Data Mining - Lab 02

- Full name: Đinh Anh Huy
- Student ID: 18110103

In [1]:
```
from google.colab import drive
drive.mount('/content/gdrive')

!ln -s /content/gdrive/My\ Drive/ /mydrive
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_rem
ount=True).
ln: failed to create symbolic link '/mydrive/My Drive': File exists
```

In [2]:
```
path = "/mydrive/Colab Notebooks/Data Mining/Lab02"
import os
os.chdir(path)
```

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")

pd.set_option("max.columns", 100)
pd.set_option("max.rows", 500)

import xgboost
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.ensemble import (GradientBoostingRegressor, GradientBoostingClassifier)
from sklearn.model_selection import train_test_split
```

In [4]:
```python
# Read data
data = pd.read_csv('Dataset/Titanic.csv')

print(">> Display the first 5 rows of data:")
display(data.head())
print(">> Shape of data: ", data.shape)
print("    * Number of rows: ", data.shape[0])
print("    * Number of columns: ", data.shape[1])
```

```
>> Display the first 5 rows of data:
```

	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

```
>> Shape of data:  (891, 12)
   * Number of rows:  891
   * Number of columns:  12
```

Description of Titanic Dataset

- **pclass**: A proxy for socio-economic status (SES)
  - 1st = Upper
  - 2nd = Middle
  - 3rd = Lower
- **age**: Age is fractional if less than 1. If the age is estimated, is it in the form of xx.5
- **sibsp**: The dataset defines family relations in this way...

  - Sibling = brother, sister, stepbrother, stepsister
  - Spouse = husband, wife (mistresses and fiancés were ignored)
- **parch**: The dataset defines family relations in this way...
  - Parent = mother, father
  - Child = daughter, son, stepdaughter, stepson
  - Some children travelled only with a nanny, therefore parch=0 for them.

In [5]:
```
def find_missing_percent(data , showresult = True):
  total = data.isnull().sum().sort_values(ascending=False)
  percent = (data.isnull().sum() / data.isnull().count()).sort_values(ascending=False)
  miss_df = pd.concat([total, percent], axis=1, keys=['TotalMissingValues', 'PercentOfMissing'])

  miss_df = miss_df[miss_df["PercentOfMissing"] > 0.0]
  miss_df = miss_df.reset_index().rename(columns={'index': 'ColumnName'})
  if(showresult):
    print("* Check missing values:")
    print(">> Shape of data: ", data.shape)
    if miss_df.shape[0] == 0:
      print(">> There is no missing value in this data.")
    else:
      print(">> The table of percentage of missing values:")
      display(miss_df)
  return miss_df
```

Check missing values and drop the columns that have the percent of missing value > 60%.

In [6]:
```python
miss_df = find_missing_percent(data)
```

```
* Check missing values:
>> Shape of data:  (891, 12)
>> The table of percentage of missing values:
```

	ColumnName	TotalMissingValues	PercentOfMissing
0	Cabin	687	0.771044
1	Age	177	0.198653
2	Embarked	2	0.002245

In [7]:
```python
drop_cols = list(miss_df[miss_df['PercentOfMissing'] > 0.6].ColumnName)
print('>> The columns have the percent of missing values greater than 60%: {}\n'.format(drop_cols))

data = data.drop(drop_cols, axis=1)
miss_df = find_missing_percent(data)
```

```
>> The columns have the percent of missing values greater than 60%: ['Cabin']

* Check missing values:
>> Shape of data:  (891, 11)
>> The table of percentage of missing values:
```

	ColumnName	TotalMissingValues	PercentOfMissing
0	Age	177	0.198653
1	Embarked	2	0.002245

Missing Handling

1. **Listwise Deletion**

In [8]:
```
def listwise_deletion(data):
  for col in data.columns:
    miss_ind = data[col][data[col].isnull()].index
    data = data.drop(miss_ind, axis = 0)
  return data
```

In [9]:
```
print(">> The shape of original data: ", data.shape)
data_lwd = listwise_deletion(data)
miss_df_lwd = find_missing_percent(data_lwd)
```

```
>> The shape of original data:  (891, 11)
* Check missing values:
>> Shape of data:  (712, 11)
>> There is no missing value in this data.
```

2. **Mean and Mode Imputation**

In [10]:
```
def mean_imputation(data_numeric):
  for col in data_numeric.columns:
    mean = data_numeric[col].mean()
    data_numeric[col] = data_numeric[col].fillna(mean)
  return data_numeric

def mode_imputation(data_categoric):
  for col in data_categoric.columns:
    mode = data_categoric[col].mode().iloc[0]
    data_categoric[col] = data_categoric[col].fillna(mode)
  return data_categoric
```

In [11]:
```
numeric_cols = data.select_dtypes(['float','int']).columns
categoric_cols = data.select_dtypes('object').columns
print(f">> Numeric Columns   : {list(numeric_cols)}")
print(f">> Categoric Columns : {list(categoric_cols)}")

data_numeric = data[numeric_cols]
data_numeric_mean_imp = mean_imputation(data_numeric)
data_categoric = data[categoric_cols]
data_categoric_mode_imp = mode_imputation(data_categoric)

print(">> The shape of original data: ", data.shape)
data_imputed_value = pd.concat([data_numeric_mean_imp, data_categoric_mode_imp], axis = 1)
miss_df_imputed = find_missing_percent(data_imputed_value)
```

