศึกษา	2
1.6 นิยามศัพท์เฉพาะ	2
1.7 ประโยชน์ที่คาดว่าจะได้รับ	2
บทที่ 2 เอกสารและงานวิจัยที่เกี่ยวข้อง	3
2.1 กระบวนการออกแบบเชิงวิศวกรรม (Engineering Design Process)	3
2.2 แนวคิดเชิงคำนวณ (Computational Thinking)	4
2.3 งานวิจัยที่เกี่ยวข้อง	4
บทที่ 3 อุปกรณ์และวิธีดำเนินการทดลอง	6
3.1 เครื่องมือที่ใช้ในการทำโครงงาน	6
3.2 กระบวนการออกแบบเชิงวิศวกรรม (Engineering Design Process)	7
3.3 แนะนำ Dataset ที่ใช้ในการพัฒนาโครงงาน (Introduction to Titanic dataset)	16
3.4 ขั้นตอนการแก้ปัญหาตามแนวคิดเชิงคำนวณ (Computational Thinking)	18
Phase 1: Preprocessing	21
Phase 2: Model Training and Testing	27
Phase 3: Deployment	36

บทที่ 4 ผลการดำเนินงาน	38
4.1 การตรวจสอบและแก้ปัญหา: Preprocessing	38
4.2 การตรวจสอบและแก้ปัญหา: Training/Testing	40
4.3 Insight เล็กๆ น้อยๆ จาก Random Forest Classification	48
4.4 แนะนำ GUI ของโปรแกรม Prototype	49
บทที่ 5 สรุป อภิปรายผล และข้อเสนอแนะ	50
5.1 สรุปผล	50
5.2 อภิปรายผล	50
5.3 ข้อเสนอแนะ	51
เอกสารอ้างอิง	53
ภาคผนวก	54
ข้อมูลผู้จัดทำ	62

สารบัญตาราง

ตารางที่	หน้า
1 แนวทางการแก้ปัญหา	8
2 ข้อมูลแต่ละ feature ก่อนการ cleaning (สภาพข้อมูลดิบ)	38
3 ข้อมูลแต่ละ feature ภายหลังการ cleaning	38
4 ตารางตรวจสอบสภาพความพร้อมของ data ก่อนใช้งาน	39
5 การปรับปรุงข้อมูลให้ผ่านทุกตัวชี้วัด	40
6 ผลการทำงานของ ML models	41

สารบัญภาพ

ภาพที่	หน้า
1 ขั้นตอนการพัฒนาตั้งแต่ต้นจนจบ	10
2 Attribute ทั้ง 12 หลัง Data Cleaning	16
3 ภาพรวมการทำงานของ Prototype และ flow การใช้งานของผู้ใช้	20
4 แผนภาพสรุปตัวแปรเก็บข้อมูลจาก Phase 1	27
5 Model V1	32
6 Model V2	32
7 Model V3	32
8 Model V4	32
9 Model V5	33
10 Model RF V1	34
11 Model RF V1 predictions	34
12 Model RF V2	34
13 Model RF V2 predictions	34
14 param_grid และ การใช้ GridSearchCV	35
15 การนำ best_pames มาใช้เป็น Hyperparameter	35
16 Model VB predictions	35
17 การ save model ด้วย pickle	35
18 ความสำคัญของแต่ละ Feature ที่ใช้ Train (Model RF V1)	48
19 GUI ของ Prototype	49
20 ตัวอย่างการใช้งานโปรแกรม Prototype (1)	50
21 ตัวอย่างการใช้งานโปรแกรม Prototype (2)	50

บทที่ 1

บทนำ

1.1 ที่มาและความสำคัญ

เมื่อปลายปี 2022 ผู้จัดทำได้เข้าไปเรียน course การสร้าง Deep Learning Model ด้วย Tensorflow กับเว็บไซต์ Datacamp ซึ่งทำให้ผู้จัดทำได้พบกับ Titanic dataset ซึ่งโดยตัวผู้จัดทำเองมีความสนใจในเรื่องไททานิคอยู่เป็นทุนเดิมอยู่แล้ว ทำให้ไม่ใช่ว่าการยากที่จะหลงใหลและอยากทำ AI ตัวแรกในชีวิตกับ dataset ชุดนี้ ก่อนหน้านี้นี้ 2 ปี ผู้จัดทำเคยมีประสบการณ์ในการใช้ภาษา Python และ Data Science Libraries พัฒนางานต่างๆ มาบ้างและก็พอมีความรู้พื้นฐานเกี่ยวกับ Machine Learning และ Neural Network ทำให้ดูไม่ยากก็บอกได้ว่า ML ตัวนี้น่าจะใช้วิธีการเรียนรู้แบบ Supervised Learning กล่าวคือมีข้อมูล sample ในการ train และก็มี target ให้ ML ดูว่าผลจริงๆ ต้องเป็นอย่างไร จากนั้น ML ก็จะเรียนรู้เพื่อให้สร้างผลที่ใกล้เคียงที่สุดกับ target

ผู้จัดทำพอรู้มาบ้างว่าการทำ ML จำเป็นต้องมีกระบวนการ Preprocessing ข้อมูลที่ได้มาก่อน จึงจะนำไป Train หรือ Test ML ได้ เมื่อเสร็จแล้วเราก็สามารถ load model ลงมาใส่ prototype ของเราได้เลย หนึ่งในเรื่องที่คุณจัดทำได้เรียนรู้จากการเข้าร่วมค่าย CAI Camp 2022 ก็คือการใช้ Google Colab ในการ train AI โดยแต่ก่อนนั้นทางผู้จัดทำเคย Train AI แต่เพียงบนคอมพิวเตอร์ของตัวเอง ซึ่งใช้เวลามากและมีความยุ่งยากในการติดตั้ง Library และการ load dataset ทำให้ในครั้งนี้ผู้จัดทำตัดสินใจจะพัฒนา Model บน Google Colab Pro และนำไฟล์ dataset ต่างๆ ไปไว้ใน Repository ใน Github ของตัวเอง ซึ่งวิธีดังกล่าวได้ทำการพัฒนา ML เป็นไปได้อย่างสะดวกยิ่งขึ้น

กระทั่งมาถึงจุดนี้ ผู้จัดทำได้ตระหนักถึงความสำคัญและความเป็นไปได้ของการที่ ML ตัวนี้ที่จะสามารถนำมาทำเป็น Prototype สำหรับแนวคิดการสร้างโปรแกรมที่ทำนายความน่าจะเป็นที่จะปลอดภัยจากการเดินทางด้วยเรือสำราญได้ ด้วยจุดนี้ทางผู้จัดทำจึงได้จัดทำ ML Version ที่ดีขึ้นและสร้าง Prototype program ด้วยภาษา Python โดยโปรแกรมดังกล่าวจะทำงานผ่านแนวคิดที่ว่าผู้ใช้มีอำนาจในการเปลี่ยนแปลงสิ่งต่างๆ ที่พวกเขาทำได้ก่อนขึ้นเรือ ผู้จัดทำมีแนวคิดที่ว่าหากผู้ใช้สามารถรู้ว่าการเลือกห้องชั้นนี้ ค่าใช้จ่ายเท่านี้ หรือ จำนวนผู้คนที่โดยสารจะขึ้นเรือเท่านี้ จะทำให้ค่าความน่าจะเป็นในการรอดชีวิตของพวกเขาเป็นเท่าไร พวกเขาจะสามารถปรับเปลี่ยนข้อมูลต่างๆ ที่เข้ามาเพื่อเพิ่มความน่าจะเป็นดังกล่าวได้ แม้ว่าในความเป็นจริงความปลอดภัยบนเรือจะบอกได้ยากหากเพียงนั่งอยู่บ้าน แต่ก่อนเตรียมตัวเท่าที่เราทำได้ก็คือสิ่งเดียวที่เราควรทำไว้ก่อนเพื่อรับความเสี่ยงที่อาจเกิดขึ้น ด้วยเหตุผลทาง dataset โปรแกรมของทางผู้จัดทำจึงเป็นเพียงโปรแกรมตัวอย่างที่แสดงให้เห็นว่าโปรแกรมของจริงจะทำงานอย่างไร แต่ทางผู้จัดทำก็หวังเป็นอย่างยิ่งว่ามันจะได้รับความสนใจจนไปต่อยอดจนได้โปรแกรมดังที่ทางผู้จัดทำคาดหวังไว้

1.2 วัตถุประสงค์

1. เพื่อสร้างโปรแกรมตัวอย่างที่บอกความน่าจะเป็นในการรอดชีวิตในอุบัติเหตุทางเรือได้
2. เพื่อศึกษากระบวนการ Preprocessing ข้อมูล
3. เพื่อศึกษากระบวนการสร้าง Machine Learning Model ที่มีประสิทธิภาพ
4. เพื่อศึกษาวิธีการ Deployment ของ Machine Learning Model

1.3 ขอบเขตของการศึกษา

1. โปรแกรมนี้เป็นโปรแกรมตัวอย่างที่มีระบบการทำงานที่ควรมีครบถ้วน (Prototype)
2. Machine Learning Model ทั้งหมดถูก Train ขึ้นมาจาก Titanic dataset
3. วิธีการเรียนรู้ของ Machine Learning ที่เลือกใช้คือ Supervised Learning
4. โครงสร้าง Model ที่เลือกใช้เป็น Artificial Neural Network และ Random Forest Classification
5. โปรแกรมตัวอย่างสามารถ run ได้บน macOS (หากยึดตามที่ทดลอง) โดยหากนำไป run ที่ระบบปฏิบัติการอื่นก็อาจต้อง download ไฟล์ที่ใช้ผ่าน Repository ของทางผู้จัดทำ และอาจมีการเปลี่ยนแปลงของ GUI บ้างเล็กน้อย

1.5 นิยามศัพท์เฉพาะ

1. Y-vector : Array ของข้อมูล feature ที่เป็น Target ของ ML model
2. X-vector : Array ของข้อมูล feature ที่ใช้ optimize model ตอน Forward Propagation
3. Feature : หัวข้อมูล/คุณสมบัติของ sample นั้นๆ ที่ถูกนำไปใช้ train ML
4. Feature vector : vector ที่แปลงข้อมูลของ sample นั้นในรูปตัวเลข
5. Parameter :
 - องค์ประกอบที่เกิดการเปลี่ยนแปลงขณะเรียนรู้ของ ML model
 - ชื่อ Input ของฟังก์ชันนั้นๆ
6. Hyperparameter : parameter ที่ของโครงสร้าง Model และระดับการ Compile
7. Normalization : การ scale ค่าของข้อมูลชุดนี้ให้อยู่ในช่วงค่าที่เหมาะสม

1.7 ประโยชน์ที่คาดว่าจะได้รับ

1. โปรแกรมตัวอย่างสามารถทำงานได้เหมือนกับที่โปรแกรมจริงๆ ทำงานได้
2. โปรแกรมตัวอย่างสามารถทำนายข้อมูลของผู้ใช้ได้ด้วยความแม่นยำที่สูง
3. โปรแกรมตัวอย่างมี Model ให้เลือกเพื่อใช้ในการเปรียบเทียบค่าความน่าจะเป็นของ Model ที่ต่างกัน

4. การพัฒนาโปรแกรมในโครงการนี้จะช่วยต่อยอดและพัฒนาความรู้ด้านการพัฒนา Machine Learning Model ของผู้จัดทำต่อไป

บทที่ 2

เอกสารและงานวิจัยที่เกี่ยวข้อง

ในการจัดทำโครงงาน เรื่อง Machine Learning for ship disaster
คณะผู้จัดทำได้ศึกษาเอกสารและงานวิจัยที่เกี่ยวข้อง ตามลำดับดังนี้

2.1 กระบวนการออกแบบเชิงวิศวกรรม (Engineering Design Process)

1. ระบุปัญหา : ลูกค้าเรือสำราญไม่ทราบว่าเขาต้องเตรียมตัวอย่างไรเพื่อลดความเสี่ยงหากเกิดเหตุฉุกเฉินขึ้นกับเรือ
2. รวบรวมข้อมูล : ได้มีการรวบรวมข้อมูลเกี่ยวกับผู้โดยสารของเรือสำราญ โดยเราได้เลือก dataset ของเรือ Titanic เพราะเป็น dataset ที่มีเนื้อหา feature ครบถ้วนและบอกด้วยว่าผู้โดยสารแต่ละคนประสบชะตากรรมอย่างไรเมื่อเกิดอุบัติเหตุทางเรือ นอกจากนั้นเรายังได้รวบรวม Library ที่จำเป็นในการแก้ปัญหาเช่น Matplotlib (สำหรับแสดง Visualization ของ data และประสิทธิภาพขณะ Training) , Numpy (แปลงข้อมูลให้อยู่ในรูปที่ model ใช้ train หรือประมวลผลได้), Tensorflow (ใช้ Train AI และใช้ทำนายเมื่อ train เสร็จสิ้นแล้ว) เป็นต้น
3. ออกแบบวิธีแก้ปัญหา : ตัดสินใจการพัฒนา Machine Learning model ที่มีความสามารถในการทำนายความเสี่ยงของผู้โดยสารหากยึดจากข้อมูล (feature) ของพวกเขา โดย model เป็น Prototype เพราะถูก train มาจากข้อมูลของเรือ Titanic เท่านั้น model ถูกพัฒนาขึ้นด้วยวิธีการเรียนรู้แบบ Supervised Learning โดยใช้โครง model เป็น Neural Network (เป็นหลัก) ที่ถูกเขียนขึ้นจาก Library : Tensorflow เราได้วางแผน Phase ของการพัฒนาเป็น 3 ส่วนคือ Preprocessing (Data Preparation) , Training/Testing , และ Deployment (สร้าง UI) Computational Thinking ถูกนำมาประยุกต์ใช้ในการพัฒนา code ในหลายขั้นตอน เช่น การจับ Pattern โดยการสร้างฟังก์ชัน
4. การดำเนินการแก้ปัญหา : ได้มีการทำกระบวนการ Data Preparation และ Model Training บน Google Colab โดยใช้ภาษา Python และ Library ต่างๆ เมื่อ Train เสร็จ จะมีการ download model ลงมาบนเครื่องเพื่อนำมาใช้งานในโปรแกรมที่ Graphic User Interface (GUI) ต่อไป หากเกิดปัญหาในขั้นตอนการพัฒนา เราก็มักจะนำ Error หรือปัญหาต่างๆ ที่เจอไปสอบถาม ChatGPT เพื่อขอคำแนะนำ ไม่ก็นำไปสืบค้นในระบบ Internet
5. ทดสอบ ประมวลผล และปรับปรุงแก้ไข : มีการ Testing Model แต่ละตัวว่ามีค่า accuracy เท่าไหร่ เมื่อเทียบระหว่างผล prediction กับสิ่งที่เกิดขึ้นจริง ค่าดังกล่าว รวมถึงกราฟที่เกิดขึ้นขณะการฝึก สามารถนำมาใช้ตีความว่า model กำลัง overfitting กับข้อมูลชุด train หรือไม่ ซึ่งจะนำมาสู่

การปรับปรุงแก้ไขสถาปัตยกรรมของ model ในเวอร์ชันต่อไป เมื่อได้ model ที่ดีพอจึงนำไป deploy ด้วย tkinter อีกที

6. นำเสนอผลงาน : อยู่ในรูปคลิปนำเสนอ

2.2 แนวคิดเชิงคำนวณ (Computational Thinking)

ได้มีการใช้แนวคิดเชิงคำนวณในการพัฒนาอัลกอริทึม

1. Decomposition : เรามีการแบ่งย่อยปัญหาที่ได้ว่าไว้ในกระบวนการทางวิศวกรรมเป็น 2 ปัญหาย่อย คือ 1.ปัญหาการ Mapping จาก Feature (X) สู่ Output (Y) และ 2.ปัญหาการแสดงให้เห็นของ feature จำเป็นที่ต้องพิจารณา (ในที่นี้ feature จะเป็น Attribute ทั้งหมดของ Dataset สำหรับเรือโททานิค) ปัญหาย่อยทั้งสองเกิดขึ้นเมื่อเราพิจารณาลึกลงไปถึงการใช้งานโปรแกรมของเราโดย user

2. Abstraction : เป็นการตัดรายละเอียดให้เหลือแต่รายละเอียดที่สำคัญและจำเป็นของโปรแกรมออก โดยภายหลังจากการคิดอย่างต่อเนื่อง โดยในปัญหาย่อยแรก เราได้แบ่งส่วนการประมวลผลหลักทั้งหมด 3 ส่วน คือ Preprocessing data , Training and Validating model และ Testing model ในแต่ละส่วนจะมีการแบ่งย่อยการประมวลผลลงไปอีก ซึ่งจะกล่าวถัดไปในบทที่ 3 ส่วนปัญหาที่สอง เราก็ได้แยกปัญหาของการลงมือสร้าง Prototype program เป็น 2 ส่วนคือ UI และการประมวลผลด้านหลัง (ซึ่งใช้ AI ที่เรา Train มา และฟังก์ชันเสริมอีกเล็กน้อย)

3. Pattern Recognition : ใน Part ของการ Preprocessing และการสร้าง UI เราได้มีการจับรูปแบบซ้ำเดิมหรือใช้บ่อยๆ ในขั้นตอนการพัฒนาออกมาสร้างเป็นฟังก์ชันแยกไว้ เพื่อจะได้ทำให้การใช้งานสะดวกยิ่งขึ้น เช่น `plot_loss(x)` และ `plot_accu(x)`

4. Algorithm design : เมื่อจับการทำงานซ้ำๆ มาเป็นฟังก์ชัน เราก็แค่สร้างขั้นตอนการทำงานของโปรแกรม (ในที่นี้คือการ Preprocessing) ออกมา สำหรับขั้นตอนการทำงานของโปรแกรม เราจะแบ่งเป็น เชิงภาพรวม กับ เชิงการประมวลผล ซึ่งจะอธิบายถัดไปในบทที่ 3

2.3 Document และข้อมูลประกอบในการพัฒนาอื่นๆ เกี่ยวกับการพัฒนา Code

เอกสารเกี่ยวกับวิธีการปรับ Accuracy ของ Model

<https://www.kdnuggets.com/2020/08/tensorflow-model-regularization-techniques.html>

เอกสารเกี่ยวกับการ normalized data

<https://towardsdatascience.com/which-models-require-normalized-data-d85ca3c85388>

dataset ที่ใช้

<https://www.kaggle.com/datasets/vinicius150987/titanic3?resource=download>

อธิบายรายละเอียดและที่มาของ feature ต่างๆ ใน dataset

<https://hub.packtpub.com/introduction-titanic-datasets/>

ข้อมูลและ code ที่ได้รับคำแนะนำจาก ChatGPT (ระบบ AI จากบริษัท OpenAI)

<https://chat.openai.com/chat/>

-เนื้อหา chat และคำแนะนำ จาก ChatGPT:

https://github.com/PeterLOVANAS/Titanic-machine-learning-project/blob/main/ChatGPT_conver_this_proj.jpg

2.6 งานวิจัยที่เกี่ยวข้อง

1.เว็บไซต์ซึ่งเป็นโครงสร้างหลักสำหรับ Random Forest Classification Model :

[https://medium.com/analytics-vidhya/random-forest-on-titanic-dataset-](https://medium.com/analytics-vidhya/random-forest-on-titanic-dataset-88327a014b4d)

[88327a014b4d](https://medium.com/analytics-vidhya/random-forest-on-titanic-dataset-88327a014b4d) โดยเว็บนี้และ code ทั้งหมดถูกจัดทำขึ้นโดย Carlos Raul Morales ซึ่งมีหัวข้อเกี่ยวกับการสร้าง Random Forest Classification Model เพียงแต่ใช้ dataset อีกชุดหนึ่ง

