Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Кафедра системотехніки

Дисципліна: «Методи та системи штучного інтелекту»

ПРАКТИЧНА РОБОТА № 1

«ПОПЕРЕДНЯ ОБРОБКА ДАНИХ»

Виконала:
ст. гр. ІТКН-18-4
Левченко А.С.

Прийняв:
к.т.н., ст. викл. каф.СТ
Жернова П.Є.

Харків 2020

Мета роботи:

Отримання практичних навичок при підготовці даних для подальшого аналізу та моделювання.

Хід роботи:

1. Необхідно імпортувати необхідні пакети і класи: Pandas, Numpy, Sklearn, matplotlib.pyplot.

```
In [2]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
```

2. Імпортувати набір даних для подальшої роботи.

Для завантаження .csv файлу з даними в pandas використовується функція read_csv ().

```
In [8]: df=pd.read_csv("Data.csv")
        df
```

Out[8]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

3. Перевірити на наявність пропущених даних та заповнити їх. Необхідно скористатись декількома методами для заповнення пропусків.

Дуже часто великі обсяги даних, які готуються для подальшого аналізу, мають пропуски. Для того, щоб можна було використовувати алгоритми машинного навчання, які будують моделі за цими даними, в більшості випадків, необхідно ці пропуски чимось і якось заповнити.

У нашому прикладі, у об'єктів з індексами 4 і 6 відсутні дані в поле Age і Salary. Відсутні дані позначаються як NaN.

Для початку звернемося до методів з бібліотеки pandas, які дозволяють швидко визначити наявність елементів NaN в структурах. Якщо таблиця невелика, то можна використовувати бібліотечний метод isnull. Виглядає це так:

```
In [14]: pd.isnull(df)
```

Out[14]:

	Country	Age	Salary	Purchased
0	False	False	False	False
1	False	False	False	False
2	False	False	False	False
3	False	False	False	False
4	False	False	True	False
5	False	False	False	False
6	False	True	False	False
7	False	False	False	False
8	False	False	False	False
9	False	False	False	False

Таким чином ми отримуємо таблицю того ж розміру, але на місці реальних даних в ній знаходяться логічні змінні, які приймають значення False, якщо значення поля в об'єкта є, або True, якщо значення в даному полі - це NaN. На додаток до цього можна подивитися детальну інформацію про об'єкт, для цього можна скористатися методом info ().

```
In [15]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Country    10 non-null     object
 1   Age        9 non-null      float64
 2   Salary     9 non-null      float64
 3   Purchased  10 non-null     object
dtypes: float64(2), object(2)
memory usage: 448.0+ bytes
```

У нашому прикладі видно, що об'єкт df має чотири стовпці (Country, Age, Salary і Purchased), при цьому в стовпці Country всі об'єкти значимі - НЕ NaN, в стовпці Age - один NaN об'єкт, в стовпці Salary - один NaN об'єкт. Можна скористатися таким підходом для отримання кількості NaN елементів в записах.

```
In [16]: df.isnull().sum()
Out[16]: Country      0
         Age          1
         Salary       1
         Purchased    0
         dtype: int64
```

Такий варіант, на мою думку, зручніший.

Відсутні дані об'єктів можна замінити на конкретні числові значення, для цього можна використовувати метод fillna ().

```
In [17]: df.fillna(0)
Out[17]:
```

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	0.0	Yes
5	France	35.0	58000.0	Yes
6	Spain	0.0	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

Цей метод не змінює поточну структуру, він повертає структуру DataFrame, створену на базі існуючої, з заміною NaN значень на ті, що передані в метод в якості аргументу. Дані можна заповнити середнім значенням по стовпцю.

```
In [18]: df.fillna(df.mean())
```

Out[18]:

	Country	Age	Salary	Purchased
0	France	44.000000	72000.000000	No
1	Spain	27.000000	48000.000000	Yes
2	Germany	30.000000	54000.000000	No
3	Spain	38.000000	61000.000000	No
4	Germany	40.000000	63777.777778	Yes
5	France	35.000000	58000.000000	Yes
6	Spain	38.777778	52000.000000	No
7	France	48.000000	79000.000000	Yes
8	Germany	50.000000	83000.000000	No
9	France	37.000000	67000.000000	Yes

