Phần 1. Cài đặt môi trường

1. Cài Python 3.8 trở lên
2. Mở Commond line ở Win (Win+R)=>cmd => Enter
3. Cài các thư viện: numpy, pandas, matplotlib, scikit-learn, seaborn (có thể ko cần dùng --user)

```
C:\Users\Luong Thai Le>pip install --user matplotlib
```

4. Nâng cấp pip:

```
C:\Users\Luong Thai Le>pip install --user --upgrade pip
```

5. Cài jupyter (để sử dụng trình soạn thảo jupyter notebook)

```
C:\Users\Luong Thai Le>pip install --user jupyter
```

6. Sử dụng jupyter để soạn thảo

```
C:\Users\Luong Thai Le>jupyter notebook
```

Phần 2. Phát triển một mô hình học máy cơ bản

I. Thu thập dữ liệu và tìm hiểu dữ liệu
- Vào ws: kaggle.com => competition => lấy bộ dữ liệu Titanic
- Overview => Xem phần Goal, Metric
- Data => Download All => Copy ra desktop
- Tạo thư mục lưu dự án
- Xem các thông số mô tả dữ liệu

II. Chuẩn bị môi trường

- Sau khi mở jupiter như bước 6, trong giao diện của jupiter mở thư mục chứa project => New => Python3 => đặt tên cho Notebook mới
- Chọn Markdown để ghi các chú thích
- Chọn Code để tạo các ô lập trình. Chạy từng đoạn code = nút Run hoặc Shift+Enter
- import các thư viện cần thiết pandas, numpy

```
import pandas as pd
import numpy as np
```

III. Khám phá dữ liệu

- Đọc các file dữ liệu train và test

```
train_df = pd.read_csv("./data/train.csv")
test_df = pd.read_csv("./data/test.csv")
```

- Xem các thuộc tính của file train và test:

```
train_df.columns
```

```
test_df.columns
```

- Xem cấu trúc các ví dụ trong dữ liệu (5 ví dụ đầu)

```
train_df.head()
```

- Xem tổng quan dữ liệu (xem có bao nhiêu ví dụ, xem có thuộc tính nào bị thiếu giá trị và thiếu bao nhiêu…):

```
train_df.info()
```

```
test_df.info()
```

- Kiểm tra thuộc tính target (các lớp cần phân loại của bài toán)

```
train_df["Survived"].value_counts().to_frame()
```

- Import thêm các thư viện giúp visualize dữ liệu:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

- Vẽ đồ thị dạng cột cho thuộc tính Sex

```
sns.countplot(data=train_df, x='Sex');
```

- Vẽ biểu đồ thể hiện sự tương quan giữa giới tính (thuộc tính dạng category, vd: Sex, Pclass…) và việc survive(chỉ trong seaborn mới có chức năng này, matplotlib ko có):

```
sns.countplot(data=train_df, x='Sex', hue='Survived', palette='Blues');
```

- Xem một số thông số mô tả về các thuộc tính:

```
train_df["Age"].describe()
```

- Chia khoảng giá trị cho thuộc tính có nhiều giá trị (Numerical), vd: Fare, Age…

```
fare_categories = ['Economic', 'Standard', 'Expensive', 'Luxury']
pd.qcut(train_df['Fare'], 4, labels=fare_categories)
```

```
pd.qcut(test_df['Fare'],4,labels=fare_categories)
```

- Xem độ tương quan của fare với survive sau khi chia khoảng:

```
fare_categories=['Economic','Standard','Expensive','Luxuxy']
qt_data=pd.qcut(train_df['Fare'],4,labels=fare_categories)
sns.countplot(qt_data,hue=train_df['Survived'])
```

IV. Feature engineering (tạo các feature mới có ích…)

- Kết hợp 2 cột thuộc tính Sibsp và Parch thành 1 cột thuộc tính mới Family_Size

```
train_df['Family_Size']=train_df['SibSp'].astype('int') + train_df['Parch'].astype('int') + 1
```

```
test_df['Family_Size']=train_df['SibSp'].astype('int') + train_df['Parch'].astype('int') + 1
```

- Xem độ tương quan của Family_Size với Survived:

```
sns.countplot(data=train_df,x='Family_Size',hue='Survived')
```