2.วิธีการสร้าง Machine Learning Model ต่างๆ แต่เป็น dataset คนละชุด

<https://www.kaggle.com/competitions/titanic/code>

3.วิธีการสร้าง Machine Learning ด้วย Support Vector Machine (ใช้แค่แนวคิด)

<https://www.datacamp.com/tutorial/tutorial-kaggle-competition-tutorial-machine-learning-from-titanic#training>

บทที่ 3

วิธีดำเนินการ

3.1 เครื่องมือที่ใช้ในการทำโครงการ

- 1) เว็บไซต์ Google Colab สำหรับการ Preprocessing, Training และ Testing model
- 2) PyCharm Edu 2021.3.3 สำหรับการพัฒนาโปรแกรม Prototype และการ Deployment
- 3) MacBook Pro 2020 (Apple M1) สำหรับเป็นศูนย์กลางการพัฒนา code
- 4) iPad Pro 2020 สำหรับการจดบันทึก idea และหาข้อมูล
- 5) รายชื่อ Library ที่ใช้ตั้งแต่ขั้นการพัฒนา Model จนถึงการสร้าง Prototype

การ Preprocessing และ Visualization

1. pandas (ใช้ดึงข้อมูลและบริหาร dataset ที่ได้มาให้อยู่ในรูป dataframe)
2. numpy (ใช้แปลง dataframe เป็น Numpy Array)
3. matplotlib (ใช้สร้างกราฟในการ visualize การทำงานของ model)
4. sklearn (ใช้ shuffle ข้อมูล)
5. seaborn (ใช้สร้างกราฟในการ visualize ข้อมูล)
6. tensorflow (ใช้สร้าง Y-vector)

การ Training และ Testing Model

1. tensorflow (ใช้ในการสร้าง , Train ,Test และ Save Deep Learning Model)
2. math (สำหรับการประมวลผลค่าบางอย่าง)
3. sklearn (ใช้ในการสร้าง , Train และ Test Random forest classifier Model)
4. pickle (ใช้ save model Random forest classifier Model)

การสร้าง UI และ Deployment

1. tkinter (ใช้สร้าง GUI)
2. pickle (ใช้ load Random Forest Classification model)
3. tensorflow (ใช้ load Deep Learning model)

*บาง Library อาจใช้งานในหลายส่วน เช่น tensorflow ซึ่งใช้ในการ Train และการ Deployment

3.2 กระบวนการออกแบบเชิงวิศวกรรม (Engineering Design Process)

1) ระบุปัญหา

5W1H

1. Who:

ผู้คนที่ต้องการรู้ระดับความเสี่ยงและเตรียมตัวรับมือในการออกเดินทางด้วยเรือสำราญ

2. What:

ไม่มีเครื่องมือในการช่วยวิเคราะห์ระดับความเสี่ยงของตนเองที่ควรระวังได้

3. When:

เวลาที่ผู้คนกำลังเลือกตัดสินใจเลือกห้องในเรือ สัมภาระหรือปัจจัยอื่นๆ เมื่อพวกเขาได้เลือกเรือที่จะใช้แล้ว

4. Where:

เกิดขึ้นในจุดที่พวกเขายังสามารถเตรียมตัวได้ เช่นที่บ้าน หรือบริเวณที่เข้าถึงระบบ Internet

5. Why:

5.1. ข้อมูลส่วนมาก อยู่ในรูปของบทความและคำแนะนำธรรมดา ทำให้ผู้ใช้เห็นภาพ ไม่ชัดเจน

5.2. ข้อมูลมีความซับซ้อนมาก ทำให้ค่อยมีการสร้างโปรแกรมมาช่วยมนุษย์ในประเด็นนี้

6. How:

6.1. Machine Learning Model โดยใช้วิธีการเรียนรู้แบบ Supervised Learning เพื่อใช้ predict ค่าความน่าจะเป็นในการรอดจากเหตุการณ์ภัยพิบัติทางเรือ

6.2. Visualization Model ภาพกราฟแสดง feature ที่มีอิทธิพลต่อการอยู่รอดของผู้โดยสารเมื่อเกิดภัยพิบัติทางเรือ การสร้าง Visualization ต้องผ่านกระบวนการทาง Data Science ก่อน

2) รวบรวมข้อมูลและแนวคิดที่เกี่ยวข้องกับปัญหา

เราได้มีการเก็บรวบรวมข้อมูลที่เกี่ยวข้องกับกระบวนการทำและข้อมูลที่เกี่ยวข้องกับผู้โดยสารของเรือ และความเสี่ยงเมื่อเกิดภัยพิบัติ โดยชุดข้อมูล (Dataset) ที่เราเลือกมานั้นเป็น dataset ของเรือ Titanic ซึ่งเป็นเรือสำราญที่มีชื่อเสียงมาจากโศกนาฏกรรมเมื่อปี ค.ศ.1912 การที่เราเลือก dataset ดังกล่าวก็เป็นเพราะมันเป็นชุดข้อมูลที่มีความครบถ้วน สามารถนำมาใช้วิเคราะห์เพื่อตอบปัญหาเป็นตัวอย่างได้

Attributes ของผู้โดยสารแต่ละคนจากชุดข้อมูลของเรือไททานิค สามารถเป็นต้นแบบให้เห็นว่าผู้โดยสารจะสามารถเตรียมตัวได้อย่างไร Attributes บางอย่างเช่น จำนวนสมาชิกในครอบครัวที่จะไปด้วยหรือห้องที่เลือก สามารถปรับเปลี่ยนเพื่อเพิ่มโอกาสการรอดชีวิต (ลดอัตราความเสี่ยง) ของผู้โดยสารได้ ในขณะที่บางตัวก็เป็นข้อมูลจำเพาะของผู้โดยสารเองเช่น อายุ เพศ (sex) เป็นต้น กล่าวโดยสรุปว่าการเตรียมตัวของผู้โดยสารล่วงหน้าก็คือการปรับ Attributes ของพวกเขาเหล่านั้นให้มีโอกาสรอดมากที่สุด

สำหรับข้อมูลในการพัฒนาเช่น Library ที่ต้องใช้ หรือกระบวนการพัฒนาเบื้องต้น ส่วนมากจะได้รับการศึกษาผ่าน Courses จากเว็บไซต์ DataCamp ซึ่งสอนการทำ Machine Learning model และ Data Science ข้อสงสัยต่างๆ ที่เกิดขึ้น เช่นนิยามคำศัพท์เฉพาะหรือคำแนะนำต่างๆ เราก็จะสอบถาม ChatGPT ซึ่งเป็นระบบ AI จากบริษัท OpenAI ที่เปิดให้ใช้ฟรีในช่วงเวลาที่เรากำลังพัฒนาโครงการนี้

3) ออกแบบวิธีการแก้ปัญหา

อย่างที่กล่าวไปในขั้น How ใน 5W1H นั้น เราพบว่าเรามีสองแนวทางในการแก้ปัญหา โดยแนวทางทั้งสองจะถูกนำไป apply กับ dataset ของเรือไททานิคเพื่อเป็นโครงการตัวอย่างหรือ Prototype ต่อไป

ต่อจากนี้คือข้อดีและข้อเสียของทั้งสองแนวทาง

แนวทางการแก้ปัญหา	ข้อดี	ข้อเสีย
1. Machine Learning Model เพื่อใช้ predict ความน่าจะเป็นในการรอดชีวิต	<ol style="list-style-type: none"> ผู้โดยสารไม่จำเป็นต้องมีความรู้ในการวิเคราะห์ข้อมูลมาก ก็สามารถใช้ได้ สามารถใช้งานได้อย่างสะดวกและรวดเร็ว ในอนาคต หากมี dataset ชุดใหม่ที่ครอบคลุมขึ้น ก็สามารถ Update ได้ง่าย 	<ol style="list-style-type: none"> AI มีโอกาสทำนายผิดพลาด ทำให้เกิดความสับสนแก่ผู้ใช้งาน หากข้อมูลหรือสถาปัตยกรรมไม่ดี AI ก็ทำงานได้ไม่ดีตาม
2. Data Analysis and Visualization เพื่อสรุป feature ที่สำคัญทางสถิติ	<ol style="list-style-type: none"> เป็นการส่งต่อข้อมูลโดยตรง หากข้อมูลถูกต้องก็ไม่มีผิดพลาดใดๆ ขั้นตอนการพัฒนาสั้นและง่ายกว่า 	<ol style="list-style-type: none"> การพัฒนาไม่ยืดหยุ่น เพราะหากมีข้อมูลใหม่ ก็อาจต้องทำการวิเคราะห์หรือ Visualization ใหม่ทั้งหมด ผู้ใช้อาจตีความ Visualization หรือผลวิเคราะห์ผิดพลาดได้

ตารางที่ 1 แนวทางการแก้ปัญหา

จากผลการวิเคราะห์พบว่าแนวทางที่ 1 ซึ่งก็คือการสร้าง Machine Learning Model นั้นเป็นแนวทางที่มีประสิทธิภาพมากกว่าในการแก้ปัญหา เพราะความยืดหยุ่นในการพัฒนา ต่อยอด รวมถึงความสะดวกและชัดเจนในการใช้งานของ user อีกด้วย แม้ว่าแนวทางนี้จะมีข้อเสีย แต่เราก็สามารถแก้ไขได้ด้วยการทำ preprocessing ที่ดีและการ train Parameters และเลือก Hyperparameters ให้ model มีผลการทำนายที่แม่นยำมากยิ่งขึ้น

4) วางแผนและดำเนินการแก้ปัญหา

(ขั้นตอนการทำงานและการพัฒนาโปรแกรมจะอธิบายอย่างละเอียดใน Computational thinking และ Machine Learning Models)

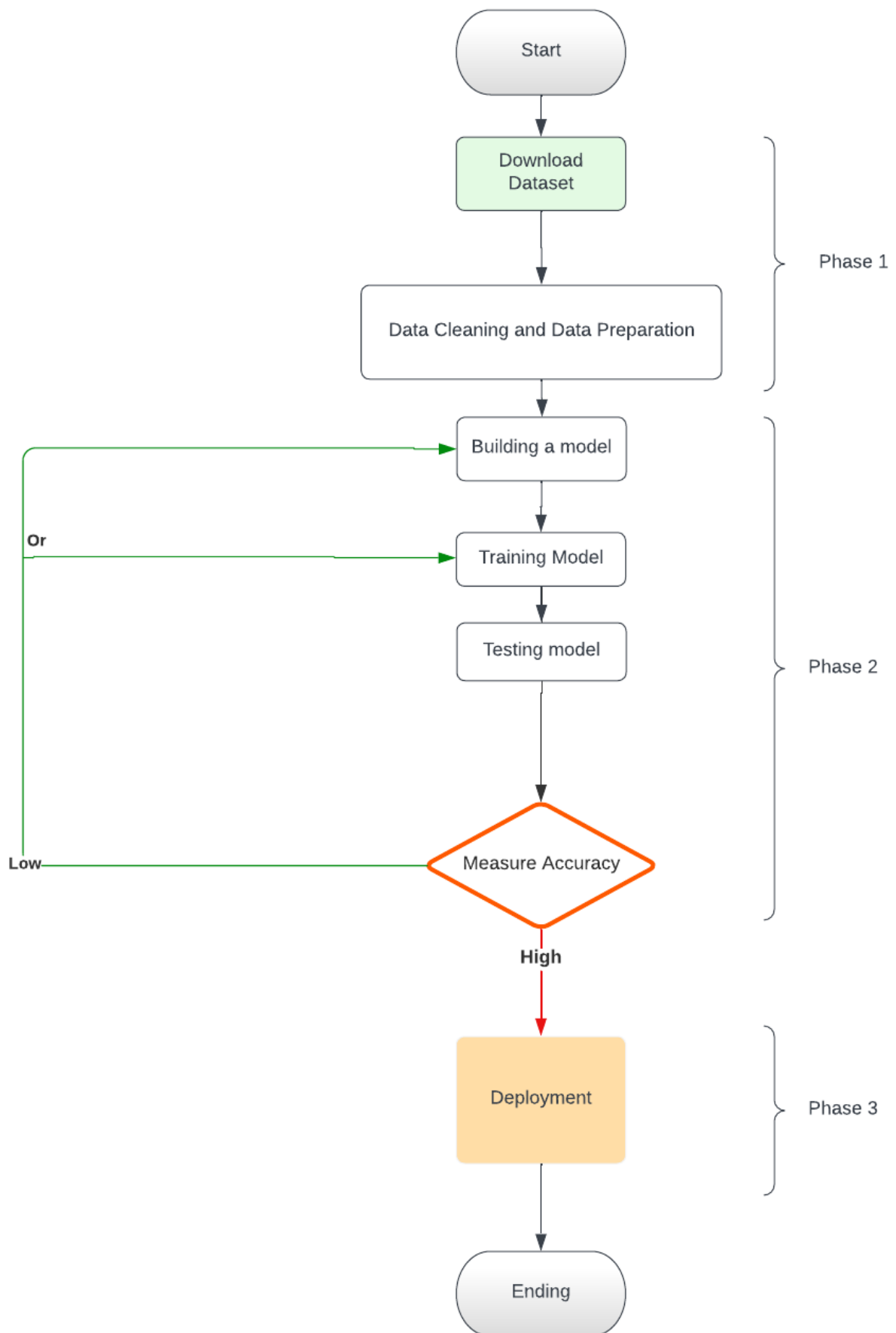
ภายหลังจากที่เราทราบแนวทางการแก้ปัญหาแล้ว เราจึงพัฒนาขั้นตอนการพัฒนาโปรแกรมของเราขึ้นมา (Algorithm เชิงภาพรวมในการพัฒนา)

ในการพัฒนา เราได้ตัดสินใจเลือกใช้ Google Colab เป็นฐานในการเขียนโปรแกรมในการ Preprocessing และ Train ML ของเรา เพราะว่า Colab ใช้ระบบการเขียนโปรแกรมแบบ Notebook ซึ่งเราสามารถเลือก run ส่วนย่อยของ code (cell) ได้ อีกทั้งยังบันทึกผลลัพธ์ไว้ให้อีกด้วย นอกจากนี้เนื่องจากเราใช้ Colab Pro ณ ขณะที่เราทำโครงการ ทำให้เราเข้าถึงพลังประมวลผลที่สูงขึ้น ทำให้การ Train ของเราสามารถทำได้ด้วยความรวดเร็ว อีกประเด็นหนึ่งที่ทำให้ Colab Pro เหมาะสมกับการทำงานของเราก็คือ การที่มี Library เตรียมไว้ให้เราหมดแล้ว อีกทั้งยังสามารถเชื่อมกับ Google Drive เพื่อ Save File ได้อย่างง่ายดาย

เมื่อ Train ML ของเราจนเสร็จสิ้นแล้ว เราจึงดาวน์โหลด Model ของเราลงมายัง Computer ส่วนตัว เพื่อนำมา Deployment หรือการขึ้นโครงโปรแกรมของเราจริงๆ ในการพัฒนา GUI และเบื้องหลังของโปรแกรม Prototype ของเรา เราจะทำที่ PyCharm IDE ซึ่งเหมาะกว่า Google Colab ในการพัฒนา GUI และเมื่อทำเสร็จ เราก็จะได้ File.py ของ Prototype เรา

ในการดำเนินการโครงการนี้เราจะแบ่ง Phase การทำงานออกเป็น 3 Phases:

1. Preprocessing phase (1): Load ข้อมูล , Data Cleaning , แบ่งกลุ่มข้อมูล , เลือก Feature
2. Processing phase (2):
Training และ Testing Deep Learning Model และ Random Forest Classification Model
3. Deployment phase (3): สร้างโปรแกรมที่มี GUI โดยใช้ Model ที่สร้างมาในการทำนาย



ภาพที่ 1 ขั้นตอนการพัฒนาตั้งแต่ต้นจนจบ

ในแผนภาพ Flowchart (ภาพที่ 1) ขั้นตอนตั้งแต่ Download Dataset จนถึง Measure Accuracy จะทำที่ Google Colab ส่วนขั้น Deployment จะทำที่ PyCharm IDE ในการพัฒนา Model ต่างๆ เราจะมี test ด้วยข้อมูลชุดทดสอบ หาก Accuracy สูงพอ (e.g. 0.79 , 0.82) ก็อาจจะนำไปใช้ต่อในโปรแกรมจริง แต่หากยังไม่สูงมาก (e.g. 0.53) ก็จะนำไปสู่การสร้าง Model ใหม่หรืออาจจะการ Train ใหม่ด้วยการปรับ Hyperparameter บางตัว หรือใช้ข้อมูลชุดที่ผ่านการ Normalized

ภายหลังจากการวางแผนเราก็มีการลงมือพัฒนา AI ใน Phase 1 และ 2 ซึ่งผลลัพธ์จากการดำเนินงานจะถูกเก็บไว้ใน Repository ของ GitHub ของ Account: Peter Lovanas (หัวหน้ากลุ่ม) : <https://github.com/PeterLOVANAS/Titanic-machine-learning-project>

โดยไฟล์ของ Colab Notebook ที่ใช้ในตลอด Phase 1-2 จะอยู่ในชื่อ
Copy_of_Copy_of_Titanic_firstML.ipynb

เมื่อ Train AI เสร็จสิ้นแล้วเราก็จะเข้าสู่ Phase 3 ซึ่งเป็นการ Deployment โดยเราจะได้ code ฉบับสมบูรณ์ดังนี้

```
import tensorflow as tf
from tkinter import *
from tkinter import ttk
import pickle
import numpy as np

# [pclass , sex , age , sibsp , parch , fare , cabin , embarked]
# Fix index of each specific features.

def replace_embarked(x):
    if x == "Southampton":
        return 0
    elif x == "Cherbourg":
        return 1
    elif x == "Queenstown":
        return 2

def replace_cabin(x):
    dic_cabin = {'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'T': 7}
    return dic_cabin[x]

def define_sex(x):
    if x == "male":
        return 1
    elif x == "female":
        return 0
```

```

def predict_state(arr):
    for p in arr:
        if p[0] > 0.5 and p[0] > p[1]:
            return "Die"
        elif p[1] > 0.5 and p[1] > p[0]:
            return "Survive"

def instance():
    pclass = float(pcc.get())
    sex = float(define_sex(sexc.get()))
    age = float(age_et.get())
    sibsp = float(sibet.get())
    parch = float(paret.get())
    fare = float(faet.get())
    cabin = float(replace_cabin(cabc.get()))
    emb_bf = emc.get()
    embark = replace_embarked(emb_bf)
    person = np.array([[pclass , sex , age , sibsp , parch , fare
, cabin , embark]])
    return person

def run_model():
    model_path_1 = "modelV5_2_2D_cop1.h5"
    model_path_2 = "modelV5_3_1D_cop1.h5"
    model_path_3 = "Model_RF_VB_1D.pkl"
    data = instance()
    ch = modc.get()
    result_et.delete(0,END)
    if ch == "Model V5":
        modelV5_2 = tf.keras.models.load_model(model_path_1)
        predictions = modelV5_2.predict(data)
        result_et.insert("", f"{predictions[0][1]*100:.2f}")
    elif ch == "Model V5 (Norm)":
        modelV5_3 = tf.keras.models.load_model(model_path_2)
        predictions = modelV5_3.predict(data)
        result_et.insert("", f"{predictions[0][1]*100:.2f}")
    elif ch == "Model VB (RF)":
        modelRF_2 = pickle.load(open(model_path_3 , "rb"))
        predictions = modelRF_2.predict(data)
        result_et.insert("", f"{predictions[0][1]*100:.2f}")

def Clear():
    age_et.delete(0,END)
    sibet.delete(0 ,END)
    paret.delete(0,END)
    faet.delete(0 ,END)