Залежно від завдання використовується той чи інший метод заповнення відсутніх елементів, це може бути нульове значення, математичне сподівання, медіана і т.п. Для заміни NaN елементів на конкретні значення, можна використовувати інтерполяцію, яка реалізована в методі interpolate (), алгоритм інтерполяції задається через аргументи методу.

```
In [20]: df.interpolate()
```

Out[20]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	59500.0	Yes
5	France	35.0	58000.0	Yes
6	Spain	41.5	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

Ще одним методом яким можна скористатися для заповнення пропусків, метод .iloc[ ], але якщо даних забагато, цю функцію використовувати не доцільно.

```
In [26]: df.iloc[4,2]=55555
         df.iloc[6,1]=33
         df
```

Out[26]:

	Country	Age	Salary	Purchased
0	France	44.0	72000	No
1	Spain	27.0	48000	Yes
2	Germany	30.0	54000	No
3	Spain	38.0	61000	No
4	Germany	40.0	55555	Yes
5	France	35.0	58000	Yes
6	Spain	33.0	52000	No
7	France	48.0	79000	Yes
8	Germany	50.0	83000	No
9	France	37.0	67000	Yes

Для того щоб позбутися NaN елементів, можно використати функцію .dropna(), в цьому випаду весь рядок буде видалено.

```
In [36]: df.dropna()
```

Out[36]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
5	France	35.0	58000.0	Yes
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

Також для обробки пропусків даних існує чотири стратегії які розглянемо далі:

In [11]:
```
from sklearn.impute import SimpleImputer
cop = df.iloc[:, :-1].values
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer.fit(cop[:, 1:3])
cop[:, 1:3] = imputer.transform(cop[:, 1:3])
cop
```

Out[11]:
```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, 55555.0],
       ['France', 35.0, 58000.0],
       ['Spain', 33.0, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

In [12]:
```
from sklearn.impute import SimpleImputer
cop = df.iloc[:, :-1].values
imputer = SimpleImputer(missing_values=np.nan, strategy='median')
imputer.fit(cop[:, 1:3])
cop[:, 1:3] = imputer.transform(cop[:, 1:3])
cop
```

Out[12]:
```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, 55555.0],
       ['France', 35.0, 58000.0],
       ['Spain', 33.0, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

In [13]:
```
from sklearn.impute import SimpleImputer
cop = df.iloc[:, :-1].values
imputer = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
imputer.fit(cop[:, 1:3])
cop[:, 1:3] = imputer.transform(cop[:, 1:3])
cop
```

Out[13]:
```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, 55555.0],
       ['France', 35.0, 58000.0],
       ['Spain', 33.0, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

```
In [14]: from sklearn.impute import SimpleImputer
         cop = df.iloc[:, :-1].values
         imputer = SimpleImputer(missing_values=np.nan, strategy='constant')
         imputer.fit(cop[:, 1:3])
         cop[:, 1:3] = imputer.transform(cop[:, 1:3])
         cop
```

```
Out[14]: array([['France', 44.0, 72000.0],
                ['Spain', 27.0, 48000.0],
                ['Germany', 30.0, 54000.0],
                ['Spain', 38.0, 61000.0],
                ['Germany', 40.0, 55555.0],
                ['France', 35.0, 58000.0],
                ['Spain', 33.0, 52000.0],
                ['France', 48.0, 79000.0],
                ['Germany', 50.0, 83000.0],
                ['France', 37.0, 67000.0]], dtype=object)
```

4. Необхідно провести кодування категоріальних даних для залежних та незалежних змінних.

Багато алгоритмів машинного навчання очікують числові вхідні дані, тому нам потрібно з'ясувати спосіб представлення наших категоріальних даних чисельним чином.

Одним з рішень цього було б довільне присвоєння числового значення для кожної категорії і відображення набору даних з вихідних категорій в кожне відповідне число. Наприклад, давайте подивимося на стовпець Purchased в нашому наборі даних. Для кодування цих даних, можна порівняти кожне значення з числом.

