รายงานสรุปการทำ Model

เสนอ

ดร. กัญณัฏฐ์ หอมทรัพย์

จัดทำโดย

นายฟาริซ หะยีมะสาและ รหัสนักศึกษา 61050258

ชั้นปีที่ 3 ภาคเรียนที่ 1 ปีการศึกษา 2563

การทำ model Feature Selection

เริ่มจากการ insert data set pima-indians-diabetes1.csv

```
link = 'https://raw.githubusercontent.com/61050960/Dataset/main/pima-indians-diabetes1.csv?raw=true'
df = pd.read_csv(link)
df
```

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure (mm Hg)	Triceps skin fold thickness (mm)	2-Hour serum insulin (mu U/ml)	Body mass index (weight in kg/(height in m)^2)	Diabetes pedigree function	Age (years)	Class variable (0 or 1)
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

หลังจากนั้นทำการสร้าง Method MI สำหรับทำการวัดคะแนนของคอลัมที่เลือกใช้งาน

```
def MI(dat):
  sum = 0
  for i in dat:
    X = df.iloc[:,i]
    y = df.iloc[:,-1]
    MI_score = mutual_info_score(X,y)
    sum += MI_score
  return sum
```

Sequential Forward Selection (SFS)

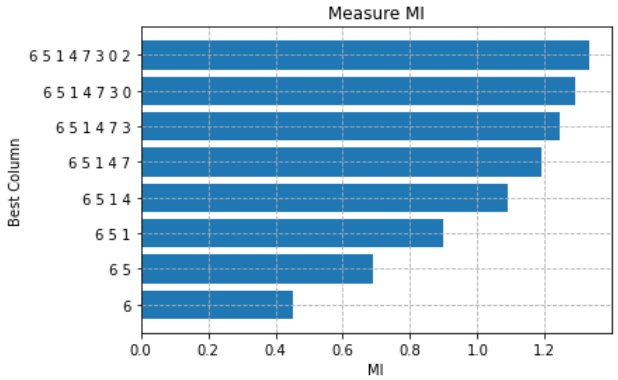
โค้ดส่วนของการทำ Sequential Forward Selection (SFS)

```
max = []
max_s = 0
count_i = len(df.columns) # จำนวน รอบ
count_j =list(range(len(df.columns)-1)) # Create list index เพื่อใช้บอก ตน. คอลัม [0,1,2,3,4,5,7]
column_old = -1
for i in range(count_i):
  for j in count_j: # ไว้เพิ่มค่า ทดลองจับคู่เพื่อเอามาหาค่า MI
    copy = max.copy()
    copy.append(j)
    if(max_s < MI(copy)): #check MI หา Max
      max_s = MI(copy)
      column_new = j
  if(column_old != column_new): # ไว้เช็ค มีคอลัมใหม่ที่ดีกว่าไหม
    column_old = column_new
    max.append(column_old)
    count_j.remove(column_old)
print(max,MI(max))
```

ได้คอลัมและคะแนนของคอลัมทั้งหมดที่ทำการเลือกมาได้คือ

```
[6, 5, 1, 4, 7, 3, 0, 2] 1.3306790760532268
```

กราฟแสดงคะแนนในการเลือกคอลัม



Sequential Backward Selection (SBS)

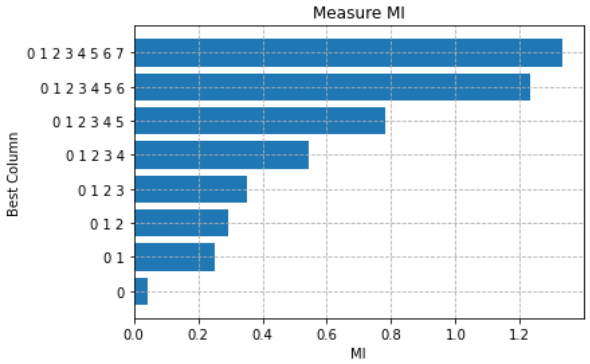
โค้ดส่วนของการทำ Sequential Backward Selection (SFS)

```
max1 = list(range(len(df.columns)-1))
max_s = MI(max1)
count_i = len(df.columns)
column_old = -1
column_new = -1
for i in range(count_i): #loop ตามจำนวนตัวทั้งหมด
  count_j = max1.copy()
  for j in count_j: #loop เพื่อหาตัวที่เอาออกแล้วมีค่ามากที่สุด
    copy = max1.copy() #copy ค่า max มาใส่เพื่อหาตัวถัดไปที่ควรเอาออก
    copy.remove(j) #ลบตัวในรอบนั้น
    if(max_s<MI(copy)): #if ไว้เช็คมาเอาออกแล้วค่ามากขึ้นไหม
      print(max_s,MI(copy))
      max_s = MI(copy)
      column_new = j
  if(column_old != column_new): #if เช็คว่าเอาออกแล้วมีค่ามากสุดยังซ้ำกับตัวเดิมไหม เพื่อกันการที่เอาค่าออกแล้วทำให้ค่า MI น้อยลงกว่าเดิม
    column_old = column_new
    max1.remove(column_old) #ลบออกค่าที่ควรเอาออกจาก max
print(max1,MI(max))
```

ได้คอลัมและคะแนนของคอลัมทั้งหมดที่ทำการเลือกมาได้คือ

```
[0, 1, 2, 3, 4, 5, 6, 7] 1.3306790760532268
```

กราฟแสดงคะแนนในการเลือกคอลัม



Sequential Forward Floating Selection (SFFS)

