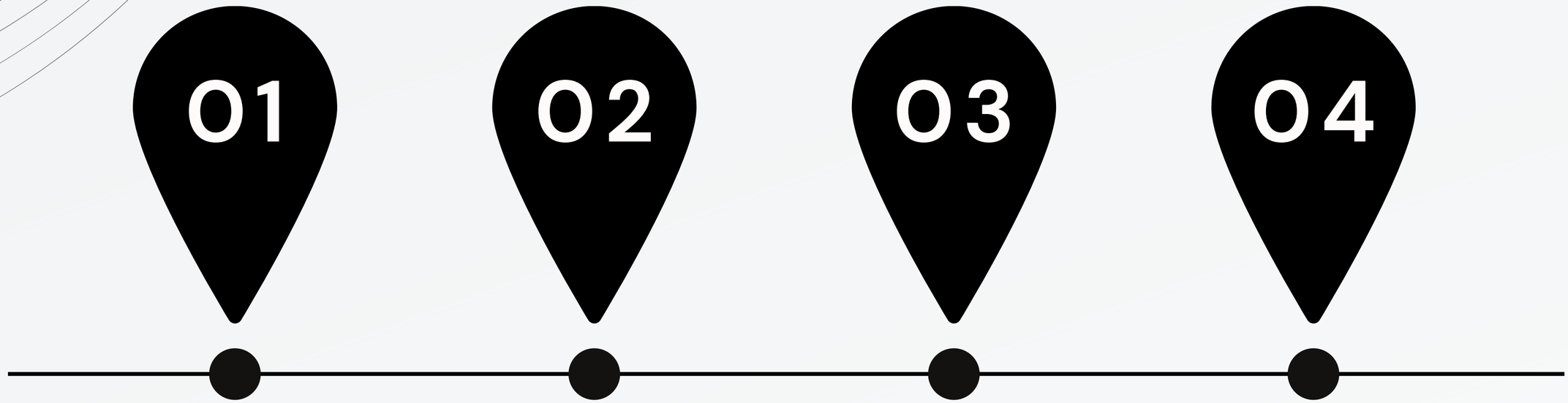
# PROJECT AI

672132003 Sitthiphong Prompalit
672132164 Kamchai Boonsri

# CONTENT

# PAHT 1

## SIMPLE NEURAL NETWORK MODEL TITANIC


01
02
03
04
CLEAN DATA
FEATURE
SPLIT DATASET
TRAIN MODEL

# IMPORT

```
1  import pandas as pd
2  import tensorflow as tf
3  import matplotlib.pyplot as plt
4  import seaborn as sns
```

# CLEAN DATA

```python
dataset = pd.read_csv('train.csv')
for x in dataset.index:
    sex = str(dataset.loc[x, "Sex"])
    embarked = str(dataset.loc[x, "Embarked"])
    pclass = str(dataset.loc[x, "Pclass"])
    if ('female' in sex) == True:
        dataset.loc[x, "female"] = 1
        dataset.loc[x, "male"] = 0
    else:
        dataset.loc[x, "male"] = 1
        dataset.loc[x, "female"] = 0
    if('C' in embarked) == True:
        dataset.loc[x, "Cherbourg"] = 1
        dataset.loc[x, "Queenstown"] = 0
        dataset.loc[x, "Southampton"] = 0
    elif('Q' in embarked) == True:
        dataset.loc[x, "Cherbourg"] = 0
        dataset.loc[x, "Queenstown"] = 1
        dataset.loc[x, "Southampton"] = 0
    elif('S' in embarked) == True:
        dataset.loc[x, "Cherbourg"] = 0
        dataset.loc[x, "Queenstown"] = 0
        dataset.loc[x, "Southampton"] = 1
```

```python
if('1' in pclass) == True:
        dataset.loc[x, "1st"] = 1
        dataset.loc[x, "2nd"] = 0
        dataset.loc[x, "3rd"] = 0
    elif('2' in pclass) == True:
        dataset.loc[x, "1st"] = 0
        dataset.loc[x, "2nd"] = 1
        dataset.loc[x, "3rd"] = 0
    elif('3' in pclass) == True:
        dataset.loc[x, "1st"] = 0
        dataset.loc[x, "2nd"] = 0
        dataset.loc[x, "3rd"] = 1
print(dataset.head())

cleaned = dataset.dropna().replace('', None)


cleaned.to_csv('cleaned.csv', index=False)
print(cleaned.head())
```