```
>> Numeric Columns   : ['PassengerId', 'Survived', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare']
>> Categoric Columns : ['Name', 'Sex', 'Ticket', 'Embarked']
>> The shape of original data:  (891, 11)
* Check missing values:
>> Shape of data:  (891, 11)
>> There is no missing value in this data.
```

3. **XGBoosting for Numerical Features and Mode Imputation for Catagorical Features**

In [12]:
```
def find_missing_index(data_numeric_xgboost, target_cols):
  miss_index_dict = {}
  for tcol in target_cols:
    index = data_numeric_xgboost[tcol][data_numeric_xgboost[tcol].isnull()].index
    miss_index_dict[tcol] = index
  return miss_index_dict

def xgboost_imputation(data_numeric_xgboost, target_cols, miss_index_dict):
  predictors = data_numeric_xgboost.drop(target_cols, axis =1)
  for tcol in target_cols:
    y = data_numeric_xgboost[tcol]
    y = y.fillna(y.mean())
    xgb = xgboost.XGBRegressor(objective="reg:squarederror", random_state=42)
    xgb.fit(predictors, y)
    predictions = pd.Series(xgb.predict(predictors),index= y.index)
    index = miss_index_dict[tcol]
    data_numeric_xgboost[tcol].loc[index] = predictions.loc[index]
  return data_numeric_xgboost
```

In [13]:
```
miss_features = miss_df["ColumnName"].values
target_cols = [feature for feature in miss_features if feature in numeric_cols]
print(f">> The numeric columns have missing values: {target_cols}")

data_numeric_xgboost = data[numeric_cols]
miss_index_dict = find_missing_index(data_numeric_xgboost, target_cols)
data_numeric_xgboost = xgboost_imputation(data_numeric_xgboost, target_cols, miss_index_dict)
data_imputed_xgboost = pd.concat([data_numeric_xgboost, data_categoric_mode_imp], axis = 1)

print(">> The shape of original data: ", data.shape)
miss_df_xgboost = find_missing_percent(data_imputed_xgboost)
```

```
>> The numeric columns have missing values: ['Age']
>> The shape of original data:  (891, 11)
* Check missing values:
>> Shape of data:  (891, 11)
>> There is no missing value in this data.
```

4. **Multiple Imputation by Chained Equations (MICE)**

In [14]:
```
def mice_imputation_numeric(train_numeric):
  iter_imp_numeric = IterativeImputer(GradientBoostingRegressor())
  imputed_train = iter_imp_numeric.fit_transform(train_numeric)
  train_numeric_imp = pd.DataFrame(imputed_train, columns = train_numeric.columns, index= train_numeric.index)
  return train_numeric_imp

def mice_imputation_categoric(train_categoric, max_iter=5, initial_strategy='most_frequent'):
  ordinal_dict={}
  for col in train_categoric:
    ordinal_dict[col] = OrdinalEncoder()
    nn_vals = np.array(train_categoric[col][train_categoric[col].notnull()]).reshape(-1,1)
    nn_vals_arr = np.array(ordinal_dict[col].fit_transform(nn_vals)).reshape(-1,)
    train_categoric[col].loc[train_categoric[col].notnull()] = nn_vals_arr

  iter_imp_categoric = IterativeImputer(GradientBoostingClassifier(), max_iter=max_iter, initial_strategy=initial_strate
  imputed_train = iter_imp_categoric.fit_transform(train_categoric)
  train_categoric_imp = pd.DataFrame(imputed_train, columns=train_categoric.columns, index=train_categoric.index).astype

  for col in train_categoric_imp.columns:
    oe = ordinal_dict[col]
    train_arr= np.array(train_categoric_imp[col]).reshape(-1,1)
    train_categoric_imp[col] = oe.inverse_transform(train_arr)

  return train_categoric_imp
```

In [15]:
```
data_numeric_mice  = mice_imputation_numeric(data_numeric)
data_categoric_mice = mice_imputation_categoric(data_categoric)

data_imputed_mice = pd.concat([data_numeric_mice, data_categoric_mice], axis = 1)

print(">> The shape of original data: ", data.shape)
miss_df_mice = find_missing_percent(data_imputed_mice)
```