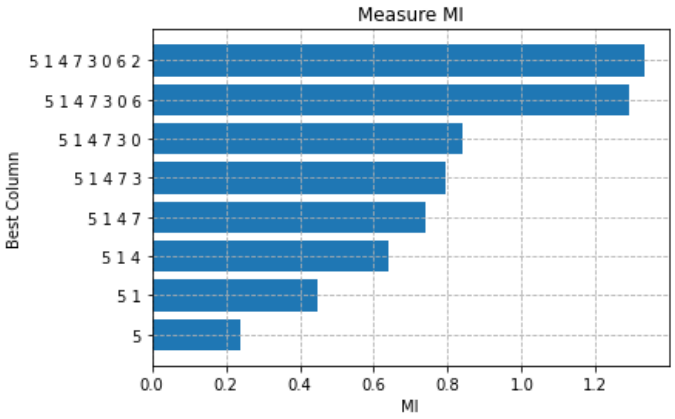
โค้ดส่วนของการทำ Sequential Forward Floating Selection (SFFS)

```
max_Global = 0
max_list_Global = []
max_list1 = []
max_list2 = []
index_col = list(range(len(df.columns)-1)) #[0,1,2,3,4,5,6,7]
index_col_SBS = list(range(len(df.columns)-1))
for i in range(len(df.columns)-1):
  max_local1 = 0
  max_local2 =0
  for j in index_col:
    if j in max_list1:
      continue
    copy = max_list1.copy()
    copy.append(j)
    if(max_local1 < MI(copy)):
      max_local1 = MI(copy)
      column_new_SFS = j
  max_list1.append(column_new_SFS)
    #SFS
  if( i >= 3):
    copy_list = max_list1.copy()
    for k in copy_list:
      temp = max_list1.copy()
      temp.remove(k)
      for m in index_col_SBS:
        if m in temp:
          continue
        temp2 = temp.copy()
        temp2.append(m)
        if(max_local2 < MI(temp2)):
          max_local2 = MI(temp2)
          max_list2 = temp2.copy()
    #SBS
  if(max_local1 < max_local2):
    max_list1 = max_list2.copy()
    max_local1 = max_local2
  if(max_Global < MI(max_list1)):
    max_Global = max_local1
    max_list_Global = max_list1.copy()
print(max_list_Global,MI(max_list_Global))
print(max_list1,MI(max_list1))
```

ได้คอลัมและคะแนนของคอลัมทั้งหมดที่ทำการเลือกมาได้คือ

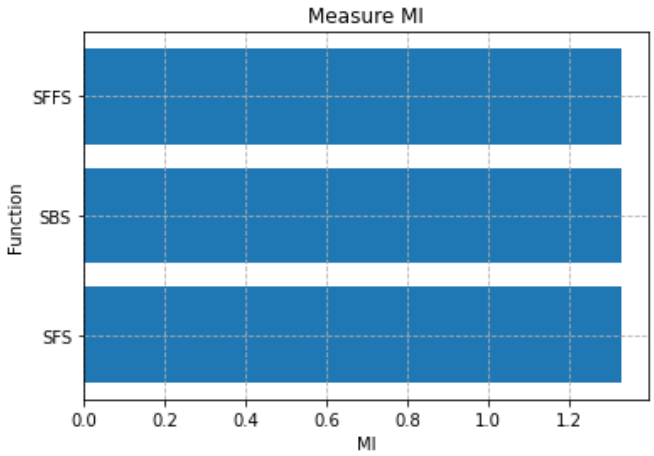
```
[5, 1, 4, 7, 3, 0, 6, 2] 1.330679076053227
```

กราฟแสดงคะแนนในการเลือกคอลัม



การวัดประสิทธิภาพของทั้งสามตัว

กราฟแสดงคะแนนการเลือกของแต่ละตัว



สรุปการวัดประสิทธิภาพ

SFS มีคะแนน MI = 1.330679076053227

SBS มีคะแนน MI = 1.330679076053227

SFFS มีคะแนน MI = 1.330679076053227

ทั้งสามวิธีได้คอลัมออกมาเหมือนกันจึงทำให้ คะแนนเท่ากัน

แต่วิธีที่ดีที่สุดในสามวิธีนี้คือ SFFS เพราะมีการรีเช็คกันระหว่างการทำงานโดยใช้วิธี SFS SBS

ปัญหา SFS SBS เพราะวิธีที่ผมทำจะเป็นแบบ greedy เมื่อเจอการใส่คอลัมแล้วทำให้ค่า MI ลดลงจะหยุด จึงติดปัญหา local maximum อาจจะมีคำตอบที่ดีกว่าแต่ไปไม่ถึงแต่ SFFS จะทำงานไปจนสุดทางและจะมีการจำคำตอบที่ดีที่สุดที่เกิดขึ้นในระหว่างทาง เพื่อแก้ไขปัญหา local maximum

SFS , SBS , SFFS โดยใช้ Library

โดยก่อนจะใช้ Library จะต้องทำการ import ตัว SFS และตัว Model ที่จะใช้วัดคะแนนมาก่อน

โค้ดส่อนของการ Import

```
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LinearRegression

X = df.iloc[:,:-1].values
y = df.iloc[:,-1].values
knn = KNeighborsClassifier(n_neighbors=2)
```

โค้ดส่วนของ SFS

```
# Sequential Forward Selection
sfs = SFS(knn,
          k_features=8,
          forward=True,
          floating=False,
          scoring='accuracy',
          cv=8,
          n_jobs=-1)
sfs = sfs.fit(X, y)
sfs1 = sfs.k_feature_idx_

pd.DataFrame.from_dict(sfs.get_metric_dict()).T
```