# CLEAN DATA

| | Survived | Pclass | Sex | Age | SibSp | Parch | Embarked | male | female | Cherbourg | Queenstown | Southampton | 1st | 2nd | 3rd |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | female | 38.0 | 1 | 0 | C | 0.0 | 1.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| 3 | 1 | 1 | female | 35.0 | 1 | 0 | S | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 1.0 | 0.0 | 0.0 |
| 6 | 0 | 1 | male | 54.0 | 0 | 0 | S | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 | 0.0 | 0.0 |
| 10 | 1 | 3 | female | 4.0 | 1 | 1 | S | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 |
| 11 | 1 | 1 | female | 58.0 | 0 | 0 | S | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 1.0 | 0.0 | 0.0 |

```python
def split_dataset(dataset, train_frac=0.3):
    train = dataset.sample(frac=train_frac)
    val = dataset.drop(train.index)
    return train, val
```

```python

train, val = split_dataset(cleaned)


xtrain = train[['female', 'male', 'Cherbourg','Queenstown','Southampton','1st','2nd','3rd']]
ytrain = train['Survived']

xtest = val[['female', 'male', 'Cherbourg','Queenstown','Southampton','1st','2nd','3rd']]
ytest = val['Survived']

print(xtrain.shape)
print(ytrain.shape)
print(xtest.shape)
print(ytest.shape)
```

```
(55, 8)
(55,)
(128, 8)
(128,)
```

```python

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.InputLayer(input_shape=(8,)))
model.add(tf.keras.layers.Dense(1000, activation='sigmoid'))
model.add(tf.keras.layers.Dense(500, activation='sigmoid'))
model.add(tf.keras.layers.Dense(300, activation='sigmoid'))
model.add(tf.keras.layers.Dense(100, activation='sigmoid'))
model.add(tf.keras.layers.Dense(1, activation='linear'))

print(model.summary())

loss = 'mse'
metric = 'mae'
epochs = 1000
model.compile(loss=loss, optimizer='sgd', metrics=[metric])
history = model.fit(xtrain, ytrain, epochs=epochs, batch_size=64, verbose=1, validation_data=(xtest, ytest))


```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_35 (Dense)            (None, 1000)              9000

 dense_36 (Dense)            (None, 500)               500500

 dense_37 (Dense)            (None, 300)               150300

 dense_38 (Dense)            (None, 100)               30100

 dense_39 (Dense)            (None, 1)                 101

=================================================================
Total params: 690,001
Trainable params: 690,001
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/1000
1/1 [==============================] - 0s 249ms/step - loss: 0.4820 - mae: 0.5059 - val_loss: 0.2263 - val_mae: 0.4020
Epoch 2/1000
1/1 [==============================] - 0s 20ms/step - loss: 0.2400 - mae: 0.4157 - val_loss: 0.2177 - val_mae: 0.4344
Epoch 3/1000
...
Epoch 999/1000
1/1 [==============================] - 0s 31ms/step - loss: 0.2258 - mae: 0.4520 - val_loss: 0.2181 - val_mae: 0.4442
Epoch 1000/1000
1/1 [==============================] - 0s 58ms/step - loss: 0.2258 - mae: 0.4520 - val_loss: 0.2181 - val_mae: 0.4442
```

# ACCURACY

```python
_, accuracy = model.evaluate(xtrain, ytrain)
print('Accuracy: %.2f' % (accuracy*100))
```

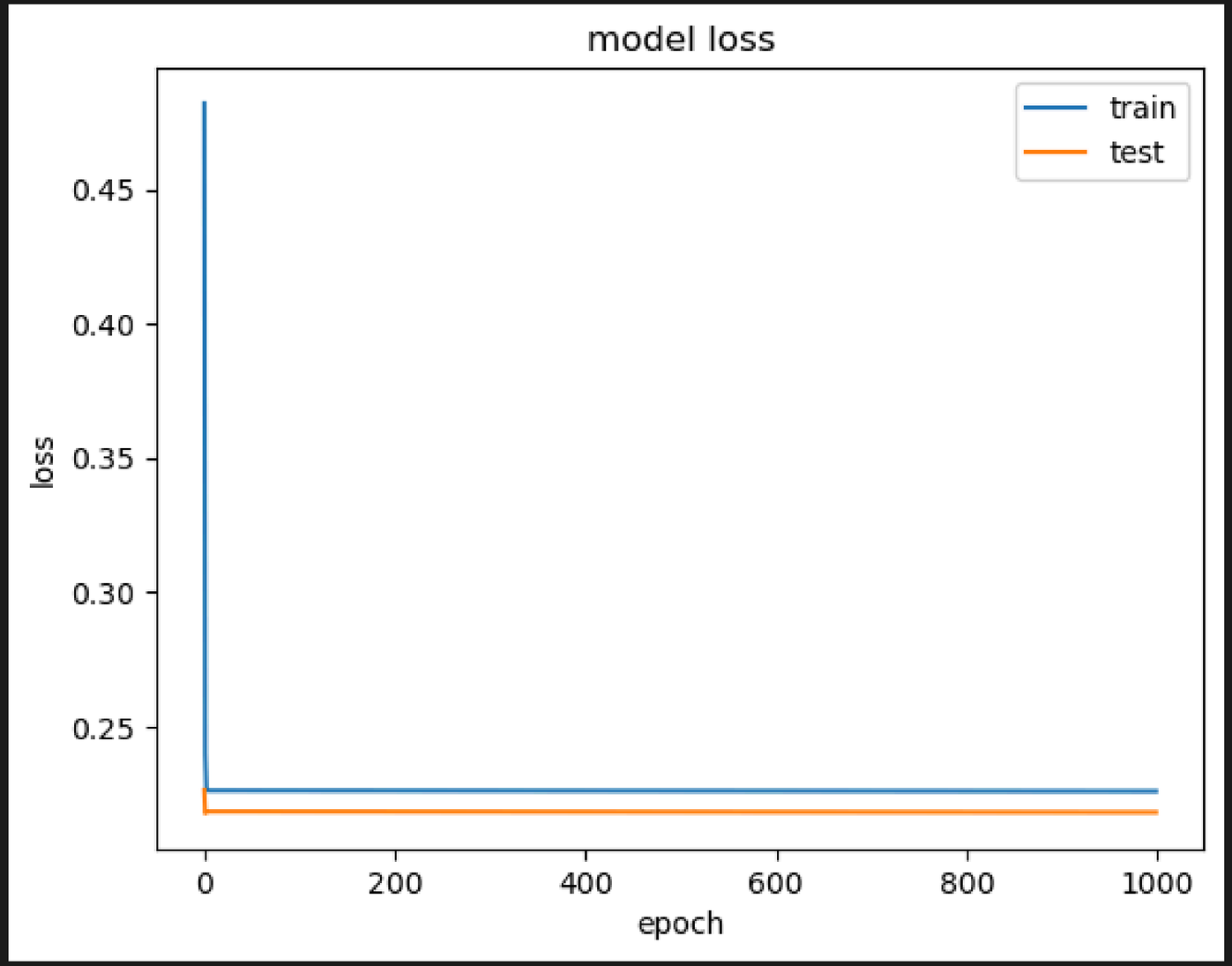
```
2/2 [==============================] - 0s 2ms/step - loss: 0.2258 - mae: 0.4520
Accuracy: 45.20
```

# 45.20 %

Split data 0.3

# Split Data 0.3

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'])
plt.show()
```