```

```

Home = Tk()
Home.geometry("1000x1000")
style = ttk.Style()
style.configure("Bold.TLabel", font=("Times", 9, "roman"))
frame = Frame(Home)
frame.pack()

# model drop down menu
model_tup = ("Model V5" , "Model V5 (Norm)" , "Model V2 (RF)" )
l1 = StringVar()
modc = ttk.Combobox( width = 30, textvariable = l1,
state='readonly',font=("Times", 15, "roman"))
modc["values"] = model_tup
modc.place(x= 400 , y= 10)
modc.current(0)
lab_model = Label(text = "Models",font=("Times", 15, "roman"))
lab_model.place(x = 320, y =10)

# passenger class drop down menu
pclass_tup = ("1" , "2" , "3")
l2 = StringVar()
pcc = ttk.Combobox( width = 30, textvariable = l2,
state='readonly',font=("Times", 15, "roman"))
pcc["values"] = pclass_tup
pcc.place(x = 400 , y = 50)
pcc.current(0)
lab_pc = Label(text = "Passenger class",font=("Times", 15,
"roman"))
lab_pc.place(x = 300 ,y =50)

# sex drop down menu
sex_tup = ("male" , "female")
l3 = StringVar()
sexc = ttk.Combobox( width = 30, textvariable = l3,
state='readonly',font=("Times", 15, "roman"))
sexc["values"] = sex_tup
sexc.place(x = 400 , y = 90)
sexc.current(0)
lab_sex = Label(text = "Sex",font=("Times", 15, "roman"))
lab_sex.place(x = 320 , y= 90)

# age entry
age_et = ttk.Entry(frame, width=90,font=("Times", 15, "roman"))
age_et.pack(pady = 130 , padx = 28)# y= 130 , x = 28
lab_age = Label(text = "Age",font=("Times", 15, "roman"))
lab_age.place(x =300 , y= 130)

# Sibsp entry
sibet = ttk.Entry(frame, width=90,font=("Times", 15, "roman"))
sibet.place(x = 110 , y =170)

```

```

lab_sib = Label(text = "Number of Siblings and
Spouses",font=("Times", 15, "roman"))
lab_sib.place(x = 225, y= 170)

# parch entry
paret = ttk.Entry(frame, width=90,font=("Times", 15, "roman"))
paret.place(x = 110 , y= 210)
lab_par = Label(text = "Number of Parents and
child",font=("Times", 15, "roman"))
lab_par.place(x = 230 , y= 210)

# fare entry
faet = ttk.Entry(frame, width=90,font=("Times", 15, "roman"))
faet.place(x = 110 , y = 250)
lab_faet = Label(text = "Fare",font=("Times", 15, "roman"))
lab_faet.place(x = 350 , y = 250)

# cabin drop down menu
cab_tup = ("A" , "B" , "C" , "D" , "E" , "F" , "G" , "T")
l4 = StringVar()
cabc = ttk.Combobox( width = 30, textvariable = l4,
state='readonly',font=("Times", 15, "roman"))
cabc["values"] = cab_tup
cabc.place(x = 400 , y = 290)
cabc.current(0)
lab_cab = Label(text = "Cabins level",font=("Times", 15,
"roman"))
lab_cab.place(x = 300 , y = 290)

# embarked drop down menu
em_tup = ("Southampton" , "Cherbourg" , "Queenstown")
l5 = StringVar()
emc = ttk.Combobox( width = 30, textvariable = l5,
state='readonly',font=("Times", 15, "roman"))
emc["values"] = em_tup
emc.place(x = 400 , y = 330)
emc.current(0)
lab_em = Label(text = "Embarked",font=("Times", 15, "roman"))
lab_em.place(x = 325 , y= 330)

# result (Survived / died)
result_et = ttk.Entry(frame, width=40,font=("Times", 25,
"roman"))
result_et.pack(pady = 130 , padx = 28)
lab_result = Label(text = "This person survival chance
is",font=("Times", 15, "roman"))
lab_result.place(x = 260 , y = 430)
lab_result2 = Label(text = "%",font=("Times", 15, "roman"))
lab_result2.place(x = 1000 , y= 430)

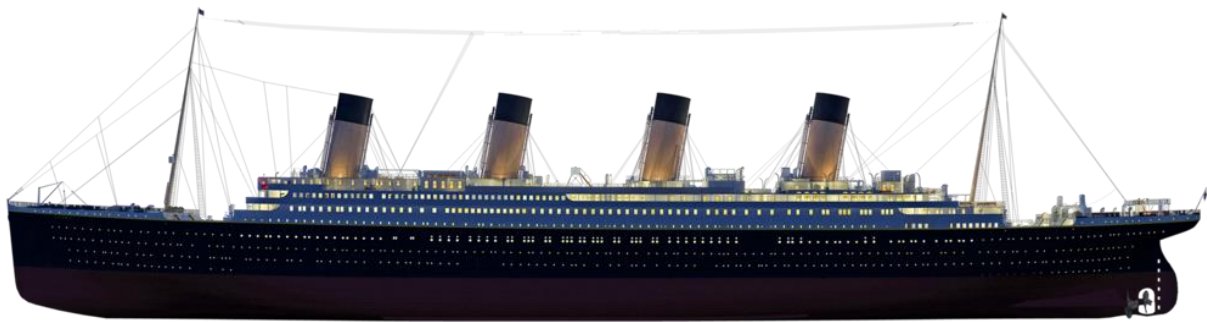
```

```
# Clear button
clear_but = Button(Home , text = "clear" , command= Clear)
clear_but.place(x = 800 , y = 370) # x = 500 , y = 370

# run model
run_but = Button(Home , text = "Run Model" , command= run_model)
run_but.place(x = 500 , y = 370)

# Image of Titanic
photo = PhotoImage(file = "NicePng_cargo-ship-png_513814.png" )
phot = Label(Home , image= photo , width = 1000 , height = 250)
phot.place(x = 300 , y = 500)

title = Label(text = "Titanic : Machine Learning from disaster",
font=("Times", 50, "roman"))
title.place(x = 300 , y = 800)
frame.mainloop()
```



NicePng_cargo-ship-png_513814.png

5) ทดสอบ ประเมินผล และปรับปรุงแก้ไข

ดังที่ได้กล่าวไปว่า ระหว่างการทำงานใน Phase 2 จะมีการทดสอบ Model ด้วย Testing dataset เพื่อวัด Accuracy ว่าสูงดีหรือไม่ หากยังไม่ดีพอก็มีการพัฒนาและปรับปรุงแก้ไขใน Model รุ่นต่อไป นอกจากการทดสอบใน Phase 2 แล้วยังมีการทดสอบย่อยใน Phase 3 เพื่อประเมินว่า Data พร้อมที่นำไปใช้ในการ Train หรือไม่ และเมื่อการสร้าง Prototype เสร็จลง เราก็จะมีการทดสอบใช้ โปรแกรมของเราอีกที

6) นำเสนอ วิธีการแก้ปัญหา ผลการแก้ปัญหาหรือชิ้นงาน

วิธีแก้ปัญหาก็ได้รับการนำเสนอในผลการดำเนินงาน หรือบทที่ 4

3.3 แนะนำ Dataset ที่ใช้ในการพัฒนาในโครงการ (Introduction to Titanic dataset)

เราได้นำ Titanic dataset ชุดนี้มาจาก

<https://www.kaggle.com/datasets/vinicius150987/titanic3?resource=download>

ซึ่งภายในจะเป็นตารางข้อมูลของผู้โดยสารจำนวน 1309 คน และมี 15 attributes ของผู้โดยสาร (มี 1309 records และ 15 fields) ภายใน dataset มีข้อมูลว่าง (NULL) จำนวนมาก และข้อมูลบางส่วนก็อยู่ในรูป String ซึ่งไม่สามารถนำไป Train ML ได้จนกว่าจะถูกแปลงเป็นข้อมูลประเภท float

ด้วยเหตุนี้เราจึงต้องมีการ Preprocessing หรือการเตรียมข้อมูลให้พร้อมก่อนนำไปพัฒนา AI ซึ่งในขั้นตอนดังกล่าวก็จะประกอบไปด้วย

1. Data Cleaning
2. Feature Extraction
3. Conversion each record to feature vector (String to Float)
4. Data Splitting (Train, Validation, Testing)
5. Normalization (เป็นข้อมูลแยกอีกชุด)

จากการดำเนินงานใน Phase 1 เราพบว่า Attribute สำคัญมีอยู่เพียง 12 attributes

PassengerId	0
pclass	0
survived	0
name	0
sex	0
age	0
sibsp	0
parch	0
ticket	0
fare	0
cabin	0
embarked	0

ภาพที่ 2 Attribute ทั้ง 12 หลัง Data Cleaning

1. PassengerId : เลขประจำตัวของผู้โดยสารเรือไททานิกที่เรียงจาก 1 ถึง 1309
2. pclass : ย่อมาจาก Passenger Class โดยในเรือไททานิกจะแบ่งคลาสผู้โดยสารเป็น 3 คลาสคือ ผู้โดยสารชั้น 1 (1) , ผู้โดยสารชั้น 2 (2) และ ผู้โดยสารชั้น 3 (3)
3. survived : บ่งบอกชะตากรรมของผู้โดยสารว่า รอดชีวิต (1) หรือ ตาย (0)

4. name : ชื่อผู้โดยสาร
5. sex : เพศของผู้โดยสาร ; male/female
6. age : อายุผู้โดยสารอยู่ในข้อมูลแบบ float
7. sibsp : ย่อมาจาก Siblings OR Spouses ซึ่งก็คือจำนวนพี่น้องหรือคู่สมรสที่เดินทางไปด้วย
8. parch : ย่อมาจาก Parents OR Child ซึ่งก็คือจำนวนพ่อแม่หรือลูกที่ไปเดินทางไปด้วย
9. ticket : รหัสตัวโดยสารของเรือไททานิค (อยู่ในรูป String)
10. fare : ค่าตั๋วเรือ (British Pound) อยู่ในรูป float
11. cabin : เรือไททานิคมี deck หรือชั้นของเรืออยู่ 8 ระดับคือ A,B,C,D,E,F,G,T ข้อมูลใน Attribute จะอยู่ในรูปรหัสของห้องที่ผู้โดยสารเข้าไว้ขณะเดินทาง ซึ่งในรหัสจะมีอักษร deck เป็นส่วนหนึ่ง ข้อมูลชุดนี้อยู่ในรูป String
12. embarked : เรือไททานิคมีการรับคนขึ้นเรือที่เมืองต่างๆ 3 เมืองคือ Southampton (S) , Cherbourg (C) , Queenstown (Q) ข้อมูลในชุดนี้อยู่ในรูป String

ใน Attribute 12 อันนี้ เราจะตัด PassengerId , name และ ticket ออกจากการเป็น feature เพราะไม่มีประโยชน์ในการ Train และอาจทำให้ผลลัพธ์ผิดพลาด ส่วน survived จะถูกแยกออกไปเป็น target vector เมื่อขั้นตอนการ Preprocessing ที่ 1-3 เราจะได้ Numpy Array ขนาด (1309,8) และจำนวน feature ทั้งหมดที่ใช้ในการ train จะมีแค่ **8 features** เท่านั้น

3.4 ขั้นตอนการแก้ปัญหาตามแนวคิดเชิงคำนวณ (Computational Thinking)

1) การแยกส่วนประกอบและย่อยปัญหา (Decomposition)

ปัญหาย่อยที่ 1: ปัญหาการ Mapping จาก Feature สู่ Output

สามารถแยกเป็นปัญหาย่อยลงไปอีกได้ทั้งหมด 3 ปัญหาคือ

1. ส่วนการทำ Preprocessing data
2. ส่วนการ Training และ Validating Model
3. ส่วนการ Testing

โดยในปัญหาที่ 2 และ 3 จะใช้ข้อมูลที่ผ่านมาจากการแก้ปัญหาที่ 1 มาแล้ว

ปัญหาย่อยที่ 2: ปัญหาการสร้าง Prototype

สามารถแยกเป็นปัญหาย่อยลงไปอีกได้ทั้งหมด 3 ปัญหาคือ

1. ส่วนการสร้าง GUI
2. ส่วนการประมวลผลข้อมูลนำเข้าจาก GUI
3. ส่วนการนำ ML model มาใช้ทำนาย

2) การคิดเชิงนามธรรม (Abstraction)

สิ่งที่มีความจำเป็นต่อการทำงานของโปรแกรมและสิ่งที่เราต้องคำนึงต่อการพัฒนาของโปรแกรม หรือที่เรียกว่า Essential feature ของการทำงานทั้งหมด

1.ส่วนการทำ Preprocessing

- ส่วนการ Load dataset
- ส่วนการแปลงข้อมูลจาก Attribute ต่างๆ จาก String เป็น float
- ส่วนการสกัด Feature (Feature Extraction)
- ส่วนการแทนข้อมูลที่ว่าง ด้วยค่าทางสถิติจากทั้ง Attribute นั้นๆ
- ส่วนการแปลง dataframe เป็น Numpy Array
- ส่วนการแยก Attribute survive ออกมาเป็น Y-vector
- ส่วนการแบ่ง Array สำหรับการ Training, Validation, Testing
- ส่วนการแปลงข้อมูล raw เป็นข้อมูลที่ถูกระบุ Normalized

2.ส่วนการ Training , Validating และ Testing Machine Learning Model

- ส่วนการสร้างกราฟแสดงผลการเรียนรู้ (loss/accuracy) [Utility]
- *ใน Deep Learning Model แต่ละ Version จะมีส่วนการทำงานดังนี้
- ส่วนการออกแบบสถาปัตยกรรม Model (กำหนดจำนวน Neurons และ layers)

- ส่วนการ Training (กำหนด Hyperparameter, epochs, ข้อมูลที่ใช้ฝึก)
- ส่วนการ Testing
- ส่วนการ save model ลง Google drive
- ส่วนการสร้าง Output array (ผลลัพธ์จากการ predict)
- *ใน Random Forest Classification Model จะมีส่วนการทำงานดังนี้
 - ส่วนการแสดงผล Visualization ความสำคัญของแต่ละ Feature
 - ส่วนการ Training และการปรับ Hyperparameters
 - ส่วนการ save model ลง Google Drive

3.ส่วนการสร้าง Prototype

- ส่วนการสร้าง Graphic User Interface เพื่อรอ Input จาก user
- ส่วนการเลือก Model
- ส่วนการประมวลผลข้อมูลจาก GUI ให้อยู่ใน Feature vector
- ส่วนการ load model
- ส่วนการนำ ML model มาใช้ predict เป็น Output Array
- ส่วนการตีความ Output Array (Original Prototype)
- ส่วนการตีความความน่าจะเป็นในการรอดชีวิต (Current Prototype)

3) การหารูปแบบ (Pattern Recognition)

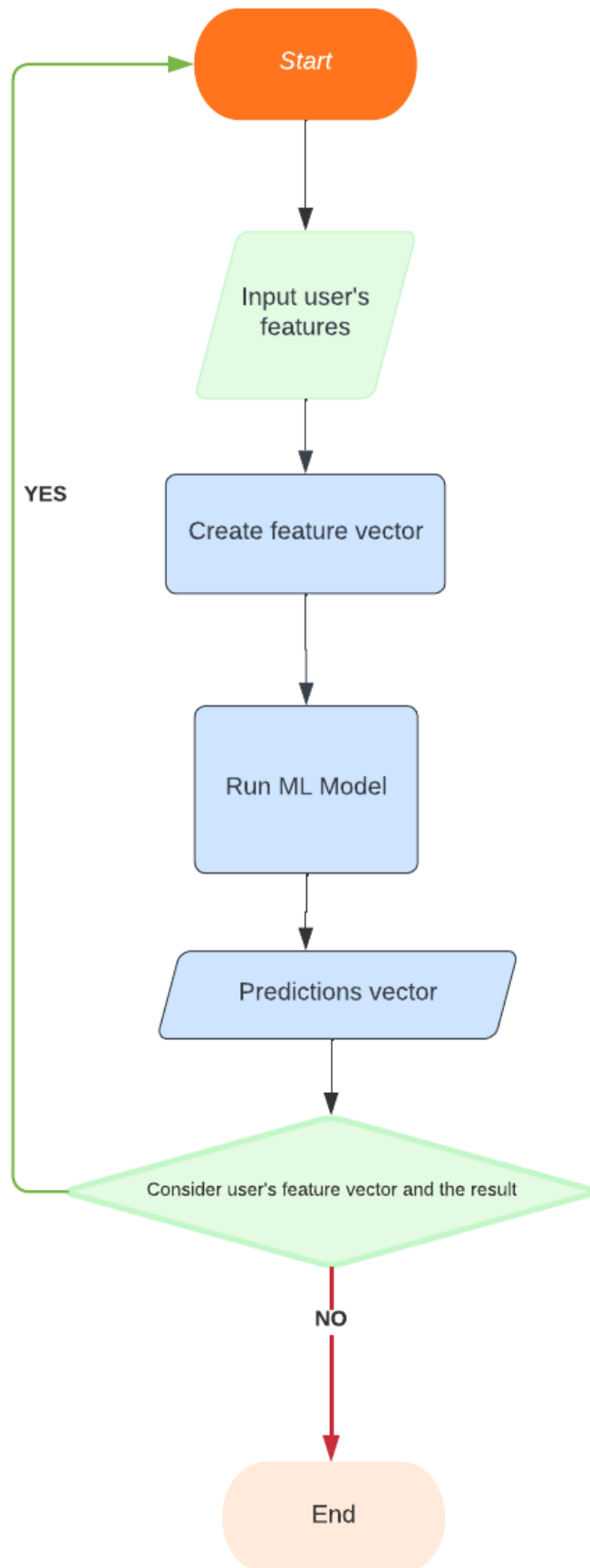
การทำงานบางส่วนของโปรแกรมสามารถจับมารวมไว้ที่เดียวกันได้ เราพบ Pattern ของ code จำนวนมากในทุก Phase ซึ่งเราสามารถจัดกลุ่มเป็นฟังก์ชัน เพื่อความสะดวกในการใช้งานและความเป็นหมวดหมู่ของ code เช่น ฟังก์ชัน `plot_loss(x)` และ `plot_accu(x)` ซึ่งทั้งสองชุดคำสั่งมีความจำเป็นที่จะต้องใช้งานภายหลังการ train model หลายต่อหลายครั้ง การจับ Pattern จะช่วยแก้ปัญหาสำคัญดังนี้

- โครงสร้างการสร้างกราฟวัดประสิทธิภาพของ Model ถูกจับไว้ที่เดียว
- ขั้นตอนการ Preprocessing บางส่วนถูกจับ Pattern แล้วแปลงเป็นฟังก์ชัน
- เป็นการจัดกลุ่ม code ให้เป็นระเบียบ เช่น ฟังก์ชันการประมวลผลของ Prototype หรือ ฟังก์ชันการแปลงค่า Probability ให้อยู่ใน $\{0,1\}$

4) การออกแบบอัลกอริทึม (Algorithm Design)

4.1) ขั้นตอนเชิงภาพรวม

ในระหว่างการพัฒนา เราได้มีการคิดขั้นตอนการทำงานของ Prototype เราอย่างคร่าวๆ ออกมาได้ดังนี้