SFS ได้คอลัมและคะแนนดังนี้

	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err
1	(1,)	[0.6458333333333334, 0.6979166666666666, 0.645...	0.684896	(1,)	0.0293318	0.0336532	0.0127197
2	(1, 5)	[0.71875, 0.71875, 0.6875, 0.7083333333333334,...	0.721354	(1, 5)	0.0255789	0.0293475	0.0110923
3	(1, 5, 6)	[0.71875, 0.7291666666666666, 0.72916666666666...	0.727865	(1, 5, 6)	0.0205222	0.0235458	0.00889946
4	(1, 4, 5, 6)	[0.75, 0.71875, 0.71875, 0.6875, 0.77083333333...	0.720052	(1, 4, 5, 6)	0.023341	0.0267799	0.0101219
5	(1, 3, 4, 5, 6)	[0.7395833333333334, 0.65625, 0.69791666666666...	0.701823	(1, 3, 4, 5, 6)	0.0217412	0.0249443	0.00942807
6	(1, 2, 3, 4, 5, 6)	[0.6770833333333334, 0.75, 0.71875, 0.72916666...	0.710938	(1, 2, 3, 4, 5, 6)	0.0206785	0.0237251	0.00896724
7	(0, 1, 2, 3, 4, 5, 6)	[0.6770833333333334, 0.7291666666666666, 0.718...	0.713542	(0, 1, 2, 3, 4, 5, 6)	0.0128397	0.0147314	0.00556794
8	(0, 1, 2, 3, 4, 5, 6, 7)	[0.6666666666666666, 0.75, 0.65625, 0.6875, 0....	0.710938	(0, 1, 2, 3, 4, 5, 6, 7)	0.0293318	0.0336532	0.0127197

โค้ดส่วนของ SBS

```
sbs = SFS(knn,
          k_features=8,
          forward=False,
          floating=False,
          scoring='accuracy',
          cv=8,
          n_jobs=-1)
sbs = sbs.fit(X, y)

pd.DataFrame.from_dict(sbs.get_metric_dict()).T
```

SBS ได้คอลัมและคะแนนดังนี้

	avg_score	ci_bound	cv_scores	feature_idx	feature_names	std_dev	std_err
8	0.710938	0.0293318	[0.6666666666666666, 0.75, 0.65625, 0.6875, 0....	(0, 1, 2, 3, 4, 5, 6, 7)	(0, 1, 2, 3, 4, 5, 6, 7)	0.0336532	0.0127197

โค้ดส่วนของ SFFS

```
sffs = SFS(knn,
           k_features=8,
           forward=True,
           floating=True,
           scoring='accuracy',
           cv=8,
           n_jobs=-1)
sffs = sffs.fit(X, y)
pd.DataFrame.from_dict(sffs.get_metric_dict()).T
```

SFFS ได้คอลัมและคะแนนดังนี้

	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err
1	(1,)	[0.6458333333333334, 0.6979166666666666, 0.645...	0.684896	(1,)	0.0293318	0.0336532	0.0127197
2	(1, 5)	[0.71875, 0.71875, 0.6875, 0.7083333333333334,...	0.721354	(1, 5)	0.0255789	0.0293475	0.0110923
3	(1, 5, 6)	[0.71875, 0.7291666666666666, 0.72916666666666...	0.727865	(1, 5, 6)	0.0205222	0.0235458	0.00889946
4	(1, 4, 5, 6)	[0.75, 0.71875, 0.71875, 0.6875, 0.77083333333...	0.720052	(1, 4, 5, 6)	0.023341	0.0267799	0.0101219
5	(1, 3, 4, 5, 6)	[0.7395833333333334, 0.65625, 0.69791666666666...	0.701823	(1, 3, 4, 5, 6)	0.0217412	0.0249443	0.00942807
6	(1, 2, 3, 4, 5, 6)	[0.6770833333333334, 0.75, 0.71875, 0.72916666...	0.710938	(1, 2, 3, 4, 5, 6)	0.0206785	0.0237251	0.00896724
7	(0, 1, 2, 4, 5, 6, 7)	[0.6875, 0.7604166666666666, 0.677083333333333...	0.721354	(0, 1, 2, 4, 5, 6, 7)	0.0286206	0.0328373	0.0124113
8	(0, 1, 2, 3, 4, 5, 6, 7)	[0.6666666666666666, 0.75, 0.65625, 0.6875, 0....	0.710938	(0, 1, 2, 3, 4, 5, 6, 7)	0.0293318	0.0336532	0.0127197

การนำคอลัมที่เลือกมาสร้างโครงสร้างต้นไม้(Decision Tree)

เลือกใช้คอลัม จาก SFFS ที่ได้ Average Score มากที่สุด คือ 0.722656 ได้คอลัม คือ 0, 1, 2, 5, 6, 7

ก่อนเข้า Model Tree ต้องทำการ Import ตัว Model เข้ามาก่อน

โค้ดส่วน Import

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import sklearn as sk
from sklearn.neural_network import MLPClassifier
import xgboost as xgb
```

โค้ดส่วนของ Method Evaluation Accuracy ไว้หาคะแนน Accuracy

```
def evaluation_accuracy(prediction,actual):
  acc = sum(prediction==actual)/len(prediction)
  return acc
```

Non-Normalization (**ยังไม่ทำการ** Normalization)

ต่อมาเป็นการแบ่งส่วนของข้อมูลที่เป็นคำตอบและคอลัมที่เลือกใช้

```
max_list = [0, 1, 2, 5, 6, 7]
X = df.iloc[:,max_list] #Feature_selection_SFFS
y = df.iloc[:,-1]
```

โค้ดของการแบ่งข้อมูล Train และ Test

```
#ทำนาย model โดยใช้ข้อมูลที่ไม่ normalization
X1_train, X1_test, y1_train, y1_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

Decision Tree

```
#DecisionTree
clf_NDT = DecisionTreeClassifier(random_state=0, max_depth=2)
clf_NDT = clf_NDT.fit(X1_train,y1_train)
y_pred_NDT = clf_NDT.predict(X1_test)
print('Accuracy DecisionTree =',evaluation_accuracy(y_pred_NDT,y1_test))
#print(confusion_matrix(y_pred_NDT, y1_test))
```

ค่า Accuracy ที่ได้ของ Decision Tree

```
Accuracy DecisionTree = 0.7575757575757576
```