# ACCURACY

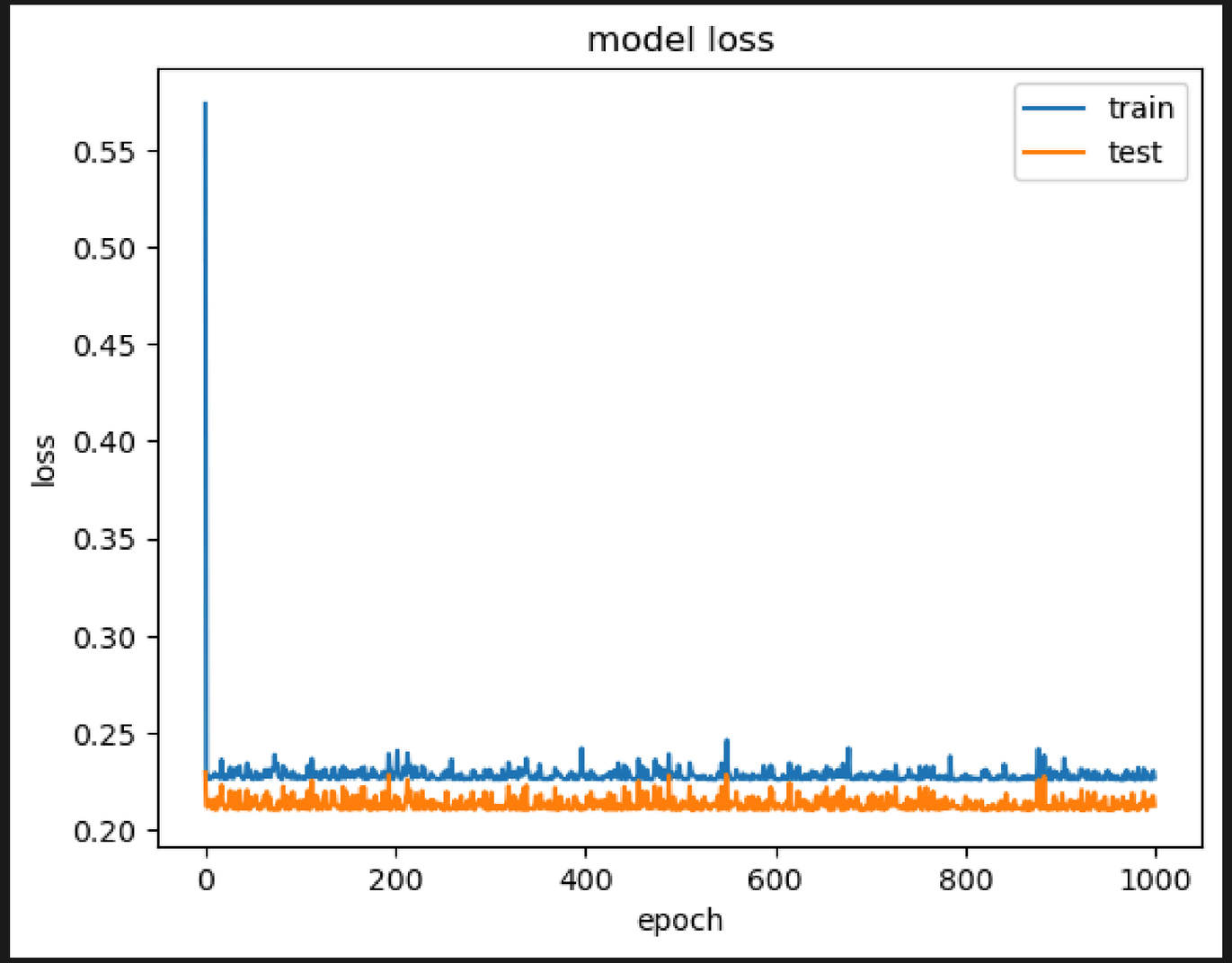
```python
_, accuracy = model.evaluate(xtrain, ytrain)
print('Accuracy: %.2f' % (accuracy*100))
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.2258 - mae: 0.4533
Accuracy: 45.33
```