=> thấy thuộc tính này ảnh hưởng đến Survived (người đi 1 mình có khả năng chết cao). Nhưng đang nhiều giá trị => chia khoảng (giống Fare, nhưng khác ở chỗ Fare chia thành các khoảng = nhau (qcut), còn Family_Size sẽ chia theo số người(cut))

```
pd.cut(train_df['Family_Size'],bins=[0,1,4,6,20],labels=['Solo','Small','Medium','Large'])
```

```
pd.cut(test_df['Family_Size'],bins=[0,1,4,6,20],labels=['Solo','Small','Medium','Large'])
```

-Đặt tên mới cho thuộc tính sau khi chia khoảng

```
train_df['Family_Cat']=pd.cut(train_df['Family_Size'],bins=[0,1,4,6,20],labels=['Solo','Small','Medium','Large'])
```

```
test_df['Family_Cat']=pd.cut(test_df['Family_Size'],bins=[0,1,4,6,20],labels=['Solo','Small','Medium','Large'])
```

- Xem độ tương quan của Family_Cat với Survived:

```
sns.countplot(data=train_df,x='Family_Cat',hue='Survived')
```

V. Data wrangling (tiền xử lý dữ liệu: điền những giá trị missing, clean dữ liệu…)

- Xác định lại các thuộc tính dùng để train mô hình:

```
num_features=['Age','Fare']
cat_features=['Sex','Pclass','Embarked','Family_Cat']
feature_cols=num_features + cat_features
print(feature_cols)
```

- Kiểm tra xem thuộc tính nào bị thiếu giá trị:

```
def display_missing(df, feature_cols):
    for col in feature_cols:
        missing_count= df[col].isnull().sum()
        if missing_count >0 :
            print(f'Col {col} has {missing_count} missing value')
display_missing(train_df,feature_cols)
display_missing(test_df,feature_cols)
```

- Điền giá trị bị thiếu:

+ tìm độ tuổi trung bình theo thuộc tính Sex và Pclass:

```
train_df.groupby(['Sex','Pclass']).median()['Age']
```

+ điền các giá trị thiếu của Age bởi giá trị trung bình tương ứng dựa theo Sex và Pclass:

```
train_df['Age']=train_df.groupby(['Sex', 'Pclass'])['Age'].apply(lambda x: x.fillna(x.median()))
```

```
test_df['Age']=test_df.groupby(['Sex', 'Pclass'])['Age'].apply(lambda x: x.fillna(x.median()))
```

+ Chuẩn bị dữ liệu để train mô hình và thực hiện các thao tác chuẩn hoá:

```
X=train_df[feature_cols]
```

```
X.tail()
```

```
y=train_df['Survived']
```

+ Age thiếu nhiều nên phải điền, còn Embarked và Fare thiếu ít hơn thì sẽ dùng scikit-learn để tự động điền:

- Embarked là kiểu categories nên điền theo most frequence
- Fare kiểu num nên điền theo mean

+ Mã hoá OnehotEncoder cho các thuộc tính kiểu cat, vd: Sex, Embarked…

+ Scaler cho các thuộc tính num

```
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
num_transformer=Pipeline(steps=[
    ('imputer',SimpleImputer(strategy='median')),
    ('scaler',StandardScaler())
])
cat_transformer=Pipeline(steps=[
    ('imputer',SimpleImputer(strategy='most_frequent')),
    ('encoder',OneHotEncoder(handle_unknown='ignore'))
])
```

```
preprocessor = ColumnTransformer(transformers=[
    ('num',num_transformer,num_features),
    ('cat',cat_transformer,cat_features)
])
```

```
X=preprocessor.fit_transform(X)
```

```
X_test=test_df[feature_cols]
```

```
X_test=preprocessor.transform(X_test)
```

VI. Huấn luyện mô hình:

- Chia dữ liệu huấn luyện thành tập train và validation:

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_val, y_train, y_val = train_test_split(X,y,test_size=0.2)
```

- Xem số lượng dữ liệu sau khi chia:

```
X_train.shape
```

- **Huấn luyện mô hình với Decision Tree**

```
#DecisionTree
decision_tree=DecisionTreeClassifier()
decision_tree.fit(X_train,y_train)
```

+ Xem độ chính xác của mô hình trên tập va lidation

```
decision_tree.score(X_val,y_val)
```

+ Thay đôi các tham số của mô hình để nâng cao độ cx:

```
decision_tree=DecisionTreeClassifier(criterion = "entropy", max_depth = 5, random_state=2022)
decision_tree.fit(X_train,y_train)
```