Random Forest

```
#RandomForest
clf_NRF = RandomForestClassifier(max_depth=2, random_state=0)
clf_NRF = clf_NRF.fit(X1_train,y1_train)
y_pred_NRF = clf_NRF.predict(X1_test)
print('Accuracy RandomForest =',evaluation_accuracy(y_pred_NRF,y1_test))
#print(confusion_matrix(y1_test, y_pred_NRF))
```

ค่า Accuracy ที่ได้ของ Random Forest

```
Accuracy RandomForest = 0.7532467532467533
```

XGBoost

```
#XGBoost
regressor = xgb.XGBClassifier(n_estimators=100, reg_lambda=1, gamma=0, max_depth=3)
regressor.fit(X1_train,y1_train)
y_pred_XGB = regressor.predict(X1_test)
print('Accuracy XGBoost =',evaluation_accuracy(y_pred_XGB,y1_test))
```

ค่า Accuracy ที่ได้ของ XGBoost

```
Accuracy XGBoost = 0.7922077922077922
```

Normalization (ทำการ Normalization)

โค้ดของการ Normalization

```
df_nor = df.copy() #df_normalization
for i in range(len(df_nor.columns)-1): #normalization
  df_nor.iloc[:,i] = (df_nor.iloc[:,i] - df_nor.iloc[:,i].min()) / (df_nor.iloc[:,i].max() - df_nor.iloc[:,i].min())
df_nor
```

แบ่งข้อมูลคำตอบและคอมลัมที่ใช้

```
X_nor = df_nor.iloc[:,max_list] #Feature_selection_SFFS
y_nor = df_nor.iloc[:,-1]
```

โค้ดส่วนของการแบ่งส่วน Train และ Test

```
#ทำนาย model โดยใช้ข้อมูลที่ normalization
X_train, X_test, y_train, y_test = train_test_split(X_nor, y_nor, test_size=0.3, random_state=1)
```

Decision Tree

```
#DecisionTree
clf_NDT = DecisionTreeClassifier(random_state=0, max_depth=2)
clf_NDT = clf_NDT.fit(X1_train,y1_train)
y_pred_NDT = clf_NDT.predict(X1_test)
print('Accuracy DecisionTree =',evaluation_accuracy(y_pred_NDT,y1_test))
#print(confusion_matrix(y_pred_NDT, y1_test))
```

ค่าของ Accuracy ที่ได้ของ Decision Tree

```
Accuracy DecisionTree = 0.7575757575757576
```

Random Forest

```
#RandomForest
clf_NRF = RandomForestClassifier(max_depth=2, random_state=0)
clf_NRF = clf_NRF.fit(X1_train,y1_train)
y_pred_NRF = clf_NRF.predict(X1_test)
print('Accuracy RandomForest =',evaluation_accuracy(y_pred_NRF,y1_test))
#print(confusion_matrix(y1_test, y_pred_NRF))
```

ค่าของ Accuracy ที่ได้ของ Random Forest

```
Accuracy RandomForest = 0.7532467532467533
```

XGBoost

```
#XGBoost
regressor = xgb.XGBClassifier(n_estimators=100, reg_lambda=1, gamma=0, max_depth=3)
regressor.fit(X1_train,y1_train)
y_pred_XGB = regressor.predict(X1_test)
print('Accuracy XGBoost =',evaluation_accuracy(y_pred_XGB,y1_test))
```

ค่า Accuracy ที่ได้ของ XGBoost

```
Accuracy XGBoost = 0.7922077922077922
```

สรุปการนำข้อมูลเข้า Model

ค่า Accuracy ของก่อนทำการ Normalize และ หลังทำการ Normalize มีค่าเท่ากัน อาจเกิดจากการที่ข้อมูลมีการจัดการมาแล้วจึงทำให้ค่าของก่อนและหลังทำการ Normalize จึงมีค่าเท่ากัน

Multiple Regression

เริ่มจากการ Import Library และ Import Data Set

โค้ดส่วนการ Import Library

```
import pandas as pd
from sklearn.metrics import mutual_info_score
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

โค้ดส่วนของการ Import Data Set

```
link = 'https://raw.githubusercontent.com/farisknight13/Python_ML/main/%E0%B8%99%E0%B9%89%E0%B8%B3%E0%B8%A1%E0%B8%B1%E0%B8%99%E0%B8%A3%E0%B8%A7%E0%B8%A1.csv'
df = pd.read_csv(link)
df
```

ตัวอย่าง Dataset

	CALENDAR_YEAR	MONTH_DESC	MONTH_CD	TAX_AMT	TOTAL_TAX_AMT	VOLUMN	TOTAL_VOLUMN	PRICE	PRICE_FACTORY	TOTAL_VOLUMN_CAPA	CAPA	VAL_RATE	TAXQTY_UNIT	ART_TAXV	ART_TOTALTAXV	LAG_TIME_TOTAL_TAX_AMT
0	2554	มกราคม	1	2.222302e+10	1.371691e+10	4.344679e+09	2.863381e+09	0.000000	0.000000	3.633069e+07	1.286312e+07	11.053448	5.076811	3.704450	5.150043	0.000000e+00
1	2554	กุมภาพันธ์	2	2.164869e+10	1.407221e+10	4.220640e+09	2.914440e+09	0.000000	0.000000	7.478745e+07	3.279386e+06	11.229805	4.969179	5.021377	6.429500	1.371691e+10
2	2554	มีนาคม	3	2.511702e+10	1.353646e+10	4.896327e+09	2.865928e+09	0.000000	0.000000	3.713671e+07	3.346770e+07	11.589345	5.101931	3.734156	5.349773	1.407221e+10
3	2554	เมษายน	4	2.261955e+10	1.324597e+10	4.597110e+09	2.848384e+09	0.000000	0.000000	3.694402e+07	3.134440e+07	11.864044	4.974018	3.488816	5.101580	1.353646e+10
4	2554	พฤษภาคม	5	8.343328e+09	4.914525e+09	4.743274e+09	2.967009e+09	0.000000	0.000000	6.134105e+07	5.331400e+07	11.384126	3.887956	2.687062	3.752981	1.324597e+10
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
103	2562	สิงหาคม	8	3.106545e+10	2.093361e+10	5.434510e+09	3.818035e+09	49.936558	67.880428	5.201406e+09	2.133556e+08	0.009600	5.100187	10.758439	11.020983	2.248958e+10
104	2562	กันยายน	9	3.081077e+10	2.113849e+10	5.380393e+09	3.795921e+09	471.270093	490.512497	4.631266e+09	1.352323e+08	0.064000	5.085954	7.666068	7.959199	2.093361e+10
105	2562	ตุลาคม	10	5.135828e+10	2.907713e+10	9.029945e+09	5.490935e+09	398.927588	410.381595	7.118240e+09	4.039597e+08	0.050032	5.129802	5.415956	6.702060	2.113849e+10
106	2562	พฤศจิกายน	11	2.919940e+10	1.639425e+10	5.134057e+09	3.117933e+09	282.659714	314.805547	4.061296e+09	4.277479e+08	0.004174	5.142831	6.913818	7.873968	2.907713e+10
107	2562	ธันวาคม	12	3.016830e+10	1.697095e+10	5.277681e+09	3.062632e+09	152.550383	185.922437	4.044626e+09	2.655927e+08	0.009031	5.159901	7.837104	8.708511	1.639425e+10

108 rows × 52 columns

Non-Normalization

โค้ดส่วนของการแบ่งข้อมูล Train 2554-2561 Test 2562