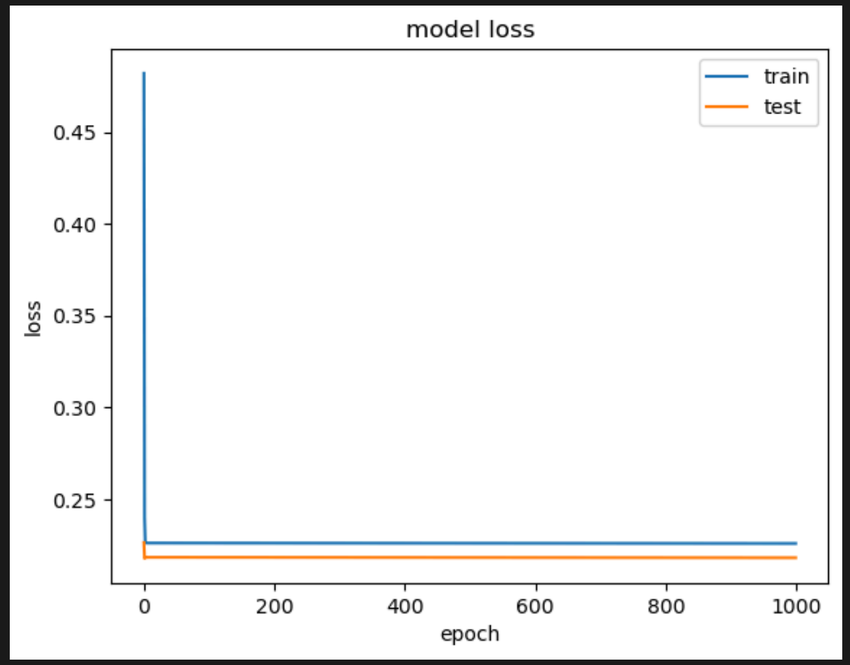
# 45.33 %

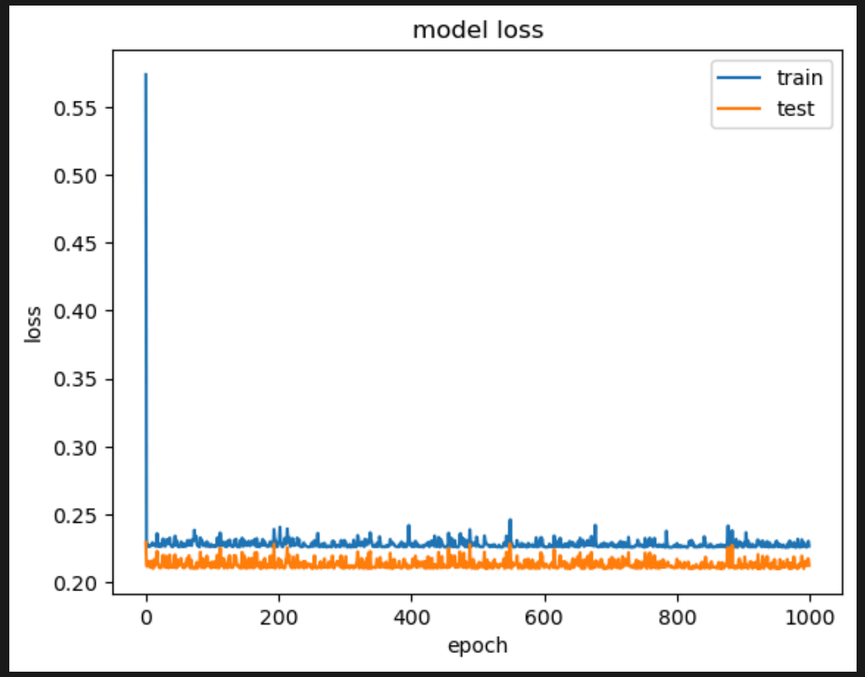
Split data 0.6

# Split Data 0.6



# Result



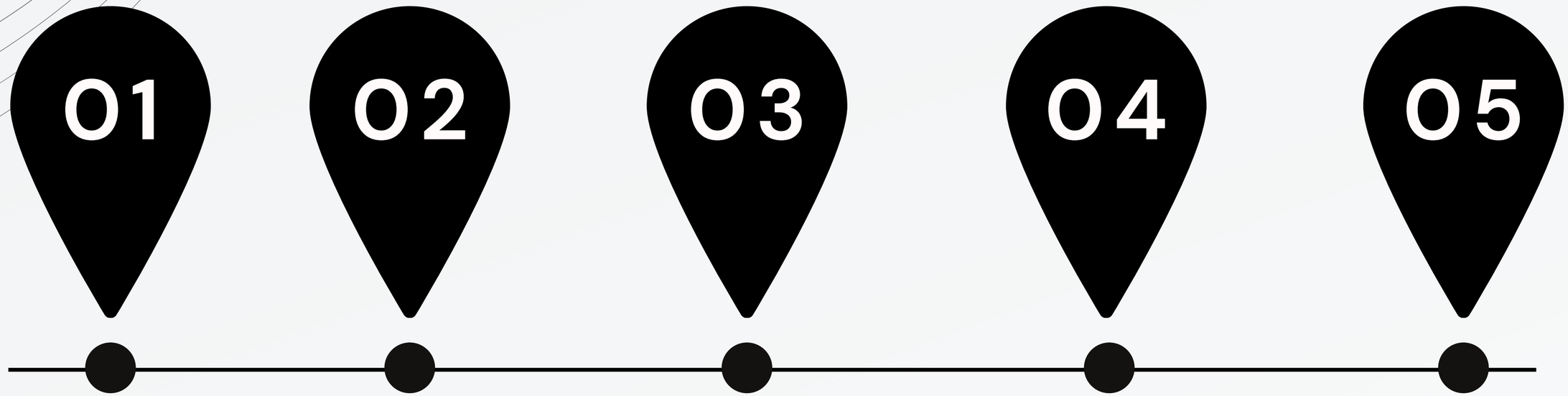




**Data มีค่าความสัมพันธ์กันระหว่าง Feature น้อยเกินไป**

# PAHT 2

## TITANIC FROM KAGGLE COMPETITION

# Titanic Model with 90% accuracy by VK_ds




IMPORT AND LOAD THE DATASET

FILLING MISSING VALUES

FEATURE ENGINEERING

MODELING

HYPER-PARAMETERS TUNING

# Import and load the Dataset

```
1  import numpy as np
2  import pandas as pd
3  import matplotlib.pyplot as plt
4  import seaborn as sns
5  import warnings
6  warnings.filterwarnings('ignore')
7  %matplotlib inline
```

```
1  train_df.head()
```