```
>> The shape of original data:  (891, 11)
* Check missing values:
>> Shape of data:  (891, 11)
>> There is no missing value in this data.
```

Data Modelling

In [16]:
```
def FeatureEngineering(df1):
  import re
  df = df1.copy()
  df['Title'] = df['Name'].apply(lambda x: re.findall(r"\S+\. ", x)[0].strip()[0:-1])
  df['Title'].loc[~df.Title.isin(['Mr', 'Mrs', 'Miss', 'Master'])] = 'Others'
  df['Is_child'] = np.select([df['Title'].str.lower() == 'master'], ['Y'], 'N')
  df['Nb_Fmly_Mem'] = df['SibSp'].fillna(0) + df['Parch'].fillna(0)
  df = df.drop(['PassengerId', 'Ticket', 'Name'], axis= 1)
  return df
```

In [49]:
```
def LabelEncoder(df):
  data = df.copy()
  from sklearn.preprocessing import LabelEncoder
  label = LabelEncoder()
  data_colums = data.dtypes.pipe(lambda X: X[X=='object']).index
  for col in data_colums:
      data[col] = label.fit_transform(data[col])
  return data
```

In [18]:
```
def DataSplitTrainTest(data_modelling, test_size=0.3, random_state=0):
    train = data_modelling.copy()
    y = train['Survived']
    X = train.drop('Survived', axis=1)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=random_state)
    print(">> Shape of Train Data :", X_train.shape)
    print(">> Shape of Test Data  :", X_test.shape)
    return X_train, X_test, y_train, y_test
```

In [28]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import Lasso, LassoCV, Ridge, RidgeCV
from sklearn import metrics
```

In [47]:
```
def XGBoostModel(X_train, y_train, X_test, y_test):
    abc = XGBClassifier(base_score=0.5, eval_metric='logloss',
              learning_rate=0.300000012,)
    abc = abc.fit(X_train, y_train)
    y_pred_abc = abc.predict(X_test)
    return metrics.accuracy_score(y_test, y_pred_abc)

#Logistic Regression
def logistic_regression(X_train,y_train,X_test,y_test):
  '''
      Purpose: Perform Logistic Regression
      Input: X_train,y_train,X_test,y_test - DataFrame
      Output: The accuracy score of logistic regression
  '''
  lr = LogisticRegression(max_iter=2000,random_state=334)
  lr = lr.fit(X_train, y_train)
  y_pred = lr.predict(X_test)
  return metrics.accuracy_score(y_test, y_pred)

#Random Forest
def random_forest(X_train,y_train,X_test,y_test):
  '''
      Purpose: Perform Random Forest Classifier
      Input: X_train,y_train,X_test,y_test - DataFrame
      Output: The accuracy score of Random Forest
  '''
  rdf=RandomForestClassifier(random_state=334)
  rdf.fit(X_train,y_train)
  y_pred=rdf.predict(X_test)
  return metrics.accuracy_score(y_test,y_pred)

def call_function(X_train, y_train, X_test, y_test, model='XGBoost'):
  if model == 'XGBoost':
    return XGBoostModel(X_train, y_train, X_test, y_test)
  elif model == 'Lasso':
    return BuildLassoModel(X_train, y_train, X_test, y_test)
  elif model == 'Ridge':
    return BuildRidgeModel(X_train, y_train, X_test, y_test)
  elif model == 'LogisticRegression':
    return logistic_regression(X_train, y_train, X_test, y_test)
  elif model == 'RandomForest':
```

```
    return random_forest(X_train, y_train, X_test, y_test)
```

In [54]:

```
datasets = {'Listwise Deletion': data_lwd, 'Mean/Mode Imputation': data_imputed_value, 'XGBoosting': data_imputed_xgboos
algs = ['XGBoost', 'LogisticRegression', 'RandomForest']
accs = {}
for key, data in datasets.items():
  print(f"* {key}")
  df1 = FeatureEngineering(data)
  df = LabelEncoder(df1)
  X_train, X_test, y_train, y_test = DataSplitTrainTest(df)
  # acc = XGBoostModel(X_train, y_train, X_test, y_test)
  accs[key] = []
  for alg in algs:
    acc = call_function(X_train, y_train, X_test, y_test, model=alg)
    accs[key].append(acc)

print("* The accuracy table of multiple modelling in different datas which have various missing handling method:")
accs = pd.DataFrame(accs, index=algs)
display(accs)
```

```
* Listwise Deletion
>> Shape of Train Data : (498, 10)
>> Shape of Test Data  : (214, 10)
* Mean/Mode Imputation
>> Shape of Train Data : (623, 10)
>> Shape of Test Data  : (268, 10)
* XGBoosting
>> Shape of Train Data : (623, 10)
>> Shape of Test Data  : (268, 10)
* MICE
>> Shape of Train Data : (623, 10)
>> Shape of Test Data  : (268, 10)
* The accuracy table of multiple modelling in different datas which have various missing handling method:
```

	Listwise Deletion	Mean/Mode Imputation	XGBoosting	MICE
XGBoost	0.803738	0.828358	0.843284	0.828358
LogisticRegression	0.771028	0.809701	0.820896	0.809701
RandomForest	0.752336	0.809701	0.809701	0.809701

Comment:

- Từ bảng kết quả trên, ta thấy rằng phương pháp sử dụng SGBoosting cho Numerical Features kết hợp Mode Iputation cho Categorical Features cho ra accuracy cao nhất trong 4 phương pháp Missing Handle ở cả 3 mô hình máy học.
- Phương pháp Listwise Deletion cho ra kết quả kém nhất, thấp hơn phương pháp XGBoosting đến 5% accuracy.
- 2 phương pháp còn lại cho ra kết quả tương tự nhau.