```
In [50]: y=df['Purchased'].map({'No': 0, 'Yes': 1 })
         y
```

```
Out[50]: 0    0
         1    1
         2    0
         3    0
         4    1
         5    1
         6    0
         7    1
         8    0
         9    1
         Name: Purchased, dtype: int64
```

Цей процес відомий як Label Encoding і sklearn може зробити це за нас.

In [53]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
y= le.fit_transform(y)
print(y)
```

```
[0 1 0 0 1 1 0 1 0 1]
```

In [21]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder='passthrough')
cop = np.array(ct.fit_transform(cop))
cop
```

Out[21]:
```
array([[0.0, 1.0, 0.0, 0.0, 44.0, 72000.0],
       [1.0, 0.0, 0.0, 1.0, 27.0, 48000.0],
       [1.0, 0.0, 1.0, 0.0, 30.0, 54000.0],
       [1.0, 0.0, 0.0, 1.0, 38.0, 61000.0],
       [1.0, 0.0, 1.0, 0.0, 40.0, 55555.0],
       [0.0, 1.0, 0.0, 0.0, 35.0, 58000.0],
       [1.0, 0.0, 0.0, 1.0, 33.0, 52000.0],
       [0.0, 1.0, 0.0, 0.0, 48.0, 79000.0],
       [1.0, 0.0, 1.0, 0.0, 50.0, 83000.0],
       [0.0, 1.0, 0.0, 0.0, 37.0, 67000.0]], dtype=object)
```

5. Розділити набір даних на тестову та тренувальну вибірку даних.

Перший спосіб:

In [19]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=1)
print(X_train)
print(X_test)
print(Y_train)
print(Y_test)
```

```
[[0.0 0.0 1.0 33.0 52000.0]
 [0.0 1.0 0.0 40.0 55555.0]
 [1.0 0.0 0.0 44.0 72000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [1.0 0.0 0.0 48.0 79000.0]
 [0.0 1.0 0.0 50.0 83000.0]
 [1.0 0.0 0.0 35.0 58000.0]]
[[0.0 1.0 0.0 30.0 54000.0]
 [1.0 0.0 0.0 37.0 67000.0]]
[0 1 0 0 1 1 0 1]
[0 1]
```

Другий спосіб:

In [85]:
```
from sklearn.model_selection import train_test_split
X, y = np.arange(10).reshape((5, 2)), range(5)
X_trn, X_tst, y_trn, y_tst = train_test_split(X, y, test_size=0.2, random_state=0)
print(X_trn)
print(X_tst)
print(y_trn)
print(y_tst)
```

```
[[0 1]
 [2 3]
 [6 7]
 [8 9]]
[[4 5]]
[0, 1, 3, 4]
[2]
```

Третій спосіб:

In [100]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
train, test = train_test_split(df, test_size=0.2)
print(train)
print('-----------------------------------')
print(test)
```

```
   Country   Age   Salary Purchased
9   France  37.0  67000.0       Yes
6    Spain   NaN  52000.0        No
2  Germany  30.0  54000.0        No
8  Germany  50.0  83000.0        No
5   France  35.0  58000.0       Yes
4  Germany  40.0      NaN       Yes
0   France  44.0  72000.0        No
3    Spain  38.0  61000.0        No
-----------------------------------
  Country   Age   Salary Purchased
1   Spain  27.0  48000.0       Yes
7  France  48.0  79000.0       Yes
```

6. Виконати масштабування даних.

```
In [9]: from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train[:, 3:] = sc.fit_transform(X_train[:, 3:])
X_test[:, 3:] = sc.transform(X_test[:, 3:])
print(X_train)
print(X_test)
```

```
[[0.0 0.0 1.0 -1.425716469028381 -0.8222256503648497]
 [0.0 1.0 0.0 0.16863313074529238 -1.131769424619852]
 [1.0 0.0 0.0 0.6592022383679611 0.7254932209101616]
 [0.0 0.0 1.0 -0.07665142306604199 -0.12575215829109468]
 [0.0 0.0 1.0 -1.425716469028381 -1.131769424619852]
 [1.0 0.0 0.0 1.1497713459906298 1.2671948258564156]
 [0.0 1.0 0.0 1.3950558998019642 1.5767386001114179]
 [1.0 0.0 0.0 -0.44457825378304355 -0.3579099889823464]]
[[0.0 1.0 0.0 -1.0577896383113794 -0.6674537632373486]
 [1.0 0.0 0.0 -0.19929369997170918 0.33856350309140876]]
```

Висновки:

Почавши з основ, ми познайомилися з природним робочим процесом Jupyter Notebooks. Навчилися перевіряти наявність пропущених даних та заповнювати їх декількома способами. Провели кодування категоріальних даних для залежних та незалежних змінних. Розділили набір даних на тестову та тренувальну вибірку даних.

Під час даної практичної роботи отримано практичні навички при підготовці даних для подальшого аналізу та моделювання.