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 3 | Braund, Mr. Owen Harris | male | 22.0 | 1 | 0 | A/5 21171 | 7.2500 | NaN | S |
| 1 | 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Th... | female | 38.0 | 1 | 0 | PC 17599 | 71.2833 | C85 | C |
| 2 | 3 | 1 | 3 | Heikkinen, Miss. Laina | female | 26.0 | 0 | 0 | STON/O2. 3101282 | 7.9250 | NaN | S |
| 3 | 4 | 1 | 1 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35.0 | 1 | 0 | 113803 | 53.1000 | C123 | S |
| 4 | 5 | 0 | 3 | Allen, Mr. William Henry | male | 35.0 | 0 | 0 | 373450 | 8.0500 | NaN | S |

```
print('__Test_DataSet_')
test_df.head()
```

```
__Test_DataSet_
```

| | PassengerId | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 892 | 3 | Kelly, Mr. James | male | 34.5 | 0 | 0 | 330911 | 7.8292 | NaN | Q |
| 1 | 893 | 3 | Wilkes, Mrs. James (Ellen Needs) | female | 47.0 | 1 | 0 | 363272 | 7.0000 | NaN | S |
| 2 | 894 | 2 | Myles, Mr. Thomas Francis | male | 62.0 | 0 | 0 | 240276 | 9.6875 | NaN | Q |
| 3 | 895 | 3 | Wirz, Mr. Albert | male | 27.0 | 0 | 0 | 315154 | 8.6625 | NaN | S |
| 4 | 896 | 3 | Hirvonen, Mrs. Alexander (Helga E Lindqvist) | female | 22.0 | 1 | 1 | 3101298 | 12.2875 | NaN | S |

```python
def missingdata(data):
    total = data.isnull().sum().sort_values(ascending = False)
    percent = (data.isnull().sum()/data.isnull().count()*100).sort_values(ascending = False)
    ms= pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
    ms= ms[ms["Percent"] > 0]
    return ms
```

```python
missingdata(train_df)
```

| | Total | Percent |
|---|---|---|
| Cabin | 687 | 77.104377 |
| Age | 177 | 19.865320 |
| Embarked | 2 | 0.224467 |

```python
missingdata(test_df)
```

| | Total | Percent |
|---|---|---|
| Cabin | 327 | 78.229665 |
| Age | 86 | 20.574163 |
| Fare | 1 | 0.239234 |

# Filling missing Values

```python
# หาค่าค่าเฉลี่ยของอายุ
test_df['Age'].mean()
```

= 30.272590361445783

```python
# เติมค่าที่ขาดหายไป ในคอลัมน์ 'Embarked' ด้วยค่าที่ปรากฏบ่อยที่สุด ในคอลัมน์นั้นเอง
train_df['Embarked'].fillna(train_df['Embarked'].mode()[0], inplace = True)
```

```python
# เติมค่าที่ขาดหายไป ในคอลัมน์ 'Fare' ด้วยค่าที่ปรากฏบ่อยที่สุด ในคอลัมน์นั้นเอง
test_df['Fare'].fillna(test_df['Fare'].median(), inplace = True)
```

```python
# "Cabin" มีข้อมูลขาดหายมากกว่า 75% ทั้งใน Train และ Test เขาจึงตัด feature นี้ออกไป
drop_column = ['Cabin']
train_df.drop(drop_column, axis=1, inplace = True)
test_df.drop(drop_column,axis=1,inplace=True)
```

```python
# "Age" ในข้อมูล Train และ Test มีข้อมูลขาดหายมากกว่า 15% เขาจึงเติมข้อมูลที่ขาดหายด้วยค่ามัธยฐาน (median)
test_df['Age'].fillna(test_df['Age'].median(), inplace = True)
train_df['Age'].fillna(train_df['Age'].median(), inplace = True)
```

```python
# เช็ค missing value อีกครั้ง
print('check the nan value in train data')
print(train_df.isnull().sum())
print('___'*30)
print('check the nan value in test data')
print(test_df.isnull().sum())

```

```
check the nan value in train data
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
__________________________________________________________________________________
check the nan value in test data
PassengerId    0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

# Feature Engineering

```python
# รวมข้อมูล train และ test เขาด้วยกัน
all_data=[train_df,test_df]
```

```python
# สร้าง feature ใหม่ ด้วยการรวม SibSp และ Parch
# SibSp จำนวนพี่น้อง/คู่สมรสบนเรือไททานิค
# Parch พ่อแม่/ลูกบนเรือไททานิค
# บวกด้วย 1 (ตัวเอง)
# FamilySize เป็น feature รวมจำนวนทั้งหมดในครอบครัวของคนหนึ่งคน

for dataset in all_data:
    dataset['FamilySize'] = dataset['SibSp'] + dataset['Parch'] + 1
```