```
feature_train = df.iloc[:96,:19]
label_train = df.iloc[:96,4]
feature_test = df.iloc[96:,:19]
label_test = df.iloc[96:,4]
```

การ Drop คอลัมที่ไม่ได้เกี่ยวข้องกับการคำนวน เช่น ปี เดือน คำตอบ

```
feature_train.drop(['CALENDAR_YEAR', 'MONTH_CD','MONTH_DESC','TOTAL_TAX_AMT'], axis=1, inplace=True)
feature_test.drop(['CALENDAR_YEAR', 'MONTH_CD','MONTH_DESC','TOTAL_TAX_AMT'], axis=1, inplace=True)
feature_train
```

โค้ดส่วนของการ Drop คอลัมที่มี Correlation มากกว่า 0.90

```
corr_matrix = feature_train.corr().abs()

# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))

# Find features with correlation greater than 0.90
to_drop = [column for column in upper.columns if any(upper[column] > 0.90)]

# Drop features
feature_train.drop(to_drop, axis=1, inplace=True)
feature_test.drop(to_drop, axis=1, inplace=True)
```

โค้ดส่วนของการจัดการข้อมูลที่มีค่าเป็น 0 โดยการใส่ค่า Mean เข้าไปแทน

```
mean_p = feature_train['PRICE'].mean()
print(mean_p)
mean_pf = feature_train['PRICE_FACTORY'].mean()
print(mean_pf)
```

```
47.843799518531256
125.69256250418319
```

```
feature_train['PRICE'] = feature_train['PRICE'].replace([0],mean_p)
feature_train['PRICE_FACTORY'] = feature_train['PRICE_FACTORY'].replace([0],mean_pf)
feature_train
```

โค้ดการนำข้อมูลเข้า Model เพื่อทำนายค่า predict ของปี 2562

```
regr = linear_model.LinearRegression()
regr.fit(feature_train,label_train)
pred = regr.predict(feature_test)
```

ค่าที่ทำนายออกมา

```
print(label_test)
print(pred)
```

```
96     2.184130e+10
97     2.230267e+10
98     2.150852e+10
99     2.282755e+10
100    2.283282e+10
101    2.235165e+10
102    2.248958e+10
103    2.093361e+10
104    2.113849e+10
105    2.907713e+10
106    1.639425e+10
107    1.697095e+10
Name: TOTAL_TAX_AMT, dtype: float64
[2.23015524e+10 2.07644159e+10 2.07022085e+10 2.12683291e+10
 2.20828420e+10 2.07081932e+10 2.05654707e+10 1.83059166e+10
 1.96042017e+10 3.31769866e+10 1.74147455e+10 1.84965765e+10]
```

หลังจากนั้นนำค่าที่ทำนายออกมาไปเทียบกับคำตอบ แล้วคิดค่า MEA% ออกมา

```
test = df.iloc[96:,[0,2]]
test['ACTUAL'] = label_test
test['Predict'] = pred
test['MAE%'] = (test['ACTUAL']-test['Predict']).abs()/test['ACTUAL']*100
test
```

	CALENDAR_YEAR	MONTH_CD	ACTUAL	Predict	MAE%
96	2562	1	2.184130e+10	2.230155e+10	2.107269
97	2562	2	2.230267e+10	2.076442e+10	6.897157
98	2562	3	2.150852e+10	2.070221e+10	3.748822
99	2562	4	2.282755e+10	2.126833e+10	6.830430
100	2562	5	2.283282e+10	2.208284e+10	3.284642
101	2562	6	2.235165e+10	2.070819e+10	7.352715
102	2562	7	2.248958e+10	2.056547e+10	8.555573
103	2562	8	2.093361e+10	1.830592e+10	12.552510
104	2562	9	2.113849e+10	1.960420e+10	7.258283
105	2562	10	2.907713e+10	3.317699e+10	14.099939
106	2562	11	1.639425e+10	1.741475e+10	6.224740
107	2562	12	1.697095e+10	1.849658e+10	8.989635

Normalization

แบ่งข้อมูลที่จะทำการ Normalization และทำการ Normalization

โค้ดการแบ่งข้อมูลที่จะทำ Normalization

```
df_ = df.iloc[:,4:]
df_
```

โค้ดการ Normalization

```
df_nor = df_.copy() #df_normalization
for i in range(len(df_nor.columns)-1): #normalization
  df_nor.iloc[:,i] = (df_nor.iloc[:,i] - df_nor.iloc[:,i].min()) / (df_nor.iloc[:,i].max() - df_nor.iloc[:,i].min())
df_nor
```

โค้ดการแบ่งข้อมูล Train 2554-2561 Test 2562