ภาพที่ 3 ภาพรวมการทำงานของ Prototype และ flow การใช้งานของผู้ใช้

จาก Flowchart ที่ได้แสดงไป (ภาพที่ 3) ขั้นตอนการทำงานของ Prototype จะอยู่ในสีน้ำเงินอ่อน และขั้นตอนที่ผู้ใช้ต้องเป็นคนกระทำจะเป็นสีเขียว ซึ่งจากแผนภาพ เราจะสังเกตได้ว่า เมื่อ user ได้รับคำตอบจาก ML model ที่เลือกแล้ว ผู้ใช้จะต้องพิจารณาที่จะปรับ feature vector ของตน เพื่อให้ผลลัพธ์ (ค่าความน่าจะเป็นในการรอดชีวิต/ปลอดภัย) อยู่ในระดับที่สูงจนพอใจ ซึ่งหากพวกเขาไม่ประสงค์จะปรับค่า predictions แล้ว (No) ก็จะมีกระบวนการ แต่หากอยากปรับค่า predictions (Yes) ก็จำเป็นต้องย้อนกลับไปใส่ข้อมูลใหม่ลงระบบก่อน

4.2) ขั้นตอนเชิงการประมวลผล

ในขั้นตอนเชิงการประมวลผล เราจะอธิบายการทำงานของ Algorithm ในส่วนของการ Preprocessing (Phase 1) , Model Training and Testing (Phase 2) และการทำงานด้านหลังของ Prototype (Phase 3)

Phase 1: Preprocessing

LIBRARY : pandas , numpy , sklearn ,tensorflow

1. Loading data

```
SET data = CALL pandas.read_csv(path_of_dataset)
```

```
SET df = data.copy()
```

คำอธิบาย : code บรรทัดนี้ใช้ method pandas.read_csv() ในการดึง Titanic dataset ซึ่งอยู่ในรูปไฟล์ .csv ส่วน path_of_dataset คือที่อยู่ของไฟล์ dataset ซึ่งใน code จริงก็คือภายใน repository (Folder : dataset) ส่วน df คือตัวแปรของ dataframe ของชุดข้อมูลของเรา (ประเภทข้อมูลคือ pandas.dataframe)

2. Preprocessing on each attribute

a. sex

```
def to_numeric_sex():
```

```
    INPUT : df["col_name"]
```

```
    IF val_in_sex is equal to "male" THEN
```

```
        OUTPUT : 1
```

```
    ELSE IF val_in_sex is equal to "female" THEN
```

```
        OUTPUT : 0
```

```
SET df["sex"] = CALL to_numeric_sex(df["sex"])
```

คำอธิบาย : val_in_sex คือข้อมูลใน column: sex ซึ่งฟังก์ชัน to_numeric_sex(x) จะแปลงค่าภายใน sex ให้เป็น 0 หากเป็นหญิง และ 1 หากเป็นชาย บรรทัดสุดท้ายเป็นการก่อให้เกิดการเปลี่ยนแปลงในตัวแปร df

*df["col_name"] หมายถึงข้อมูลทั้ง column นั้นๆ (ทุก record)

b. age

```
SET df["age"] = df["age"].fillna(numpy.mean(df["age"]))
```

คำอธิบาย : Method df["col_name"].fillna(x) จะใช้ในการเติมค่าที่ว่างใน column นั้นๆ ในที่นี้คือ age โดยค่าที่เราเติมลงไปแทนจุดที่ว่างก็คือค่า mean ของอายุผู้โดยสารทั้งหมดที่มีใน column

c. embarked

```
SET df["embarked"] = df["embarked"].fillna("S")
```

```
def to_numeric_embarked():
```

```
    INPUT : df["col_name"]
```

```
    IF val_in_em is "S":
```

```
        OUTPUT : 0
```

```
    ELSE IF val_in_em is "C":
```

```
        OUTPUT : 1
```

```
    ELSE IF val_in_em is "Q":
```

```
        OUTPUT : 2
```

```
SET df["embarked"] = CALL to_numeric_embarked(df["embarked"])
```

คำอธิบาย : ในขั้นตอนแรก เราต้องเติมค่าที่ว่างด้วย "S" ก่อน (ค่าที่ common ที่สุด) จากนั้นค่อยใช้ฟังก์ชัน to_numeric_embarked(x) ในการแปลงตัวย่อของเมืองให้เป็นตัวเลข 0-2

d. cabin

```
SET df["cabin"] = df["cabin"].fillna("G6")
```

```
SET dic_cabin = {'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'T': 7}
```

```
def to_numeric_cabin():
```

```
    INPUT : df["col_name"]
```

```
    OUTPUT : dic_cabin.get_value(val_in_cab[first_index])
```

```
SET df["cabin"] = CALL to_numeric_cabin(df["cabin"])
```

คำอธิบาย : ในขั้นตอนแรกเราต้องเติมค่าที่ว่างด้วย “G6” ก่อน (ค่าที่ common ที่สุด) จากนั้นค่อยใช้ฟังก์ชัน to_numeric_cabin(x) ในการแปลงตัวอักษรแรกของแต่ละค่าข้อมูลใน column cabin เช่น “G6” เราจะดึงมาแค่อักษรตัวแรกซึ่งก็คือ G จากนั้นเราจะไปประมวลผลในฟังก์ชัน จนได้ว่าตัวเลขของ G คือ 6 เมื่อถึงบรรทัดสุดท้าย ทุกค่าข้อมูลจะกลายเป็นตัวเลขที่ยึดจากตัวอักษรแรกสุด (ชื่อ deck)

e. fare

```
SET df["fare"] = df["fare"].fillna(numpy.mean(df[pclass = 3]))
```

คำอธิบาย : เป็นการเติมค่าที่ว่างด้วย ค่าเฉลี่ยของ fare ในผู้โดยสารที่เป็นผู้โดยสารชั้นสาม (pclass = 3) ทั้งนี้เนื่องจากค่าที่ว่างของ column นี้เกิดขึ้นในผู้โดยสารชั้นสาม เราจึงควรเอาค่าเฉลี่ยของผู้โดยสารชั้นสาม แทนที่จะเป็นทุกคนเพื่อป้องกันข้อมูลที่ผิดเพี้ยนไป

3. Conversion to Numpy Array

a. Shuffle data

```
SET df = sklearn.shuffle(df)
```

คำอธิบาย : เราจำเป็นต้อง Shuffle ข้อมูลเนื่องจากใน dataset นี้มีการจัดเรียงผู้โดยสารโดยเรียงจากผู้โดยสารชั้น 1 ไป 3 ซึ่งเมื่อเราต้องแบ่งข้อมูลด้วย index อันใดอันหนึ่ง ปริมาณข้อมูลจะไม่สมดุลระหว่างข้อมูลการเรียนรู้และการทดสอบที่แยก

ออกมา เมื่อเรา shuffle record ต่างๆ จะกระจายอย่าง random ทำให้ไม่มีปัญหาดังกล่าว

b. Conversion to array

```
SET unim_feature = ["survived", "name", "ticket", "PassengerId"]
```

```
SET predicset = to_numpy(df.drop(unim_feature))
```

คำอธิบาย : ในขั้นแรกเรามีการตัด feature ที่เราจะไม่รวมไว้เป็น feature สำหรับการ train (X-vector) ออก ซึ่ง feature เหล่านั้นจะอยู่ในตัวแปร unim_feature (unimportant feature) เมื่อทำเสร็จเราก็แปลง df ที่ตัด column เหล่านั้นทิ้งให้กลายเป็น Numpy array ซึ่งเป็นรูปแบบข้อมูลที่จะเข้าไปใช้ train ML ได้ต่อไป

c. Creation of Y-vector (Target)

```
SET targetset = tensorflow.to_categorical(df["survived"])
```

คำอธิบาย : แปลง df["survived"] ให้อยู่ใน Array ที่มี 2 columns โดย column แรกจะเกี่ยวกับสถานะว่าตาย อีก column จะเกี่ยวกับสถานะรอดชีวิต หากค่าใน column ไหนเป็น 1 หมายถึงว่าสถานะนั้นเป็นจริง

4. Splitting dataset

a. Splitting between predictors and target

```
SET predictors = predicset[:892]
```

```
SET target = targetset[:892]
```

คำอธิบาย : predicset และ targetset ถูกแบ่งที่ index เดียวกัน นี่ก็เหมือนกับ การแบ่ง df ที่ index ที่ 892 แต่เพียงตอนนี้ข้อมูลอยู่ในรูป numpy array เท่านั้นเอง การเลือก index ด้วย [:892] หมายถึงเลือกมาตั้งแต่ record ที่มี index เท่ากับ 0 ถึง 891

b. Splitting predictors into predictors for training and for validation

```
SET ratio = math.trunc(length(predictors)*0.8)
```

```
SET predictors_train = predictors[:ratio]
```

```
SET predictors_valid = predictors[ratio:]
```

```
SET target_train = target[:ratio]
```

```
SET target_valid = target[ratio:]
```

คำอธิบาย : ในการแบ่งข้อมูลภายใน predictors ออกเป็น 2 ส่วนให้พร้อมสำหรับการใช้ train และ validation ต้องมีการตั้งค่า ratio หรือค่า index ที่ใช้แบ่ง ซึ่งฟังก์ชัน math.trunc() ได้ช่วยให้ค่า ratio เป็นจำนวนเต็ม (index ต้องเป็นจำนวนเต็ม) เมื่อแบ่งเสร็จข้อมูลที่จะใช้ train จริงๆ ก็คือ predictors_train , target_train, predictors_valid และ target_valid

5. Normalization data

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

การ Normalized ข้อมูล raw ในบาง Column จะทำให้ข้อมูลใน Column นั้นไม่ส่งผลต่อการปรับค่า parameters ของ model จนมากเกินไป เพราะค่าทุกค่าอยู่ในช่วง **[0,1]**

a. Normalized data

```
def normalized()
```

```
    INPUT : df
```

```
    FOR col_name IN ['age', 'fare']:
```

```
        SET max_val = df[col_name].max()
```

```
        SET min_val = df[col_name].min()
```

```
        SET df[col_name] = (df[col_name] - min_val) / (max_val - min_val)
```

```
    OUTPUT : df
```

```
SET unim_fea = ["name", "ticket", "PassengerId"]
SET df_norm = CALL normalized(df.drop(unim_fea))
SET predictors_norm_set = df_norm.drop(["survived"])
SET target_norm_set = tensorflow.to_categorical(df_norm["survived"])
```

คำอธิบาย : เราเลือก feature : age และ fare มา normalized เพราะสองตัวนี้มีค่าที่มากในบางผู้โดยสาร เมื่อ normalized เสร็จเราก็ทำแบบเดียวกับที่ตอนที่เรากำกับ df ซึ่งเป็น raw data เพียงแต่รอบนี้ df_norm เป็น data ที่ผ่านการ scaling มาแล้วเท่านั้น

6. Testing dataset preparation

a. Raw data

```
SET predictors_test = predicset[892:]
SET target_test = targetset[892:]
```

คำอธิบาย : predicset เป็น array ที่มี raw data เมื่อขั้นตอนก่อนๆ เราแยกชุดข้อมูลก่อน index ที่ 892 ออกไปเป็นชุดข้อมูลฝึก ที่เหลือจากนั้นก็แค่ให้เป็น Testing dataset เท่านั้น

b. Normalized data

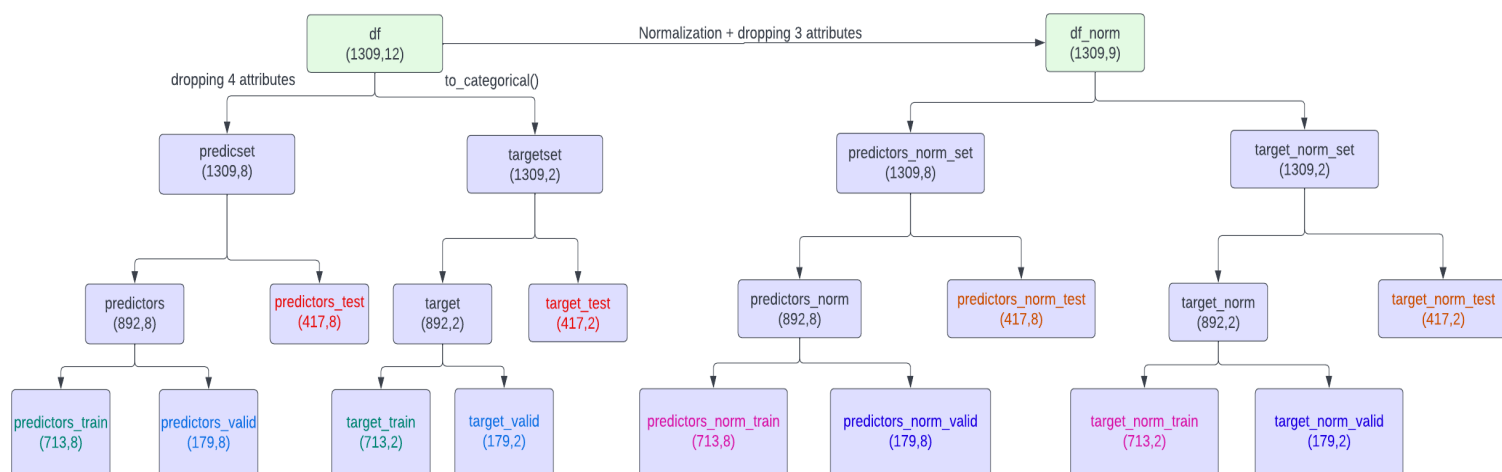
```
SET predictors_norm_test = predictors_norm_set[892:]
SET target_norm_test = target_norm_set[892:]
```

คำอธิบาย : หลักการเหมือนกับของ raw data เพียงแต่ข้อมูลที่ใช่แบ่งออกมาเป็น normalized dataframe เท่านั้น

หลักการของชื่อตัวแปร (e.g. predictors_norm_train)

- {a}_{b}_{c} => a คือ หน้าที่ของข้อมูลชุดนี้เช่น ใช้ฝึก หรือ ใช้เทียบเพื่อ Optimizes
- => b คือ รูปแบบภายในข้อมูลเช่น Normalized อาจจะมีหรือไม่มีก็ได้
- => c คือ ขั้นตอนที่น่าข้อมูลชุดนี้ไปใช้งาน เช่น ชั้น Train , ชั้น test

สรุปตัวแปรเก็บข้อมูลจาก Phase 1



ภาพที่ 4 แผนภาพสรุปตัวแปรเก็บข้อมูลจาก Phase 1

ในแผนภาพ ตัวแปรที่อยู่ปลายสุดของแผนภาพทุกกิ่งจะเป็นตัวแปรที่จะใช้ในการ train, test หรือ validate ใน Phase 2 ทั้งหมด โดยตัวแปรที่มีสีอักษรเดียวกัน (e.g. แดง-แดง) คือตัวแปรที่เป็นคู่ X-Y กัน (Input-Appropriate Output) และกล่องที่มีสีเขียวอ่อนคือตัวแปรที่เก็บข้อมูลประเภท pandas dataframe ส่วนสีน้ำเงินอ่อนจะเก็บข้อมูลประเภท NumPy array

Phase 2: Model Training and Testing

Deep Learning Model

เมื่อเราได้แปลง dataset ของเราให้อยู่ในรูปแบบที่นำไปใช้ train, test หรือ validate ML ได้แล้ว ก็ถึงเวลาที่เราจะพัฒนา Algorithm ของ ML (Deep Learning) กัน ในที่นี้ เราจะอธิบายโครง Algorithm ทั่วไปที่เราใช้ในทุกระดับชั้นของ ML model ที่เราทำขึ้นในโครงการนี้

LIBRARY : tensorflow , matplotlib

Objects from tensorflow

- 1.tensorflow.keras.Sequential()
- 2.tensorflow.keras.layers.Dense(x)

1. Model Architecture

```
SET Model_version_n = Sequential()  
Model_version_n.add(Dense(num_neurons, activation,input_shape))  
Model_version_n.add(Dense(num_neurons , activation))  
Model_version_n.add(Dense(num_neurons = 2 , activation))
```

คำอธิบาย :

Deep Learning model ของเราถูกพัฒนาขึ้นมาโดยใช้แนวคิดที่ว่า Layers ทุกอันของเราจะเรียงซ้อนเป็นเส้นตรง (Linear stack of layers) ซึ่งหมายความว่ามีการส่งค่าระหว่าง layer จากบนลงล่างของ code นี้ไปตามลำดับ (เป็นเส้นตรง) ซึ่งการจัดรูป Model ให้เป็นแบบดังกล่าวต้องใช้ Sequential() ซึ่งเป็นวัตถุทาง Python ของ tensorflow

จากนั้นเราก็ add layers ต่างๆ ที่เราจะใส่ลงไป เราได้เลือก layer : Dense(x) ที่เป็นแบบ Fully connected Model ของเราจะแบ่ง layers ภายในเป็น 3 ส่วนจากบนลงล่างคือ

1. Input Layer
2. Hidden Layers (มีจำนวน layers ตามความเหมาะสม)
3. Output Layer

โดยภายใน Dense(x) จะมี parameters 3 ตัวให้ใส่คือ

1. num_neurons : จำนวน neurons ใน layer นั้นๆ โดยใน Output layer จะมี num_neurons เท่ากับ 2 เท่านั้น โดยใน neuron ตัวแรกให้ค่าความน่าจะเป็นที่รอดชีวิต ส่วนตัวที่สองให้ค่าความน่าจะเป็นที่จะตาย
2. activation : ในหลักการทำงานของ Neural Network activation function จะเป็นตัว scale ค่าใน neuron ให้อยู่ในช่วงหรือค่าที่เหมาะสมก่อนส่งออกไป โดย activation ที่เราใช้ก็คือ relu ซึ่งเป็นฟังก์ชันคณิตศาสตร์ดังนี้

$$f(x) = \max(0, x)$$

อีกอันหนึ่งก็คือ softmax ซึ่งใช้ใน layer สุดท้ายเพื่อให้ออกมาเป็นค่าความน่าจะเป็นที่จะอยู่ในช่วง $[0,1]$ ของแต่ละ *class* (รอดหรือตาย)

3. `input_shape` : ใช้แค่ใน Layer แรกเท่านั้น เพราะเป็นการกำหนดว่าจะ input ของ model จะต้องมีความยาว (n,num_feature) โดย num_feature คือจำนวน Feature ทั้งหมดที่ Model จะรับไป ส่วน n คือจำนวน record ของเรา ในกรณีนี้เราได้กำหนดให้ `input_layer = (n_col,)` เสมอ ซึ่งหมายความว่าไม่มีการ subset จำนวน feature ของเรา และรับจำนวน feature ทั้งหมดไปเลย ซึ่งจำนวน feature ทั้งหมดก็เก็บอยู่ในตัวแปร `n_col` (`n_col = 8`)

2. Compile a model

```
Model_version_n.compile(optimizer, loss, metrics)
SET hist_V{n}_A{k}= Model_version_n.fit(predictors_train,
                                         target_train,
                                         validation_data,
                                         epochs)
```

คำอธิบาย : Model จะเริ่ม Train ได้ก็ต่อเมื่อเราเรียกใช้ 2 Method คือ `compile(x)` และ `fit(x)` โดยภายใน `compile(x)` จะมี parameters 3 ตัวดังนี้คือ