```python
import re
# function หาคำนำหน้าชื่อ
def get_title(name):
    title_search = re.search(' ([A-Za-z]+)\.', name)
    if title_search:
        return title_search.group(1)
    return ""

# สร้างชื่อ "Title" ซึ่งเก็บคำนำหน้าชื่อ
for dataset in all_data:
    dataset['Title'] = dataset['Name'].apply(get_title)

# รวมคำนำหน้าชื่อที่ไม่พบบ่อยทั้งหมดเป็นกลุ่มเดียวกันชื่อ "Rare"
for dataset in all_data:
    dataset['Title'] = dataset['Title'].replace(['Lady', 'Countess','Capt', 'Col','Don',
                                                 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare')
    dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')
```

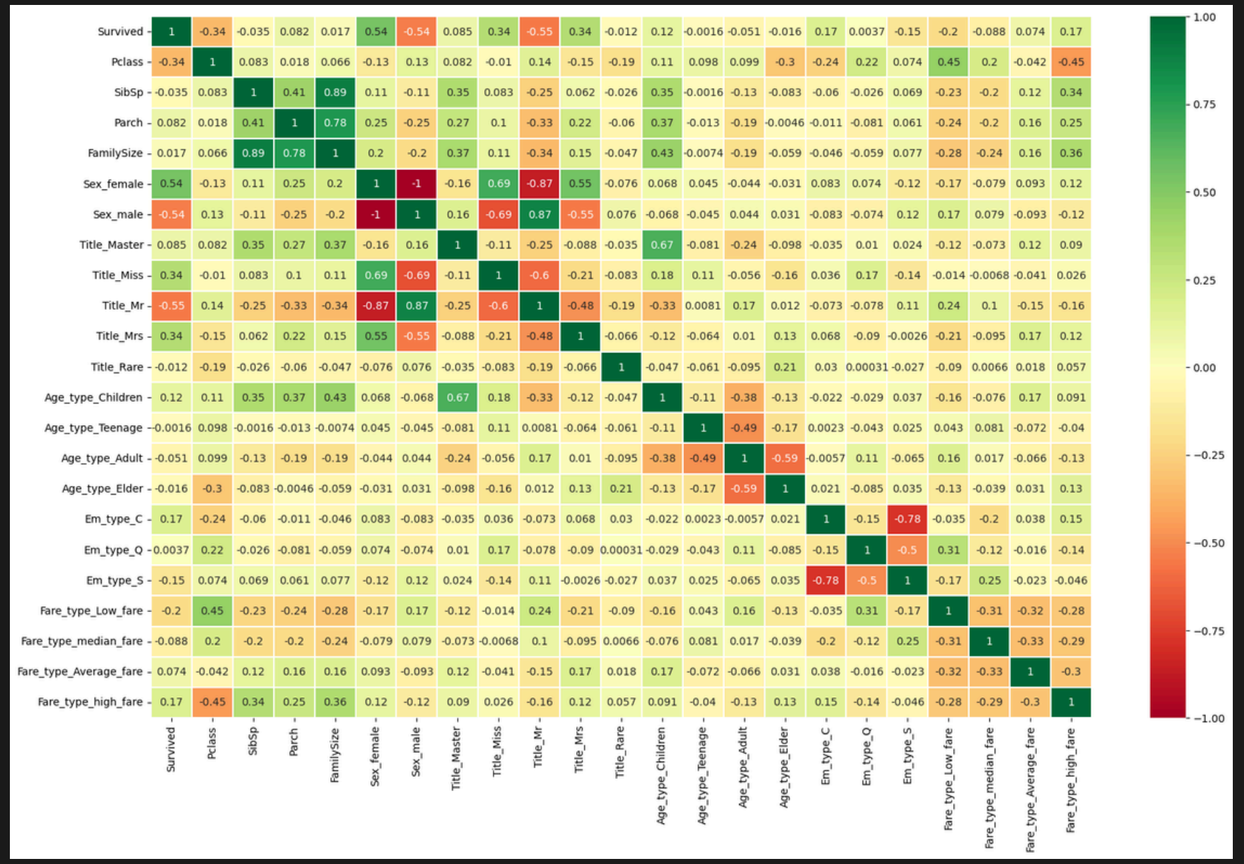
```python
# สร้าง "Age_bin" แบ่งค่าในคอลัมน์ "Age" ออกเป็นกลุ่มต่างๆ ตามช่วงอายุที่กำหนด
for dataset in all_data:
    dataset['Age_bin'] = pd.cut(dataset['Age'], bins=[0,12,20,40,120], labels=['Children','Teenage','Adult','Elder'])
```