```
feature_train_nor = df_nor.iloc[:96,1:15]
label_train_nor = df_nor.iloc[:96,0]
feature_test_nor = df_nor.iloc[96:,1:15]
label_test_nor = df_nor.iloc[96:,0]
```

Drop คอลัมที่มี Correlation มากกว่า 0.90

```
corr_matrix = feature_train_nor.corr().abs()

# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))

# Find features with correlation greater than 0.90
to_drop = [column for column in upper.columns if any(upper[column] > 0.90)]

# Drop features
feature_train_nor.drop(to_drop, axis=1, inplace=True)
feature_test_nor.drop(to_drop, axis=1, inplace=True)
```

จัดการข้อมูลที่เป็นค่า 0 โดยใส่ค่า Mean

```
mean_p_nor = feature_train_nor['PRICE'].mean()
print(mean_p_nor)
mean_pf_nor = feature_train_nor['PRICE_FACTORY'].mean()
print(mean_pf_nor)
```

```
0.054240438300075855
0.03891287311553241
```

```
feature_train_nor['PRICE'] = feature_train_nor['PRICE'].replace([0],mean_p_nor)
feature_train_nor['PRICE_FACTORY'] = feature_train_nor['PRICE_FACTORY'].replace([0],mean_pf_nor)
feature_train_nor
```

โค้ดการนำข้อมูลเข้า Model เพื่อทำนายผลปี 2562

```
regr_nor = linear_model.LinearRegression()
regr_nor.fit(feature_train_nor,label_train_nor)
pred_nor = regr_nor.predict(feature_test_nor)
```

ค่าที่ได้

```
print(label_test_nor)
print(pred_nor)
```

```
96     0.736077
97     0.752469
98     0.724254
99     0.771118
100    0.771305
101    0.754209
102    0.759110
103    0.703827
104    0.711107
105    0.993161
106    0.542546
107    0.563036
Name: TOTAL_TAX_AMT, dtype: float64
[0.73721514 0.69476578 0.71937967 0.71639632 0.76118347 0.71546259
 0.703508   0.61933099 0.65287727 1.13526496 0.59247505 0.59777292]
```

นำค่าที่ได้มาเทียบกับคำตอบเพื่อหาค่า MAE%

```
test_nor = df.iloc[96:,[0,2]]
test_nor['ACTUAL'] = label_test_nor
test_nor['Predict'] = pred_nor
test_nor['MAE%'] = (test_nor['ACTUAL']-test_nor['Predict']).abs()/test_nor['ACTUAL']*100
test_nor
```

	CALENDAR_YEAR	MONTH_CD	ACTUAL	Predict	MAE%
96	2562	1	0.736077	0.737215	0.154662
97	2562	2	0.752469	0.694766	7.668498
98	2562	3	0.724254	0.719380	0.672946
99	2562	4	0.771118	0.716396	7.096360
100	2562	5	0.771305	0.761183	1.312238
101	2562	6	0.754209	0.715463	5.137366
102	2562	7	0.759110	0.703508	7.324619
103	2562	8	0.703827	0.619331	12.005242
104	2562	9	0.711107	0.652877	8.188540
105	2562	10	0.993161	1.135265	14.308215
106	2562	11	0.542546	0.592475	9.202658
107	2562	12	0.563036	0.597773	6.169519

ค่า Intercept

```
print('intercept',regr_nor.intercept_)
```

```
intercept 0.09910156964953709
```

ค่า Coefficient

```
print('coefficient',regr_nor.coef_)
```

```
coefficient [ 0.92265515  0.09954554  0.03477391  0.02266729  0.04106307 -0.08074786
  0.09858102 -0.01929255  0.76735397 -0.8097284 ]
```

ค่า Adjusted r square

```
print('adjusted r square',r2_score(label_test_nor,pred_nor))
```

```
adjusted r square 0.6721293522973857
```

การทำนายผลปี 2563

หาค่าเฉลี่ยของแต่ละคอลัม

```
x0 = feature_train_nor.iloc[:,0].mean()
x1 = feature_train_nor.iloc[:,1].mean()
x2 = feature_train_nor.iloc[:,2].mean()
x3 = feature_train_nor.iloc[:,3].mean()
x4 = feature_train_nor.iloc[:,4].mean()
x5 = feature_train_nor.iloc[:,5].mean()
x6 = feature_train_nor.iloc[:,6].mean()
x7 = feature_train_nor.iloc[:,7].mean()
x8 = feature_train_nor.iloc[:,8].mean()
x9 = feature_train_nor.iloc[:,9].mean()
```

ทำนายผล

```
pred_2563 = regr_nor.predict([[x0,x1,x2,x3,x4,x5,x6,x7,x8,x9]])
print('predict 2563 =',pred_2563)
```

```
predict 2563 = [0.35673017]
```

Outlier

ใช้ Data set pima-indians-diabetes1.csv