1. `optimizer` : Algorithm ที่ทำหน้าที่ปรับ parameters ภายหลังจากการหาค่า loss จาก loss function ไปแล้ว ส่วนมากในโครงงานนี้เราใช้ adam (Adaptive Moment Estimation) เป็น optimizer
2. `loss` : Loss function ที่ทำหน้าที่ประเมินค่า loss ซึ่งค่าเป็นค่าที่ใช้เทียบว่าค่าที่ predict มีความแตกต่างจากค่าจริงภายใน Y-vector อย่างไรแล้วจึงนำไปปรับปรุงต่อกับ sample อื่นๆ ใน Training dataset ในที่นี้เราใช้ loss เป็น `categorical_crossentropy` ซึ่งนิยมใช้ในงานประเภท classification แบบเดียวกับโครงงานเรา
3. `metrics` : มาตรวัดประสิทธิภาพของ Model เราเมื่อ train ผ่านไปในแต่ละ epochs (ครบทุก sample/record) โดยเทียบกับ Y-vector เช่นเดิม จะอยู่ในรูป Percentage ในกรณีนี้เราใช้ accuracy เป็นมาตรวัด

ส่วนภายใน `fit(x)` จะมี parameters 4 ตัวดังนี้

1. `predictors_train` : ชุดข้อมูลที่ใช้ train โดยการตัดส่วนที่เป็นผลที่ควรเป็นของแต่ละ sample ออกไป ชุดข้อมูลดังกล่าวเป็นชุดข้อมูลที่ใช้ run ในช่วง Forward propagation โดยของเรามีขนาด (713,8)

2. `target_train` = ชุดข้อมูลเป้าหมาย ที่ `loss function` ใช้ประเมินเพื่อนำไปสู่การ optimize model ในกระบวนการ Backpropagation โดยชุดข้อมูลดังกล่าวจะมีขนาด (713,2) ซึ่งเหตุผลที่เป็นแบบนี้ก็เพราะเราได้วางโครงให้ Model ของเรามี Output neuron 2 ตัว ตัวหนึ่งจะบอกค่าความน่าจะเป็นในการที่รอดชีวิต อีกตัวจะบอกความน่าจะเป็นที่จะเสียชีวิตในภัยพิบัติเรือครั้งนี้ ซึ่งเมื่อ `target variable` ของเราในแต่ละ sample มีบอกมาทั้งสองค่า ก็จะทำให้การเทียบกันระหว่างค่าที่ทำนายกับค่าที่เป็นจริงแยกกันระหว่างความน่าจะเป็นของสองเหตุการณ์นี้

EX. $[[0.489, 0.897]] \xrightarrow{\text{compare}} [[0, 1]]$

โดยในกระบวนการเปรียบเทียบนี้ `loss function` ของเราก็จะผลิตค่า `loss` ขึ้นมาสำหรับ Sample ที่ทำการเปรียบเทียบอยู่

3. `validation data` : ข้อมูลเพื่อตรวจสอบความถูกต้องขณะ Training เพื่อเป็นการ Estimate Performance ของ model ข้อมูลชุดดังกล่าวมีผลต่อการเลือกปรับ Hyperparameter ของเราในการ train ใน Attempt อื่นๆ (Model โครงเดิม) `validation data` ต้องใส่ค่าเป็น tuple; (`predictors_valid`, `target_valid`)
4. `epochs` : เป็น Hyperparameter ตัวหนึ่งของการ train โดยมันจะกำหนดจำนวนรอบในการ train ซึ่งใน 1 `epochs` Model จะต้องผ่าน `predictors_train` และ `target_train` ทั้งชุด (เห็นทุก samples)

สุดท้าย `hist_V{n}_A{k}` คือตัวแปรเก็บข้อมูล `loss` และ `accuracy` ในแต่ละ epoch โดยในชื่อตัวแปรของเราจะบ่งบอกว่าเป็น history (`hist`) ของ Model รุ่นใด (`V{n}`) และ Attempt การ train ครั้งที่เท่าไร (`A{k}`) ข้อมูลจริงๆ สามารถดึงได้จากการเรียก attribute : `history` ของมัน ซึ่งเมื่อดึงแล้ว ข้อมูลจะอยู่ในรูป dictionary

3. Plotting Model performance

LIBRARY : `matplotlib`

3.1 กราฟของค่า `loss`

`def plot_loss():`

 INPUT : `hist` , style

`matplotlib.plot(loss_data , label = "loss")`

`matplotlib.plot(validation_loss_data , label = "loss")`

`matplotlib.set_x_label("Epochs")`

```
matplotlib.set_y_label("loss")
matplotlib.show_graph()
```

คำอธิบาย : loss_data และ validation_loss_data สามารถดึงมาได้จาก dictionary : hist ที่เป็น Input ของฟังก์ชัน โดยภายหลังจากการสร้างกราฟ ก็จะมีการตั้งชื่อแกนในกราฟ ก่อนจะแสดงกราฟ

3.2 กราฟของค่า accuracy

```
def plot_validation():
    INPUT : hist , style
    matplotlib.plot(accuracy_data , label = "loss")
    matplotlib.plot(validation_accuracy_data , label = "loss")
    matplotlib.set_x_label("Epochs")
    matplotlib.set_y_label("accuracy")
    matplotlib.show_graph()
```

คำอธิบาย : : accuracy_data และ validation_accuracy_data สามารถดึงมาได้จาก dictionary : hist ที่เป็น Input ของฟังก์ชัน โดยภายหลังจากการสร้างกราฟ ก็จะมีการตั้งชื่อแกนในกราฟ ก่อนจะแสดงกราฟ

หากเราจะ plot กราฟเพื่อดูผลก็แค่เรียกสองฟังก์ชันนี้มาใช้งานได้ทุกเมื่อ

4. Testing Model

```
SET test_loss , test_acc = Model_version_n.evaluate(predictors_test
, target_test)
```

คำอธิบาย : เมื่อเรานำชุดข้อมูลการทดสอบเข้าฟังก์ชัน evaluate(x) แล้ว มันจะให้ค่าผลลัพธ์มาสองค่า (เก็บด้วยสองตัวแปร) พร้อมแสดงผลให้เราดู

5. Saving Model

```
Model_version_n.save ("path")
```

คำอธิบาย : ใช้ฟังก์ชัน save ของ tensorflow ไปเก็บไว้ในจุดที่ต้องการ ในกรณีเราใช้ Google Drive

Deep Learning Model ทั้ง 5 versions ของเรา

1. Model V1:

```
modelV1 = Sequential()  
modelV1.add(Dense(32, activation = "relu" , input_shape = (n_col ,)))  
modelV1.add(Dense(32 , activation = "relu"))  
modelV1.add(Dense(2 , activation = "softmax"))
```

ภาพที่ 5

2. Model V2:

```
modelV2 = Sequential()  
modelV2.add(Dense(32, activation = "relu" , input_shape = (n_col ,)))  
modelV2.add(tf.keras.layers.Dropout(0.5))  
  
modelV2.add(Dense(32 , activation = "relu"))  
modelV2.add(tf.keras.layers.Dropout(0.5))  
  
modelV2.add(Dense(2 , activation = "softmax"))
```

ภาพที่ 6

3. Model V3:

```
modelV3 = Sequential()  
modelV3.add(Dense(32, activation = "relu" , input_shape = (n_col ,)))  
modelV3.add(BatchNormalization())  
modelV3.add(tf.keras.layers.Dropout(0.5))  
  
modelV3.add(Dense(32 , activation = "relu"))  
modelV3.add(BatchNormalization())  
modelV3.add(tf.keras.layers.Dropout(0.5))  
  
modelV3.add(Dense(2 , activation = "softmax"))
```

ภาพที่ 7

4. Model V4:

```
modelV4 = Sequential()  
modelV4.add(Dense(32, activation = "relu" , input_shape = (n_col ,), kernel_regularizer='l1'))  
modelV4.add(BatchNormalization())  
modelV4.add(tf.keras.layers.Dropout(0.5))  
  
modelV4.add(Dense(32 , activation = "relu"))  
modelV4.add(BatchNormalization())  
modelV4.add(tf.keras.layers.Dropout(0.5))  
  
modelV4.add(Dense(2 , activation = "softmax"))
```

ภาพที่ 8

5. Model V5:

```
modelV5 = Sequential()
modelV5.add(Dense(16, activation = "relu" , input_shape = (n_col ,)))
modelV5.add(BatchNormalization())
modelV5.add(tf.keras.layers.Dropout(0.5))

modelV5.add(Dense(16 , activation = "relu"))
modelV5.add(BatchNormalization())
modelV5.add(tf.keras.layers.Dropout(0.5))

modelV5.add(Dense(16 , activation = "relu"))
modelV5.add(BatchNormalization())
modelV5.add(tf.keras.layers.Dropout(0.5))

modelV5.add(Dense(2 , activation = "softmax"))
```

ภาพที่ 9

จะสังเกตให้เห็นว่าในแต่ละ Model จะมีการเพิ่มส่วนประกอบไปเล็กน้อย เช่น Dropout layer หรือ การประมวลผลบางอย่างชั้นกลางเช่น BatchNormalization() นอกจากนี้ยังมีการปรับ Hyperparameter เช่น จำนวน neurons ใน Hidden Layer และ Input Layer หรือการเพิ่มจำนวน Hidden Layers

ในส่วนของการละเอียดในการทำ หากต้องการเห็นกระบวนการทำทั้งหมด สามารถดูได้ใน

https://github.com/PeterLOVANAS/Titanic-machine-learning-project/blob/main/Copy_of_Copy_of_Titanic_firstML.ipynb

Random Forest Classification Model

เรายังได้มีการพัฒนาโมเดลเพิ่มเติมเพื่อเป็นการเปรียบเทียบกับ Model ที่เราทำมาด้วย Neural Network ด้วย Model นี้จะเป็น Random Classification Model (ใช้ library : sklearn) โดยในโครงการนี้ เราได้สร้าง Model นี้ขึ้นมา 3 Version ได้แก่

1. Model Random Forest Version 1
 - a. Training

```
modelRF_V1 = RandomForestClassifier(n_estimators=100)
modelRF_V1.fit(predictors_train , target_train)
```

ภาพที่ 10

คำอธิบาย : Model รอบนี้ใช้ Algorithm เป็น RandomForestClassifier(x) ซึ่งจะมีการสร้าง Decision Tree ตามจำนวนที่กำหนดไว้ใน n_estimators เมื่อเราสร้าง Model Object แล้วเราจึงนำข้อมูลใส่ลงไปด้วย Method fit()

b. Testing

```
pred = modelRF_V1.predict(predictors_test)
print(" Accuracy: {}".format(metrics.accuracy_score(target_test, pred)))
```

ภาพที่ 11

คำอธิบาย : ขั้นแรกเราจะให้มันทำนายออกมาก่อน แล้วเก็บผลทำนายในรูป Array แบบเดียวกับ Output ของ Model ที่ใช้ Deep Learning เพียงแต่จะบอกเป็นค่าตรงๆ (1 หรือ 0) จากนั้นจึงนำมาเทียบกับ target_test เพื่อวัดค่า accuracy อีกที่

2. Model Random Forest Version 2 (Use Normalized data)

a. Training

```
modelRF_V2 = RandomForestClassifier(n_estimators=100)
modelRF_V2.fit(predictors_norm , target_norm)
```

ภาพที่ 12

คำอธิบาย : เป็นวิธีการสร้าง Model เดียวกันกับเวอร์ชันแรก

b. Testing

```
pred = modelRF_V2.predict(predictors_test)
print(" Accuracy: {}".format(metrics.accuracy_score(target_test, pred)))
```

ภาพที่ 13

คำอธิบาย : หลักการเดียวกับตอนทำ Model Version แรก

3. Model Random Forest Best Version (Use Normalized data)

a. Finding Best Hyperparameter

```

param_grid = {
    'n_estimators': [50, 100, 500, 800],
    'max_depth': [None, 50, 90, 120],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(modelRF_V2, param_grid, cv=5, scoring='accuracy')

grid_search.fit(predictors_norm, target_norm)

```

ภาพที่ 14

คำอธิบาย : Dictionary ของค่า Hyperparameters ทั้ง 3 ตัวที่ต้องการให้มีการปรับเพื่อหาชุดของค่า Hyperparameters ที่สร้าง accuracy ของชุดข้อมูลทดสอบที่ดีที่สุด Algorithm ที่ทำสิ่งที่ว่ามาได้ก็คือ GridSearchCV(x) ซึ่งต้องใส่ param_grid ที่เราทำมา และ ตัวมาตรวัด (accuracy) ลงไปเป็น parameters จากนั้นจึงใส่ข้อมูลลงไป

b. Training

```

best_params = grid_search.best_params_

# Use the best hyperparameters to create a new model
best_model = RandomForestClassifier(**best_params)

# Fit the new model to the training data
best_model.fit(predictors_norm, target_norm)

```

ภาพที่ 15

คำอธิบาย : เมื่อได้ชุด Hyperparameters ที่ดีที่สุดแล้วจึงนำมาสู่การนำมันมาใส่ใน Model ซึ่งเราจะเรียกว่า Best Model จากนั้นจึงใส่ข้อมูลแบบ Normalized ลงไป

c. Testing

```

# Evaluate the new model on the test set
test_accuracy = best_model.score(predictors_test, target_test)

print(f'Test accuracy: {test_accuracy:.2f}')

```

ภาพที่ 16

คำอธิบาย : หลักการเดียวกับตอนทำ Model Version แรก

จากนั้นเราจะ save ด้วย (Save Best version of Model)

```

pickle.dump(best_model , open("/content/drive/MyDrive/Colab_models/Model_RF_VB_1D.pkl" , "wb"))

```

ภาพที่ 17

Phase 3: Deployment

เมื่อเลือก Model ที่ต้องการจะใส่ใน Prototype แล้วโดย Model V5 รุ่นที่มีการ train ด้วยข้อมูลที่ไม่มีการ normalized เป็นรุ่นที่มี accuracy ดีที่สุด ส่วน Model V5 รุ่นที่มีการ train ด้วยข้อมูลที่มีการ normalized และ Random Forest Model best version จะถูกนำไปวางประกอบ

1. Instance()

def instance():

```
    SET pclass = float(pcc.get())
    SET sex = float(CALL define_sex(sexc.get()))
    SET age = float(age_et.get())
    SET sibsp = float(sibet.get())
    SET parch = float(paret.get())
    SET fare = float(faet.get())
    SET cabin = float(replace_cabin(cabc.get()))
    SET emb_bf = emc.get()
    SET embark = replace_embarked(emb_bf)
    SET person = np.array([[pclass , sex , age , sibsp , parch , fare , cabin
, embark]])
    OUTPUT : person
```

คำอธิบาย : ฟังก์ชันนี้มีการดึงข้อมูลจาก drop down menu, entry ที่ให้ผู้ใช้กรอก จากนั้นจึงมีการนำค่ามาแปลงเป็นตัวเลข (float) เมื่อเสร็จจึงรวมเป็น feature vector (Numpy Array) แล้วส่งต่อไปที่ฟังก์ชันที่จะใช้

2. run_model()

def run_model():

```
    SET model_path_1 = "modelV5_2_2D_cop1.h5"
    SET model_path_2 = "modelV5_3_1D_cop1.h5"
    SET model_path_3 = "Model_RF_VB_1D.pkl"
    SET data = instance()
```

```

IF ch == "Model V5" THEN
    SET modelV5_2 = tf.keras.models.load_model(model_path_1)
    SET predictions = modelV5_2.predict(data)
    result_et.insert("", f"{predictions[0][1]*100:.2f}")
ELSE IF ch == "Model V5 (Norm)" THEN
    SET modelV5_3 = tf.keras.models.load_model(model_path_2)
    SET predictions = modelV5_3.predict(data)
    result_et.insert("", f"{predictions[0][1]*100:.2f}")
ELSE ch == "Model VB (RF)" THEN
    SET modelRF_2 = pickle.load(open(model_path_3 , "rb"))
    SET predictions = modelRF_2.predict(data)
    result_et.insert("", f"{predictions[0][1]*100:.2f}")

```

คำอธิบาย : ใน drop down menu จะมีให้เลือกว่าจะเอา model ไหน เมื่อกดปุ่ม run model ก็จะทำให้ฟังก์ชันนี้ทำงาน และมีการ capture ค่าที่กรอกทั้งหมดเข้าฟังก์ชัน instance เพื่อแปลงข้อมูลให้อยู่ในรูป feature vector ก่อนส่งเข้า model ตามที่ผู้ใช้เลือก

บทที่ 4

ผลการดำเนินงาน

จากโครงการ Machine Learning for ship disaster ทางผู้จัดทำได้มีการตรวจสอบสภาพของข้อมูลที่จะนำไปฝึกและนำไปทดสอบและได้ทำการทดลองปรับปรุง Model หลายต่อหลายเวอร์ชัน เพื่อได้โมเดลที่ดีที่สุดที่จะนำไปใส่ในตัว Prototype จริง

โดยใน Phase 1 ทางผู้จัดทำขอเสนอตารางการตรวจสอบความแต่ละ columns ของข้อมูลก่อน Train

Column's name	Not NULL	In Numeric form
age	Not pass	pass
embarked	Not pass	Not pass
cabin	Not pass	Not pass
pclass	pass	pass
fare	Not pass	pass
sibsp	pass	pass
parch	pass	pass
sex	pass	Not pass

ตารางที่ 2 ข้อมูลแต่ละ feature ก่อนการ cleaning (สภาพข้อมูลดิบ)

เมื่อผ่านกระบวนการ Preprocessing (ในบทที่ 3) แล้ว ทุก requirement จึงผ่านใน column ที่มีปัญหา

Column's name	Not NULL	In Numeric form
age	pass	pass
embarked	pass	pass
cabin	pass	pass
fare	pass	pass
sex	pass	pass

ตารางที่ 3 ข้อมูลแต่ละ feature ภายหลังการ cleaning

เมื่อ การ cleaning สิ้นสุด dataframe ก็จะมีข้อมูลครบถ้วนและพร้อมที่จะถูกนำไปแปลงเป็น Numpy Array

ตัวชี้วัดความพร้อม	ผลลัพธ์จากการทดสอบ
Sample ทั้งหมดใน df เรียงสลับอย่าง random	pass
ชุดข้อมูลรวม มี feature แค่ 8 อัน (ห้ามมีของ target feature รวมอยู่ด้วย)	pass
Target variable มี target feature	pass
Target variable มีขนาด (1309,2)	pass
ชุดข้อมูลรวมและ Target Variable อยู่ในรูป Numpy Array	pass
ข้อมูล Testing ถูกแยกจากข้อมูล Training	pass
ข้อมูลชุด Training ที่แยกมาแล้วถูกแยกย่อยเป็น สำหรับการ Training และการ Validation	pass
df ที่ถูก copy ได้รับการ normalized เฉพาะแค่ feature ที่จำเป็น	Not pass
ชุดข้อมูลรวมที่ normalized เป็น Numpy Array	pass
ขนาดของชุดข้อมูล Training ทั้ง raw และ normalized เป็น (713,8)	pass
ขนาดของชุดข้อมูล Validation ทั้ง raw และ normalized เป็น (179,8)	pass
ขนาดของ Target Variable (Training) ทั้ง raw และ normalized เป็น (713,2)	Not pass
ขนาดของ Target Variable (Validation) ทั้ง raw และ normalized เป็น (179,2)	Not pass
ขนาดของชุดข้อมูลทดสอบ ทั้ง raw และ normalized เป็น (417,8)	pass
ขนาดของ Target Variable (Testing) ทั้ง raw และ normalized เป็น (417,2)	Not pass