```python
# สร้าง "Fare_bin" แบ่งค่าในคอลัมน์ "Fare" ออกเป็นกลุ่มต่างๆ ตามช่วงราคาบัตร
for dataset in all_data:
    dataset['Fare_bin'] = pd.cut(dataset['Fare'], bins=[0,7.91,14.45,31,120], labels=['Low_fare','median_fare',
                                                                                      'Average_fare','high_fare'])
```

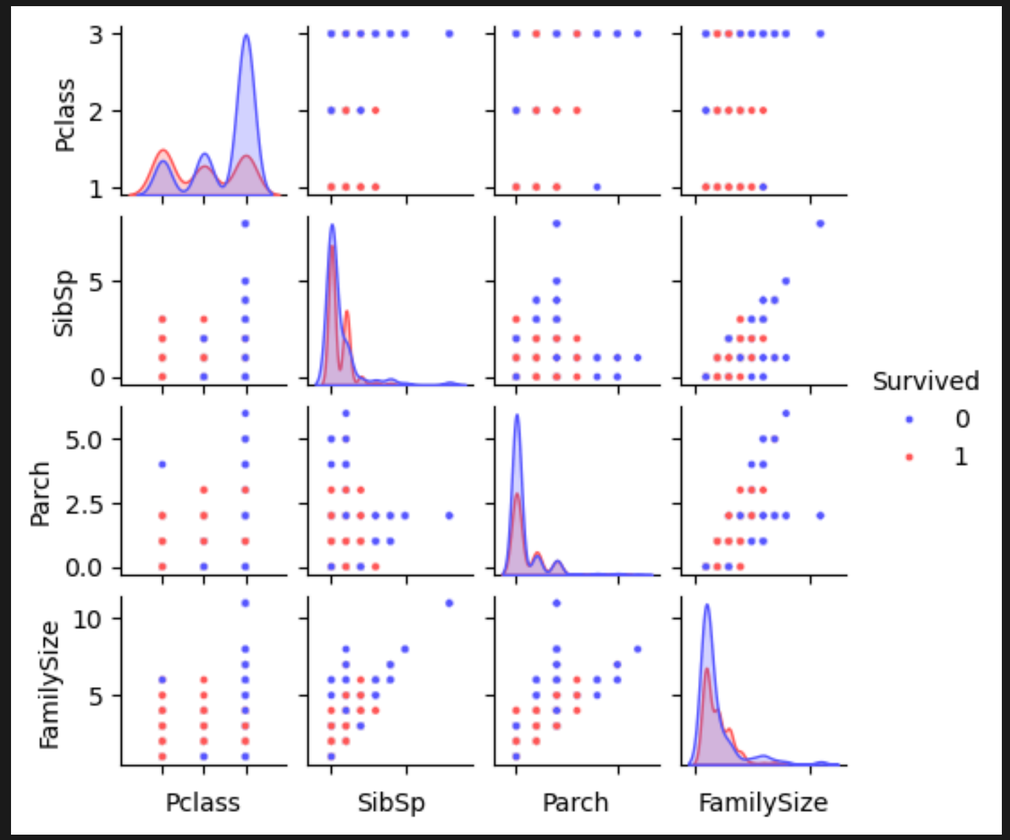
```python
# get_dummies เป็น function สำหรับแปลงข้อมูลเชิงหมวดหมู่ (categorical data) ให้เป็นข้อมูลเชิงตัวเลข (numerical data)
traindf = pd.get_dummies(traindf, columns = ["Sex","Title","Age_bin","Embarked","Fare_bin"],
                         prefix=["Sex","Title","Age_type","Em_type","Fare_type"])
```

```python
testdf = pd.get_dummies(testdf, columns = ["Sex","Title","Age_bin","Embarked","Fare_bin"],
                        prefix=["Sex","Title","Age_type","Em_type","Fare_type"])
```

```python
# แสดงความสัมพันธ์ระหว่าง feature ทั้งหมด
sns.heatmap(traindf.corr(),annot=True,cmap='RdYlGn',linewidths=0.2) #data.corr()-->correlation matrix
fig=plt.gcf()
fig.set_size_inches(20,12)
plt.show()
```



```python
# ใช้ pairplot สร้างกราฟเพื่อสังเกตการกระจายของข้อมูลระหว่าง feature และ lable (Survived) ต่างๆ โดยใช้ Seaborn ช่วย
g = sns.pairplot(data=train_df, hue='Survived', palette = 'seismic', size=1.2,diag_kind = 'kde',diag_kws=dict(shade=True),plot_kws=dict(s=10) )
g.set(xticklabels=[])
```



# Modelling

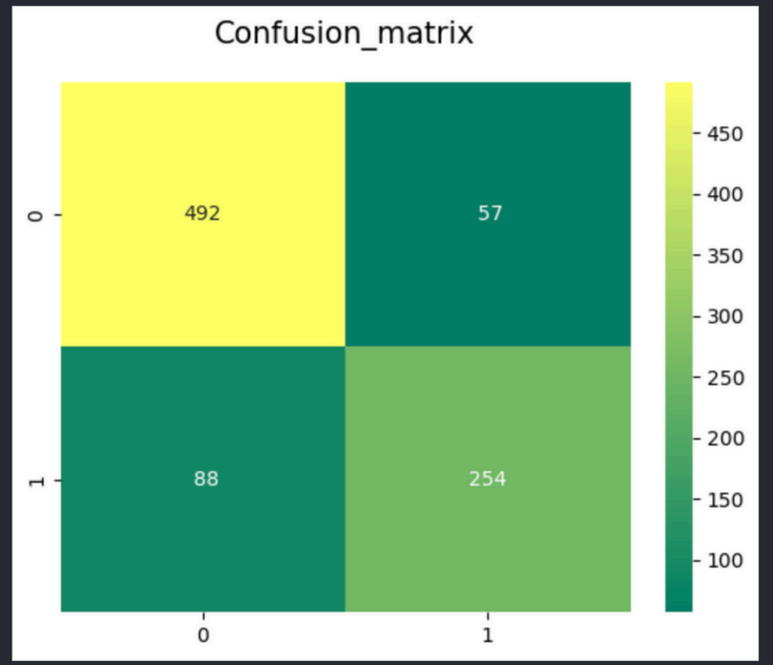
```
from sklearn.model_selection import train_test_split #for split the data
from sklearn.metrics import accuracy_score  #for accuracy_score
from sklearn.model_selection import KFold #for K-fold cross validation
from sklearn.model_selection import cross_val_score #score evaluation
from sklearn.model_selection import cross_val_predict #prediction
from sklearn.metrics import confusion_matrix #for confusion matrix
all_features = traindf.drop("Survived",axis=1)
Targeted_feature = traindf["Survived"]
X_train,X_test,y_train,y_test = train_test_split(all_features,Targeted_feature,test_size=0.3,random_state=42)
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