```
import pandas as pd
import seaborn as sns
import numpy as np

url = 'https://raw.githubusercontent.com/61050960/Dataset/main/pima-indians-diabetes1.csv?raw=true'
df = pd.read_csv(url)
df
```

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure (mm Hg)	Triceps skin fold thickness (mm)	2-Hour serum insulin (mu U/ml)	Body mass index (weight in kg/(height in m)^2)	Diabetes pedigree function	Age (years)	Class variable (0 or 1)
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

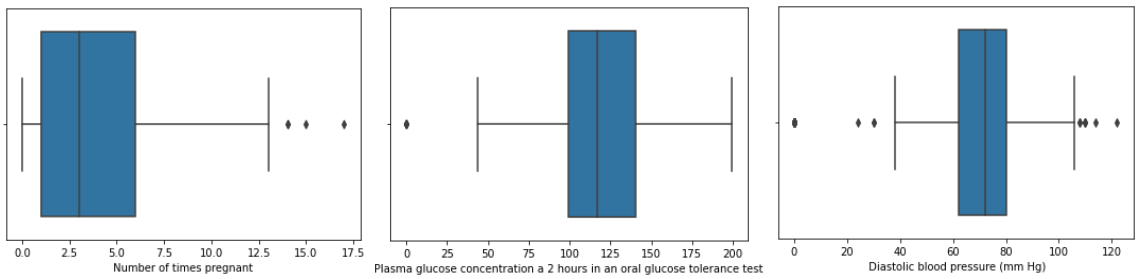
ส่วนของ Box plot แสดงค่าของข้อมูลที่ไม่ได้อยู่ในช่วง หรือข้อมูลที่เบี่ยข้อมูลที่ผิดพลาด

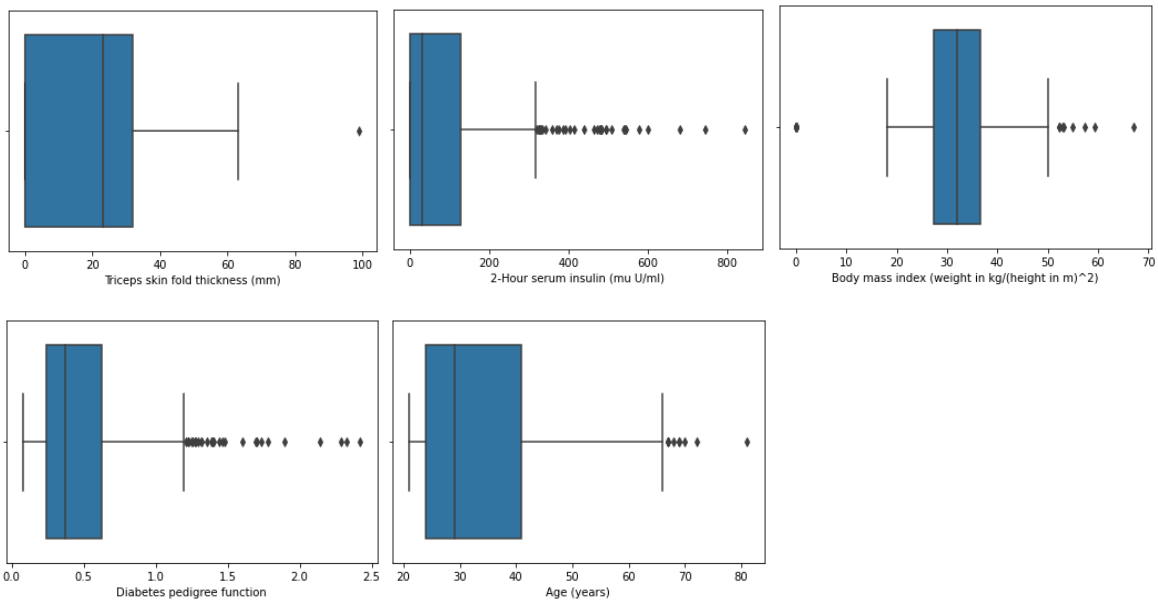
โค้ดของ Box plot ของแต่ละคอลัม

```
test = df.iloc[:,0] test1 = df.iloc[:,1] test2 = df.iloc[:,2] test3 = df.iloc[:,3]
sns.boxplot(x=test) sns.boxplot(x=test1) sns.boxplot(x=test2) sns.boxplot(x=test3)

test4 = df.iloc[:,4] test5 = df.iloc[:,5] test6 = df.iloc[:,6] test7 = df.iloc[:,7]
sns.boxplot(x=test4) sns.boxplot(x=test5) sns.boxplot(x=test6) sns.boxplot(x=test7)
```

กราฟ Box plot





โค้ดของ Method การบอก Upper limit และ Lower limit และเลือกเฉพาะค่าที่อยู่ในช่วง

```
def Outlier(dat,i):
  q25 = dat.iloc[:,i].quantile(0.25)
  q75 = dat.iloc[:,i].quantile(0.75)
  IQR = q75 - q25
  upperlimit = q75 + 1.5*IQR
  lowerlimit = q25 - 1.5*IQR
  outlier = np.bitwise_or(dat.iloc[:,i] > upperlimit,dat.iloc[:,i]< lowerlimit)
  print("upper limit => ",upperlimit)
  print("lower limit => ",lowerlimit)
  dat.iloc[outlier,i] = np.mean(dat.iloc[:,i])
```

ค่า Upper limit และ Lower limit ของแต่ละคอลัม

```
for i in range(len(df.columns)-1):
  print("column : ",i)
  Outlier(df,i)
```

```
column :  0
upper limit =>  13.5
lower limit =>  -6.5
column :  1
upper limit =>  202.125
lower limit =>  37.125
column :  2
upper limit =>  107.0
lower limit =>  35.0
column :  3
upper limit =>  80.0
lower limit =>  -48.0
column :  4
upper limit =>  318.125
lower limit =>  -190.875
column :  5
upper limit =>  50.550000000000004
lower limit =>  13.35
column :  6
upper limit =>  1.2
lower limit =>  -0.32999999999999996
column :  7
upper limit =>  66.5
lower limit =>  -1.5
```

หลังจากเลือกเฉพาะค่าที่อยู่ในช่วงก็จะนำมาทำ Sequential Forward Floating Selection ต่อเพื่อหาคอลัมที่จะนำไปใช้งาน

โค้ดของ Sequential Forward Floating Selection

```
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LinearRegression

X = df.iloc[:,:-1].values
y = df.iloc[:,-1].values
knn = KNeighborsClassifier(n_neighbors=2)
```