ตารางที่ 4 ตารางตรวจสอบสภาพความพร้อมของ data ก่อนใช้งาน

ภายหลังการ test ก็ได้มีการปรับปรุงข้อมูลให้ผ่านตัวชี้วัดดังนี้

ตัวชี้วัดความพร้อม	ปัญหา	การปรับปรุง
df ที่ถูก copy ได้รับการ normalized เฉพาะแค่ feature ที่จำเป็น	Normalized ทุก column ทำให้การข้อมูลมีความผิดเพี้ยนไปหลายจุด	Normalized แค่ feature: fare และ age เพราะตัวเลขของสอง feature นี้มีขนาดมาก อาจส่งผลตอน Training
ขนาดของ Target Variable (Training/Validation/Testing) ทั้ง raw และ normalized เป็นขนาดตามที่ต้องการจะเป็น	มีการเรียกใช้ to_categorical() สองครั้งทำให้ shape ของ array เป็น 3 มิติ	เรียกใช้ to_categorical() แค่ครั้งเดียว เพื่อให้ข้อมูลยังคงเป็นแค่ 2 มิติ

ตารางที่ 5 การปรับปรุงข้อมูลให้ผ่านทุกตัวชี้วัด

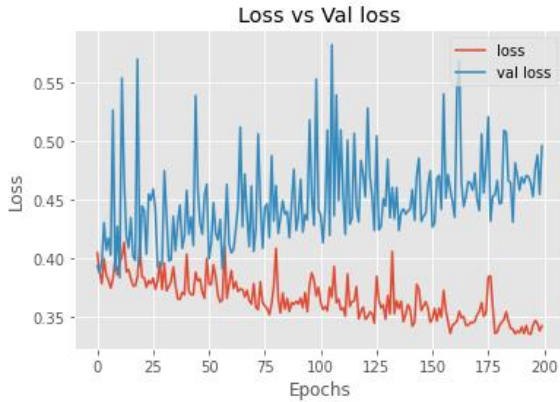
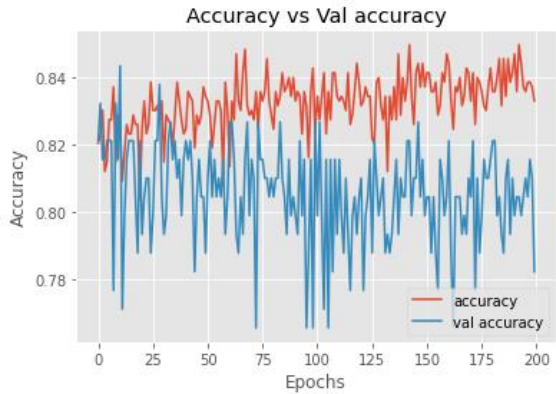
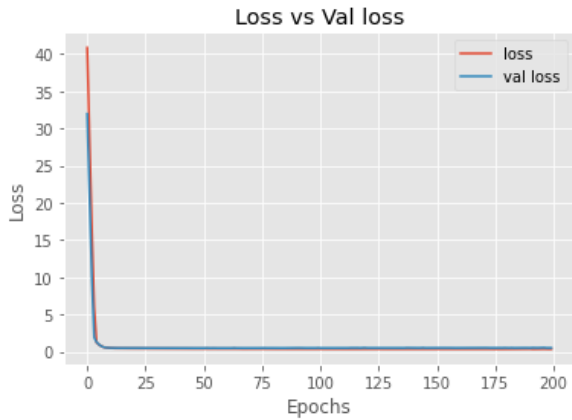
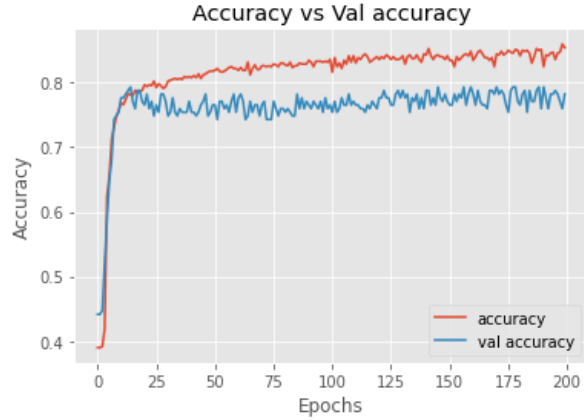
เมื่อผ่านกระบวนการ Preprocessing ทั้งหมด เราก็ได้นำข้อมูลไปใช้ Train ML model ต่อ
 ต่อจากนี้คือตารางแสดงผลการดำเนินงานของ Phase ที่ 2 ของทุก ML Model

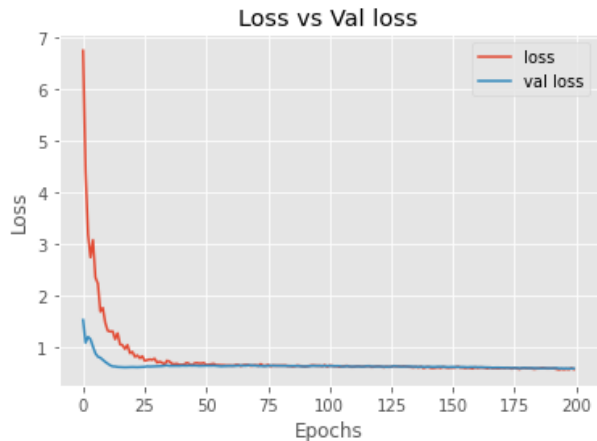
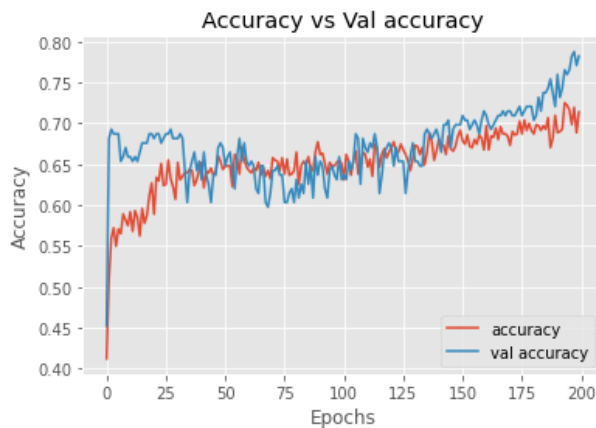
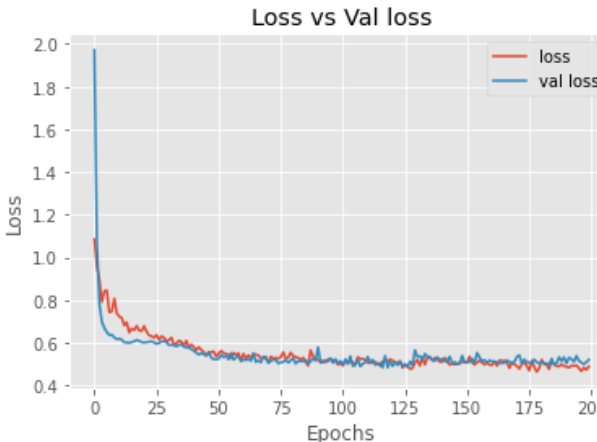
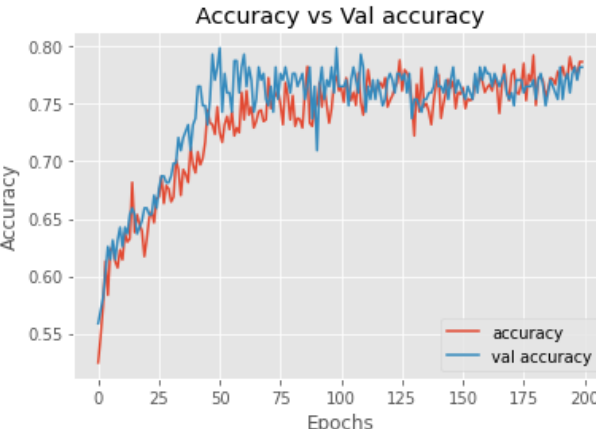
*Model_Attempt ที่ติดดาวทั้งหมดคือ Model ที่จะได้ลง Prototype จริงๆ โดย

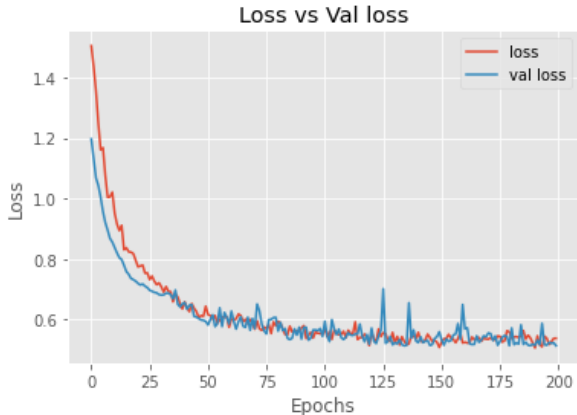
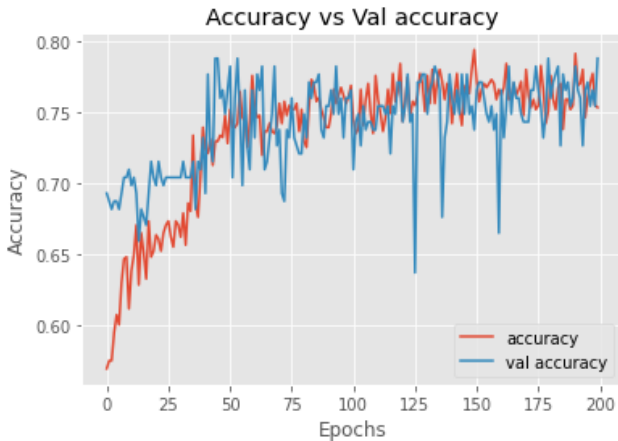
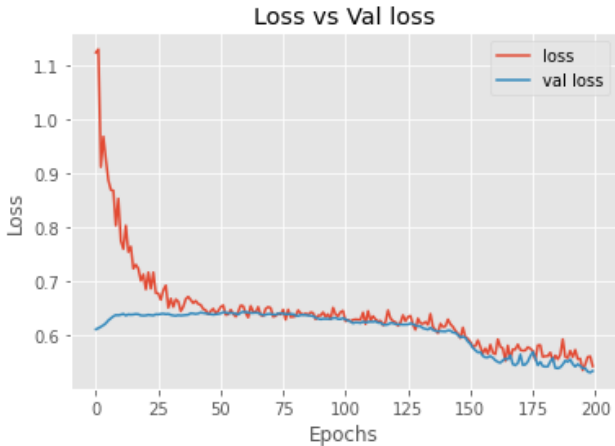
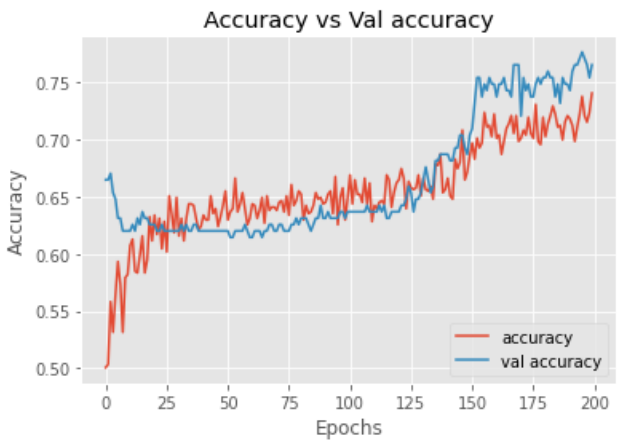
ดาวสีเหลือง หมายถึง Model หลักที่ใช้ประมวลผล

ดาวสีเงิน หมายถึง Model ที่มีความเหมาะสมรองลงมา (ใช้เทียบผลลัพธ์และประสิทธิภาพ)


ดาวสีทองแดง หมายถึง Model ที่มีความเหมาะสมน้อยที่สุดในสามอัน (ใช้เทียบผลลัพธ์และประสิทธิภาพเช่นกัน)

Model	Attempt	Raw/Norm	Loss graph (Training)	Accu. graph (Training)	Test Accu.
V1	1	R	-	-	0.7752
	2	R			0.7674
	3	N			0.7818

Model	Attempt	Raw/Norm	Loss graph (Training)	Accu. graph (Training)	Test Accu.
V2	1	R			0.7602
V3	1	R			0.7962

Model	Attempt	Raw/Norm	Loss graph (Training)	Accu. graph (Training)	Test Accu.
V4	1	R			0.7722
V5	1	R			0.7986

Model	Attempt	Raw/Norm	Loss graph (Training)	Accu. graph (Training)	Test Accu.
	2 ★	R			0.8225
	3 ★	N			0.8201

Model	Attempt	Raw/Norm	Loss graph (Training)	Accu. graph (Training)	Test Accu.
RF_V1	1	R	-	-	0.7721
RF_V2	1	N	-	-	0.6139
RF_VB	1 	N	-	-	0.7700

ตารางที่ 6 ผลการทำงานของ ML models

ตลอดการพัฒนา ML Model เราพบข้อบกพร่องทางความแม่นยำมากมาย ส่งผลให้มีการสร้าง Model ออกมาหลาย Version และในแต่ละ version ก็มีการ compile หลายครั้ง (เรียกว่าการ Attempt) โดยในแต่ละครั้งก็จะมีการเปลี่ยนชุดข้อมูลจาก raw เป็น normalized บ้าง ไม่ก็มีการปรับ Hyperparameters เช่น Learning rate เป็นต้น จากนั้นจะเป็นการอธิบายผลลัพธ์ในแต่ละ Model *กำหนดให้จำนวนรอบในการ train (epochs) มีค่าเป็น 200 ทุกครั้งที่ compile

1. Model V1

Model V1 เป็น Model ตัวเริ่มต้น

ใน Attempt 1 เราไม่ได้ Train ด้วย Validation data ทำให้ไม่มีกราฟ แต่ก็สังเกตได้ว่า Accuracy ก็อยู่ในระดับน่าพอใจ สำหรับ Model ในเวอร์ชันที่ยังไม่ปรับหรือเพิ่มอะไรเลย

ใน Attempt 2 เราได้นำ Validation data เข้าร่วมการ train ด้วย ผลลัพธ์คือเกิด Overfitting กับ Training dataset สังเกตได้จากกราฟ loss/accuracy ของข้อมูล Validation จะแยกไปคนละทาง และ accuracy ตอน Testing จะมีค่าตกลง

ใน Attempt 3 เราได้เปลี่ยนชุดข้อมูลที่นำเข้าเป็นชุดข้อมูลที่ Normalized ซึ่งผลลัพธ์คือกราฟ loss ลดลงอย่างสวยงาม ส่วนกราฟ Accuracy มีการแยกทางกันเล็กน้อย ซึ่งทางเราคิดว่าไม่เป็นของการ Overfitting ในการ Train รอบนี้

2. Model V2

Model V2 มีการเพิ่ม Layer dropout ลงไปเพื่อเพิ่ม Accuracy ส่วนจำนวน Neurons และ จำนวน Dense Layers ยังเท่าเดิมเหมือน Model V1

ใน Attempt 1 เราใช้ Raw data โดยเราพบว่ากราฟทั้งสองมีรูปแบบที่มันควรจะเป็น แต่ในกราฟ accuracy จะมีสัญญาณของการ Overfitting ส่งผลให้ Testing Accuracy มีค่าตกลงใน

3. Model V3

Model V3 มีการเพิ่มส่วนการประมวลผล BatchNormalization() ลงไประหว่าง layer โดยยังคง Dropout layer ที่เพิ่มไว้ตั้งแต่ Model V2 อยู่ ส่วนจำนวน Neurons และจำนวน Dense Layers ยังเท่าเดิมเหมือน Model V1

ใน Attempt 1 เราใช้ Raw data โดยเราพบว่ากราฟมีความ Overfitting น้อยลงอย่างมาก ซึ่งก็สอดคล้องกับการที่ Testing Accuracy มีค่าสูงขึ้น

4. Model V4

Model V4 คงทุกอย่างเหมือน Model V3 แต่เพิ่ม L1 Regularization ลงไปเพื่อหวังเพิ่ม accuracy

ใน Attempt 1 ซึ่งใช้ Raw data เราจะพบว่ามีการ Overfitting บ้าง สังเกตได้จากกราฟ Accuracy ขณะ Training ซึ่งก็สอดคล้องกับ Testing Accuracy ที่ตกลง

5. Model V5

Model V5 มีการปรับโครงสร้างหลายจุด โดยเอาข้อดีของ Model V1-3 มารวมกันเช่นการคง dropout layer และส่วนประมวลผล BatchNormalization() ไว้ ซึ่งการเติมทั้งสองอย่างจะเพิ่ม accuracy ได้จากที่เราเห็นใน Model รุ่นผ่านๆ มา นอกจากนั้น Model เวอร์ชันนี้ยังปรับ Hyperparameters อย่างจำนวน Neurons ในแต่ละ Dense layer ให้ลดจาก 32 เป็น 16 แต่ให้เพิ่มจำนวน Dense layer จาก 2 เป็น 3 แทน

ใน Attempt 1 ซึ่งใช้ Raw data เราไม่ได้มีการปรับ Hyperparameter อื่น จึงเป็นการ compile เหมือนครั้งก่อนๆ ซึ่งในครั้งนี้อาจจะเห็นว่ากราฟมีรูปแบบที่สมควรเป็น และมีสัญญาณ Overfitting ที่ค่อนข้างน้อย ส่วน Testing accuracy ก็สูงกว่าความพยายามครั้งก่อนๆ ใน Model รุ่นที่ 1-4

ใน Attempt 2 ซึ่งใช้ Raw data เช่นกัน เราได้ปรับ Hyperparameter อย่าง Learning rate ให้มีค่า 0.001 ซึ่งผลลัพธ์รอบนี้ แม้ว่าจะสร้างกราฟที่มีค่าแกว่งไปบ้าง แต่ก็มีหลายจุดในกราฟ Accuracy ซึ่งค่าระหว่าง Training data กับ Validation data ไม่ต่างกันมาก ซึ่งสื่อได้ถึงแนวโน้มที่จะมีการ Overfitting ที่ในระดับที่ไม่มากเกินไป ซึ่งก็สอดคล้องกับผล Testing accuracy ซึ่งออกมาสูงเกิน 0.80 เป็นครั้งแรก

ใน Attempt 3 ได้มีการใช้ Normalized data แทน ในขณะที่ Learning rate จะยังคงมีค่าเท่ากับ Attempt 2 โดยในกราฟที่ออกมาจากการ Train ถือว่ามีรูปแบบที่น่าพึงพอใจ และ Testing accuracy ก็สูงเกิน 0.80 เช่นเดียวกับ Attempt 2

6. Model RF V1

ใน Random Forest Classification Model เราได้ set n_estimators เป็น 100 ซึ่ง Testing Accuracy ก็ถือว่าอยู่ในระดับพึงพอใจ