# Random Forest Classifier

```python
# Random Forests
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(criterion='gini', n_estimators=700,
                               min_samples_split=10,min_samples_leaf=1,oob_score=True,
                               random_state=1,n_jobs=-1)
model.fit(X_train,y_train)
prediction_rm=model.predict(X_test)
print('--------------The Accuracy of the model----------------------------')
print('The accuracy of the Random Forest Classifier is',round(accuracy_score(prediction_rm,y_test)*100,2))
kfold = KFold()
result_rm=cross_val_score(model,all_features,Targeted_feature,cv=10,scoring='accuracy')
print('The cross validated score for Random Forest Classifier is:',round(result_rm.mean()*100,2))
y_pred = cross_val_predict(model,all_features,Targeted_feature,cv=10)
sns.heatmap(confusion_matrix(Targeted_feature,y_pred),annot=True,fmt='3.0f',cmap="summer")
plt.title('Confusion_matrix', y=1.05, size=15)
```

```
--------------The Accuracy of the model----------------------------
The accuracy of the Random Forest Classifier is 82.46
The cross validated score for Random Forest Classifier is: 83.73

Text(0.5, 1.05, 'Confusion_matrix')
```

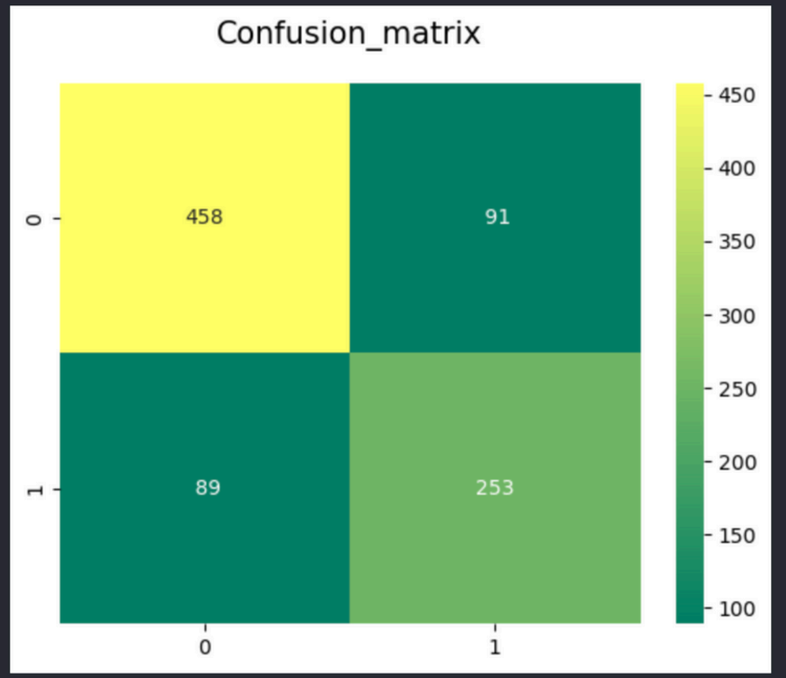


# Gaussian Naive Bayes

```python
# Gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB
model= GaussianNB()
model.fit(X_train,y_train)
prediction_gnb=model.predict(X_test)
print('--------------The Accuracy of the model----------------------------')
print('The accuracy of the Gaussian Naive Bayes Classifier is',round(accuracy_score(prediction_gnb,y_test)*100,2))
kfold = KFold() # k=10, split the data into 10 equal parts
result_gnb=cross_val_score(model,all_features,Targeted_feature,cv=10,scoring='accuracy')
print('The cross validated score for Gaussian Naive Bayes classifier is:',round(result_gnb.mean()*100,2))
y_pred = cross_val_predict(model,all_features,Targeted_feature,cv=10)
sns.heatmap(confusion_matrix(Targeted_feature,y_pred),annot=True,fmt='3.0f',cmap="summer")
plt.title('Confusion_matrix', y=1.05, size=15)
```

```
--------------The Accuracy of the model----------------------------
The accuracy of the Gaussian Naive Bayes Classifier is 79.48
The cross validated score for Gaussian Naive Bayes classifier is: 79.8

Text(0.5, 1.05, 'Confusion_matrix')
```



# Hyper-Parameters Tuning

```
# Random Forests
from sklearn.ensemble import RandomForestClassifier
random_forest = RandomForestClassifier(
            bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=400, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False
 )
random_forest.fit(train_X, train_Y)
Y_pred_rf = random_forest.predict(test_X)
random_forest.score(train_X,train_Y)
acc_random_forest = round(random_forest.score(train_X, train_Y) * 100, 2)

print("Important features")
pd.Series(random_forest.feature_importances_,train_X.columns).sort_values(ascending=True).plot.barh(width=0.8)
print('__'*30)
print(acc_random_forest)

```

# Result After Tuning

```
Important features
```

```
89.11
```