```
sffs = SFS(knn,
           k_features=8,
           forward=True,
           floating=True,
           scoring='accuracy',
           cv=8,
           n_jobs=-1)
sffs = sffs.fit(X, y)

pd.DataFrame.from_dict(sffs.get_metric_dict()).T
```

ได้ผลลัพธ์

	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err
1	(1,)	[0.6145833333333334, 0.71875, 0.63541666666666...	0.682292	(1,)	0.0311214	0.0357065	0.0134958
2	(1, 5)	[0.7083333333333334, 0.7291666666666666, 0.687...	0.720052	(1, 5)	0.0258543	0.0296635	0.0112118
3	(1, 5, 6)	[0.7083333333333334, 0.71875, 0.71875, 0.6875,...	0.722656	(1, 5, 6)	0.0231193	0.0265255	0.0100257
4	(1, 4, 5, 6)	[0.71875, 0.6979166666666666, 0.73958333333333...	0.71224	(1, 4, 5, 6)	0.0217412	0.0249443	0.00942807
5	(1, 4, 5, 6, 7)	[0.7291666666666666, 0.7291666666666666, 0.708...	0.71875	(1, 4, 5, 6, 7)	0.025275	0.0289988	0.0109605
6	(0, 1, 3, 4, 5, 7)	[0.7291666666666666, 0.6875, 0.6875, 0.7395833...	0.726562	(0, 1, 3, 4, 5, 7)	0.0211709	0.02429	0.00918078
7	(0, 1, 3, 4, 5, 6, 7)	[0.71875, 0.6875, 0.6875, 0.7395833333333334, ...	0.72526	(0, 1, 3, 4, 5, 6, 7)	0.021262	0.0243945	0.00922026
8	(0, 1, 2, 3, 4, 5, 6, 7)	[0.7083333333333334, 0.7395833333333334, 0.677...	0.71875	(0, 1, 2, 3, 4, 5, 6, 7)	0.0208027	0.0238676	0.0090211

เลือกคอลัมที่มีค่า CV Score มากที่สุดคือ [0,1,3,4,5,7]

นำมาเข้า Model Decision Tree

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import sklearn as sk
import xgboost as xgb

X = df.iloc[:,[0, 1, 3, 4, 5, 7]] #Feature_selection_SFFS
y = df.iloc[:,-1]

def evaluation_accuracy(prediction,actual):
  acc = sum(prediction==actual)/len(prediction)
  return acc

#ทำนาย model โดยใช้ข้อมูลที่ไม่ normalization
X1_train, X1_test, y1_train, y1_test = train_test_split(X, y, test_size=0.3, random_state=1)

#DecisionTree
clf_NDT = DecisionTreeClassifier(random_state=0, max_depth=2)
clf_NDT = clf_NDT.fit(X1_train,y1_train)
y_pred_NDT = clf_NDT.predict(X1_test)
print('Accuracy DecisionTree =',evaluation_accuracy(y_pred_NDT,y1_test))
#print(confusion_matrix(y_pred_NDT, y1_test))

#GradientBoosting
clf_NGB = GradientBoostingClassifier(random_state=0)
clf_NGB = clf_NGB.fit(X1_train,y1_train)
y_pred_NGB = clf_NGB.predict(X1_test)
print('Accuracy GradientBoosting =',evaluation_accuracy(y_pred_NGB,y1_test))

#RandomForest
clf_NRF = RandomForestClassifier(max_depth=2, random_state=0)
clf_NRF = clf_NRF.fit(X1_train,y1_train)
y_pred_NRF = clf_NRF.predict(X1_test)
print('Accuracy RandomForest =',evaluation_accuracy(y_pred_NRF,y1_test))
#print(confusion_matrix(y1_test, y_pred_NRF))

#XGBoost
regressor = xgb.XGBClassifier(n_estimators=100, reg_lambda=1, gamma=0, max_depth=3)
regressor.fit(X1_train,y1_train)
y_pred_XGB = regressor.predict(X1_test)
print('Accuracy XGBoost =',evaluation_accuracy(y_pred_XGB,y1_test))
```

ได้ผลลัพธ์ค่า Accuracy

```
Accuracy DecisionTree = 0.7532467532467533
Accuracy GradientBoosting = 0.7662337662337663
Accuracy RandomForest = 0.7532467532467533
Accuracy XGBoost = 0.7922077922077922
```

การทำ Bagging

โค้ดส่วน Bagging

```
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import classification_report,confusion_matrix
#Bagging
clf_bg = BaggingClassifier(random_state=0)
clf = clf_bg.fit(X1_train,y1_train)
y_pred = clf.predict(X1_test)
y_score = clf.score(X1_test,y1_test)
print("Bagging")
print(classification_report(y1_test,y_pred))
print(confusion_matrix(y1_test,y_pred))
print(y_score)
```

ค่าการทำนายและ confusion matrix

```
Bagging
              precision    recall  f1-score   support

           0       0.79      0.89      0.84       146
           1       0.76      0.59      0.66        85

    accuracy                           0.78       231
   macro avg       0.77      0.74      0.75       231
weighted avg       0.78      0.78      0.77       231

[[130  16]
 [ 35  50]]
0.7792207792207793
```

การทำ Boosting

โค้ดส่วน Boosting

```
#GradientBoosting
clf_NGB = GradientBoostingClassifier(random_state=0)
clf_N = clf_NGB.fit(X1_train,y1_train)
y_pred_NGB = clf_N.predict(X1_test)
y_score = clf_N.score(X1_test,y1_test)
print("Boosting")
print(classification_report(y1_test,y_pred_NGB))
print(confusion_matrix(y1_test,y_pred_NGB))
print(y_score)
```

ค่าการทำนายและ confusion matrix

```
Boosting
              precision    recall  f1-score   support

           0       0.80      0.86      0.83       146
           1       0.72      0.62      0.67        85

    accuracy                           0.77       231
   macro avg       0.76      0.74      0.75       231
weighted avg       0.77      0.77      0.77       231

[[125  21]
 [ 32  53]]
0.7705627705627706
```