7. Model RF V2

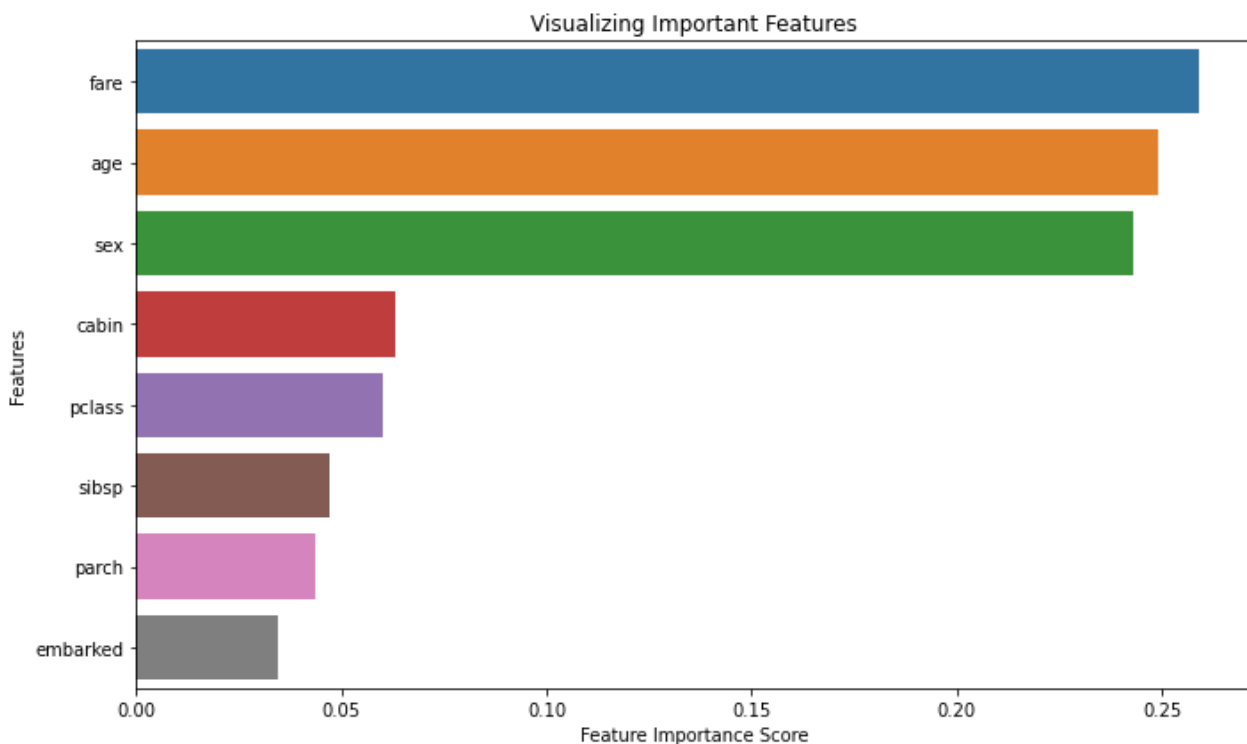
Hyperparameter เหมือนกับใน Model RF V1 แต่ใช้ Normalized data แทน ซึ่ง Testing Accuracy มีค่าตกลงมาอย่างมาก

8. Model RF V3

มีการปรับ Hyperparameter ให้เป็นชุด Hyperparameter ที่ดีที่สุด (เลือกจาก parem_grid) และใช้ Normalized data ซึ่ง Testing accuracy ก็อยู่ในระดับดีเท่ากับ Model RF V1

Insight เล็กๆ น้อยๆ จาก Random Forest Classification

Random Forest Classification Model V1 สามารถบอก Feature Importance ได้ ซึ่งมันได้แสดงเป็นกราฟดังนี้



ภาพที่ 18 ความสำคัญของแต่ละ Feature ที่ใช้ Train (Model RF V1)

กราฟแท่งชุดนี้บอกเราว่า ค่าที่มีขนาดใหญ่ใน dataset เช่น fare, age จะมีอิทธิพลต่อ Model ที่ใช้ Raw data Train มาก แต่เรายังพบว่า sex ซึ่งมีค่าเพียง 0 หรือ 1 ก็มีอิทธิพลต่อการตัดสินใจเช่นกัน ในประเด็นนี้เรายังพบอีกว่าปัจจัยเช่นชั้น cabin หรือชั้นผู้โดยสารมีอิทธิพลค่อนข้างน้อย ในขณะที่เมืองที่ขึ้นเรือมีอิทธิพลน้อยที่สุด

ใน dataset ชุดอื่นๆ ในอนาคต feature ต่างๆ ที่ใช้ Train จะต่างออกไป รวมถึงความสำคัญของแต่ละ Feature ด้วย เช่น ใน dataset ชุดอื่น ชั้นของห้องพัก (cabin) หากผ่านกระบวนการ Preprocessing ที่ดี อาจมีอิทธิพลมากในการตัดสินใจของ AI และ อายุหรือเพศอาจจะไม่มีผลใด ๆ เลยก็เป็นได้

แนะนำ GUI ของโปรแกรม Prototype

The image shows a web-based GUI for a Titanic survival prediction model. It features several input fields and dropdown menus, each with a yellow callout number. The inputs include: Model (V5), Passenger class (I), Sex (male), Age, Number of Siblings and Spouses, Number of Parents and child, Fare, Cabins level (A), and Embarked (Southampton). There are also buttons for 'Run Model' and 'clear', and a display area for the survival chance percentage. At the bottom, there is a large image of the Titanic ship.

Models: Model V5 (1)
Passenger class: I (2)
Sex: male (3)
Age: (4)
Number of Siblings and Spouses: (5)
Number of Parents and child: (6)
Fare: (7)
Cabins level: A (8)
Embarked: Southampton (9)
Run Model (10) clear (11)
This person survival chance is % (12)

Titanic : Machine Learning from disaster

ภาพที่ 19 GUI ของ Prototype

- เบอร์ 1 : Drop down menu ใช้เลือก Model ที่ต้องการใช้ประมวลผล
- เบอร์ 2 : Drop down menu ใช้เลือก Passenger Class
- เบอร์ 3 : Drop down menu ใช้เลือก เพศผู้โดยสาร
- เบอร์ 4 : Entry ใช้กรอก อายุ
- เบอร์ 5: Entry ใช้กรอกจำนวนสามีภรรยาหรือพี่น้องที่จะร่วมเดินทาง
- เบอร์ 6 : Entry ใช้กรอกจำนวนพ่อแม่หรือลูกที่จะร่วมเดินทาง
- เบอร์ 7 : Entry ใช้กรอกจำนวนเงินที่มีการใช้จ่ายเป็นค่าเดินทาง
- เบอร์ 8 : Drop down menu ใช้เลือกชั้นของห้องพัก
- เบอร์ 9 : Drop down menu ใช้เลือกเมืองที่ใช้ออกเดินทาง
- เบอร์ 10 : ปุ่มเพื่อ run model
- เบอร์ 11 : ปุ่มเพื่อ clear ข้อมูลใน Entry ทั้งหมด
- เบอร์ 12 : Entry แสดงค่าความน่าจะเป็นในการรอดชีวิตในรูปแบบ Percentage

Models **Model V5**

Passenger class **2**

Sex **male**

Age **16**

Number of Siblings and Spouses **0**

Number of Parents and child **2**

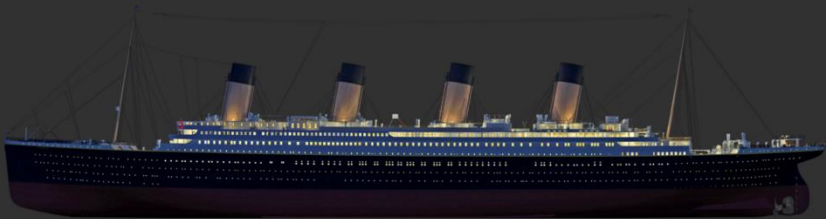
Fare **120**

Cabins level **A**

Embarked **Cherbourg**

Run Model **clear**

This person survival chance is **63.94** %



Titanic : Machine Learning from disaster

ภาพที่ 20 ตัวอย่างการใช้งานโปรแกรม Prototype เมื่อมีการใส่ค่า feature ดังเดิมลงไป พบว่าค่าความน่าจะเป็นในการรอดชีวิตอยู่ในระดับไม่สูงมาก

Models **Model V5**

Passenger class **1**

Sex **male**

Age **16**

Number of Siblings and Spouses **0**

Number of Parents and child **2**

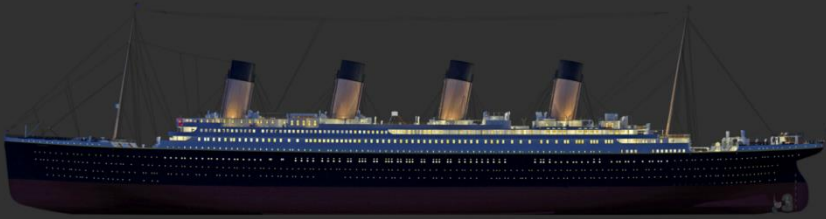
Fare **500**

Cabins level **A**

Embarked **Southampton**

Run Model **clear**

This person survival chance is **92.64** %



Titanic : Machine Learning from disaster

ภาพที่ 21 ตัวอย่างการใช้งานโปรแกรม Prototype เมื่อมีการเปลี่ยนค่า feature ให้ค่าความน่าจะเป็นสูงขึ้น

บทที่ 5

สรุป อภิปรายผล และข้อเสนอแนะ

5.1 สรุปผล

จากการพัฒนาโปรแกรมทั้งหมด ทางผู้จัดทำพบว่า โปรแกรมสามารถทำงานได้ตามวัตถุประสงค์ แม้จะเป็นเพียงโปรแกรม Prototype เท่านั้น แต่ก็สามารถถ่ายทอดรูปแบบการใช้งานได้ที่เราคาดหวังอย่างสมบูรณ์ โดย data ที่ผ่านกระบวนการ Preprocessing มีความสมบูรณ์และอยู่ในรูปแบบที่สามารถ Train ML Model ได้ เมื่อเข้าสู่ขั้นของการ Training ML model ของเราก็สามารถค่อยๆ พัฒนความแม่นยำและถูกต้องไปเรื่อยๆ จากการที่เราทำการปรับ Hyperparameter เช่น Learning rate , จำนวน neurons หรือจำนวน Layers และจากการเพิ่ม Layer อย่าง Dropout และส่วนประมวลผลอย่าง BatchNormalization() ผลลัพธ์ของกราฟในแต่ละ Attempt ในการ Train ยังสามารถบ่งบอกพฤติกรรมของ Model ว่า Overfitting หรือไม่ ซึ่งการรู้เช่นนี้ก็จะนำไปสู่การปรับปรุง Model ในรุ่นต่อไป เมื่อ Train เสร็จเราได้เลือก Model ที่เราพิจารณาว่ามี Testing Accuracy ดีที่สุดมาใช้งานจริงในโปรแกรม Prototype ของเรา ซึ่งเมื่อนำมาใช้งานจริงแล้ว ก็สามารถทำการปรับ feature เพื่อเปลี่ยนค่า accuracy ได้จริงๆ อย่างที่หวังไว้ การพัฒนาครั้งนี้ได้เปิดแนวทางให้ผู้จัดทำได้ทำความรู้จักกับการพัฒนา ML model ตั้งแต่ขั้นตอนแรก จนถึงขั้น Prototyping ซึ่งนำมาสู่ ML model ตัวแรกอย่างทางการของผู้จัดทำ อีกสิ่งหนึ่งที่ทำให้ผู้จัดทำได้เรียนรู้คือการเตรียมข้อมูลที่ดี จะมีผลต่อการประมวลผลของ Model เป็นอย่างมาก โครงการงานนี้จึงนับว่าเป็นประโยชน์ต่อความก้าวหน้าในการพัฒนา Machine Learning Model ของทางผู้จัดทำอย่างยิ่ง

5.2 อภิปรายผล

ผลทำนายที่อยู่ในระดับค่อนข้างแม่นยำของ ML model ล้วนเกิดจากกระบวนการ Preprocessing ที่ดี ซึ่งในกระบวนการ *Preprocessing* นี้ข้อมูลต้องอยู่ในรูปตัวเลขและต้องไม่มีค่าที่มากจนเกินไป ทั้งนี้เพราะ Model หลักของเรา (Deep Learning Model) มีการใช้ค่าจาก feature เหล่านี้ในการ Optimizes parameters ของมัน ซึ่งหากค่ามีขนาดใหญ่ก็อาจส่งผลกระทบต่อกระบวนการนี้ได้ นี่จึงเป็นเหตุผลที่ทำให้บาง feature ถึงมีอิทธิพลต่อการตัดสินใจของ Model มากนั่นเอง การปรับค่า Hyperparameter ของ Model เองก็มีส่วนสำคัญเช่นกัน โดยหากเราปรับถูกให้ Model ไม่มีพฤติกรรม Overfitting มันก็จะสามารถเรียนรู้ให้ generalized กับ data อื่นที่นอกเหนือจาก Training data ได้ หากสามารถทำได้ดีในทั้งสองปัจจัยที่กล่าวมา AI model จะมีประสิทธิภาพที่ดีขึ้นอย่างแน่นอน

5.3 ข้อเสนอแนะ

5.3.1 ข้อเสนอแนะจากผลการจัดทำโครงการ

ทางผู้จัดทำมีแผนที่จะนำแนวความคิดที่ได้ทั้งหมดจากการพัฒนา Prototype นี้ไปต่อยอดเป็นโปรแกรมบอกความเสี่ยงในการเกิดอุบัติเหตุจากการคมนาคม หากมี dataset ที่มากพอและมีความ general ระดับสังคม

5.3.2 ข้อเสนอแนะในการทำโครงการครั้งต่อไป

หากผู้ใดสนใจจะนำตัว prototype ไปพัฒนาต่อ ทางผู้จัดทำขอแนะนำให้มีการปรับปรุงในส่วนของการ preprocessing ให้มีการ scale ค่าของ feature บางตัวให้เหมาะสม หรืออาจจะมีการ feature engineering เพื่อหา feature โดยใช้ dataset ที่มีอยู่ ส่วนโครงสร้างของ Model ก็อาจจะลองพิจารณาใช้ Machine Learning Model ประเภทอื่นๆ ที่เกี่ยวข้องกับงาน Classification เช่น Support Vector Machine เพื่อการประมวลผลและความแม่นยำที่ดียิ่งขึ้น

และหากผู้ใดสนใจจะนำแนวคิดของโครงการนี้ไปพัฒนาต่อ ทางผู้จัดทำขอแนะนำให้ลองหา dataset ที่มีขนาดใหญ่และมีความ General ในระดับสังคม ทั้งนี้หากได้ข้อมูลที่ดีมา Train AI ก็จะได้ Model ที่มีประโยชน์กับผู้คนในสังคมในระดับวงกว้างมากขึ้น

เอกสารอ้างอิง

Mahbub Jhankar. (2020). Don't Buy Alexa! Build Your Own. Create a Virtual Assistant with Python | Python Project | Jarvis AI. Retrieved 20 August 2021 from <https://youtu.be/AWvsXxDtEkU>

Jie Jenn. (2020). Plot Grouped Bar Graph With Python and Pandas. Retrieved 15 September 2021 from <https://youtu.be/1h0LvDg9NA>

Ahmad Anis. (2020). 4 ways to improve your TensorFlow model – key regularization techniques you need to know. Retrieved 20 December 2022 from <https://www.kdnuggets.com/2020/08/tensorflow-model-regularization-techniques.html>

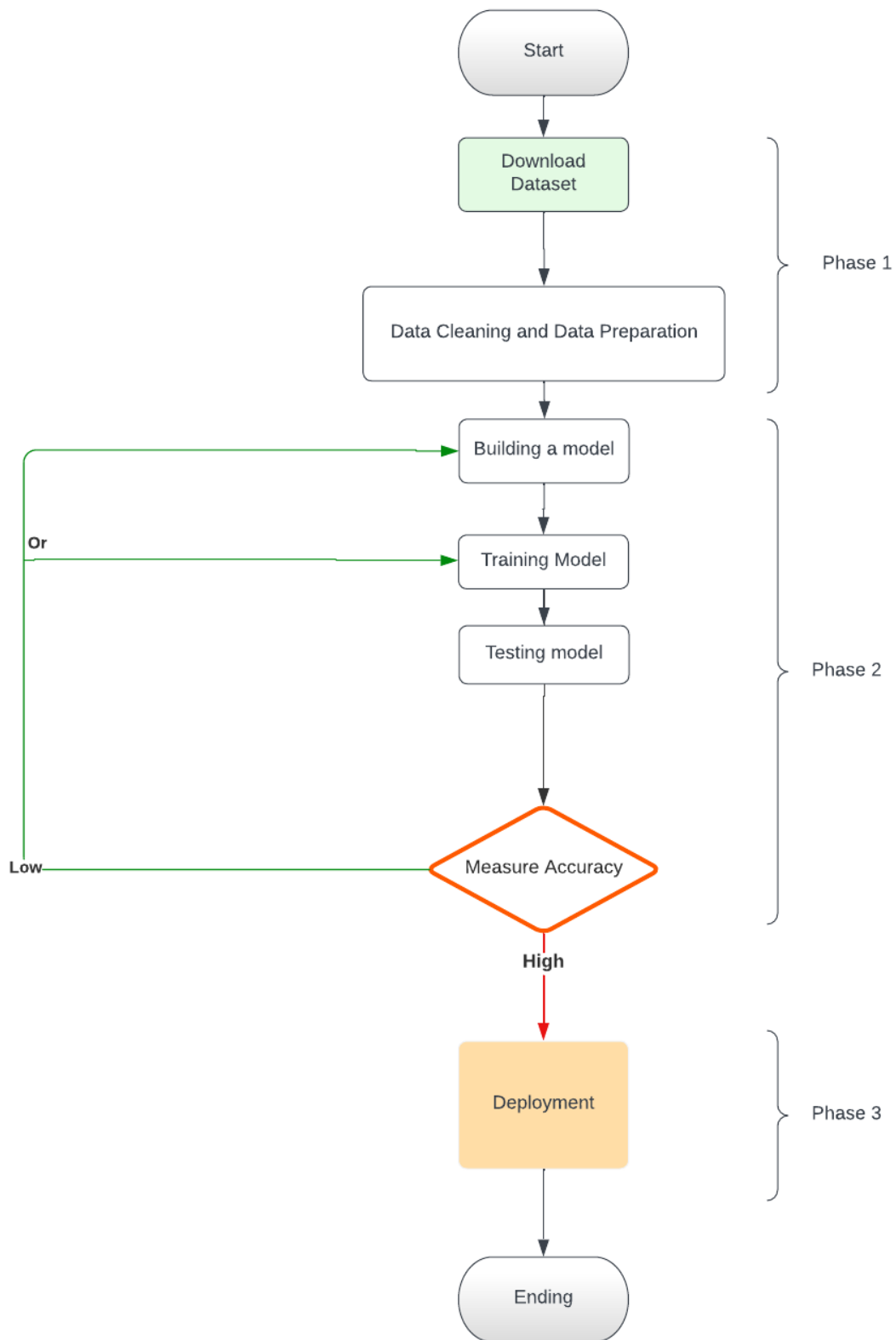
Gianluca Malato. (2022). Which models require normalized data?
A brief overview about models that need pre-processed data. Retrieved 18 December 2022 from <https://towardsdatascience.com/which-models-require-normalized-data-d85ca3c85388>

Packt. (2017). Introduction to Titanic Datasets. Retrieved 15 December 2022 from <https://hub.packtpub.com/introduction-titanic-datasets/>

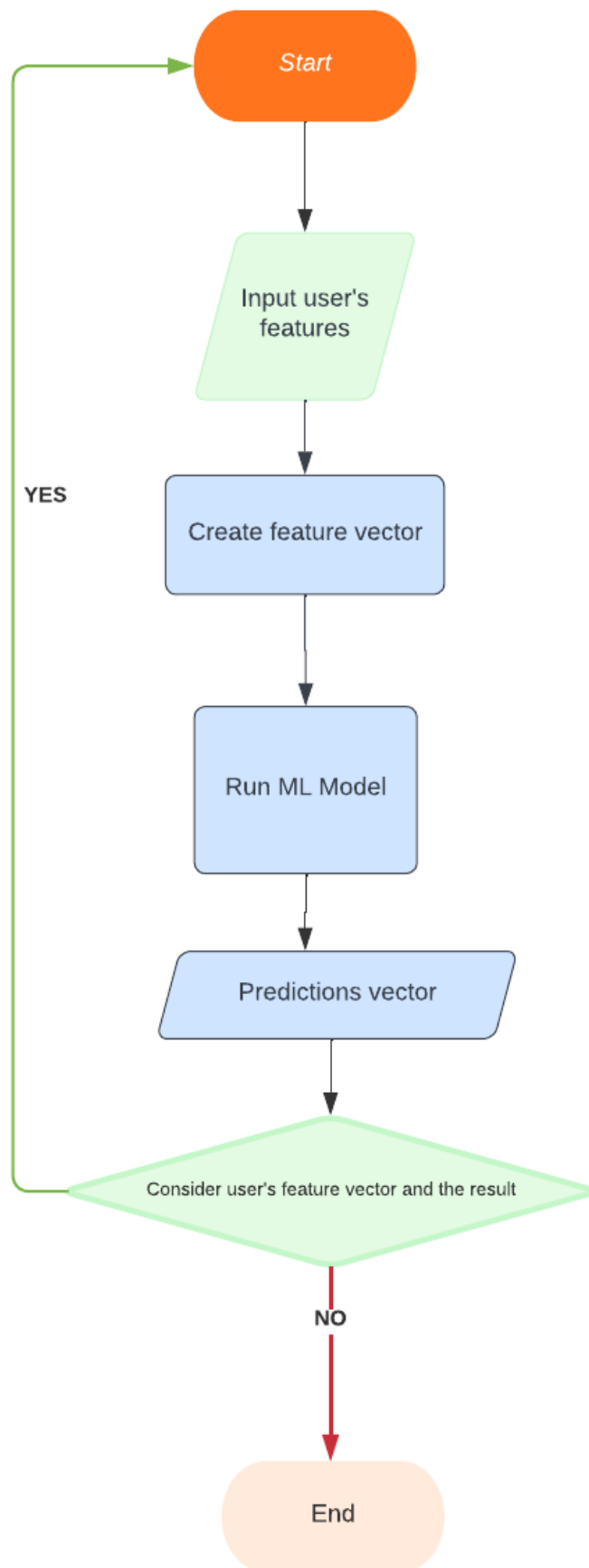
Carlos Raul Morales. (2020). Random Forest on Titanic Dataset. Retrieved 26 December 2022 from <https://medium.com/analytics-vidhya/random-forest-on-titanic-dataset-88327a014b4d>

Çağlar Uslu. (2022). Kaggle Competition Tutorial: Machine Learning from the Titanic. Retrieved 29 November 2022 from <https://www.datacamp.com/tutorial/tutorial-kaggle-competition-tutorial-machine-learning-from-titanic#training>

ภาคผนวก



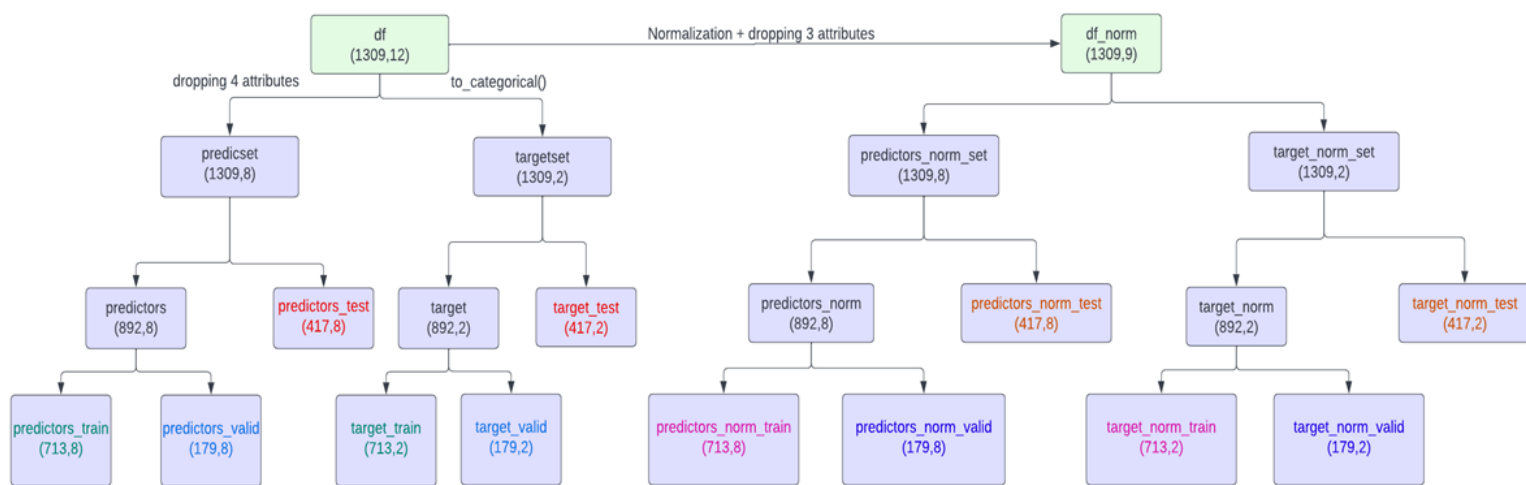
ภาพที่ 1 ขั้นตอนการพัฒนาตั้งแต่ต้นจนจบ



ภาพที่ 3 ภาพรวมการทำงานของ Prototype และ flow การใช้งานของผู้ใช้

PassengerId	0
pclass	0
survived	0
name	0
sex	0
age	0
sibsp	0
parch	0
ticket	0
fare	0
cabin	0
embarked	0

ภาพที่ 2 Attribute ทั้ง 12 หลัง Data Cleaning



ภาพที่ 4 แผนภาพสรุปตัวแปรเก็บข้อมูลจาก Phase 1

```
modelV1 = Sequential()
modelV1.add(Dense(32, activation = "relu" , input_shape = (n_col ,)))
modelV1.add(Dense(32 , activation = "relu"))
modelV1.add(Dense(2 , activation = "softmax"))
```

ภาพที่ 5 Model V1

```
modelV2 = Sequential()
modelV2.add(Dense(32, activation = "relu" , input_shape = (n_col ,)))
modelV2.add(tf.keras.layers.Dropout(0.5))

modelV2.add(Dense(32 , activation = "relu"))
modelV2.add(tf.keras.layers.Dropout(0.5))

modelV2.add(Dense(2 , activation = "softmax"))
```

ภาพที่ 6 Model V2


```

modelV3 = Sequential()
modelV3.add(Dense(32, activation = "relu" , input_shape = (n_col ,)))
modelV3.add(BatchNormalization())
modelV3.add(tf.keras.layers.Dropout(0.5))

modelV3.add(Dense(32 , activation = "relu"))
modelV3.add(BatchNormalization())
modelV3.add(tf.keras.layers.Dropout(0.5))

modelV3.add(Dense(2 , activation = "softmax"))

```

ภาพที่ 7 Model V3

```

modelV4 = Sequential()
modelV4.add(Dense(32, activation = "relu" , input_shape = (n_col ,), kernel_regularizer='l1'))
modelV4.add(BatchNormalization())
modelV4.add(tf.keras.layers.Dropout(0.5))

modelV4.add(Dense(32 , activation = "relu"))
modelV4.add(BatchNormalization())
modelV4.add(tf.keras.layers.Dropout(0.5))

modelV4.add(Dense(2 , activation = "softmax"))

```

ภาพที่ 8 Model V4

```

modelV5 = Sequential()
modelV5.add(Dense(16, activation = "relu" , input_shape = (n_col ,)))
modelV5.add(BatchNormalization())
modelV5.add(tf.keras.layers.Dropout(0.5))

modelV5.add(Dense(16 , activation = "relu"))
modelV5.add(BatchNormalization())
modelV5.add(tf.keras.layers.Dropout(0.5))

modelV5.add(Dense(16 , activation = "relu"))
modelV5.add(BatchNormalization())
modelV5.add(tf.keras.layers.Dropout(0.5))

modelV5.add(Dense(2 , activation = "softmax"))

```

ภาพที่ 9 Model V5

```
modelRF_V1 = RandomForestClassifier(n_estimators=100)
modelRF_V1.fit(predictors_train , target_train)
```

ภาพที่ 10 Model RF V1

```
pred = modelRF_V1.predict(predictors_test)
print("    Accuracy: {}".format(metrics.accuracy_score(target_test, pred)))
```

ภาพที่ 11 Model RF V1 predictions

```
modelRF_V2 = RandomForestClassifier(n_estimators=100)
modelRF_V2.fit(predictors_norm , target_norm)
```

ภาพที่ 12 Model RF V2

```
pred = modelRF_V2.predict(predictors_test)
print("    Accuracy: {}".format(metrics.accuracy_score(target_test, pred)))
```

ภาพที่ 13 Model RF V2 predictions

```
param_grid = {
    'n_estimators': [50, 100, 500, 800],
    'max_depth': [None, 50, 90, 120],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(modelRF_V2, param_grid, cv=5, scoring='accuracy')
grid_search.fit(predictors_norm, target_norm)
```

ภาพที่ 14 param_grid และ การใช้ GridSearchCV

```
best_params = grid_search.best_params_

# Use the best hyperparameters to create a new model
best_model = RandomForestClassifier(**best_params)

# Fit the new model to the training data
best_model.fit(predictors_norm, target_norm)
```

ภาพที่ 15 การนำ best_pames มาใช้เป็น Hyperparameter

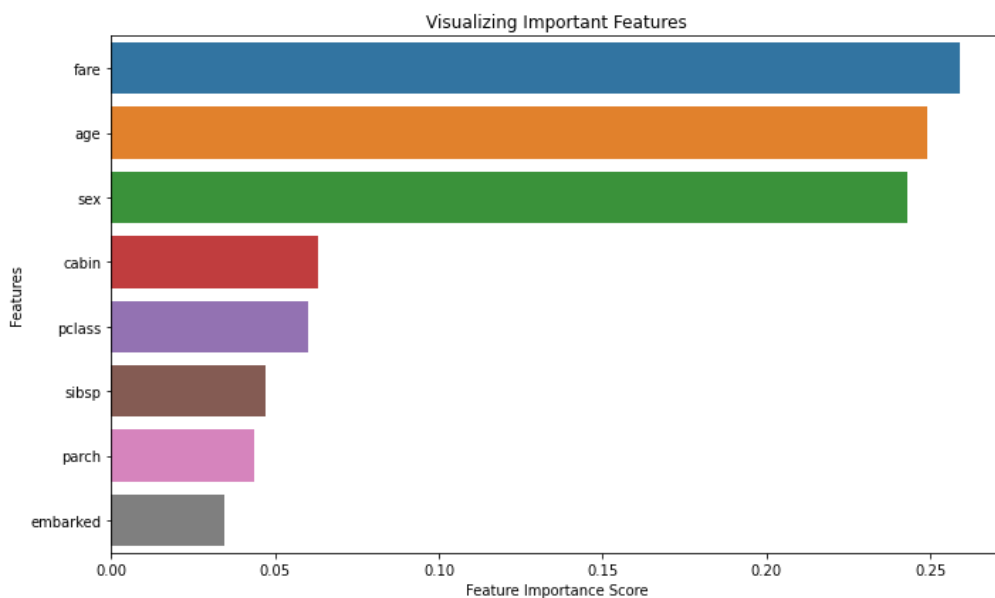
```
# Evaluate the new model on the test set
test_accuracy = best_model.score(predictors_test, target_test)

print(f'Test accuracy: {test_accuracy:.2f}')
```

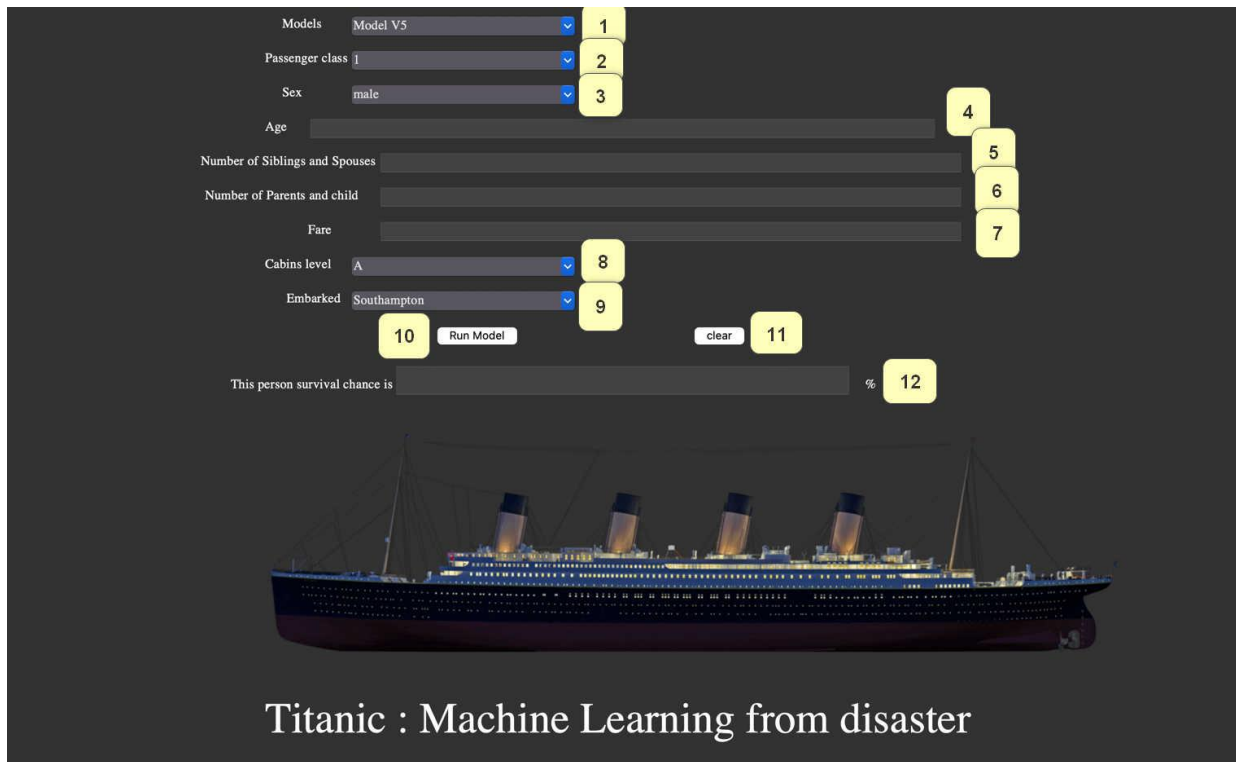
ภาพที่ 16 Model VB predictions

```
pickle.dump(best_model , open("/content/drive/MyDrive/Colab_models/Model_RF_VB_1D.pkl" , "wb"))
```

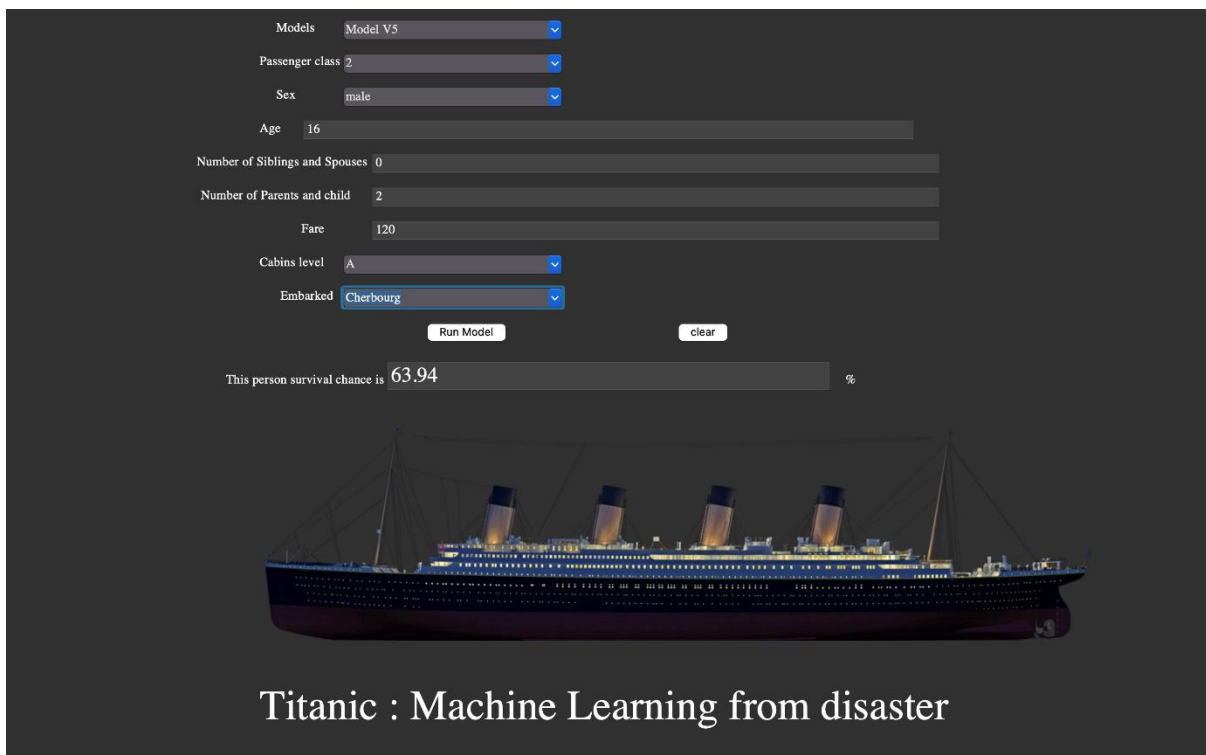
ภาพที่ 17 การ save model ด้วย pickle



ภาพที่ 18 ความสำคัญของแต่ละ Feature ที่ใช้ Train (Model RF V1)



ภาพที่ 19 GUI ของ Prototype



ภาพที่ 20 ตัวอย่างการใช้งานโปรแกรม Prototype เมื่อมีการใส่ค่า feature ตั้งเดิมลงไป พบว่าค่าความน่าจะเป็นในการรอดชีวิตอยู่ในระดับไม่สูงมาก

Models **Model V5**

Passenger class **I**

Sex **male**

Age **16**

Number of Siblings and Spouses **0**

Number of Parents and child **2**

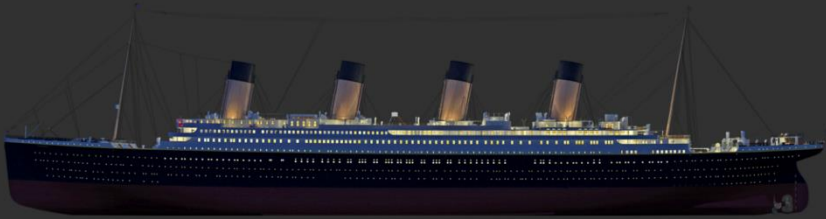
Fare **500**

Cabins level **A**

Embarked **Southampton**

Run Model **clear**

This person survival chance is **92.64** %



Titanic : Machine Learning from disaster

ภาพที่ 21 ตัวอย่างการใช้งานโปรแกรม Prototype เมื่อมีการเปลี่ยนค่า feature ให้ค่าความน่าจะเป็นสูงขึ้น

ข้อมูลผู้จัดทำ



ชื่อ นายภาค นามสกุล หล่อวิจิตร
อายุ 16 ปี
ที่อยู่ 7/68 สวนสยาม 24 แยก 4 แขวงคันนายาว เขตคันนายาว จังหวัด
กรุงเทพมหานคร 10230
เบอร์โทรศัพท์ 0844654515
Email : paklovichit@gmail.com Line ID : eiffelland